Exorcising Spectres with Secure Compilers – CSC Report

Marco Patrignani[1,2]

23rd June 2021

1 CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

2 Stanford University

Talk Outline

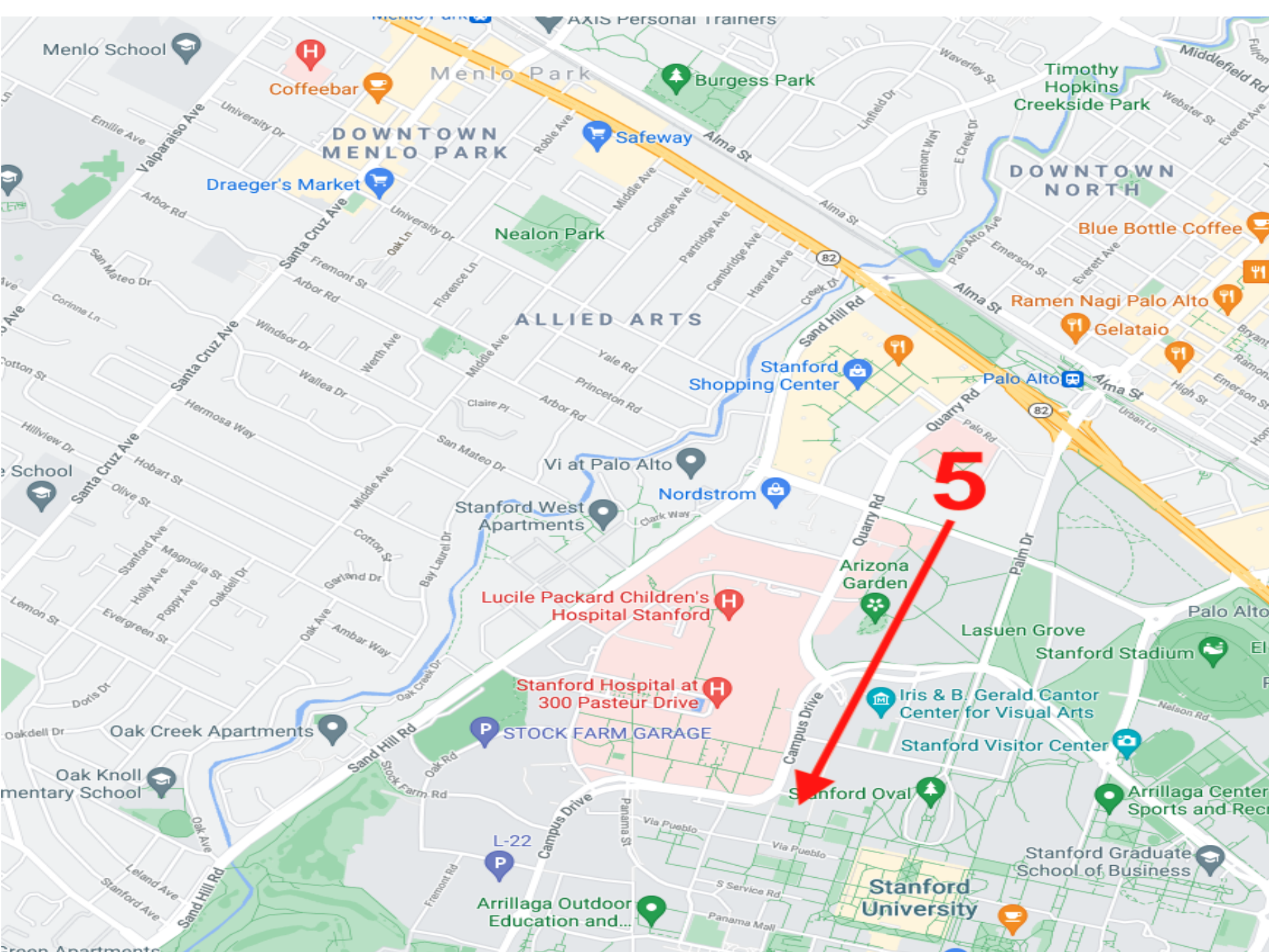
Who Am I ?

Exorcising Spectres with Secure Compilers Patrignani & Guarnieri CCS'21

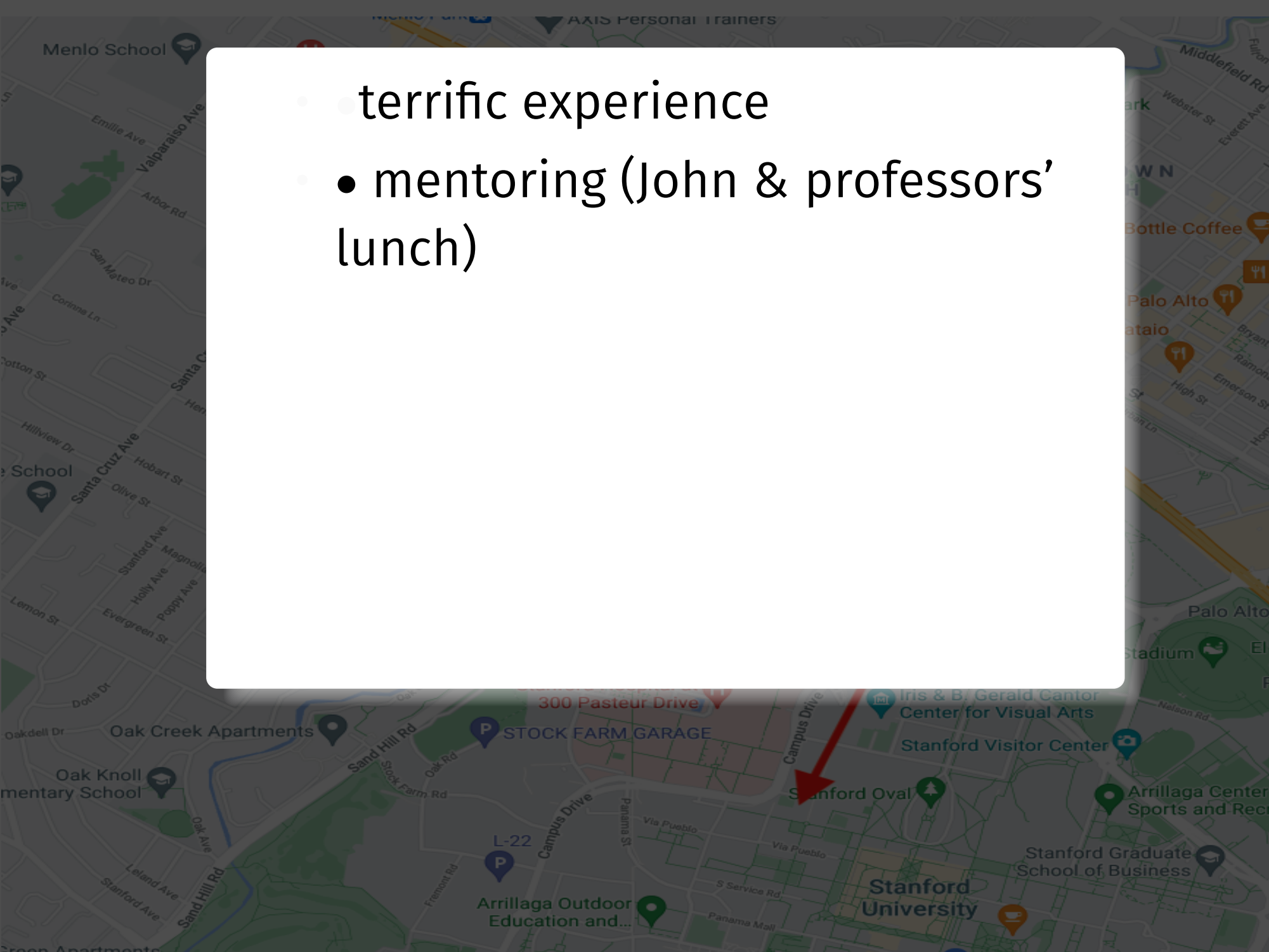
Future Outlook

Who Am I ?

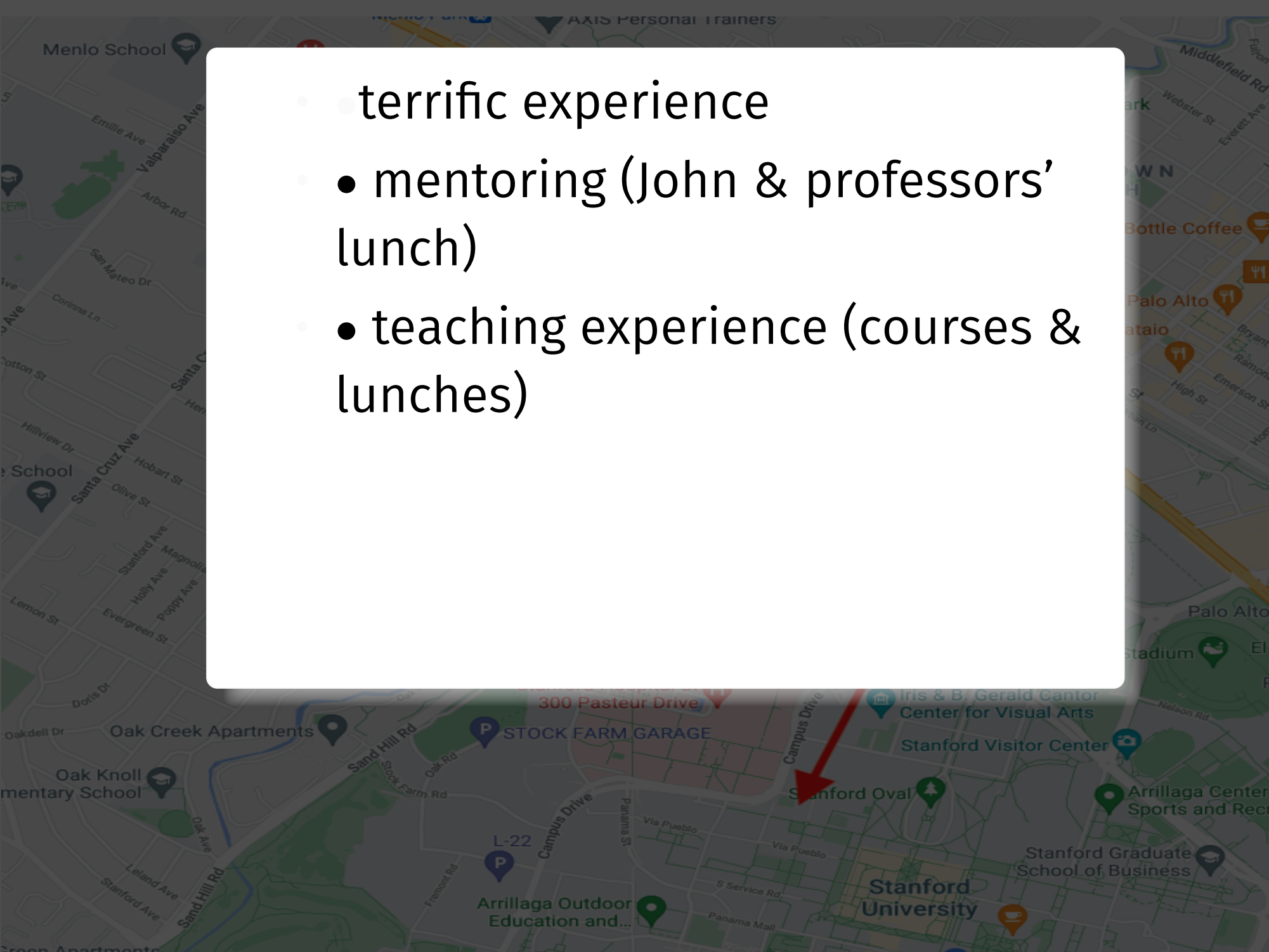

Germany
Belgium
Luxembourg
France
Czech Rep
Slovenia
Croatia
Bruges
Antwerp
Brussels
Lille
Amiens
Saint-Quentin
Reims
Düsseldorf
Cologne
Leipzig
Dresden
Görlitz
Chemnitz
Gera
Liberec
Prague
Wiesbade
Nuremberg
Karlsruhe
Strasbourg
Munich
Linz
Freiburg
Konstanz
Basel
Zurich
Dijon
Besançon
Bourges
Chalon-sur-Saône
Clermont-Ferrand
Lyon
Grenoble
Valence
Turin
Milan
Venice
Padua
Trieste
Rijeka
Pula
Parma
Asti
Genoa
Bologna
Ravenna
Zadar
Avignon
3
4-6
2
1
MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS
CISPA-Stanford Center
FOR CYBERSECURITY
KU LEUVEN
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA


Menlo School
Coffeebar
Menlo Park
Burgess Park
Timothy Hopkins Creekside Park
DOWNTOWN MENLO PARK
Safeway
Draeger's Market
Nealon Park
DOWNTOWN NORTH
Blue Bottle Coffee
Ramen Nagi Palo Alto
Gelataio
ALLIED ARTS
Stanford Shopping Center
Palo Alto
Vi at Palo Alto
Nordstrom
Stanford West Apartments
5
Arizona Garden
Lucile Packard Children's Hospital Stanford
Lasuen Grove
Stanford Stadium
Stanford Hospital at 300 Pasteur Drive
Iris & B. Gerald Cantor Center for Visual Arts
STOCK FARM GARAGE
Oak Creek Apartments
Stanford Visitor Center
Oak Knoll
Stanford Oval
Arrillaga Center Sports and Rec
L-22
Stanford Graduate School of Business
Arrillaga Outdoor Education and...
Stanford University
Alma St
Sand Hill Rd
Quarry Rd
Campus Drive
Santa Cruz Ave
Valparaiso Ave
University Dr
Middle Ave
Oak Creek Dr
Stock Farm Rd
Via Pueblo
Panama St
S Service Rd
Panama Mall
Fremont Rd
Palm Dr
Nelson Rd

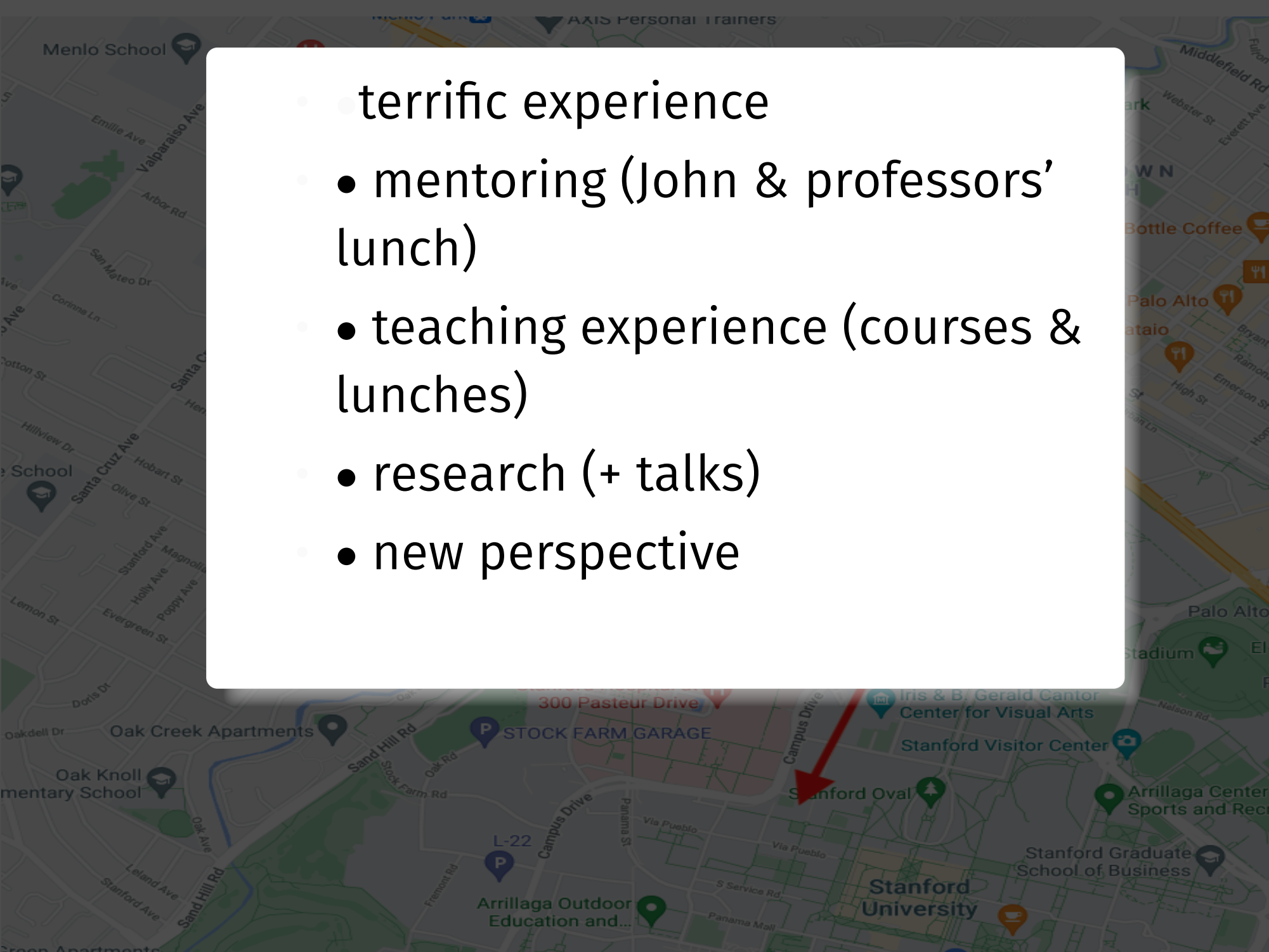
- terrific experience
- mentoring (John & professors' lunch)



- terrific experience
- mentoring (John & professors' lunch)
- teaching experience (courses & lunches)



- terrific experience
- mentoring (John & professors' lunch)
- teaching experience (courses & lunches)
- research (+ talks)



- terrific experience
- mentoring (John & professors' lunch)
- teaching experience (courses & lunches)
- research (+ talks)
- new perspective

- terrific experience
- mentoring (John & professors' lunch)
- teaching experience (courses & lunches)
- research (+ talks)
- new perspective
- skiing (who'd have thought?)

Exorcising Spectres with Secure Compilers

Patrignani & Guarnieri CCS'21

Speculative execution + branch prediction

Size of array **A**

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

Speculative execution + branch prediction

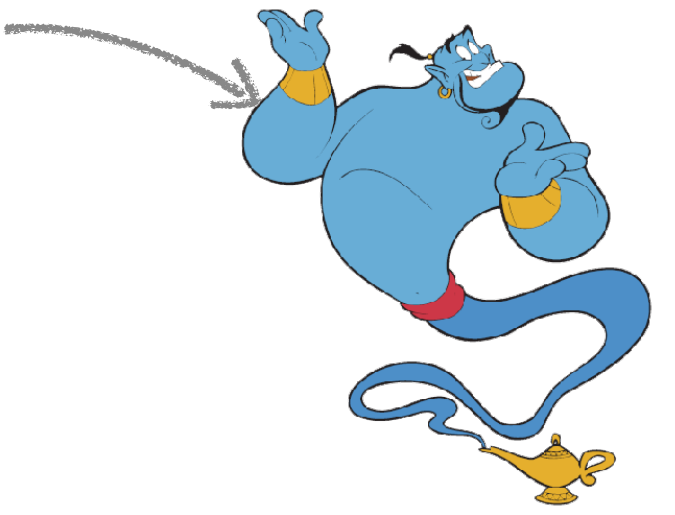
Size of array **A**

```
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***!

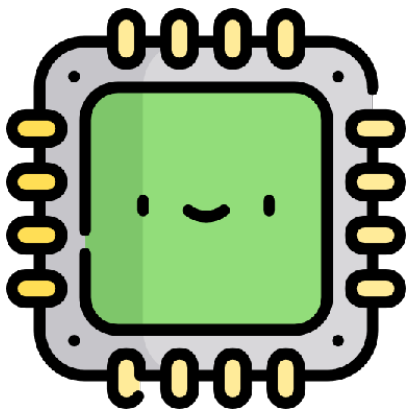
- Architectural (ISA) state
- Microarchitectural state

Spectre V1

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

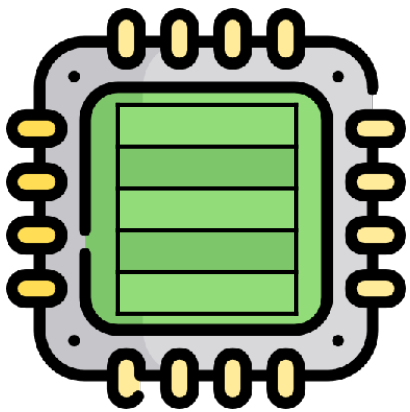
Spectre V1

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



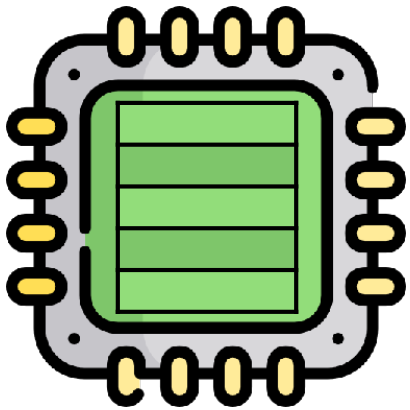
Spectre V1

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

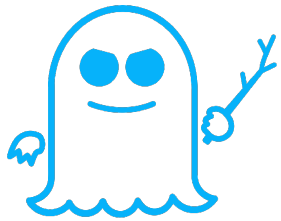


Spectre V1

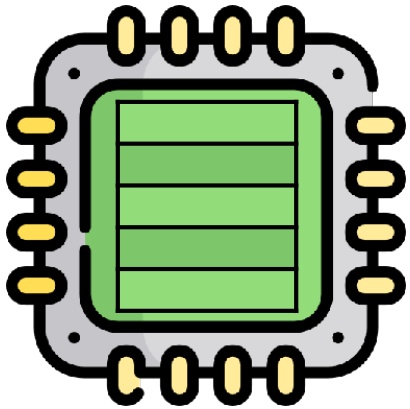
```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



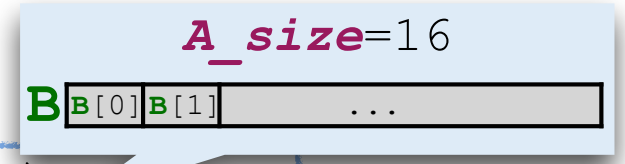
Spectre V1



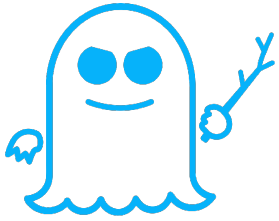
```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

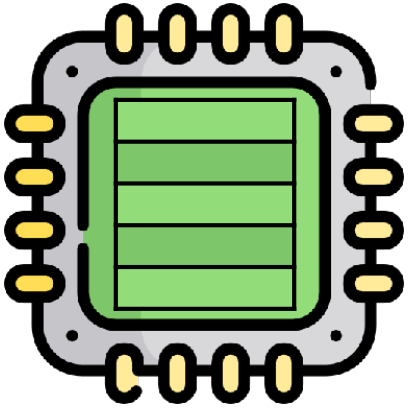


Spectre V1

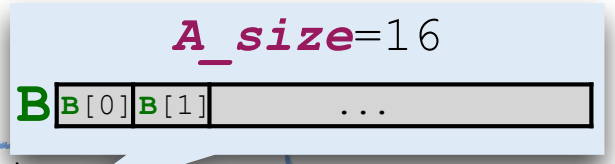



```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

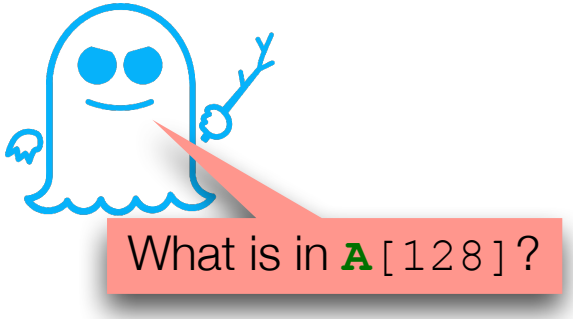


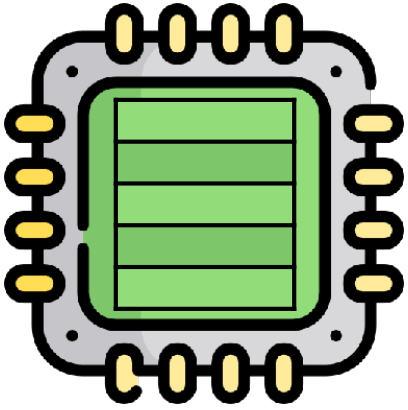


Spectre V1

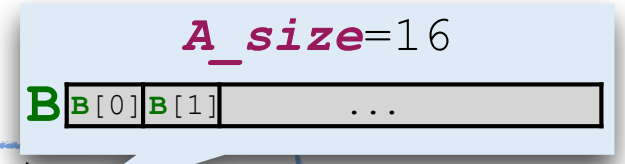



```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

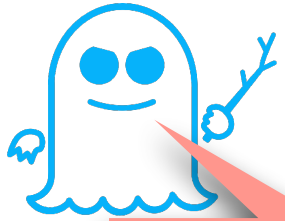




Spectre V1

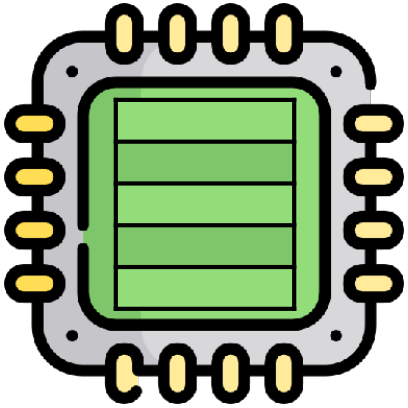



```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

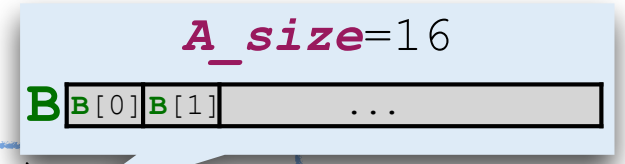


What is in A[128]?

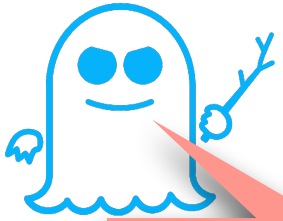
1) Training



Spectre V1

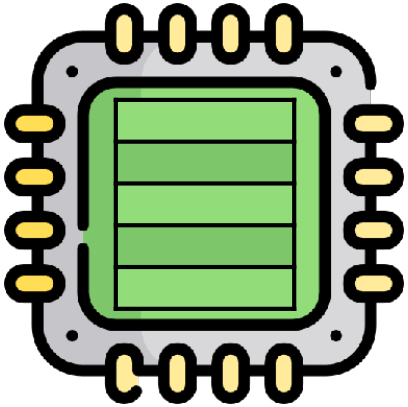



```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

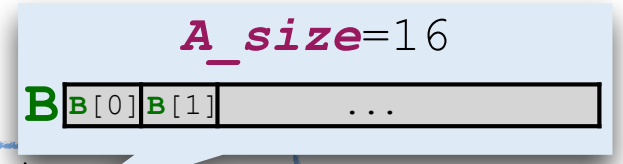


What is in A[128]?

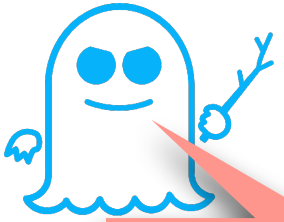
1) Training 



Spectre V1

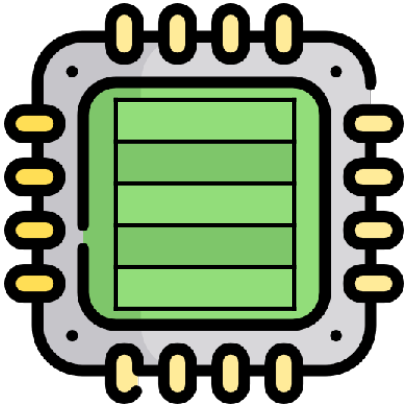



```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

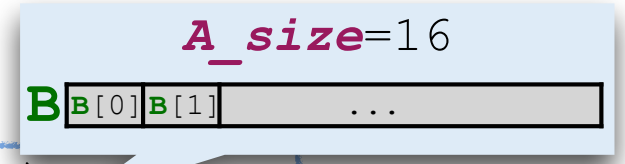


What is in A[128]?

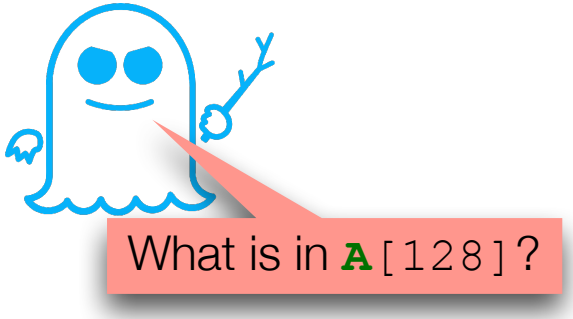
1) Training  f(0);



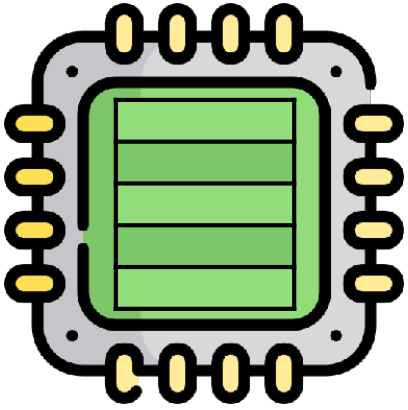
Spectre V1



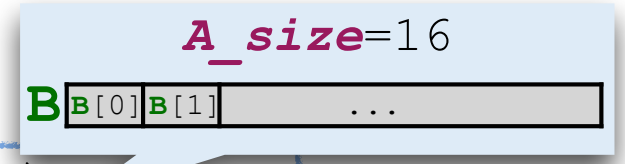

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



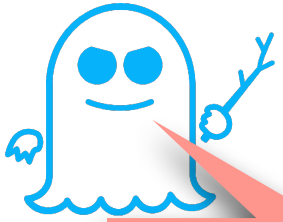
1) Training  `f(0);f(1);`



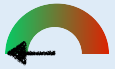
Spectre V1

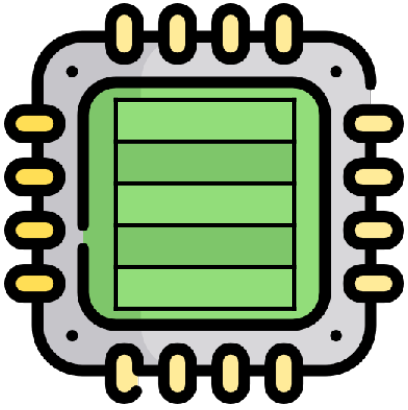



```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

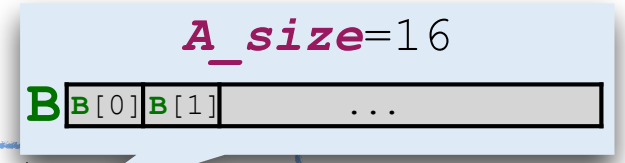


What is in A[128]?

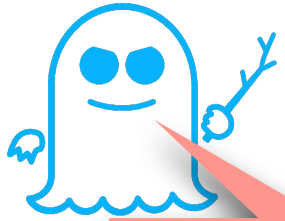
1) Training  f(0);f(1);f(2); …



Spectre V1



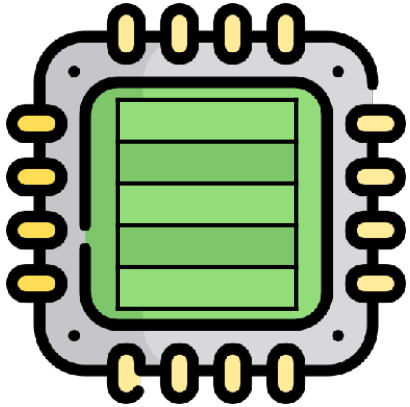

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



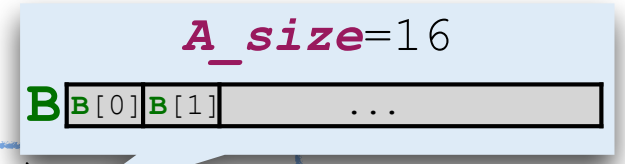
What is in `A[128]`?

1) Training  `f(0);f(1);f(2); …`

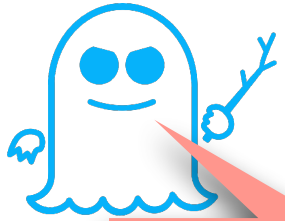
2) Prepare cache



Spectre V1



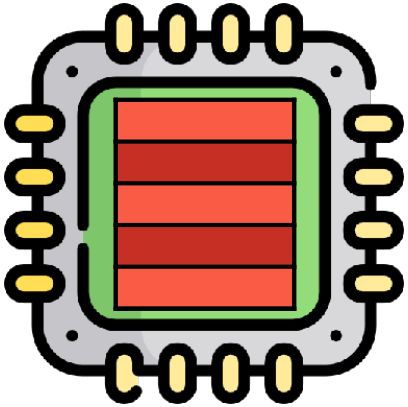

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



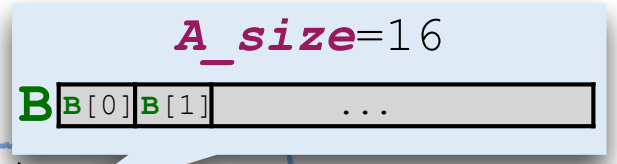
What is in `A[128]`?

1) Training  `f(0);f(1);f(2); …`

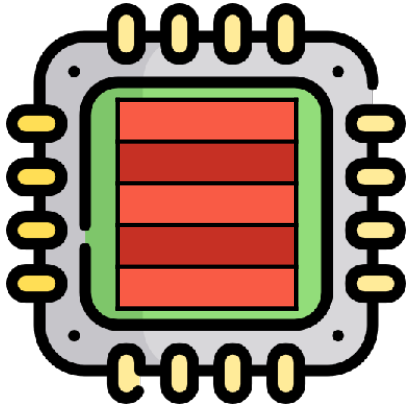
2) Prepare cache

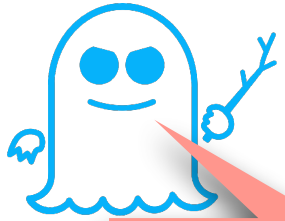


Spectre V1

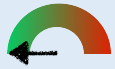



```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```





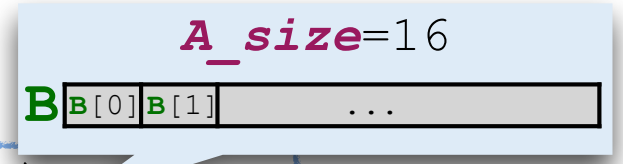
What is in `A[128]`?

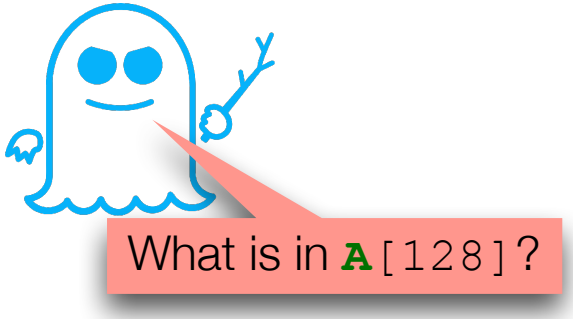
1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

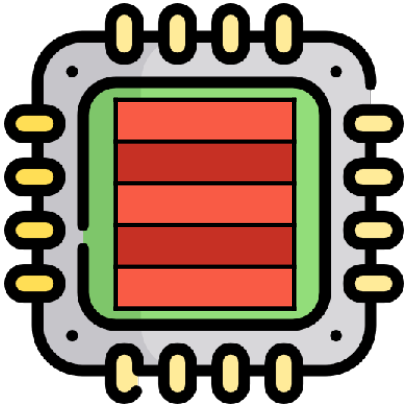
3) Run with `x = 128`

Spectre V1






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

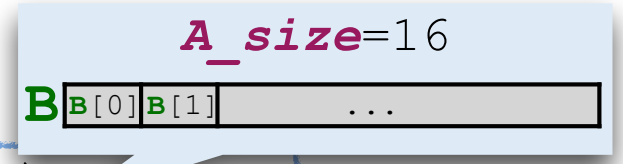


1) Training  `f(0);f(1);f(2); …`

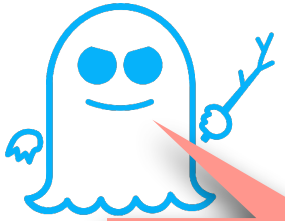
2) Prepare cache

3) Run with `x = 128`

Spectre V1



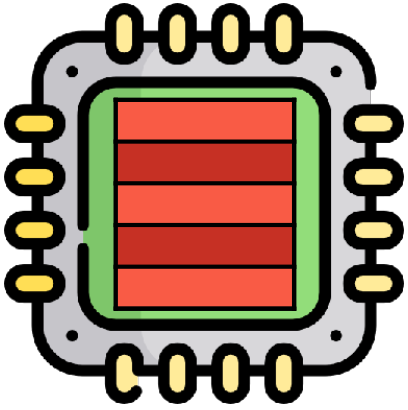

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



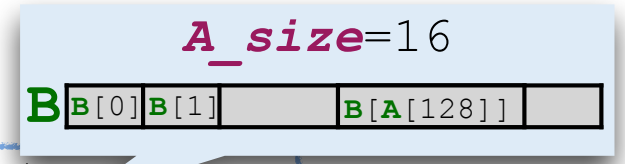
1) Training  `f(0);f(1);f(2); …`

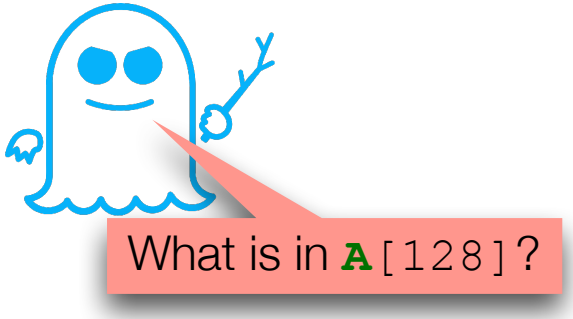
2) Prepare cache

3) Run with ***x*** `= 128`

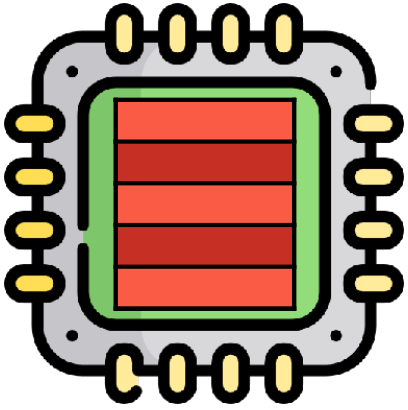


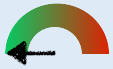
Spectre V1






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

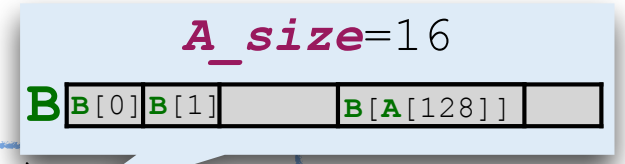


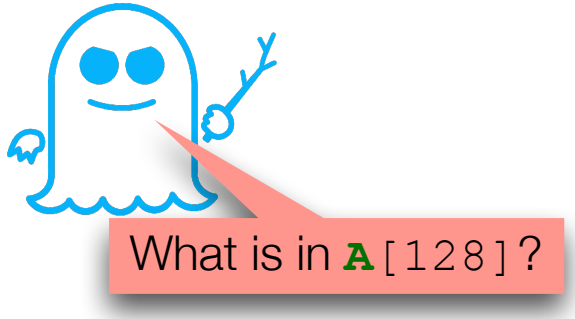
1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

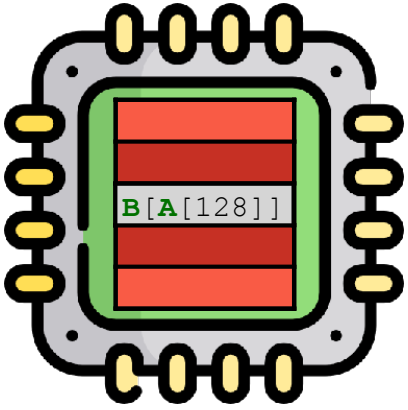
3) Run with `x = 128`

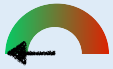
Spectre V1






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

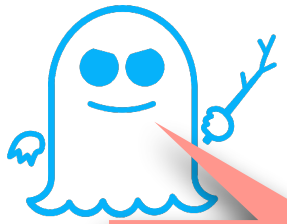


1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with ***x*** `= 128`

Spectre V1



What is in `A[128]`?

`A_size`=16

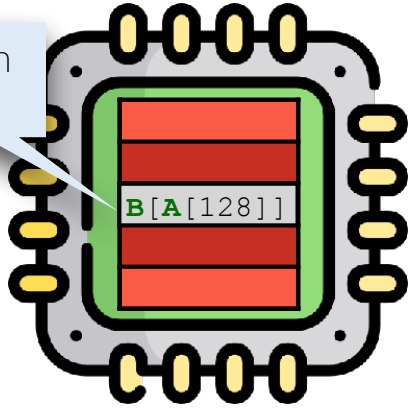

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Training  `f(0);f(1);f(2); …`

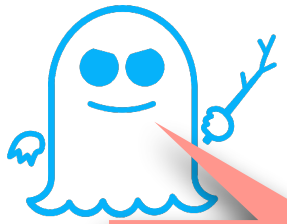
2) Prepare cache

3) Run with `x = 128`

Depends on `A[128]`



Spectre V1

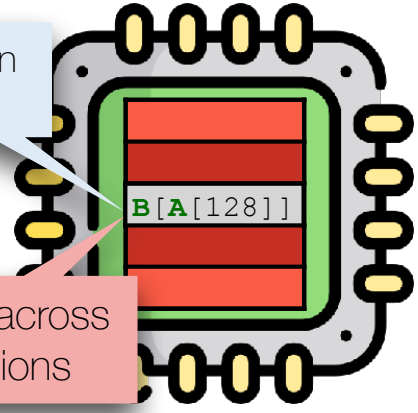


What is in `A[128]`?

`A_size`=16

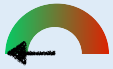
B | B[0] | B[1] | | B[A[128]] | |

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



Depends on `A[128]`

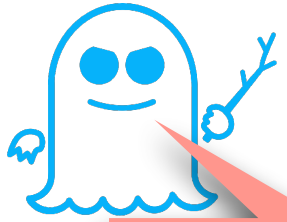
Persistent across speculations

1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with `x = 128`

Spectre V1

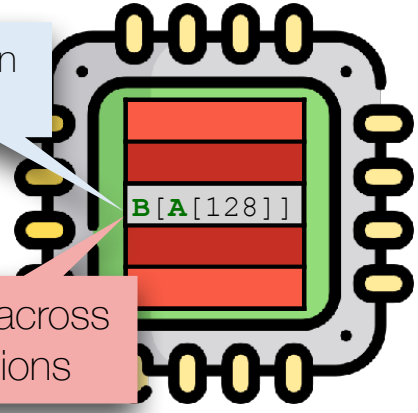


What is in `A[128]`?

`A_size`=16

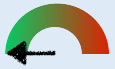
B | B[0] | B[1] | | B[A[128]] | |

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



Depends on `A[128]`

Persistent across speculations

1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with `x = 128`

4) Extract from cache

Compiler-level countermeasures

Compiler-level countermeasures

For ***Spectre V1***

Injecting speculation barriers

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

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

- In x86, ***LFENCE*** act as ***speculation barrier***
- Compiler injects LFENCE after each branch instruction
  - Microsoft Visual C++
  - Intel ICC
- Effectively ***stop speculative execution***!

Speculative load-hardening (SLH)

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

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

- Injects ***data dependencies*** and ***masking operations***
  - Combines ***conditional moves*** and ***binary operations***
- Stops ***speculative leaks***
- Does not block speculative execution!
- Implemented in Clang

Goal

1. formalise `lfence` & `SLH` compilers

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS
4. use secure compilation to preserve SS: RSSC

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS
4. use secure compilation to preserve SS: RSSC
5. prove the compilers attain RSSC

Goal Up Next

2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS
4. use secure compilation to preserve SS: RSSC

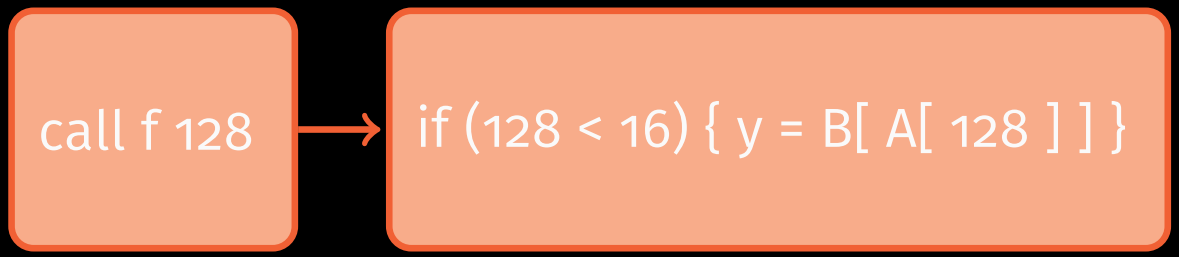
Speculative Semantics 101 “Spectector …” S&P’20

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3

call f 128

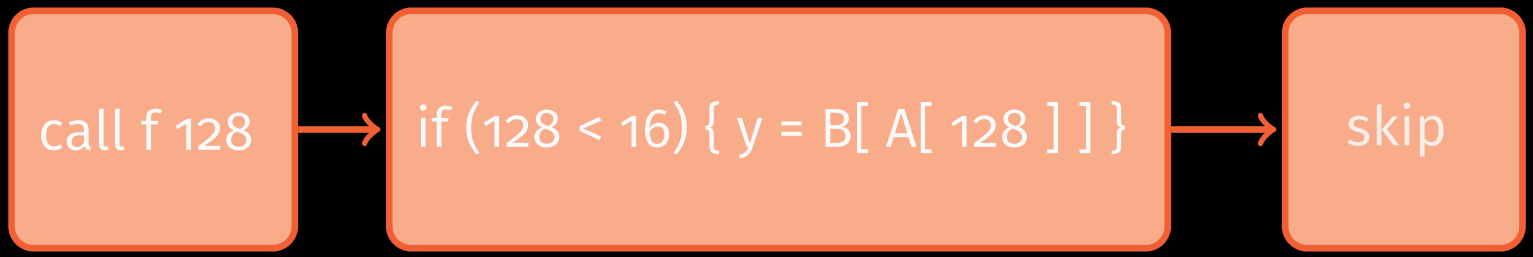
Speculative Semantics 101 “Spectector …” S&P’20

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3



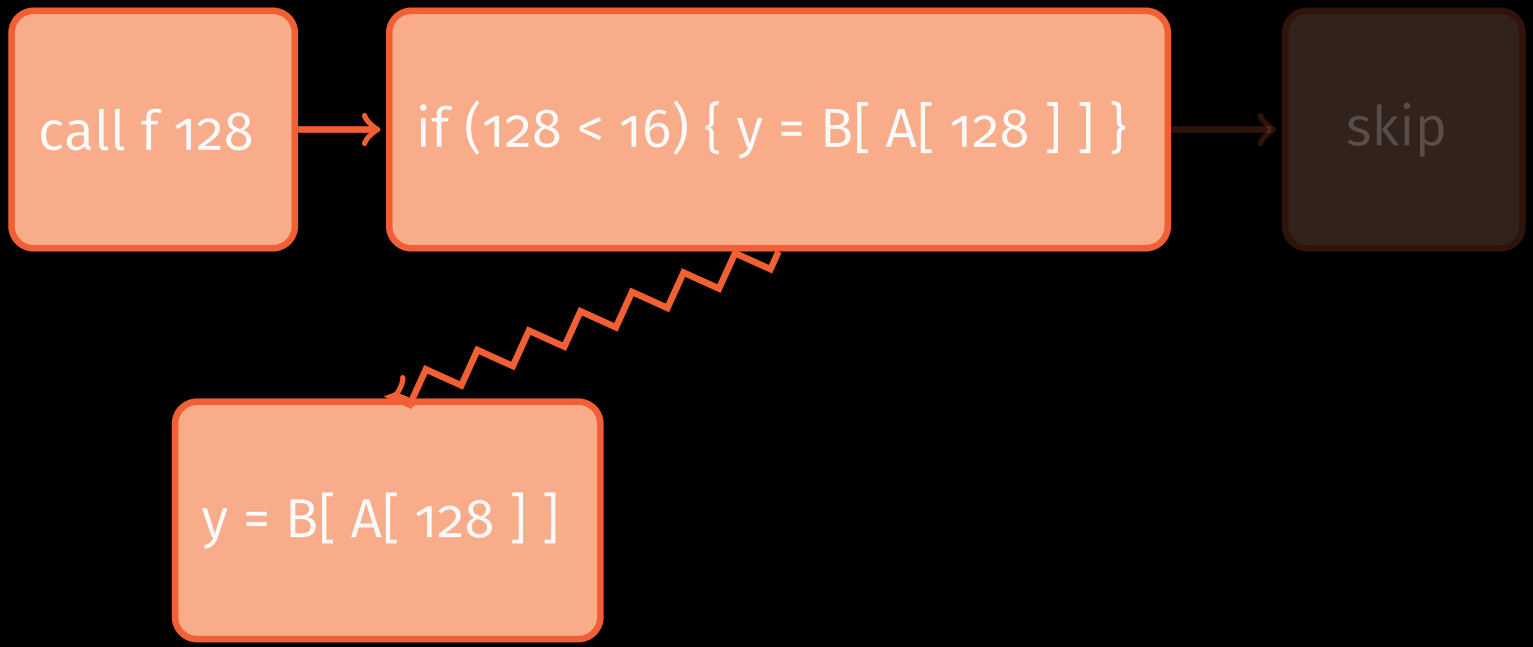
Speculative Semantics 101 “Spectector …” S&P’20

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3



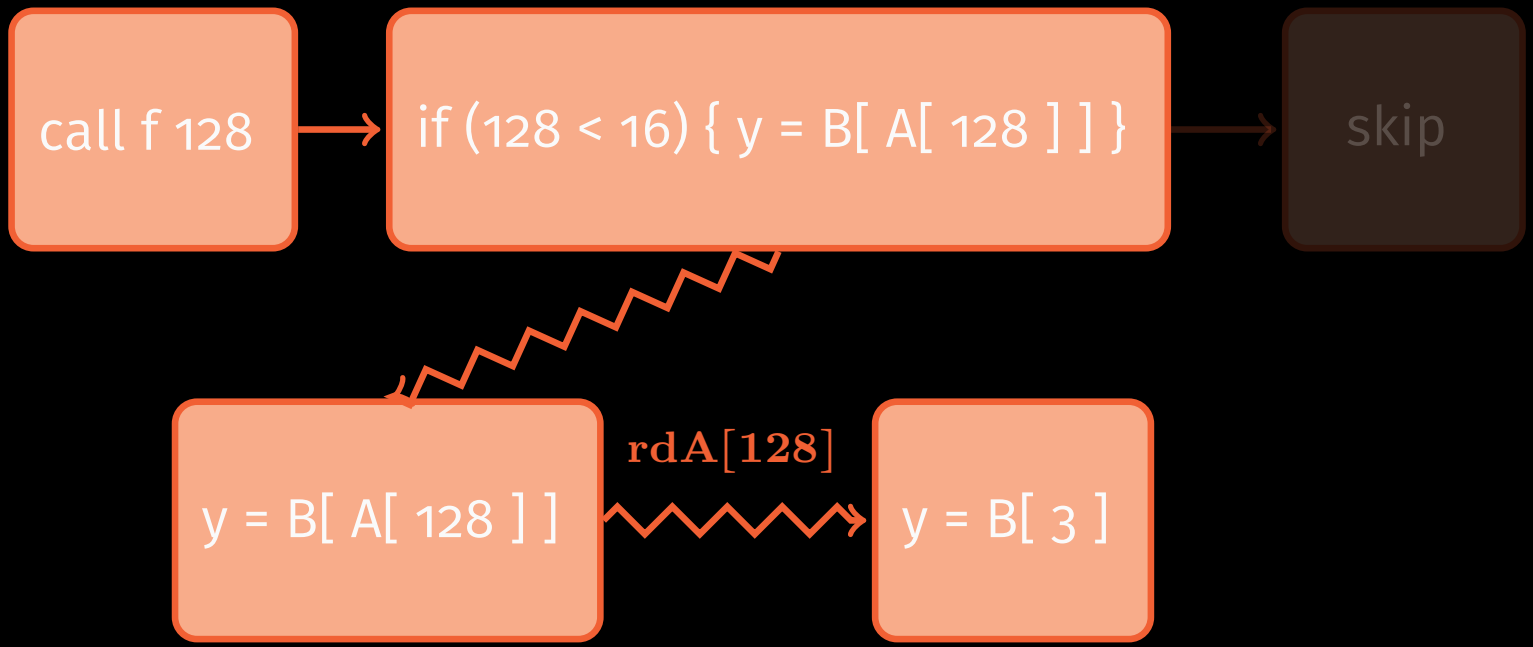
Speculative Semantics 101 “Spectector …” S&P’20

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3



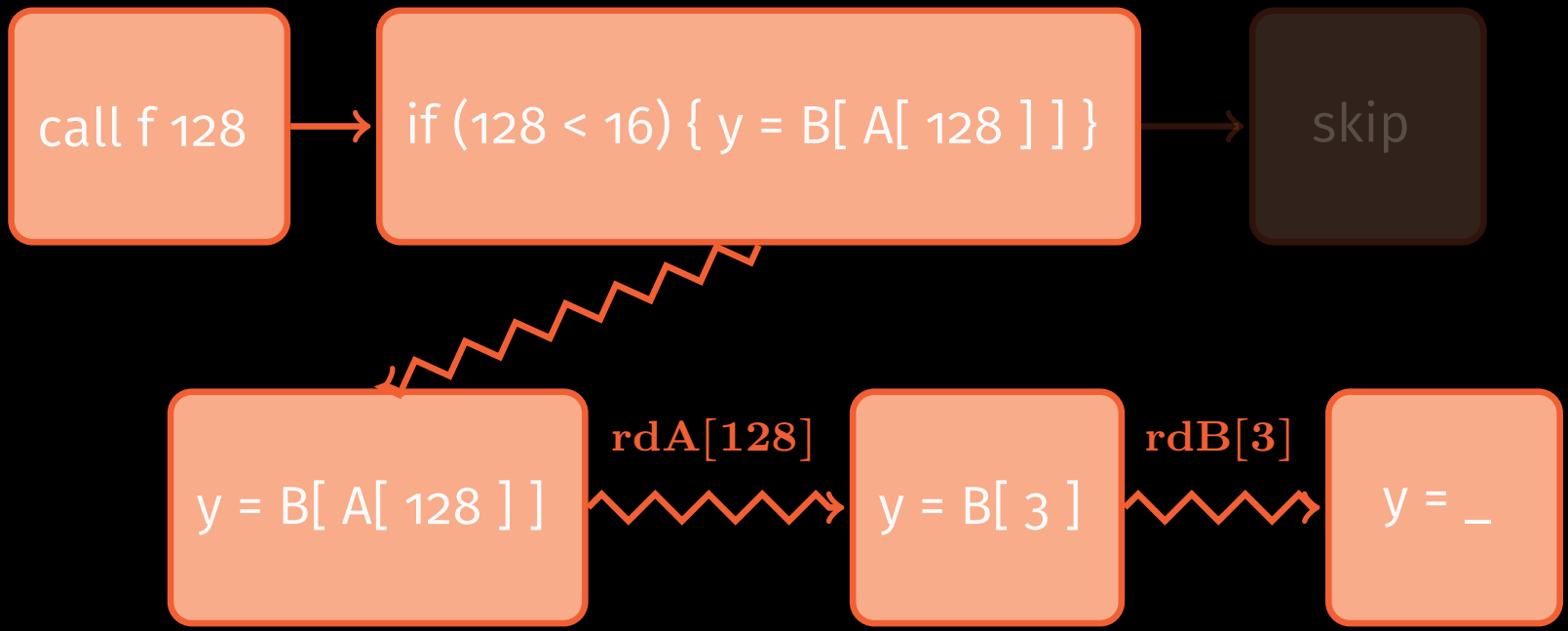
Speculative Semantics 101 “Spectector …” S&P’20

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3



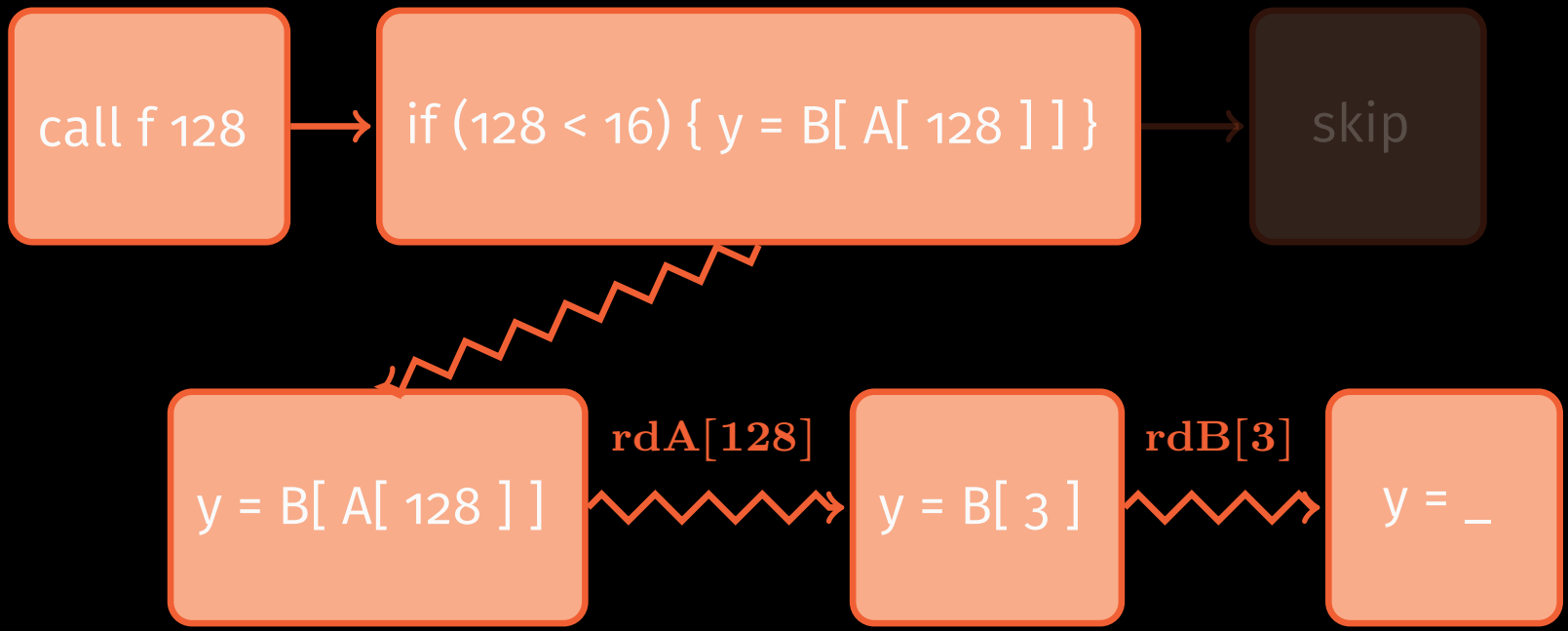
Speculative Semantics 101 “Spectector …” S&P’20

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3



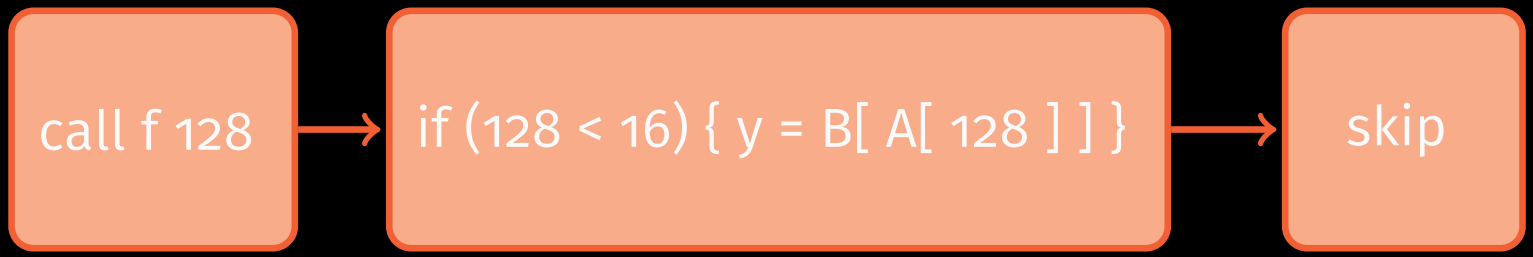
Speculative Semantics 101 “Spectector …” S&P’20

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3



Speculative Semantics 101 “Spectector …” S&P’20

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3



$\mathbf{rdA[128]}$ $\mathbf{rdB[3]}$

Speculative Safety (SS): Taint Tracking

void f (int x) $\mapsto$ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S

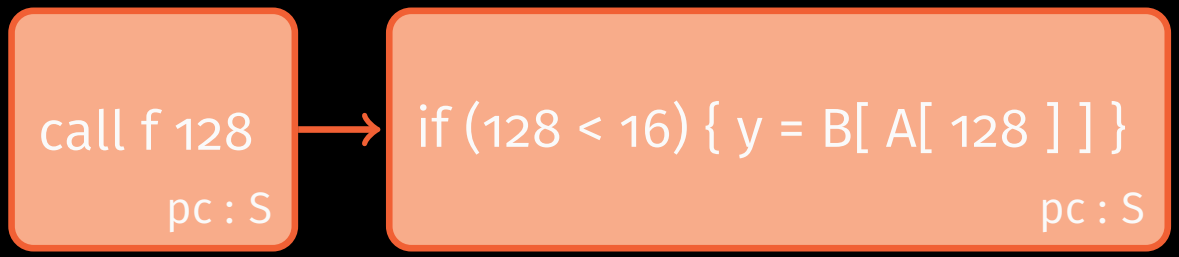
call f 128

pc : S

Speculative Safety (SS): Taint Tracking

void f (int x) $\mapsto$ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3

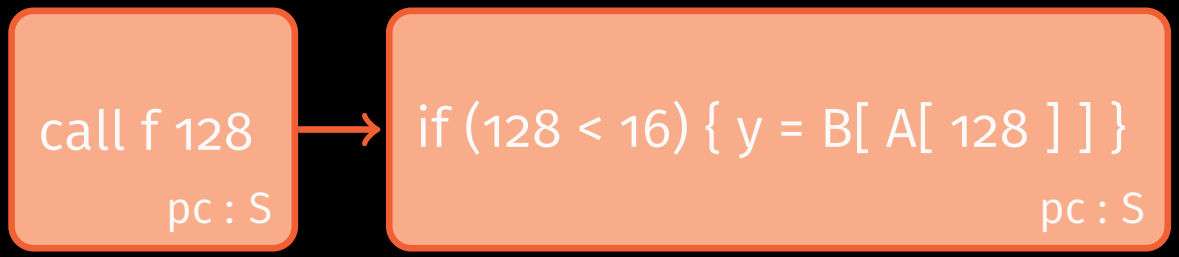
integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S



Speculative Safety (SS): Taint Tracking

void f (int x) $\mapsto$ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3

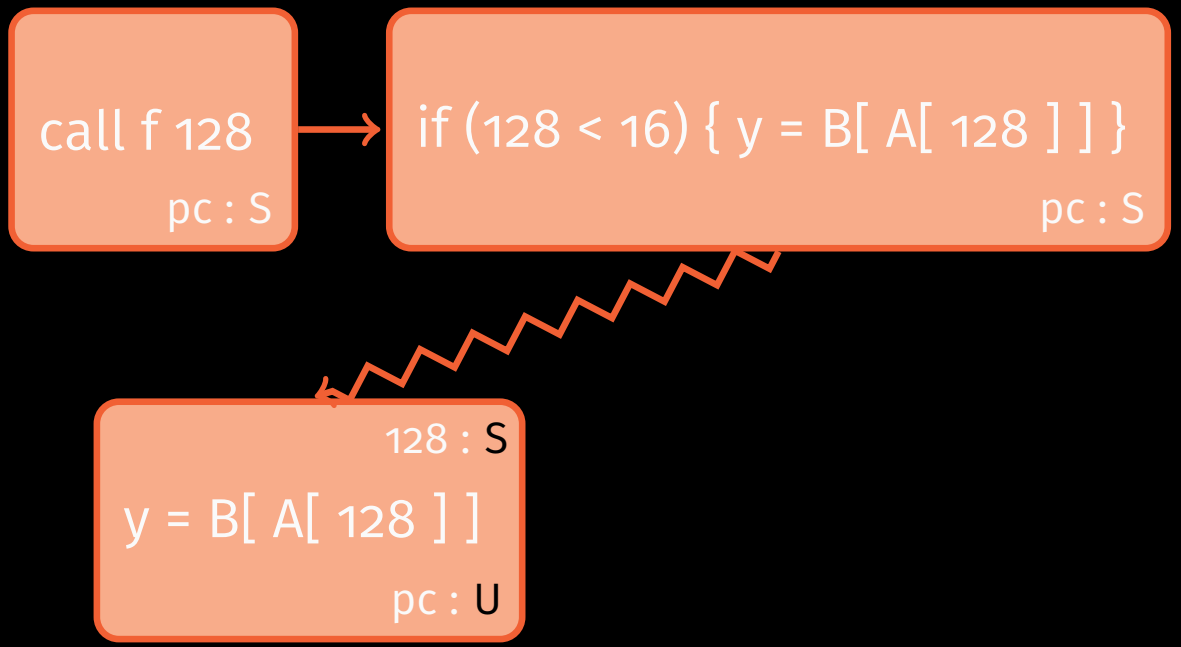
integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S



Speculative Safety (SS): Taint Tracking

void f (int x) $\mapsto$ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3

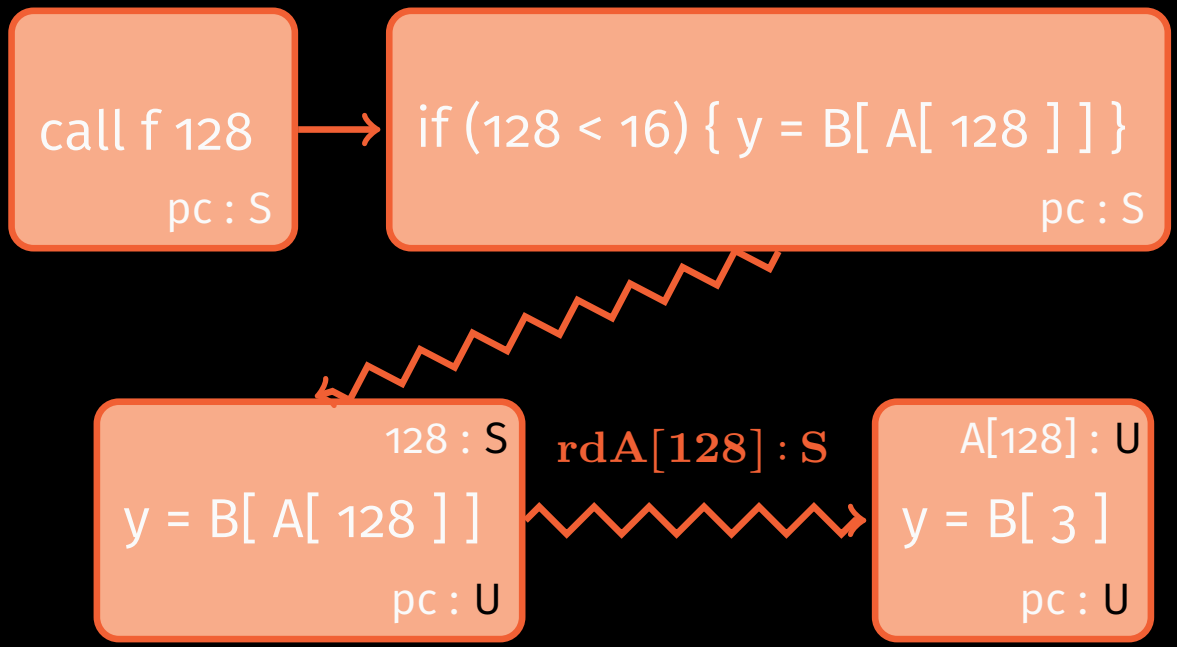
integrity lattice: $S \sqsubset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

void f (int x) $\mapsto$ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3

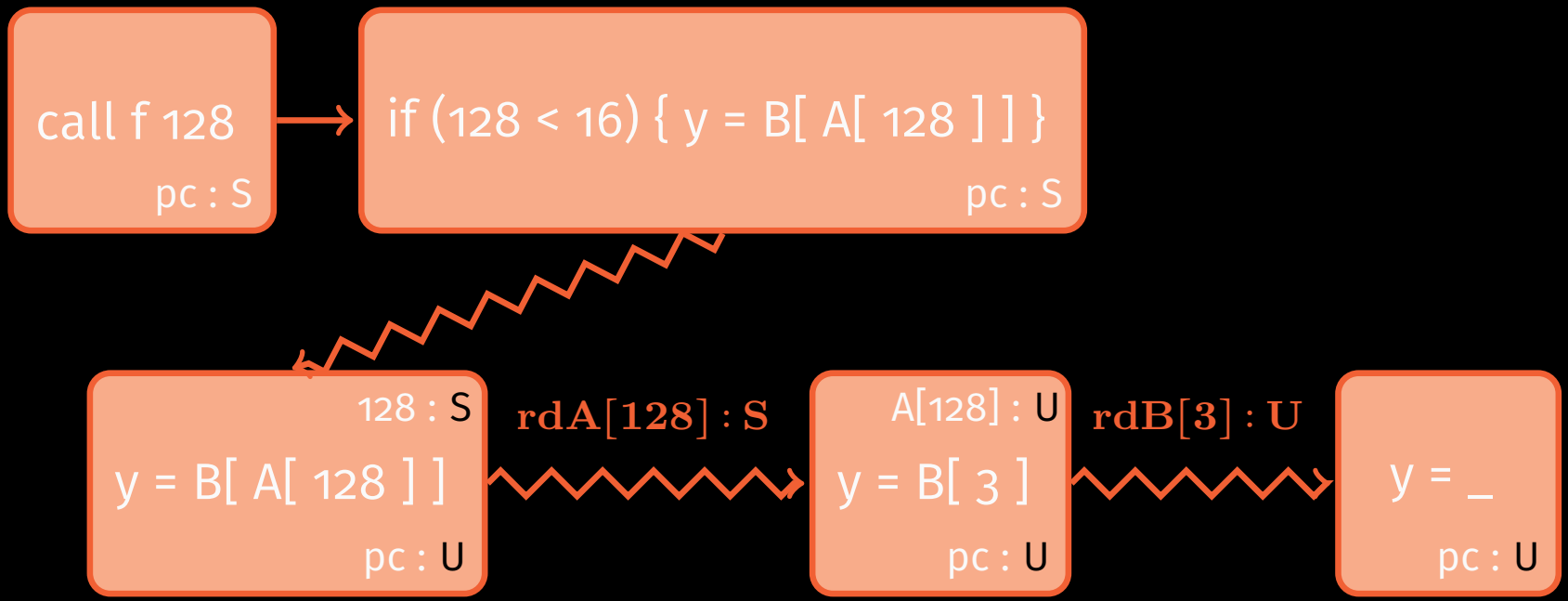
integrity lattice: $S \sqsubset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3

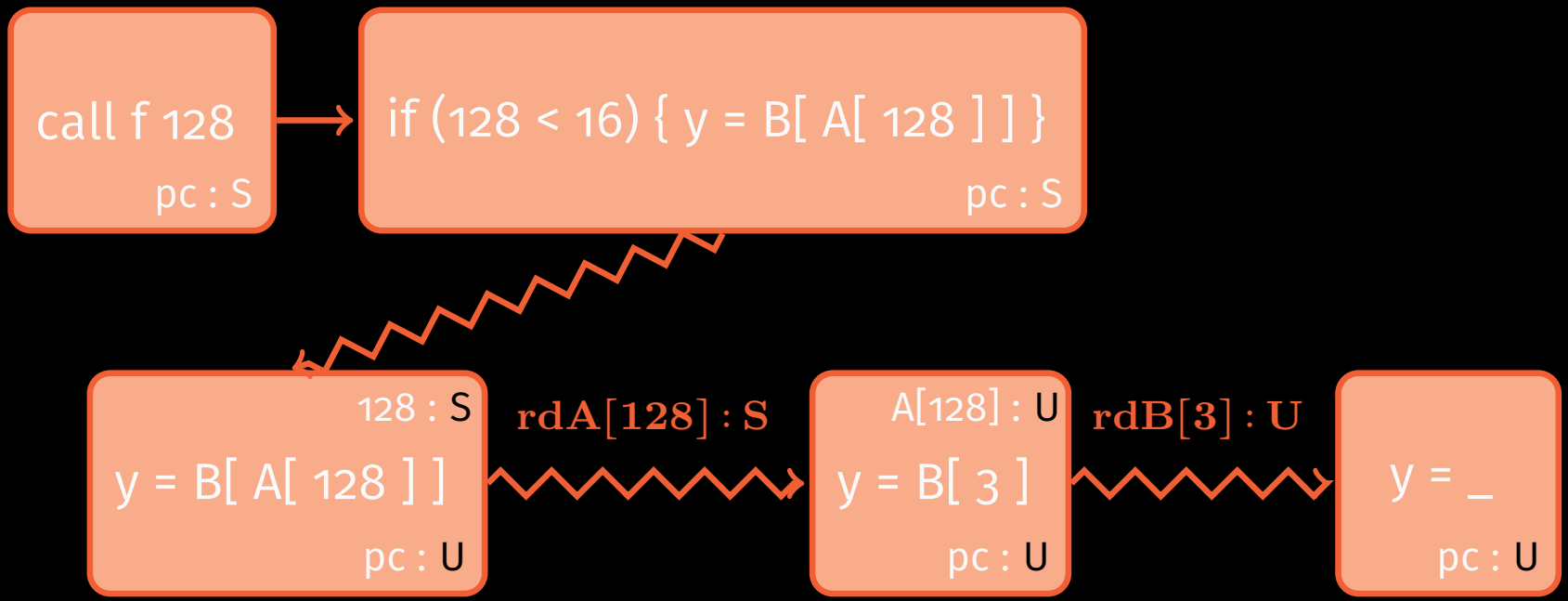
integrity lattice: $S \sqsubset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

void f (int x) ↦ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3

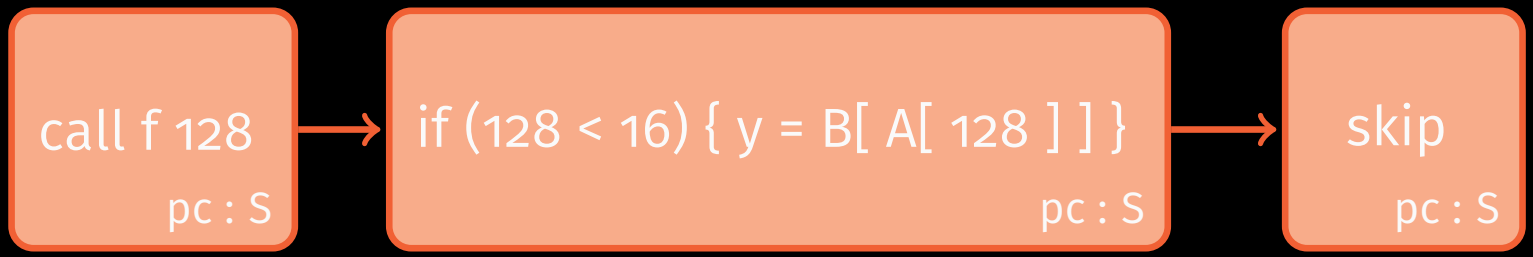
integrity lattice: $S \sqsubset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

void f (int x) $\mapsto$ if (x < A.size) { y = B[A[x]] } // A.size=16, A[128]=3

integrity lattice: $S \subset U$ $S \sqcap U = S$ U does not flow to S



$\mathbf{rdA[128] : S}$ $\mathbf{rdB[3] : U}$

SS-Preserving Compiler: RSSC & RSSP

$[\![\cdot]\!]$: RSSP $\stackrel{\text{def}}{=}$ if $\forall \mathsf{A}.\mathsf{A}\,[\mathsf{P}] : SS$ then $\forall \mathbf{A}.\mathbf{A}\,[[\![\mathsf{P}]\!]] : SS$

SS-Preserving Compiler: RSSC & RSSP

$[\![\cdot]\!] : \mathrm{RSSP} \stackrel{\text{def}}{=}$ if $\forall \mathrm{A}.\mathrm{A}\,[\mathrm{P}] : SS$ then $\forall \mathbf{A}.\mathbf{A}\,[[\![\mathrm{P}]\!]] : SS$

$[\![\cdot]\!] : \mathrm{RSSC} \stackrel{\text{def}}{=}$ if $\forall \mathbf{A}.\mathbf{A}\,[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{m}$ then $\exists \mathrm{A}.\mathrm{A}\,[\mathrm{P}] \rightsquigarrow \mathrm{m} \approx \mathbf{m}$

$\approx$= same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

SS-Preserving Compiler: RSSC & RSSP

$[\![\cdot]\!] : \text{RSSP} \stackrel{\text{def}}{=}$ if $\forall \mathrm{A}.\mathrm{A}[\mathrm{P}] : SS$ then $\forall \mathbf{A}.\mathbf{A}[[\![\mathrm{P}]\!]] : SS$

$[\![\cdot]\!] : \text{RSSC} \stackrel{\text{def}}{=}$ if $\forall \mathbf{A}.\mathbf{A}[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{m}$ then $\exists \mathrm{A}.\mathrm{A}[\mathrm{P}] \rightsquigarrow \mathrm{m} \approx \mathbf{m}$

$\approx$= same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- RSSC & RSSP are equivalent

SS-Preserving Compiler: RSSC & RSSP

$[\![\cdot]\!] : \text{RSSP} \stackrel{\text{def}}{=}$ if $\forall \mathrm{A}.\mathrm{A}\,[\mathrm{P}] : SS$ then $\forall \mathbf{A}.\mathbf{A}\,[[\![\mathrm{P}]\!]] : SS$

$[\![\cdot]\!] : \text{RSSC} \stackrel{\text{def}}{=}$ if $\forall \mathbf{A}.\mathbf{A}\,[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{m}$ then $\exists \mathrm{A}.\mathrm{A}\,[\mathrm{P}] \rightsquigarrow \mathrm{m} \approx \mathbf{m}$

$\approx$= same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- RSSC & RSSP are equivalent
- `lfence` : RSSC because it has no speculation ($\mathbf{pc}$: $\mathbf{S}$ always)

SS-Preserving Compiler: RSSC & RSSP

$[\![\cdot]\!] : \text{RSSP} \stackrel{\text{def}}{=}$ if $\forall \mathrm{A}.\mathrm{A}\,[\mathrm{P}] : SS$ then $\forall \mathbf{A}.\mathbf{A}\,[[\![\mathrm{P}]\!]] : SS$

$[\![\cdot]\!] : \text{RSSC} \stackrel{\text{def}}{=}$ if $\forall \mathbf{A}.\mathbf{A}\,[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{m}$ then $\exists \mathrm{A}.\mathrm{A}\,[\mathrm{P}] \rightsquigarrow \mathrm{m} \approx \mathbf{m}$

$\approx$= same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- RSSC & RSSP are equivalent
- `lfence` : RSSC because it has no speculation ($\mathbf{pc}$: $\mathbf{S}$ always)
- `SLH` : RSSC because masking taints as $\mathbf{S}$

RSSC **for** `lfence`

$$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}} \qquad \text{// A.size=16, A[128]=3}$$

$$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$$

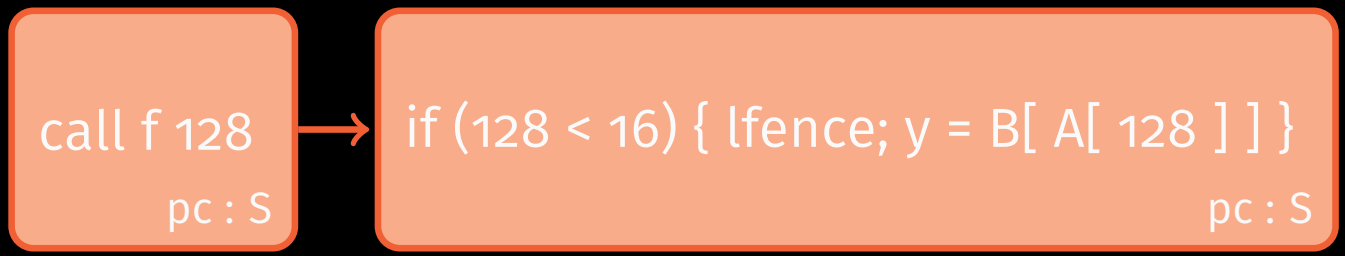
call f 128

pc : S

RSSC **for** `lfence`

$$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}} \qquad \text{// A.size=16, A[128]=3}$$

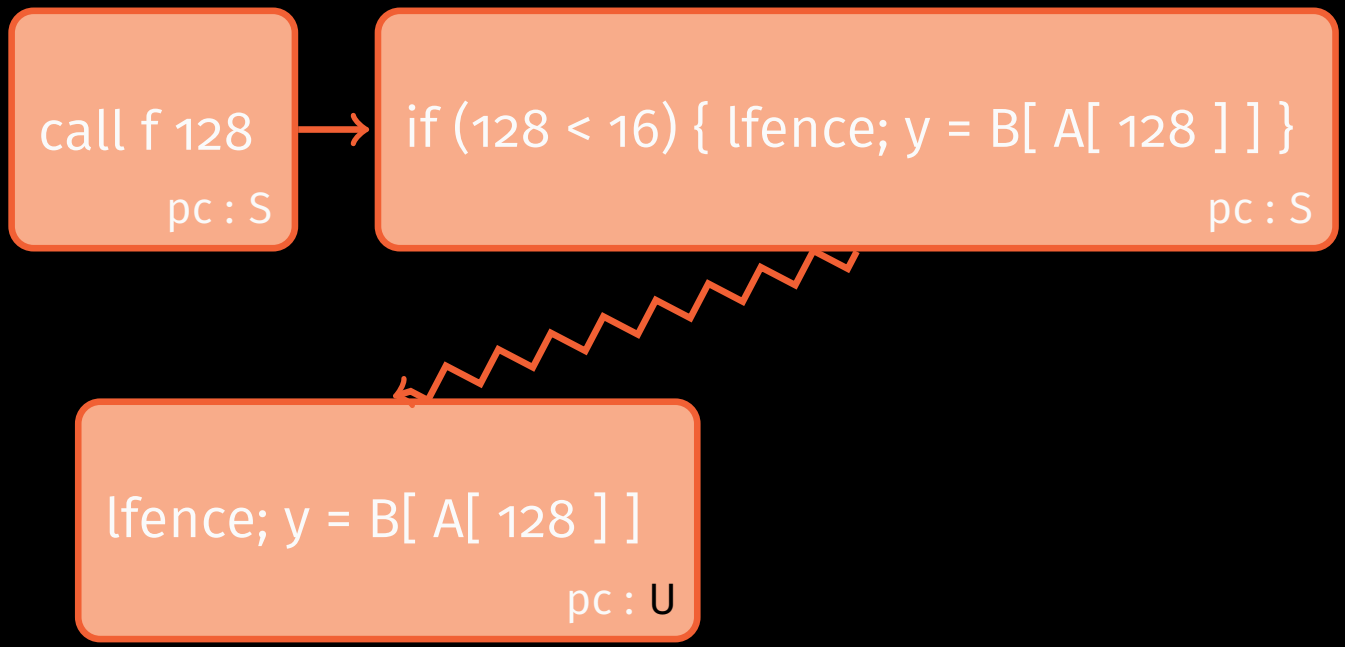
$$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$$



RSSC **for** `lfence`

$$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}} \qquad \text{// A.size=16, A[128]=3}$$

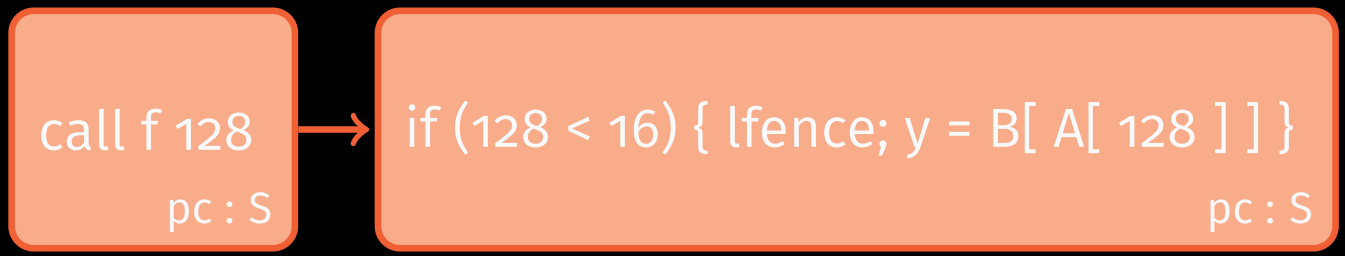
$$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$$



RSSC **for** `lfence`

$$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}} \qquad \text{// A.size=16, A[128]=3}$$

$$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$$



RSSC **for** `lfence`

$$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}} \qquad \text{// A.size=16, A[128]=3}$$

$$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$$

RSSC **for** SLH

$\mathsf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$

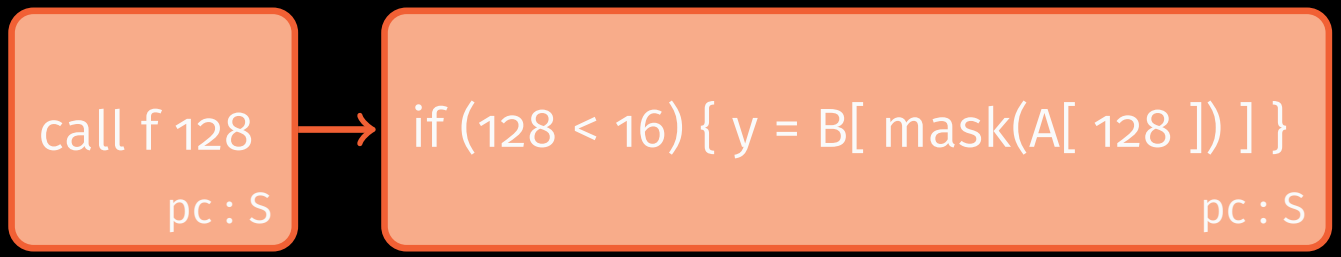
call f 128

pc : S

RSSC **for** SLH

$\mathsf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

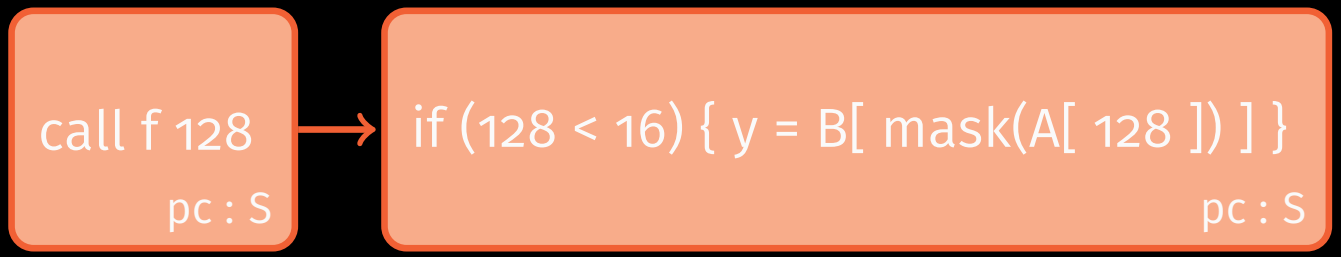
$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$



RSSC **for** SLH

$\mathsf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

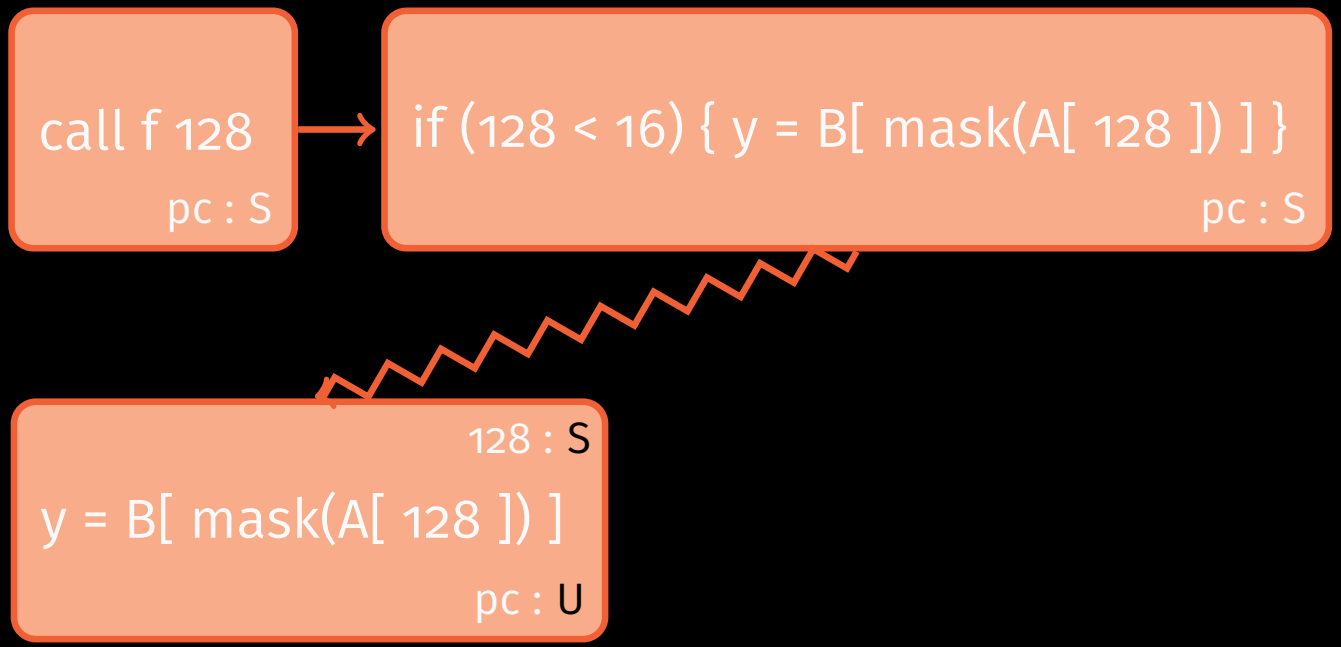
$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$



RSSC **for** SLH

$\mathsf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

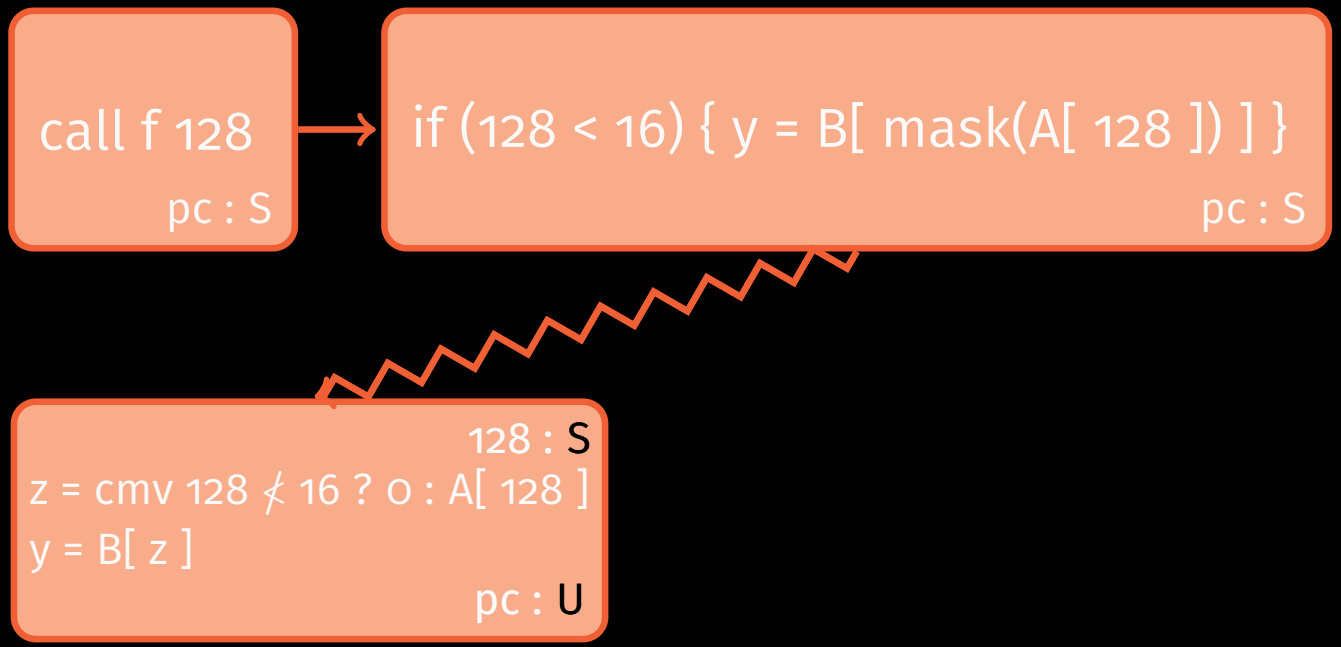
$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$



RSSC **for** SLH

void f(int x) ↦ if(x < A.size){y = B[A[x]]} // A.size=16, A[128]=3

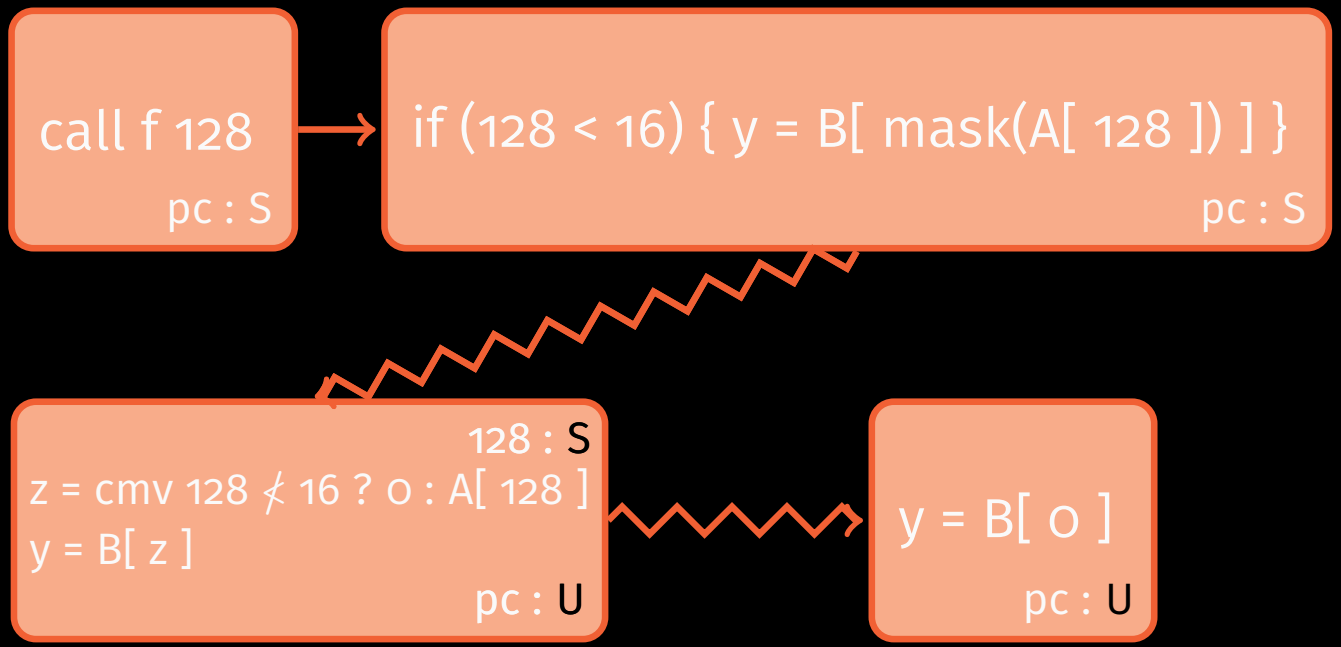
$[\![\cdot]\!]$ = $\mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$



RSSC **for** SLH

$\mathsf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

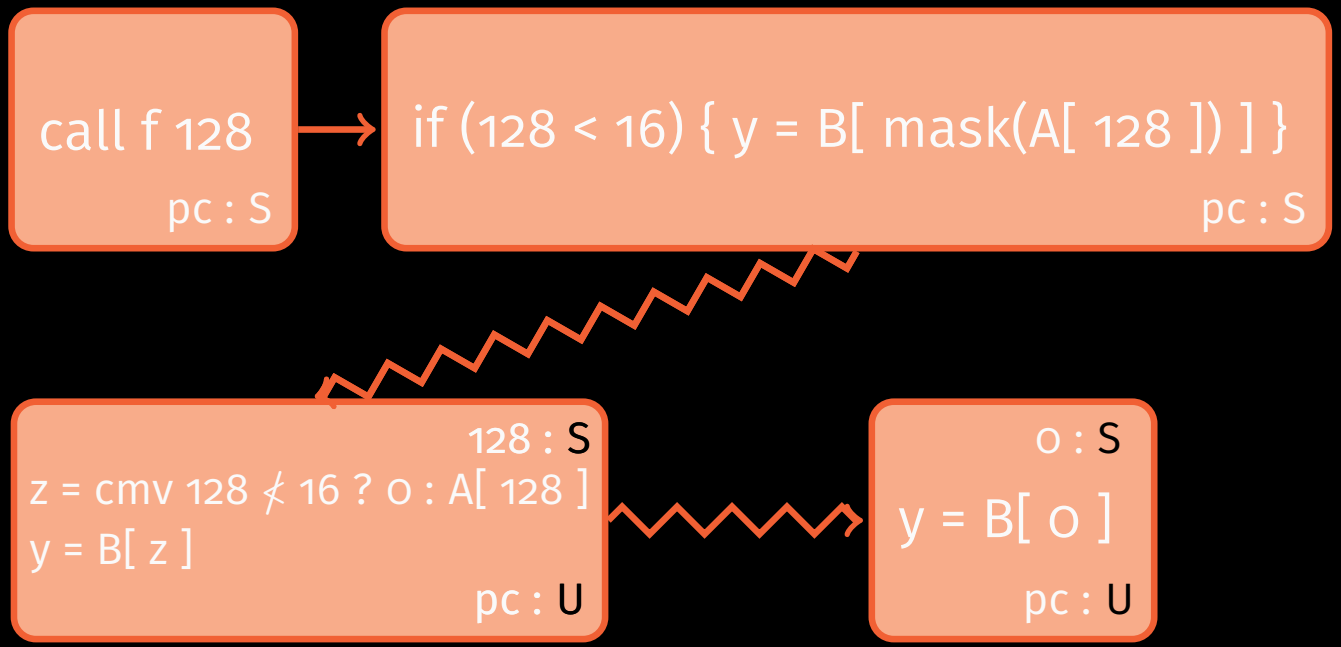
$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$



RSSC **for** SLH

$\mathsf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

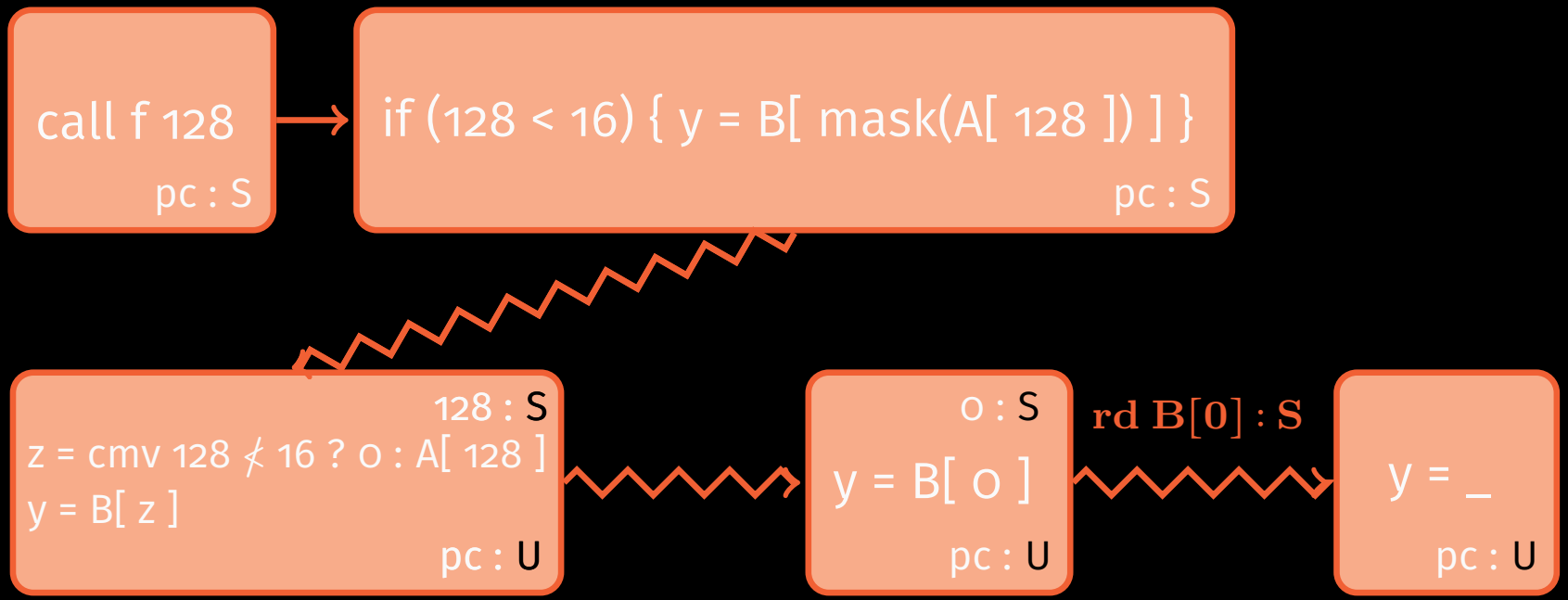
$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$



RSSC **for** SLH

$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

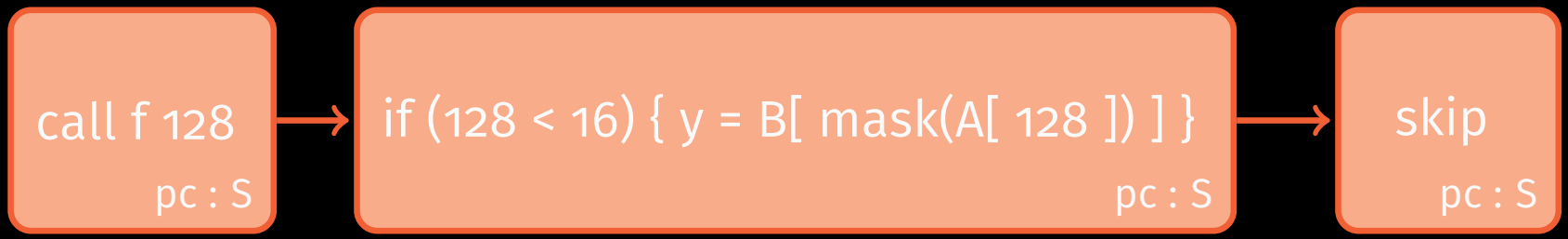
$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$



RSSC **for** SLH

void f(int x) ↦ if(x < A.size){y = B[A[x]]} // A.size=16, A[128]=3

$[\![\cdot]\!]$ = $\mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$



$\mathbf{rd\ B[0] : S}$

In the Paper

- theory scalable to Spectre v2, v4, v5

In the Paper

- theory scalable to Spectre v2, v4, v5
- 2 notions of SS for strong & weak absence of speculative leakage

In the Paper

- theory scalable to Spectre v2, v4, v5
- 2 notions of SS for strong & weak absence of speculative leakage
- proofs that RSSC means absence of leaks

In the Paper

- theory scalable to Spectre v2, v4, v5
- 2 notions of SS for strong & weak absence of speculative leakage
- proofs that RSSC means absence of leaks
- proof of security/insecurity for `lfence`, `SLH` (weakly), `interprocedural SLH`, `SSLH`

Future Outlook

What More?

What More?

- secure compilation for Spectre V2+ (w. Imdea, Cispa)

What More?

- secure compilation for Spectre V2+
  (w. Imdea, Cispa)
- secure compilation to webassembly
  (w. UCSD, Harvard, Cispa)

What More?

- secure compilation for Spectre V2+
  (w. Imdea, Cispa)
- secure compilation to webassembly
  (w. UCSD, Harvard, Cispa)
- secure compilation is universal
  composability
  (w. Stanford, Cispa)

What More?

- secure compilation for Spectre V2+
  (w. Imdea, Cispa)
- secure compilation to webassembly
  (w. UCSD, Harvard, Cispa)
- secure compilation is universal composability
  (w. Stanford, Cispa)
- secure compilation for linear languages
  (w. Novi / FB)

What More?

- secure compilation for Spectre V2+
  (w. Imdea, Cispa)
- secure compilation to webassembly
  (w. UCSD, Harvard, Cispa)
- secure compilation is universal composability
  (w. Stanford, Cispa)
- secure compilation for linear languages
  (w. Novi / FB)
- …

Questions?