

Spectector: Principled detection of speculative information flows

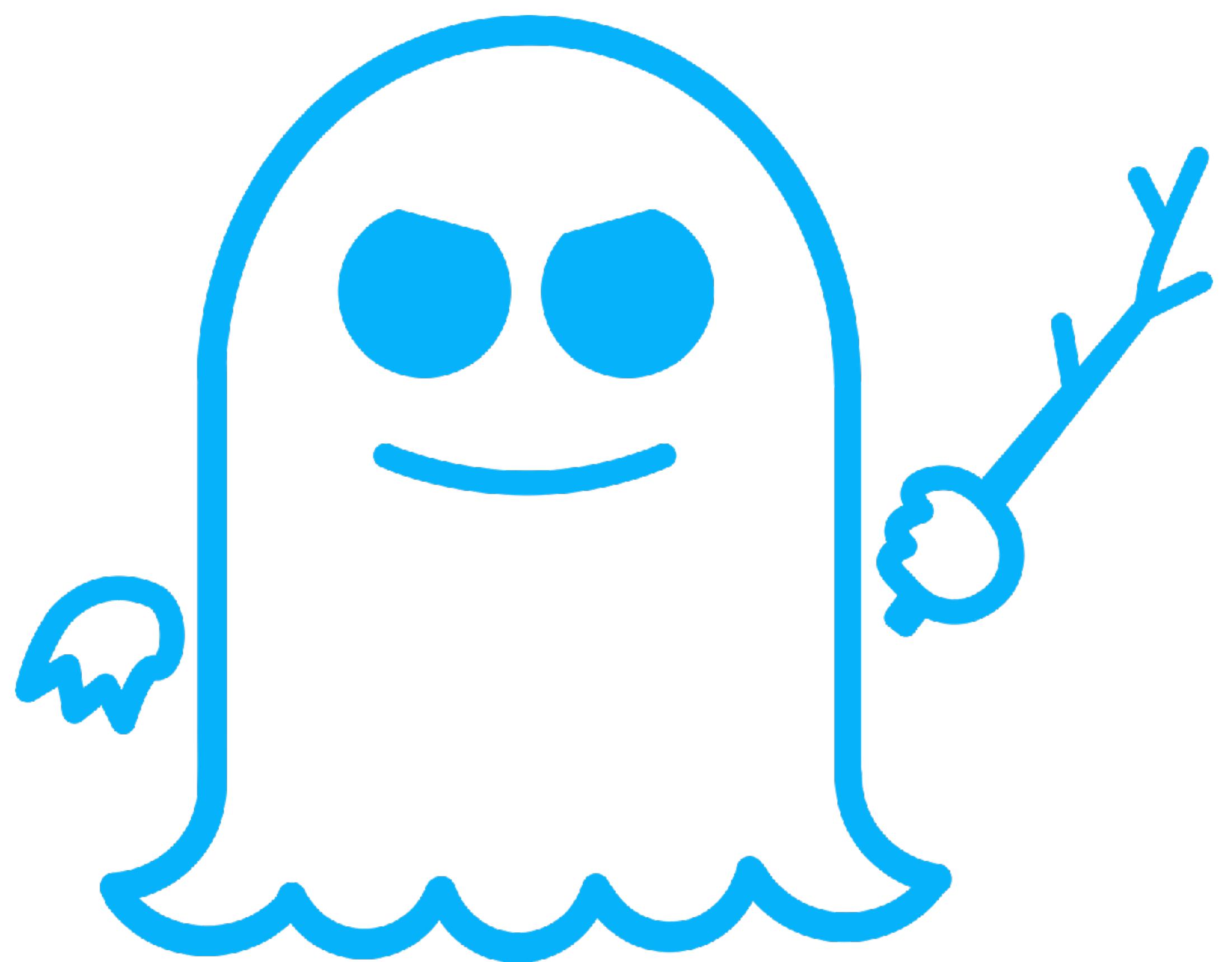
Jan Reineke @



UNIVERSITÄT
DES
SAARLANDES

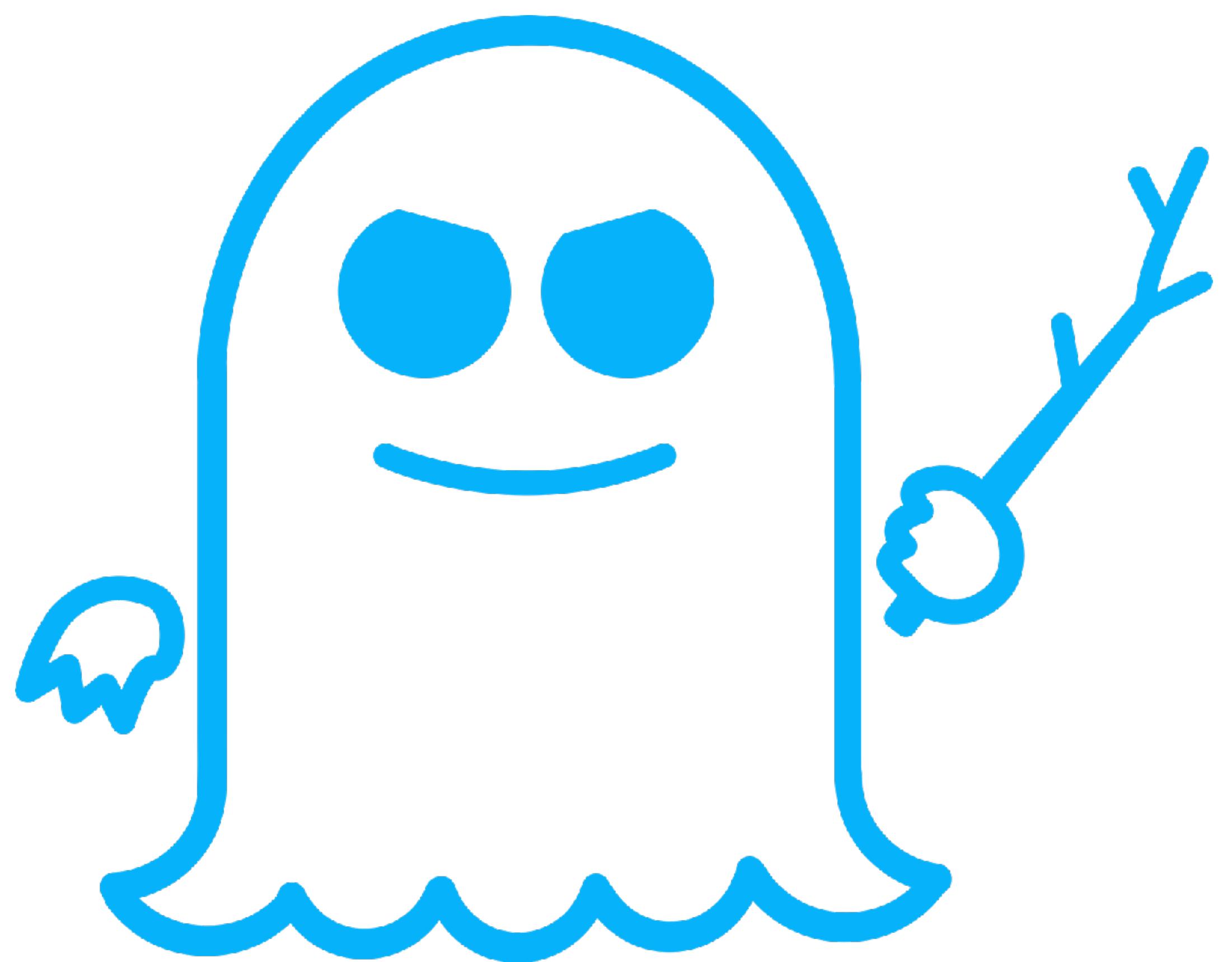
Joint work with
Marco Guarnieri, Jose Morales, Andres Sanchez @ IMDEA Software, Madrid
Boris Köpf @ Microsoft Research, Cambridge, UK

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



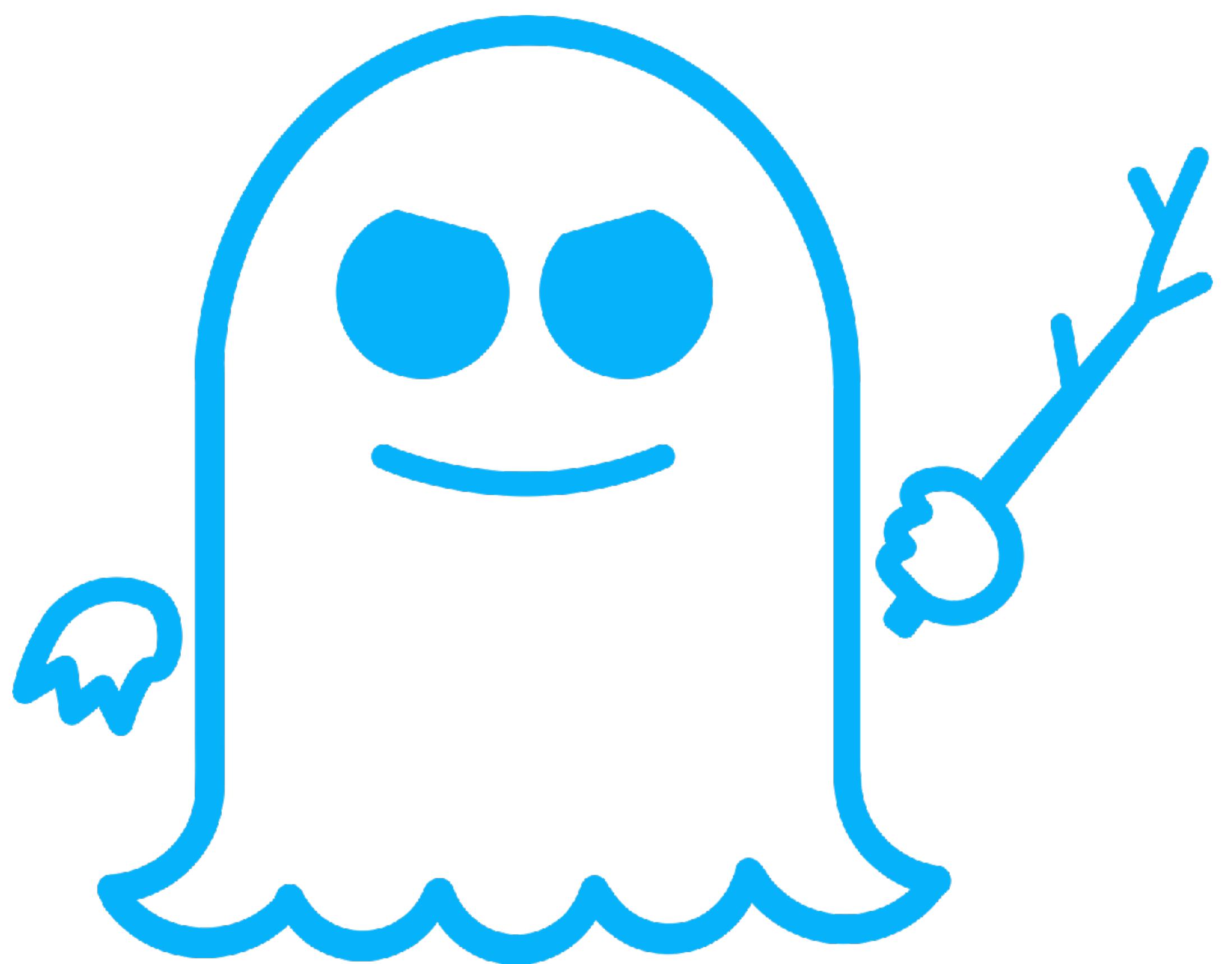
SPECTRE

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



Exploits **speculative execution** to leak sensitive information

SPECTRE



SPECTRE

Exploits **speculative execution** to leak sensitive information

Almost all modern processors are affected

Countermeasures

Countermeasures

Long Term: Co-Design of Software and Hardware countermeasures

Countermeasures

Long Term: Co-Design of Software and Hardware countermeasures

Short and Mid Term: Software countermeasures

In particular: Compiler-level countermeasures

- ✓ *Example:* insert “fences” to selectively terminate speculative execution
- ✓ Implemented in major compilers (Microsoft Visual C++, Intel ICC, Clang)

Countermeasures

Long Term: Co-Design of Software and Hardware countermeasures

Short and Mid Term: Software countermeasures

In particular: Compiler-level countermeasures

- ✓ *Example:* insert “fences” to selectively terminate speculative execution
- ✓ Implemented in major compilers (Microsoft Visual C++, Intel ICC, Clang)

PROBLEM SOLVED ?

Compiler-level countermeasures

Spectre Mitigations in Microsoft's C/C++ Compiler

Paul Kocher

February 13, 2018

Compiler-level countermeasures

Spectre Mitigations in Microsoft's C/C++ Compiler

Paul Kocher

February 13, 2018

“The countermeasure [...] is conceptually straightforward
but **challenging in practice**”

Compiler-level countermeasures

Spectre Mitigations in Microsoft's C/C++ Compiler

Paul Kocher

February 13, 2018

“The countermeasure [...] is conceptually straightforward
but **challenging in practice**”

“compiler [...] produces **unsafe code** when the
static analyzer is unable to determine whether
a code pattern will be exploitable”

Compiler-level countermeasures

Spectre Mitigations in Microsoft's C/C++ Compiler

Paul Kocher

February 13, 2018

“The countermeasure [...] is conceptually straightforward
but **challenging in practice**”

“compiler [...] produces **unsafe code** when the
static analyzer is unable to determine whether
a code pattern will be exploitable”

“there is **no guarantee** that all possible instances of
[Spectre] will be instrumented”

Compiler-level countermeasures

Spectre Mitigations in Microsoft's C/C++ Compiler

Paul Kocher

February 13, 2018

“The countermeasure [...] is conceptually straightforward
but **challenging in practice**”

“compiler [...] produces **unsafe code** when the
static analyzer is unable to determine whether
a code pattern will be exploitable”

“there is **no guarantee** that all possible instances of
[Spectre] will be instrumented”

Bottom line: No guarantees!

Goals

Goals

1. Introduce **semantic notion of security**
against speculative execution attacks

Goals

1. Introduce **semantic notion of security**
against **speculative execution attacks**
2. Static analysis to **detect vulnerability**
or to **prove security**

Outline

1. Speculative execution attacks
2. Speculative non-interference
3. Spectector: Detecting speculative leaks
4. Challenges

1. Speculative execution attacks

Background: Speculative execution

Background: Speculative execution

- Predict instructions' outcomes and speculatively continue execution

Background: Speculative execution

- Predict instructions' outcomes and speculatively continue execution
- Rollback changes if speculation was wrong

Background: Speculative execution

- Predict instructions' outcomes and speculatively continue execution
- Rollback changes if speculation was wrong



Only architectural (ISA, “logical”) state,
not microarchitectural state

Background: Branch prediction

Size of array **A**

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

Background: Branch prediction

Size of array **A**

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

Background: Branch prediction

Size of array **A**

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



Background: Branch prediction

Size of array **A**

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



Background: Branch prediction

Size of array **A**

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



Predictions based on
branch history &
program structure



Background: Branch prediction

Size of array **A**

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



Predictions based on
branch history &
program structure

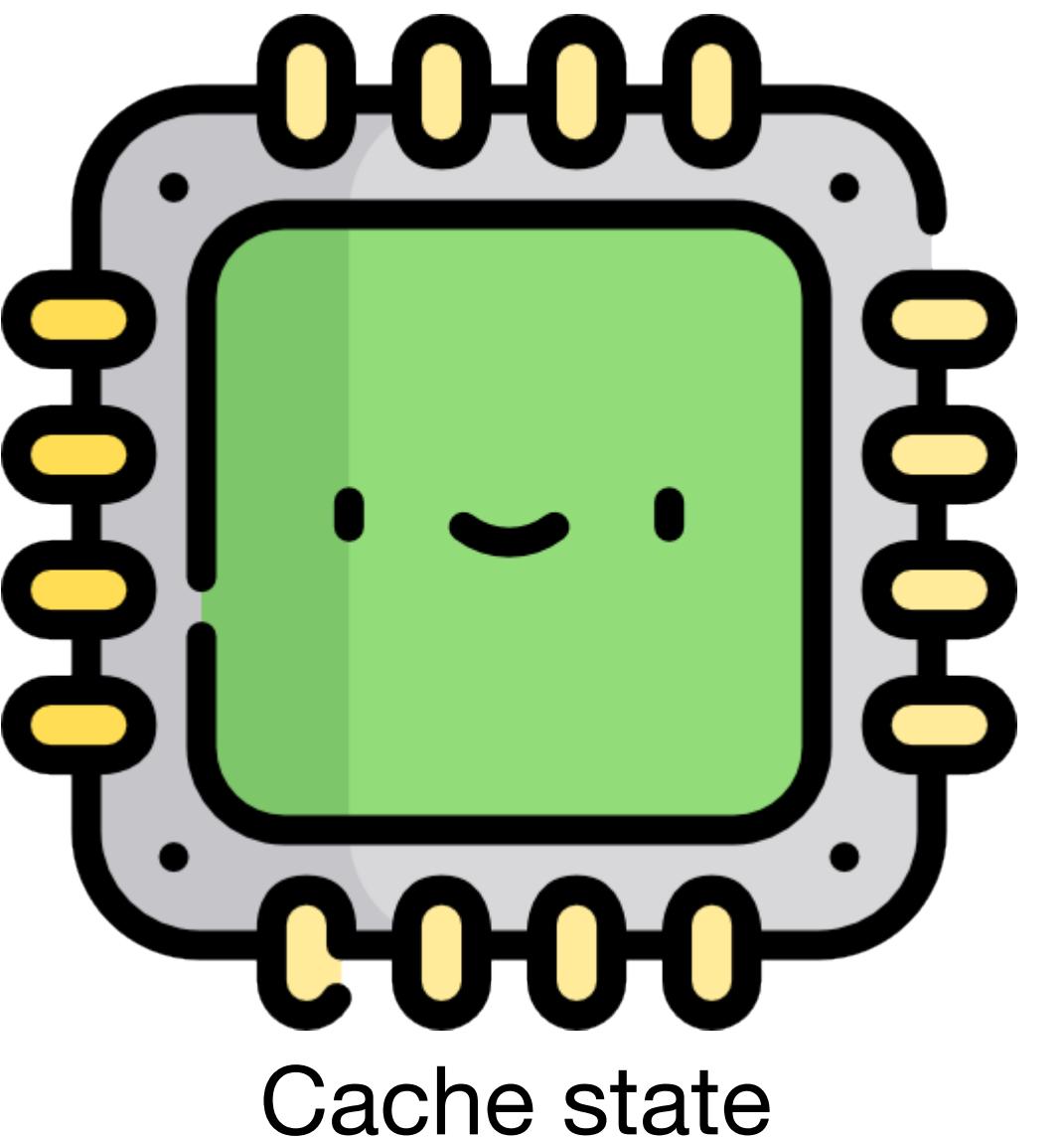


Spectre V1

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

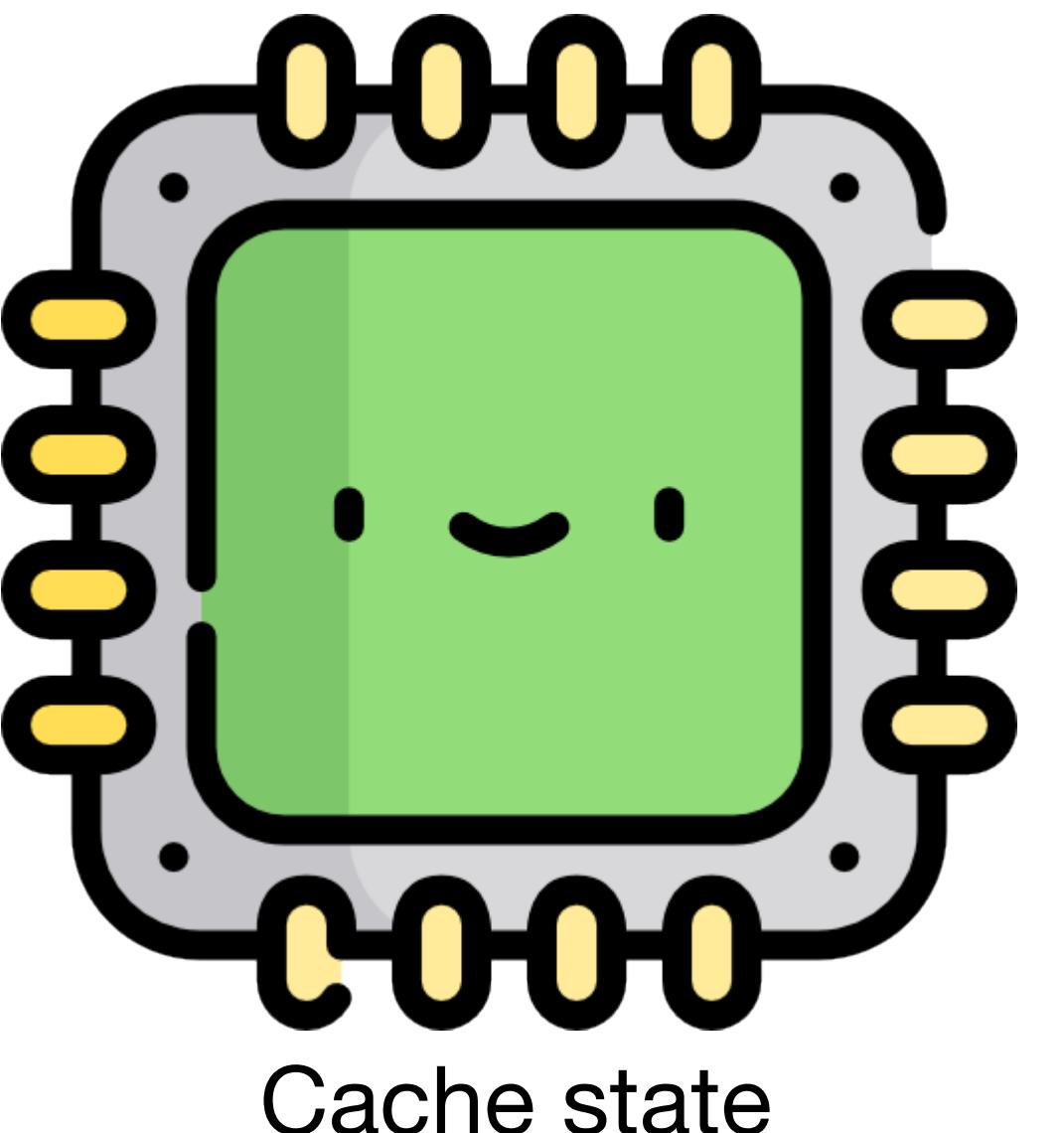
Spectre V1

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



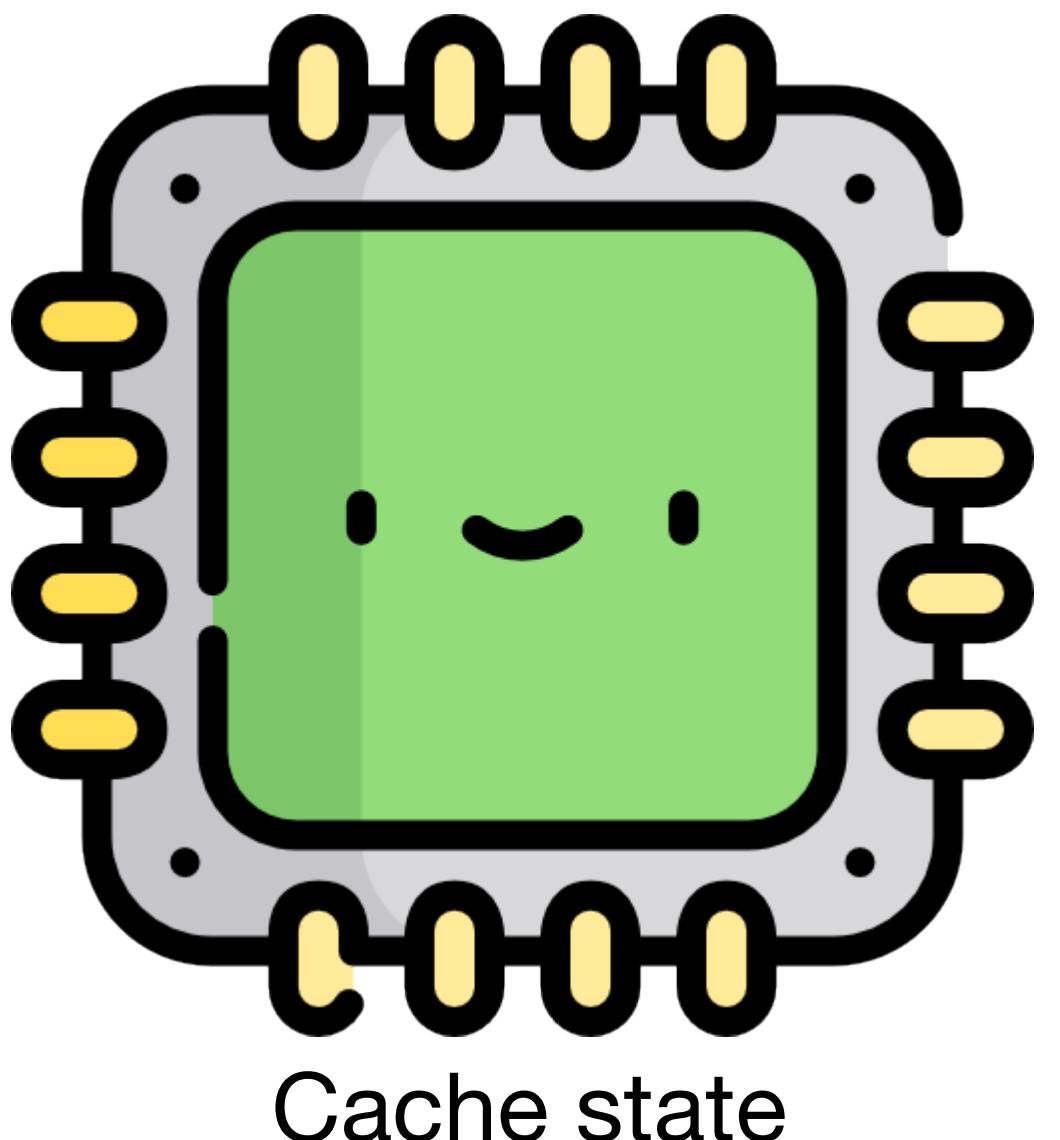
Spectre V1

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



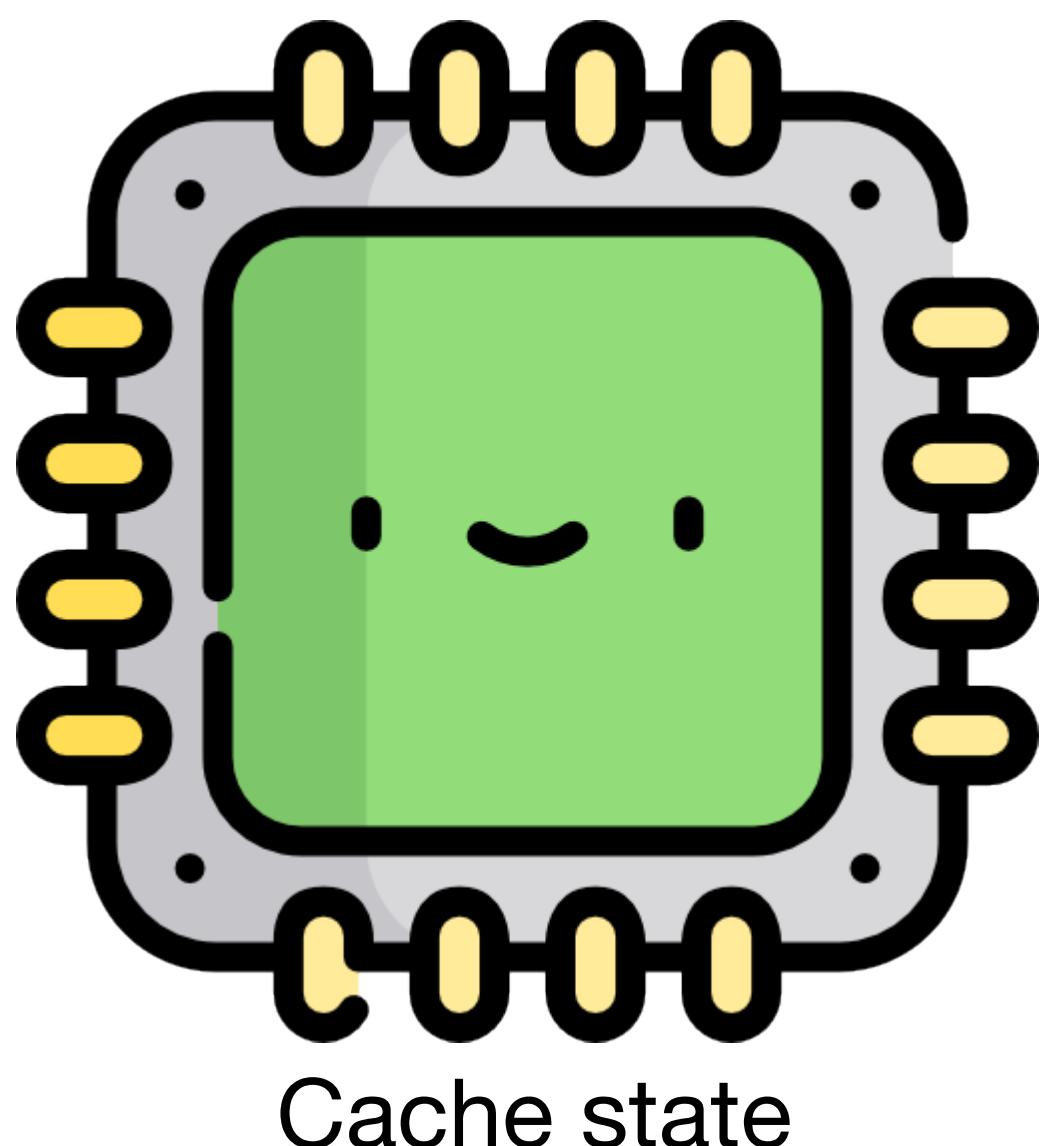
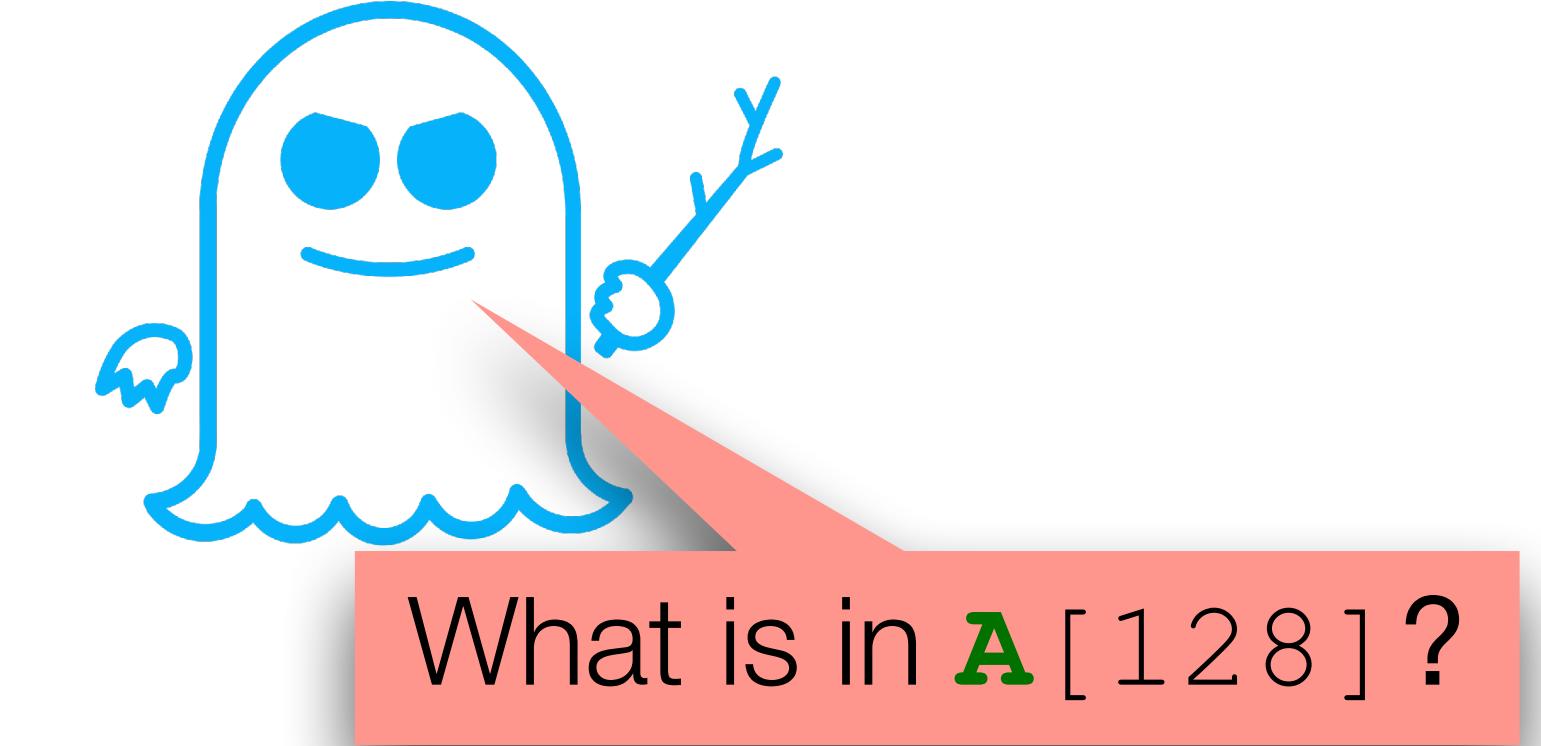
Spectre V1

```
A_size=16  
B[B[0]B[1] ...  
void f(int x)  
if (x < A_size)  
y = B[A[x]]
```



Spectre V1

```
A_size=16  
B[B[0] B[1] ...]  
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]]
```

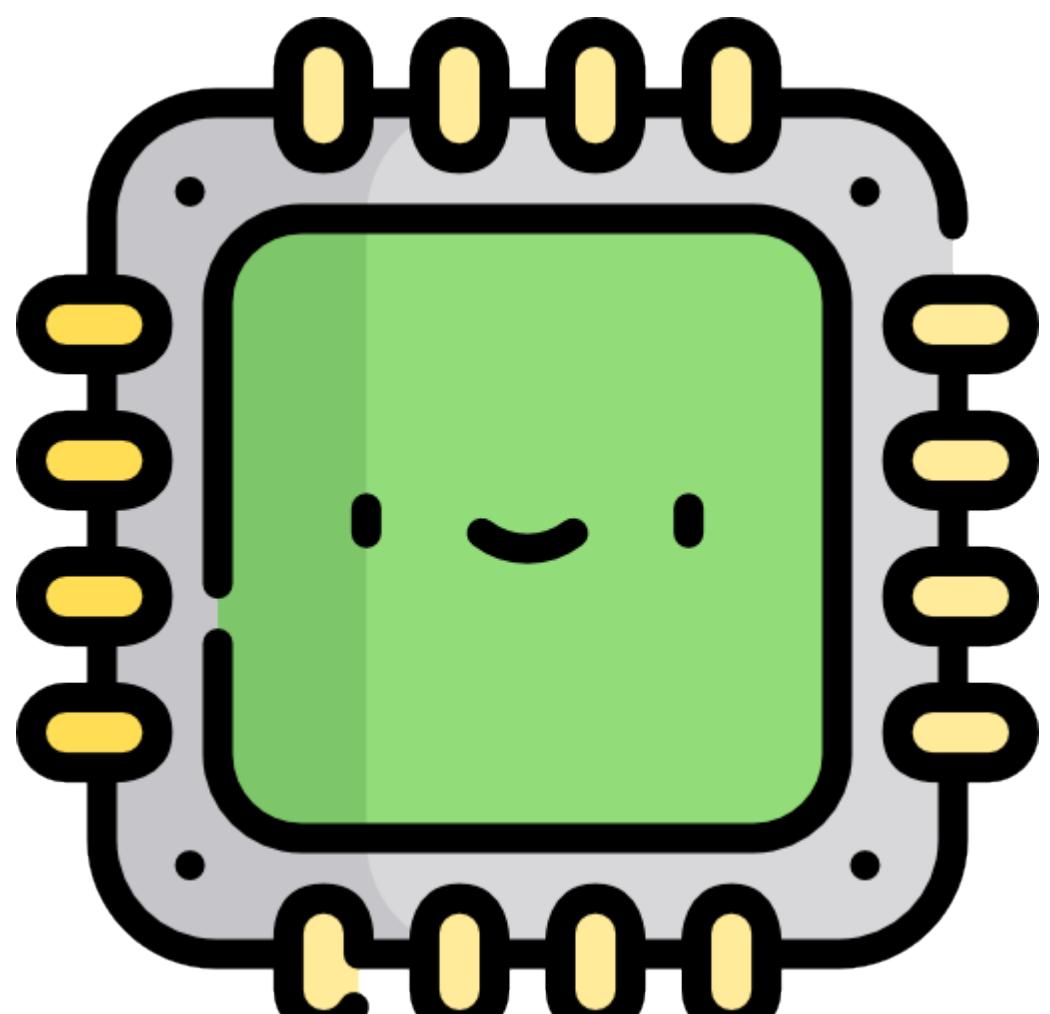
A_size=16

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



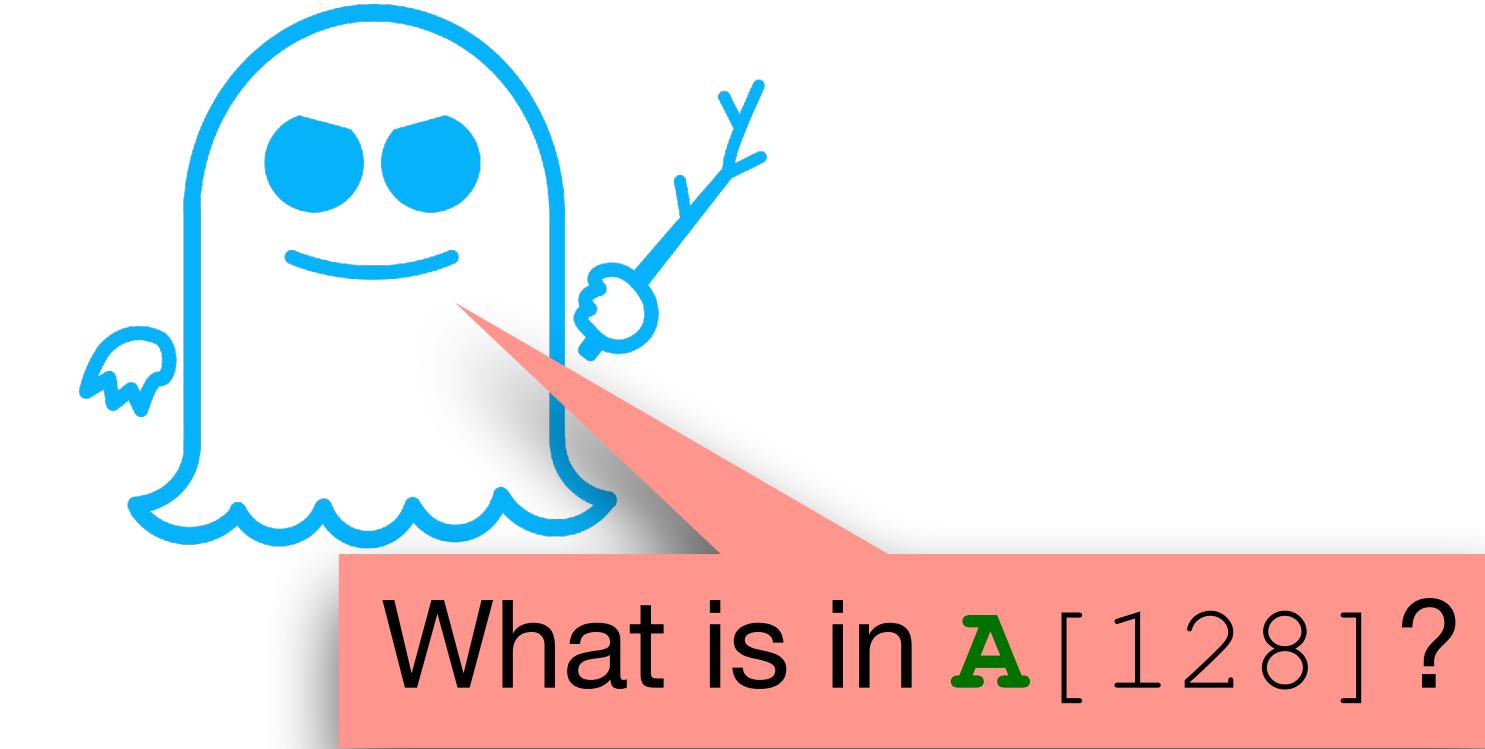
What is in A[128]?

1a) Training



Cache state

Spectre V1



$A_size = 16$

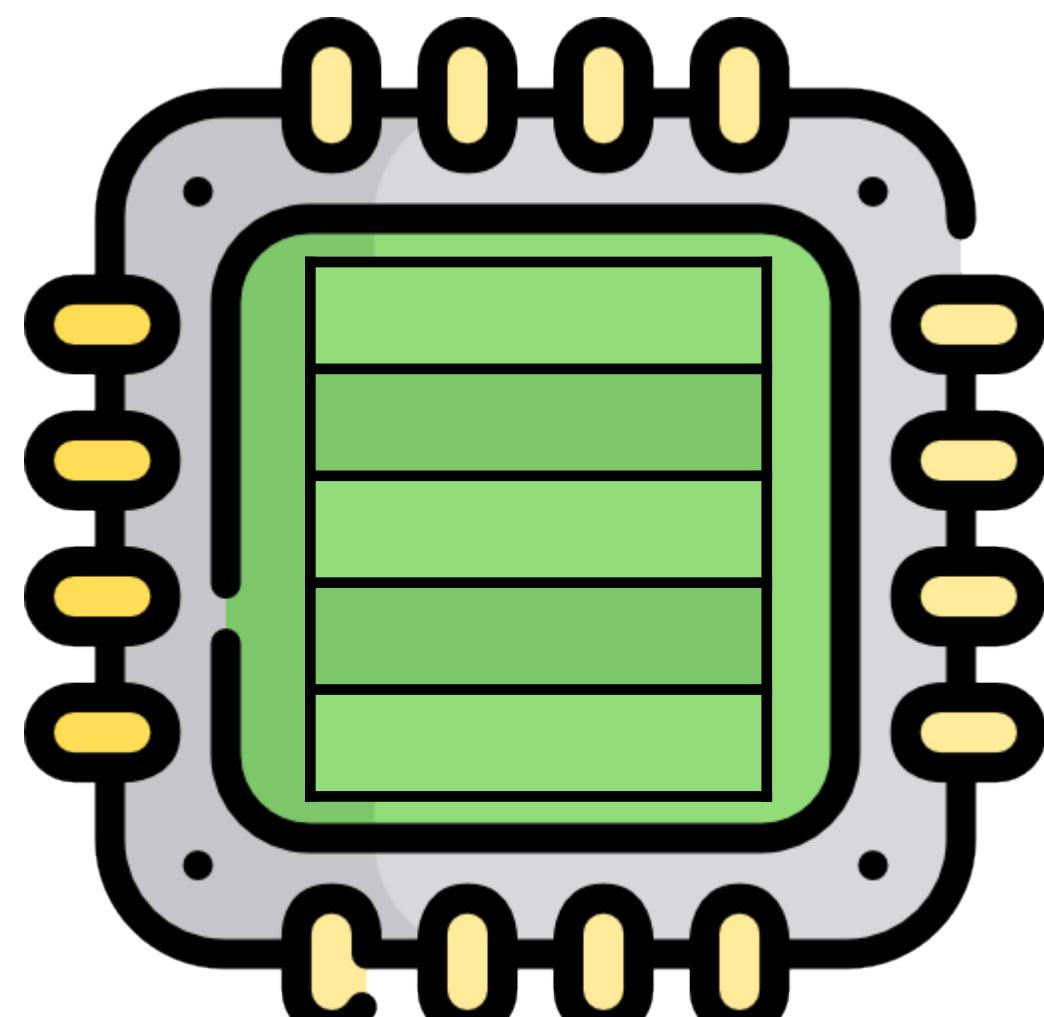
$B[B[0] B[1] \dots]$

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

1a) Training

Cache state

What is in $A[128]$?



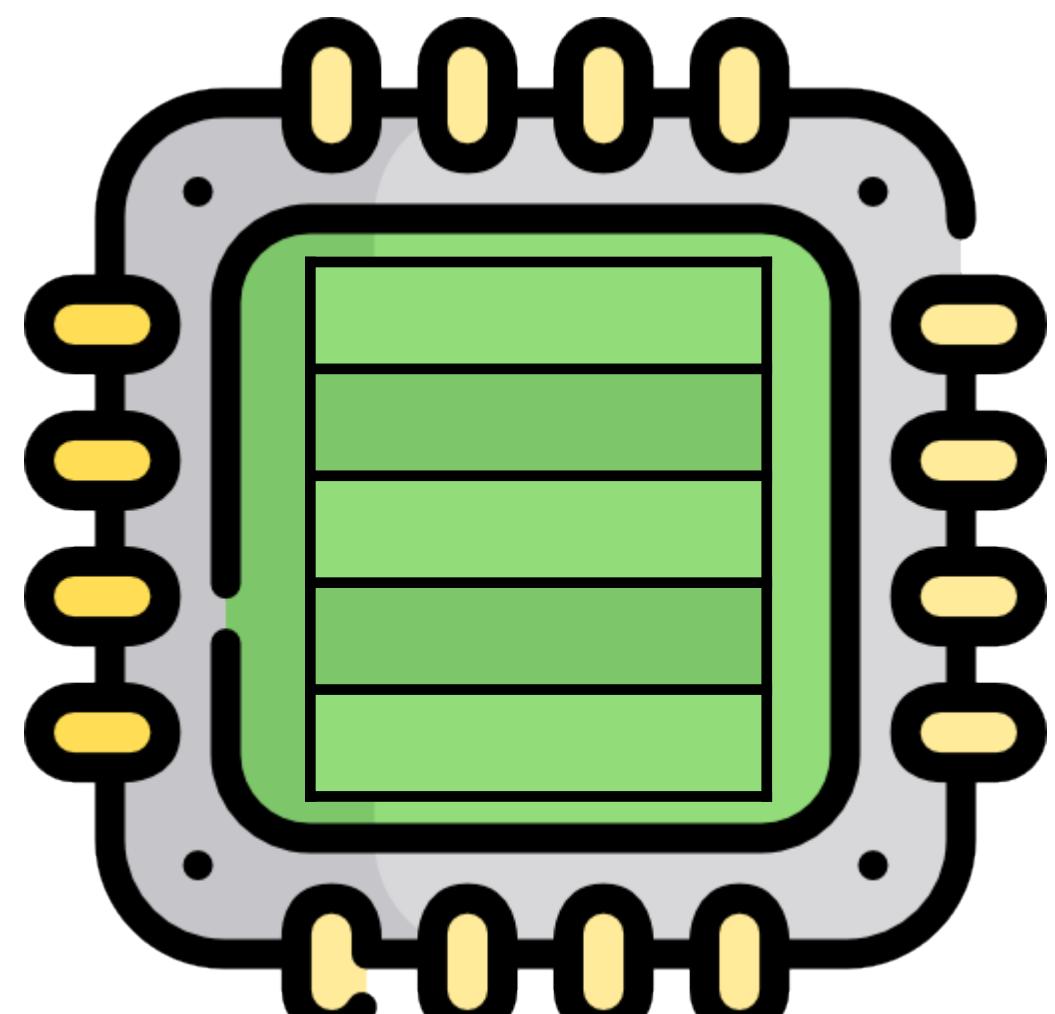
Spectre V1

```
A_size=16
B[B[0] B[1] ...]
void f(int x)
if (x < A_size)
    y = B[A[x]]
```



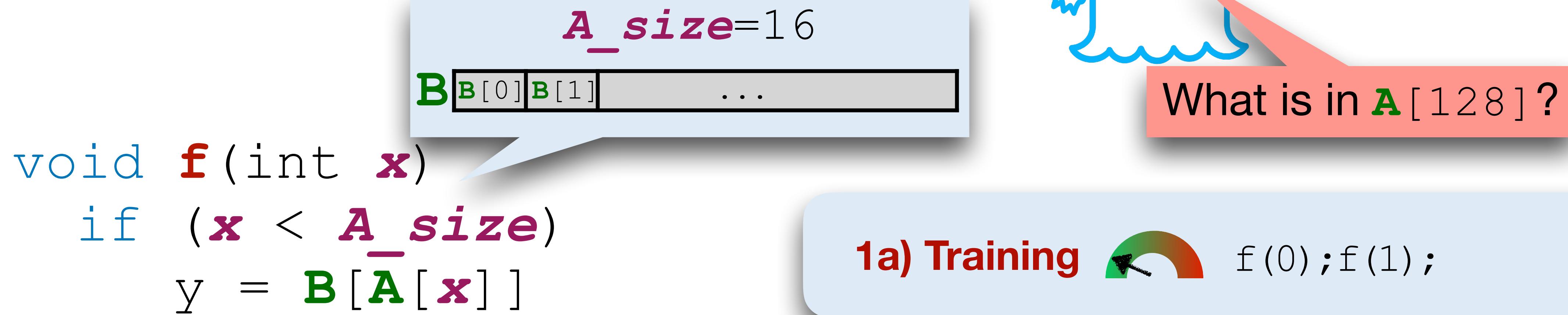
What is in A[128]?

1a) Training  f(0);



Cache state

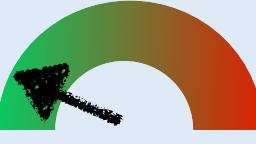
Spectre V1

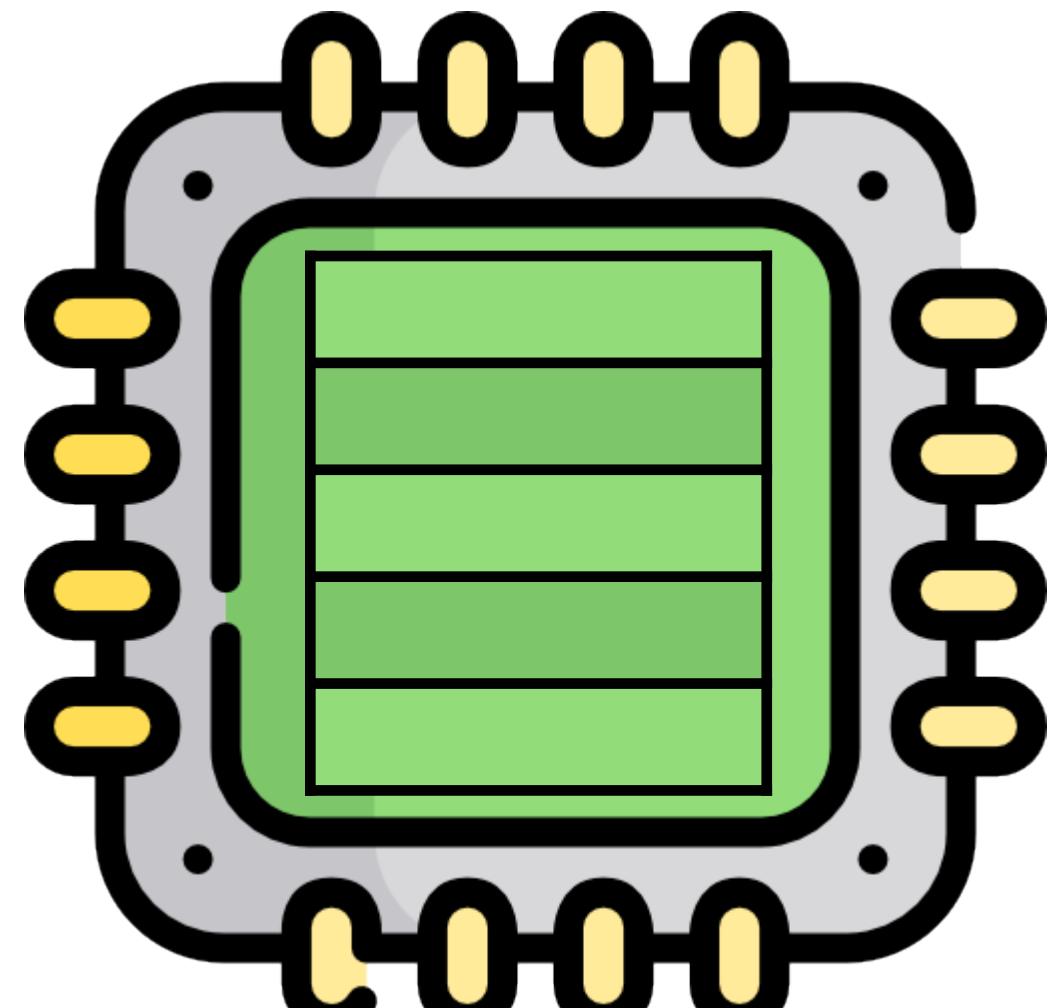


`A_size=16`

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

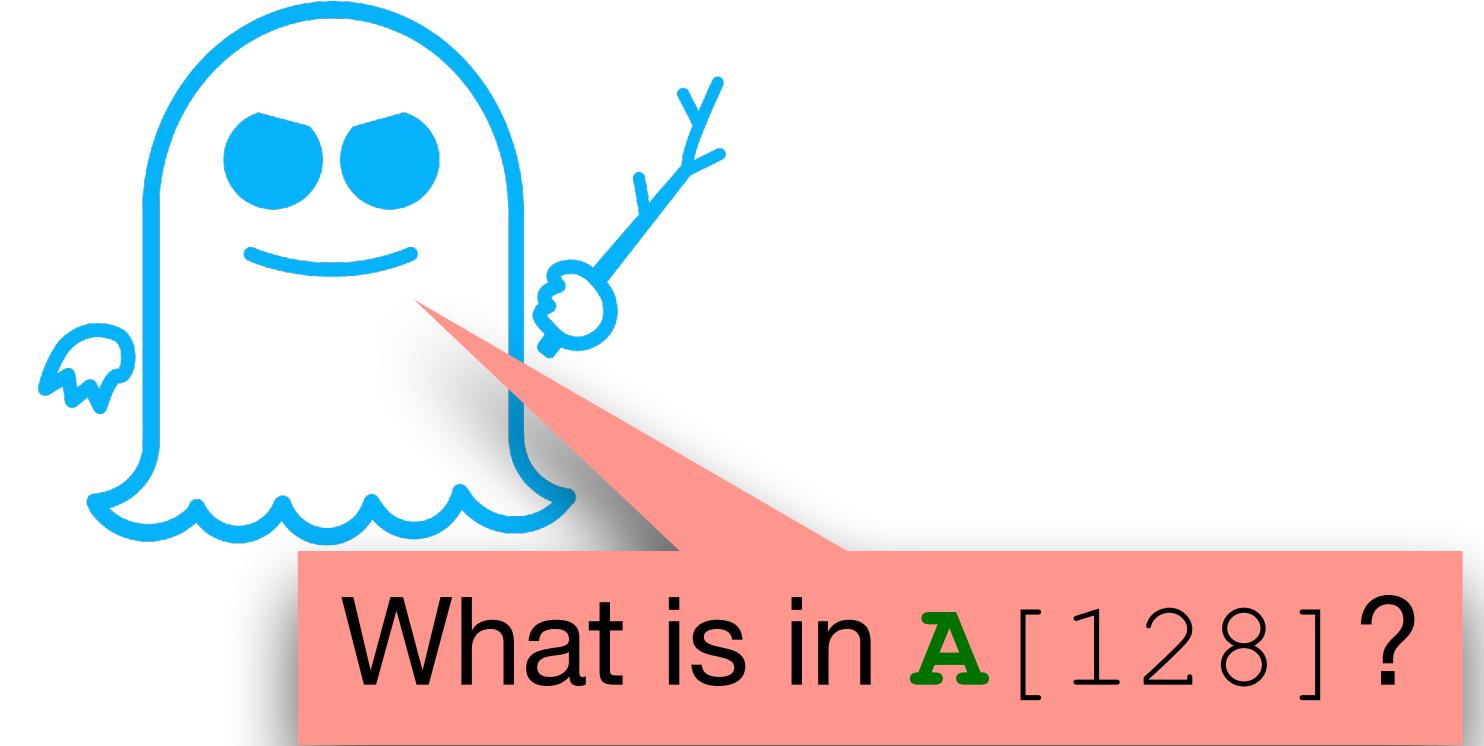
```
void f(int x)
if (x < A_size)
    y = B[A[x]]
```

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



Cache state

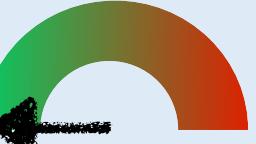
Spectre V1

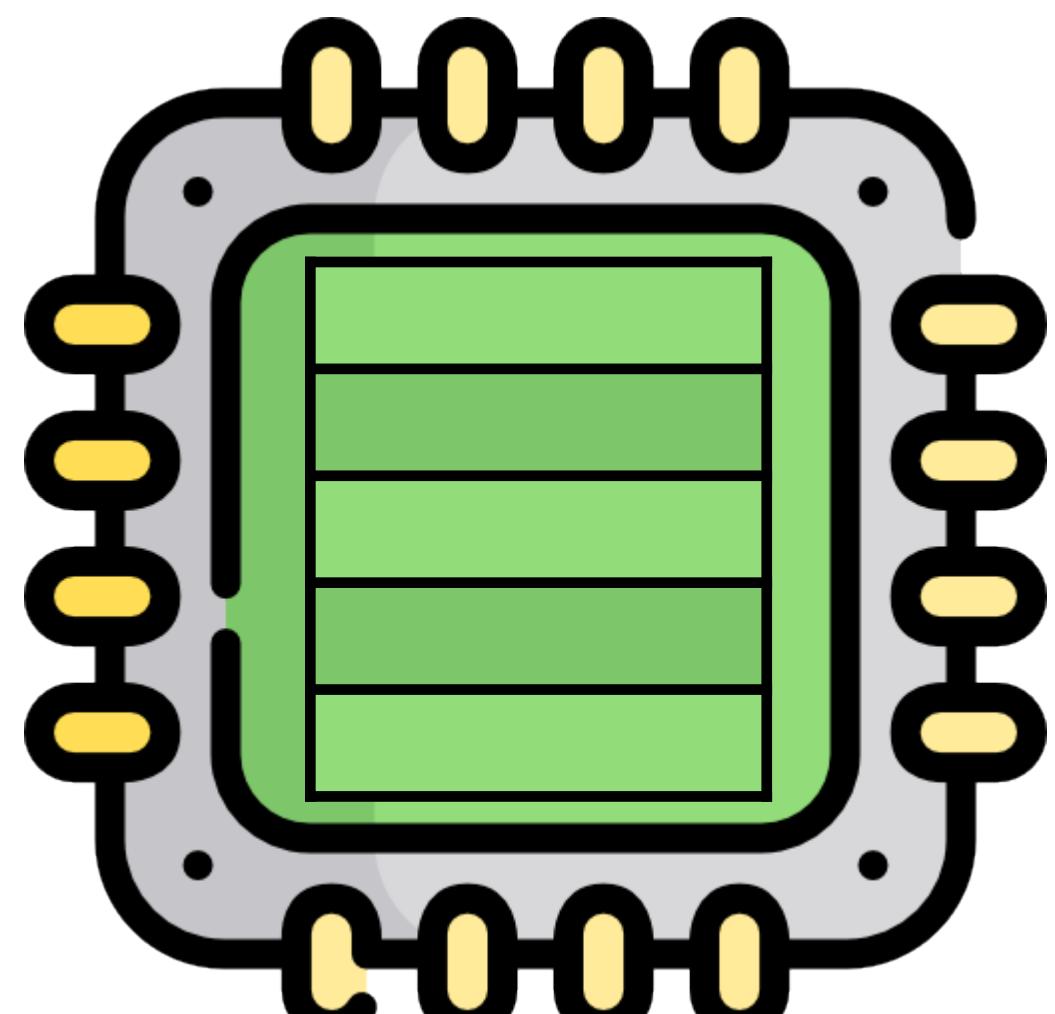


`A_size=16`

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

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

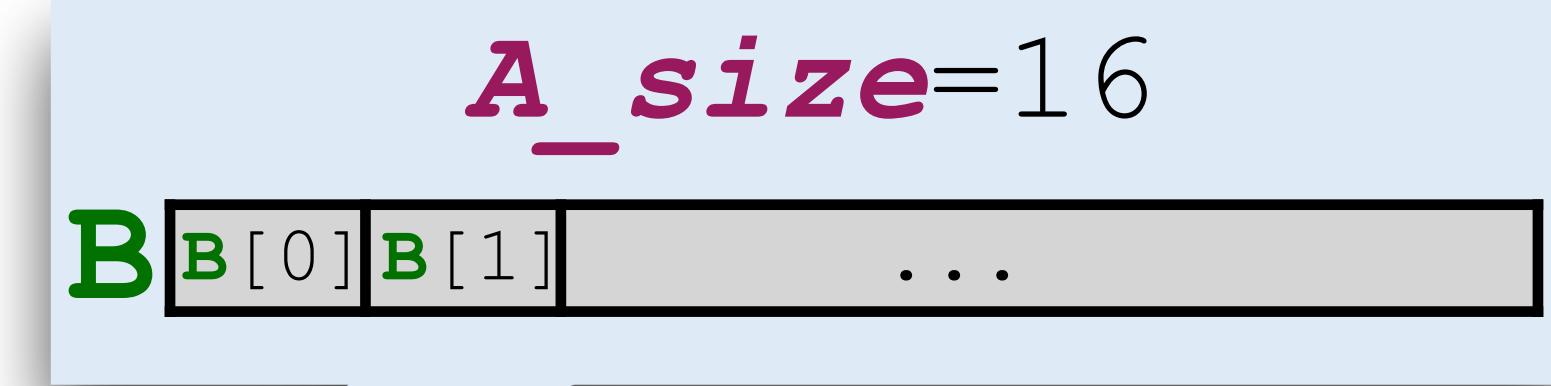
1a) Training  `f(0); f(1); f(2); ...`



Cache state

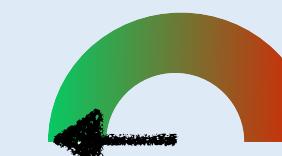
Spectre V1

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



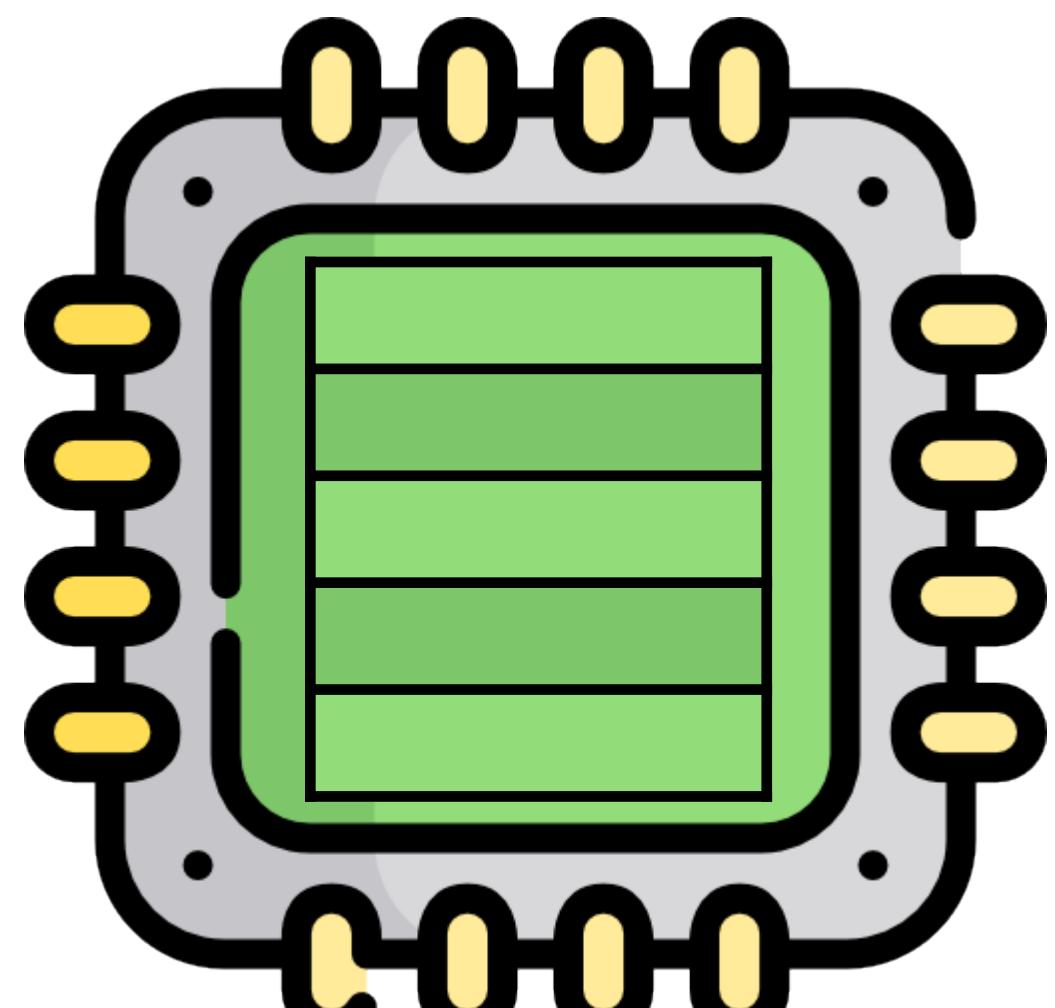
What is in $A[128]$?

1a) Training



$f(0); f(1); f(2); \dots$

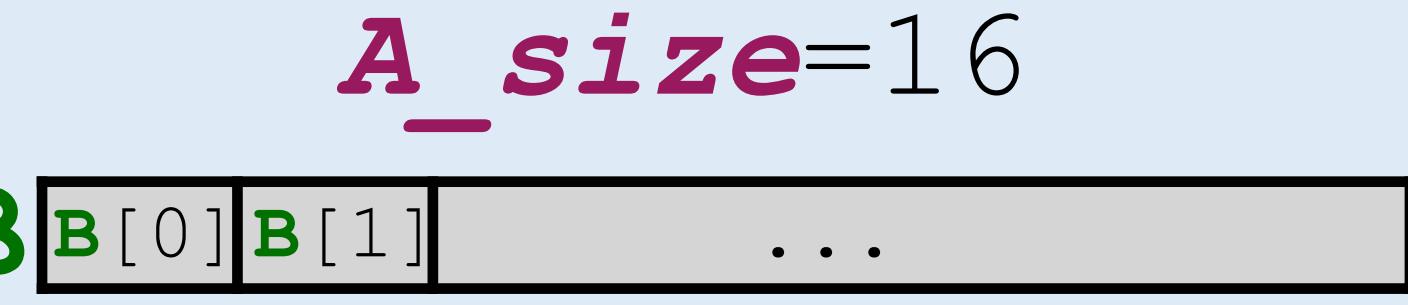
1b) Prepare cache



Cache state

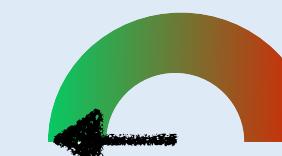
Spectre V1

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



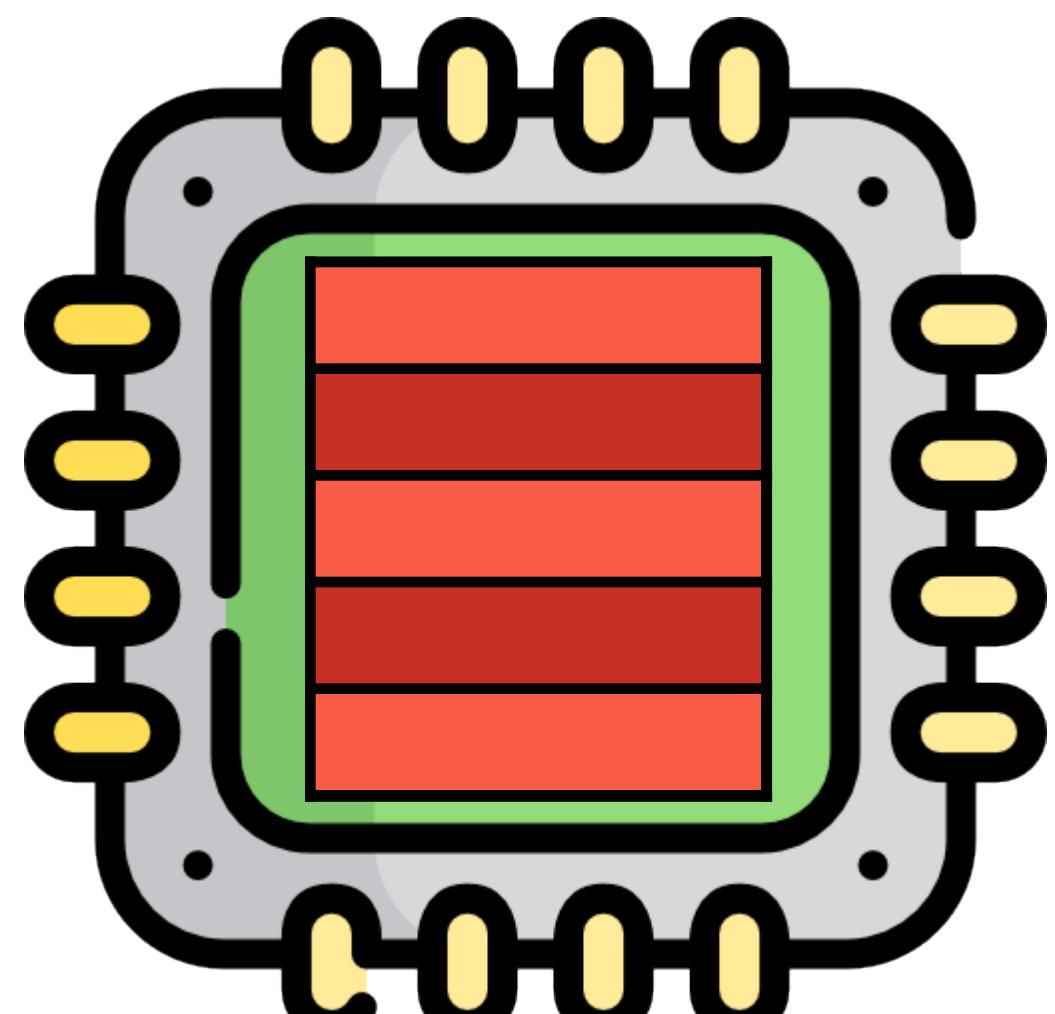
What is in $A[128]$?

1a) Training



$f(0); f(1); f(2); \dots$

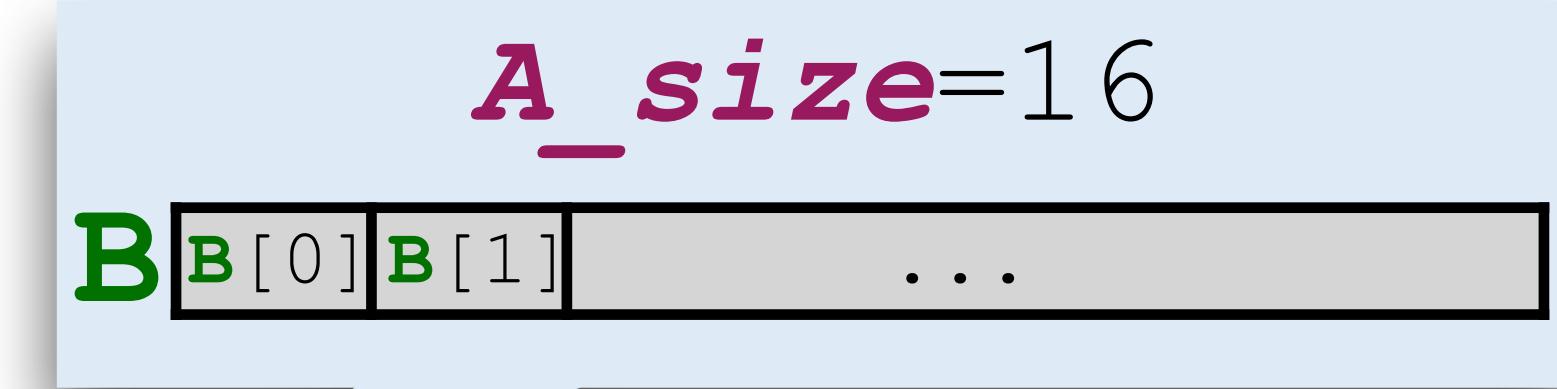
1b) Prepare cache



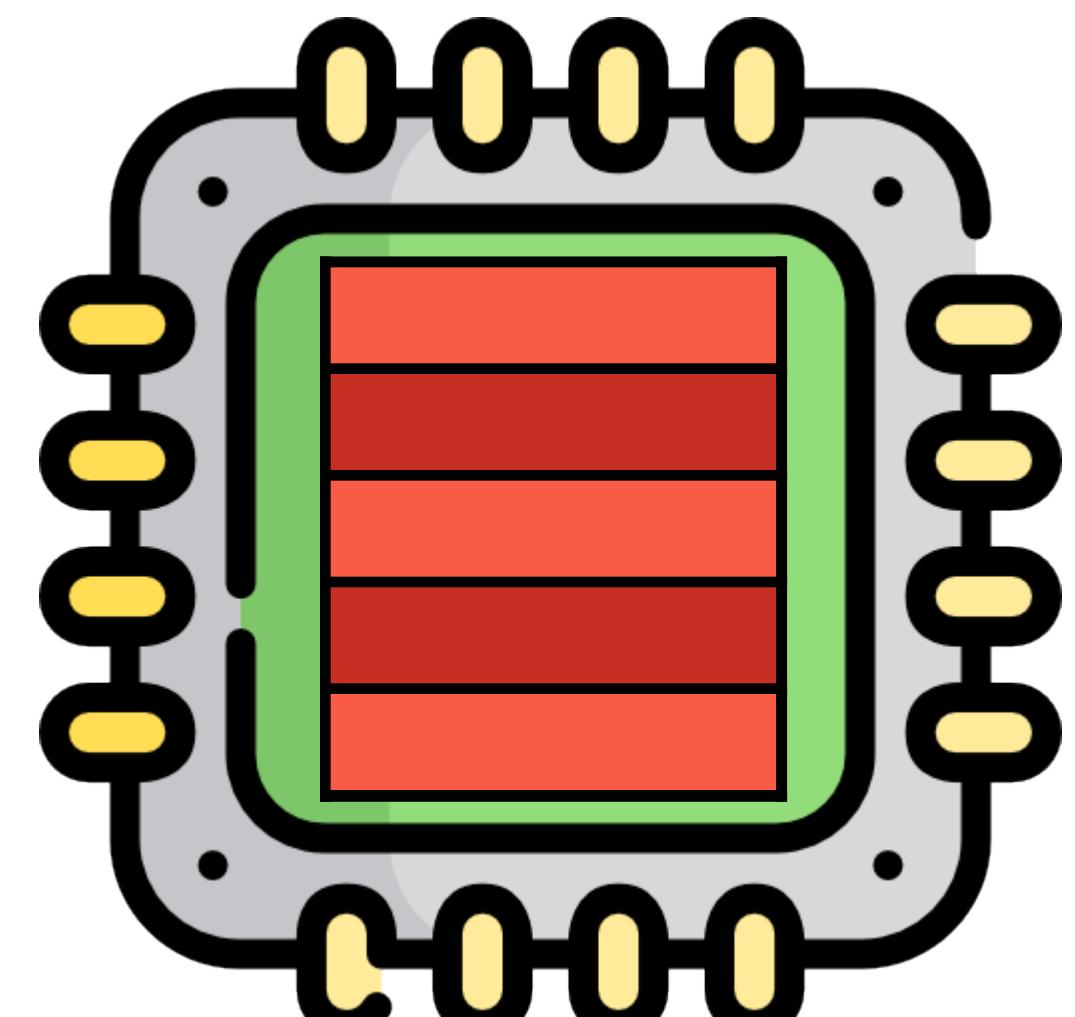
Cache state

Spectre V1

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

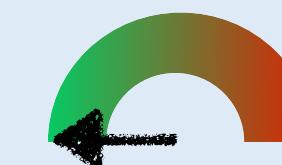


What is in A[128]?



Cache state

1a) Training



f(0); f(1); f(2); ...

1b) Prepare cache

2) Run f(128)

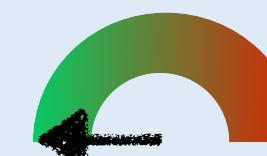
Spectre V1

```
A_size=16  
B[B[0] B[1] ...]  
void f(int x)  
if (x < A_size)  
y = B[A[x]]
```



What is in **A**[128]?

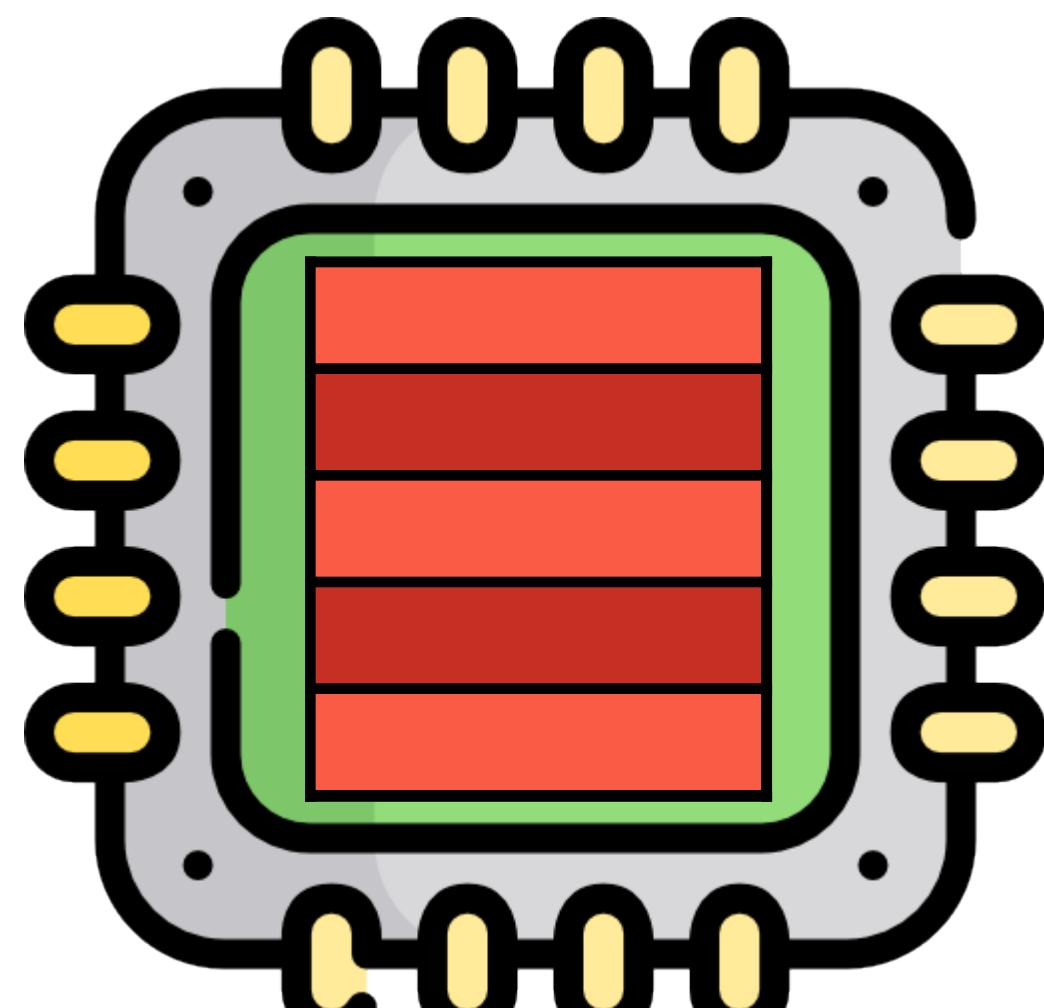
1a) Training



f(0); f(1); f(2); ...

1b) Prepare cache

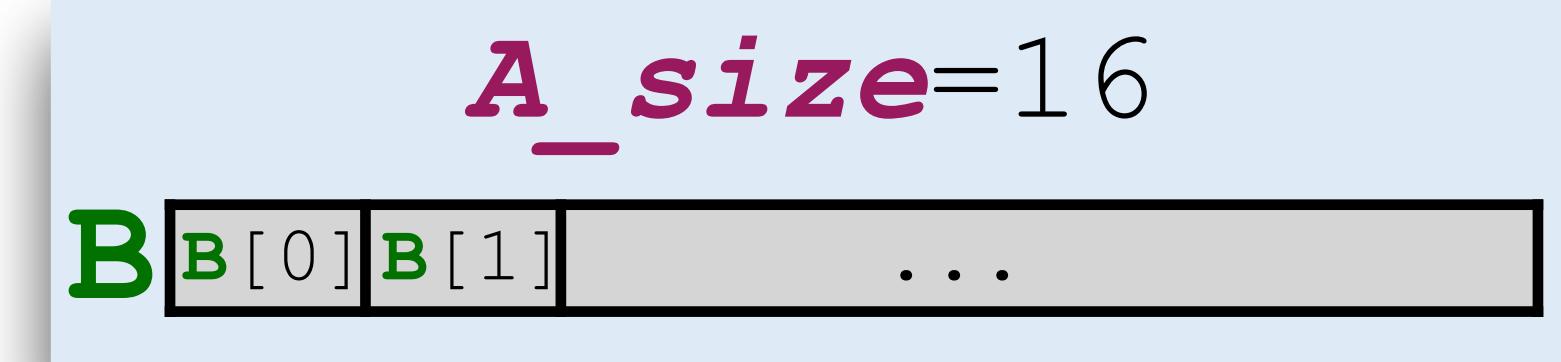
2) Run **f**(128)



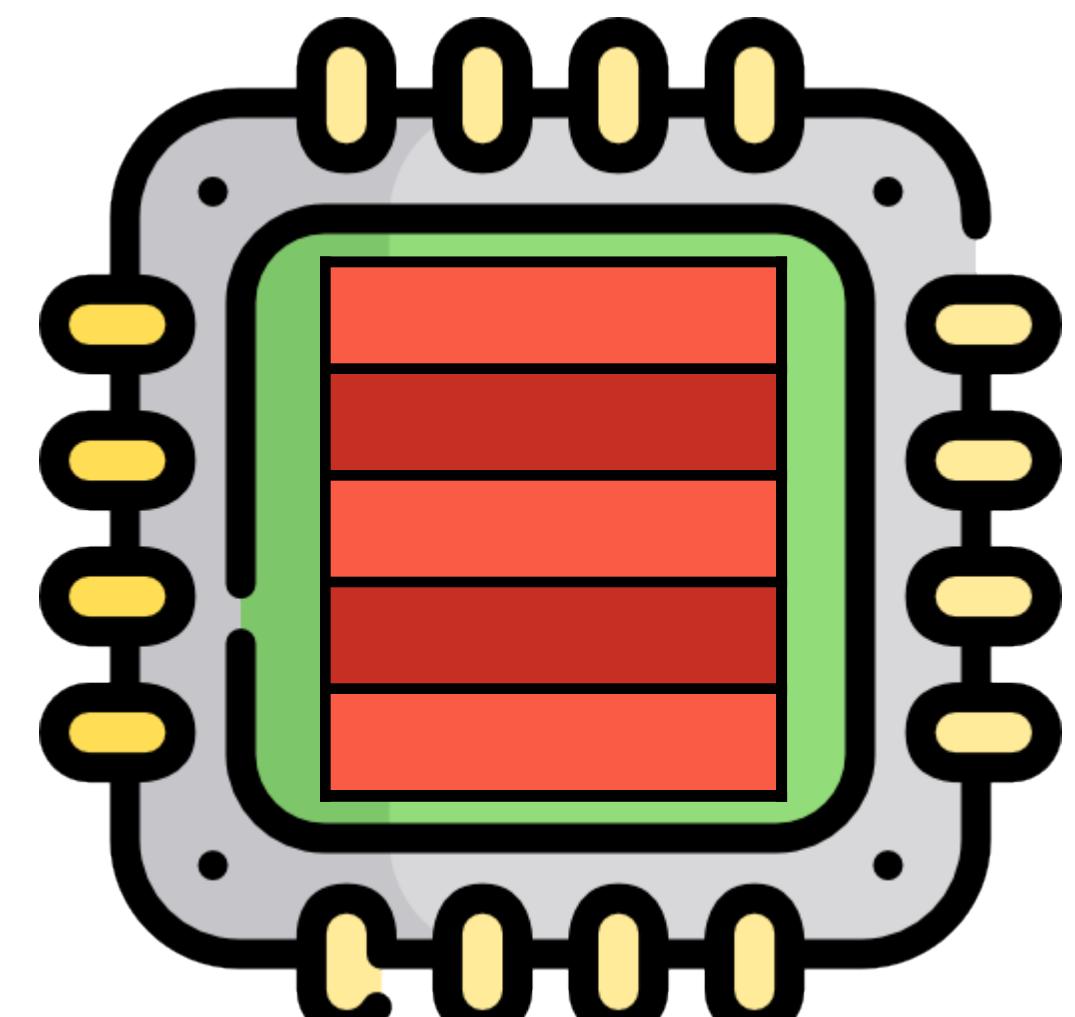
Cache state

Spectre V1

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

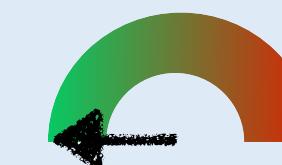


What is in A[128]?



Cache state

1a) Training



f(0); f(1); f(2); ...

1b) Prepare cache

2) Run f(128)

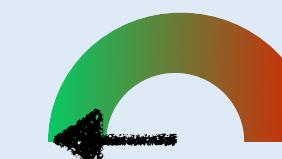
Spectre V1

```
A_size=16  
B[B[0] B[1] ... B[A[128]]]  
void f(int x)  
if (x < A_size)  
y = B[A[x]]
```



What is in **A**[128]?

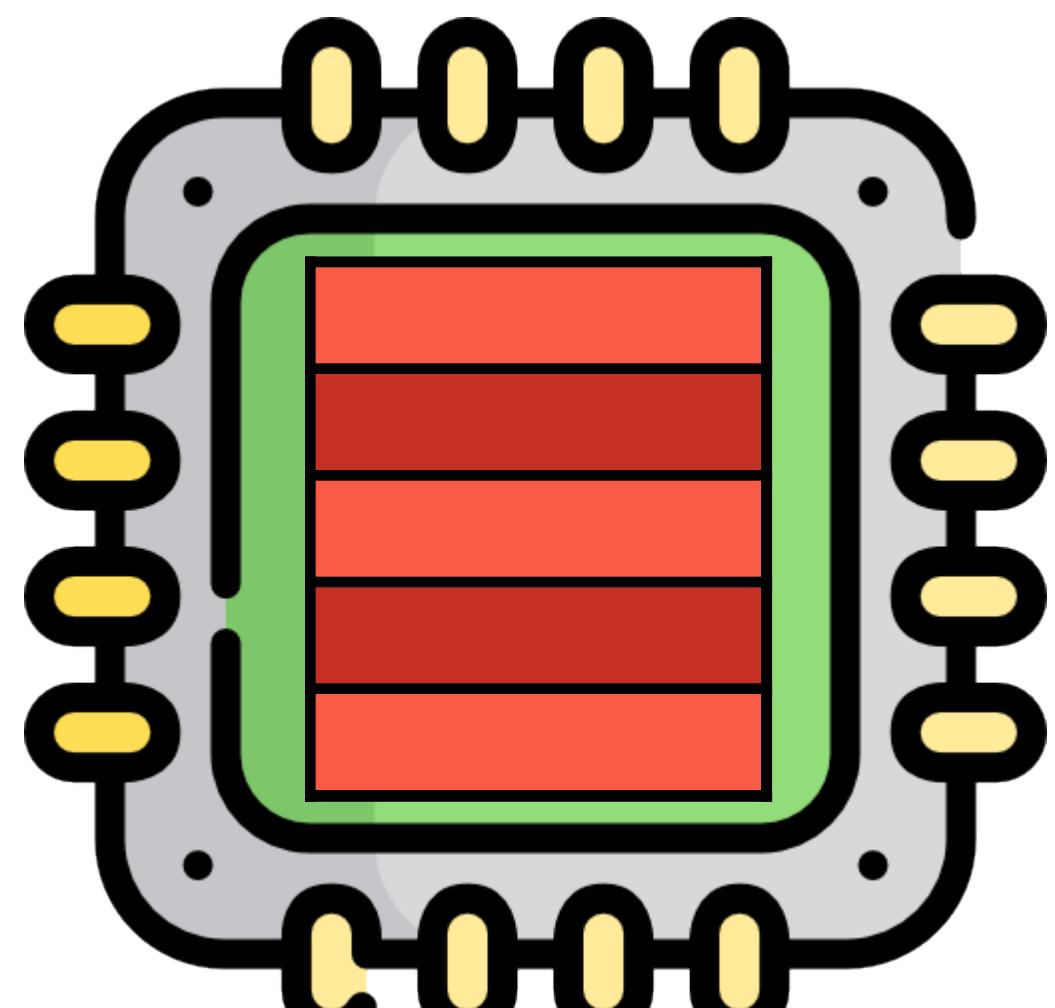
1a) Training



f(0); f(1); f(2); ...

1b) Prepare cache

2) Run **f**(128)



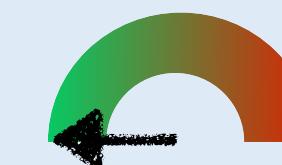
Spectre V1

```
A_size=16  
B[B[0] B[1] ... B[A[128]]]  
void f(int x)  
if (x < A_size)  
y = B[A[x]]
```



What is in A[128]?

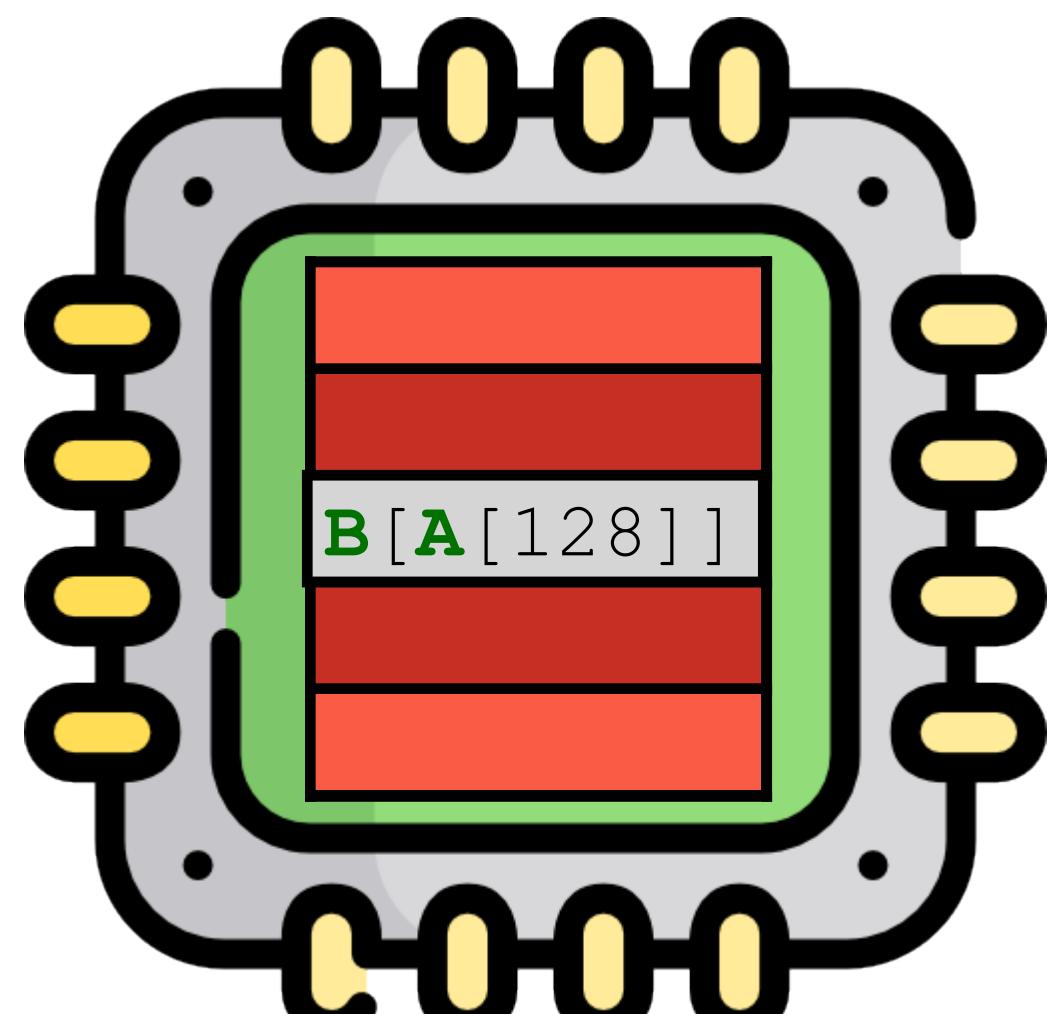
1a) Training



f(0); f(1); f(2); ...

1b) Prepare cache

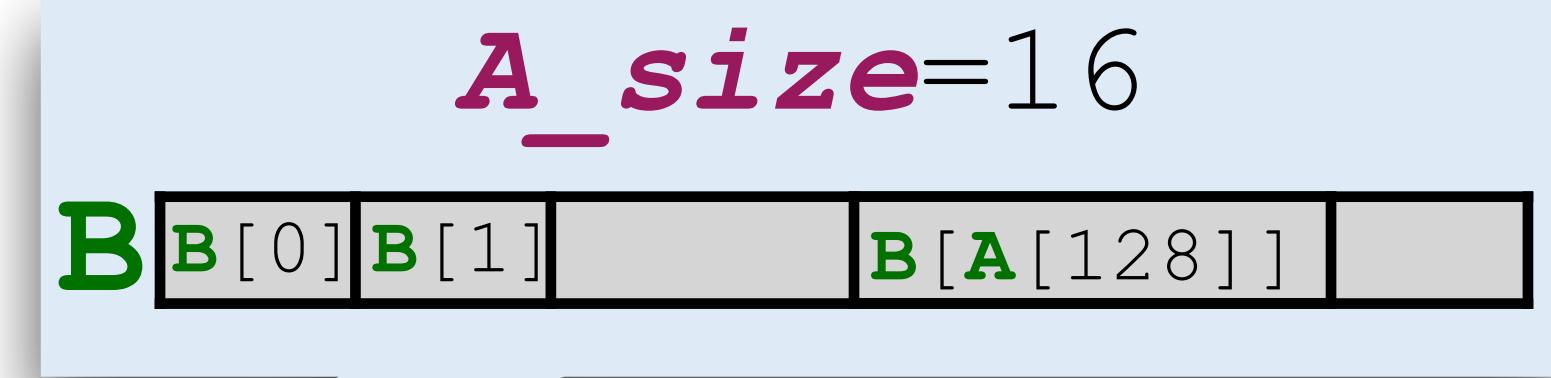
2) Run f(128)



Cache state

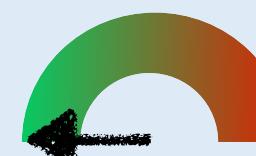
Spectre V1

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



What is in $A[128]$?

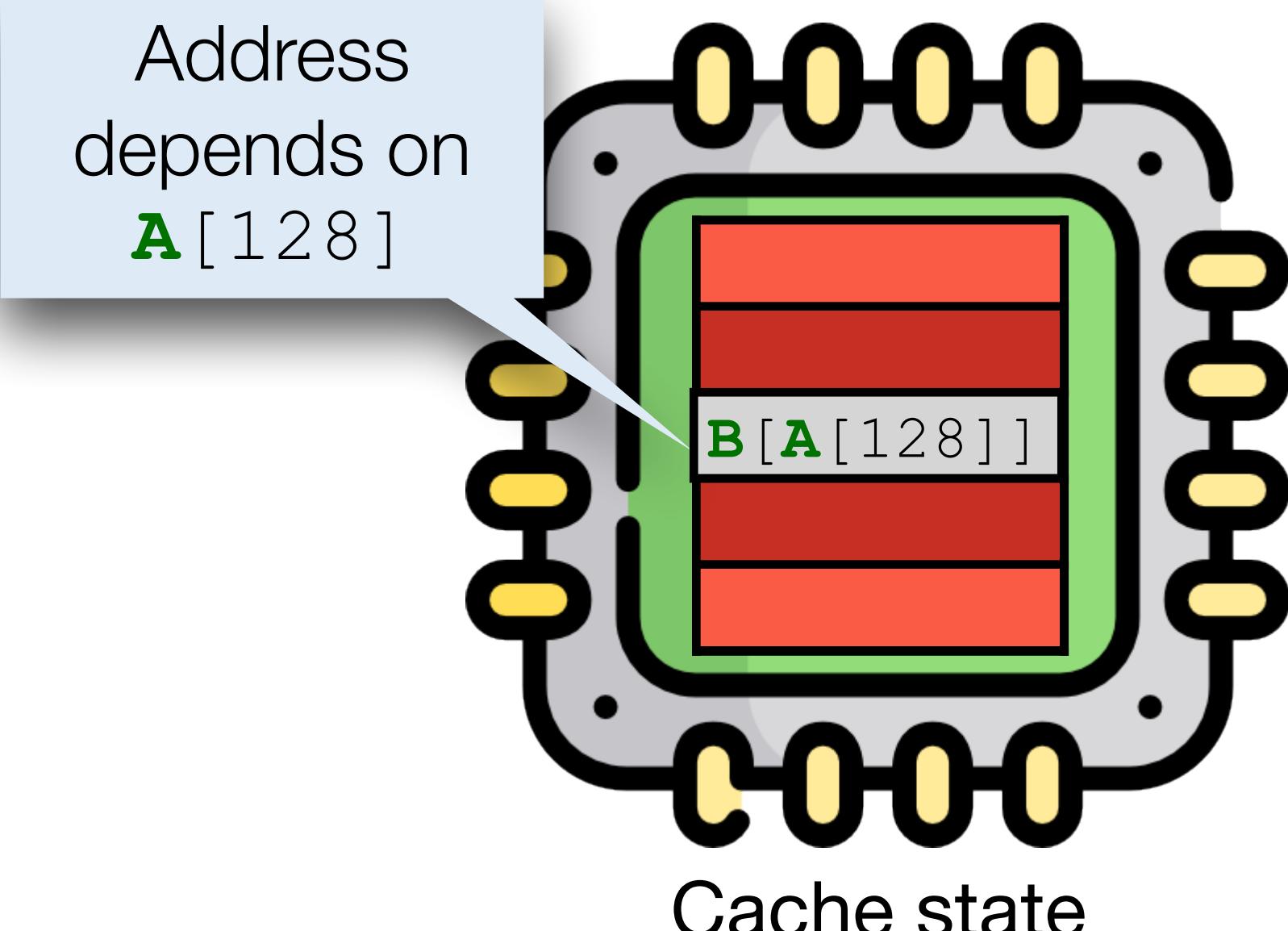
1a) Training



f(0); f(1); f(2); ...

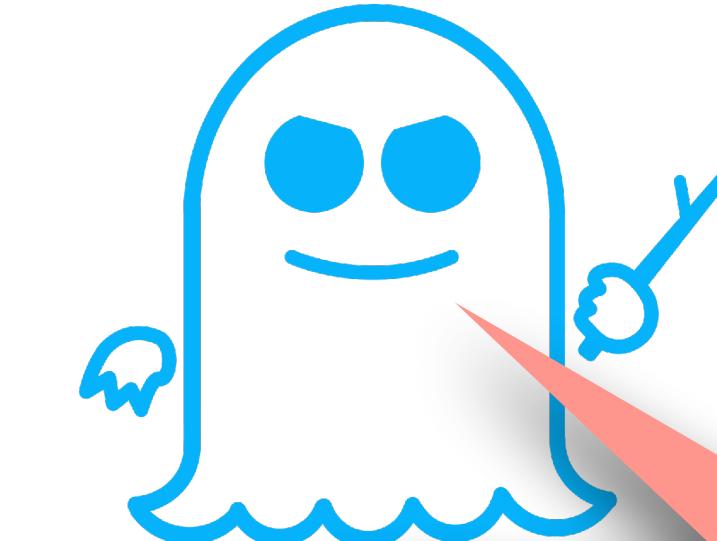
1b) Prepare cache

2) Run f(128)



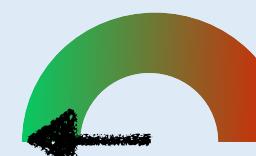
Spectre V1

```
A_size=16  
B[B[0] B[1] ... B[A[128]]]  
void f(int x)  
if (x < A_size)  
y = B[A[x]]
```



What is in **A**[128]?

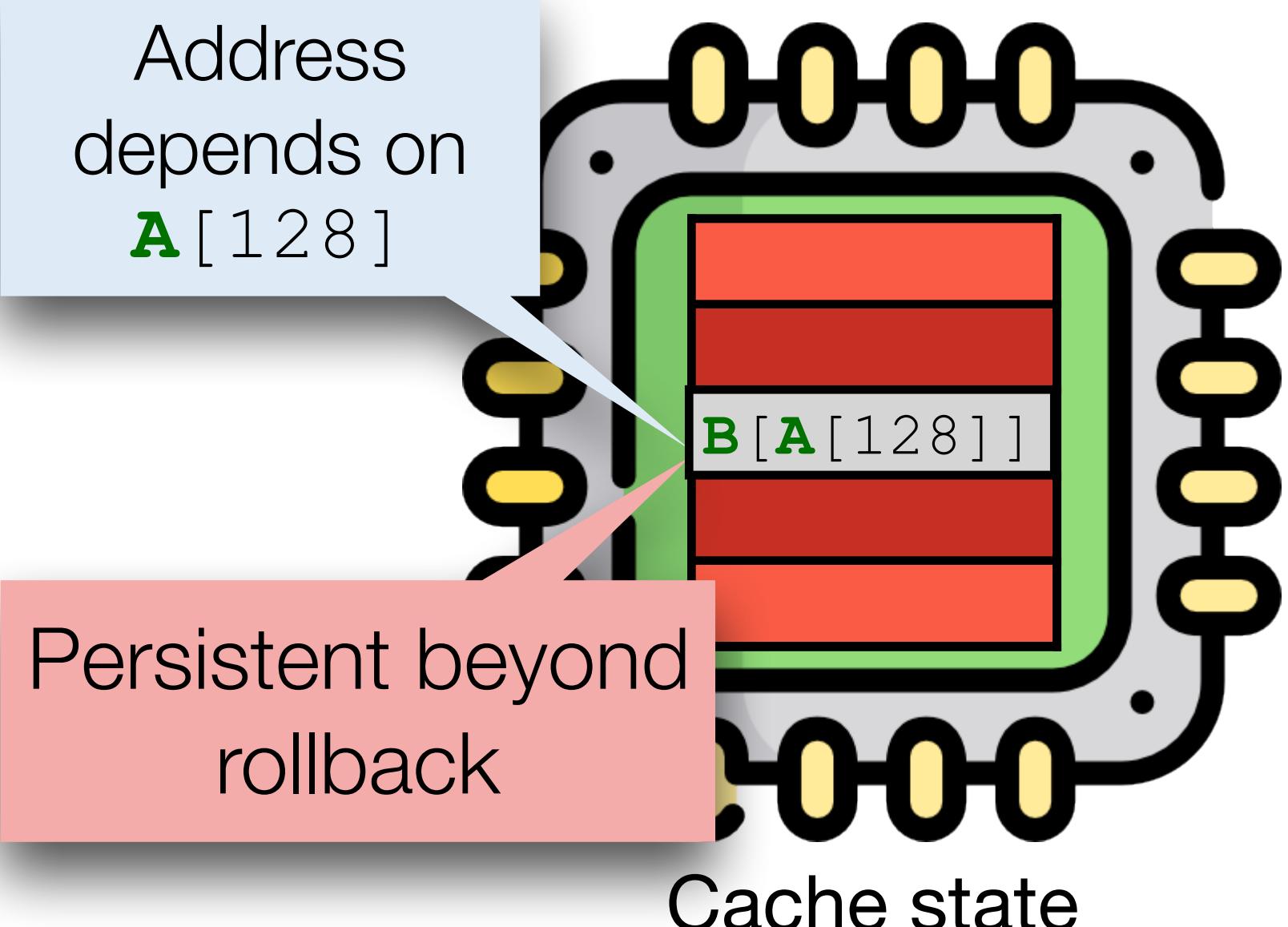
1a) Training



f(0); f(1); f(2); ...

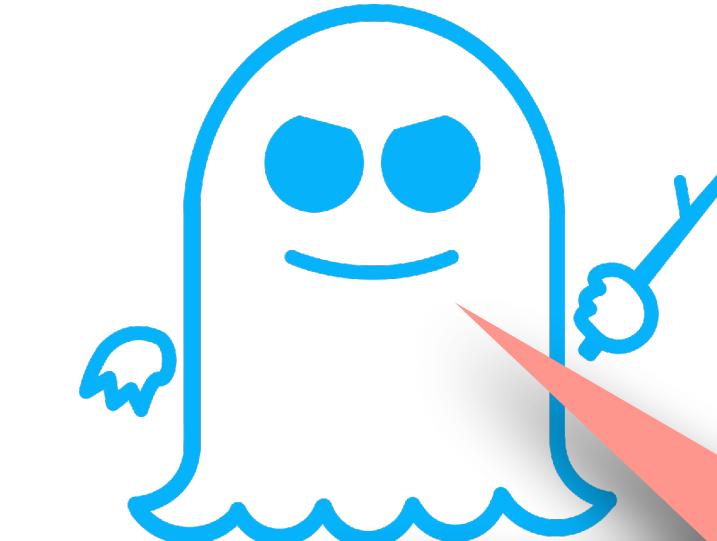
1b) Prepare cache

2) Run f(128)

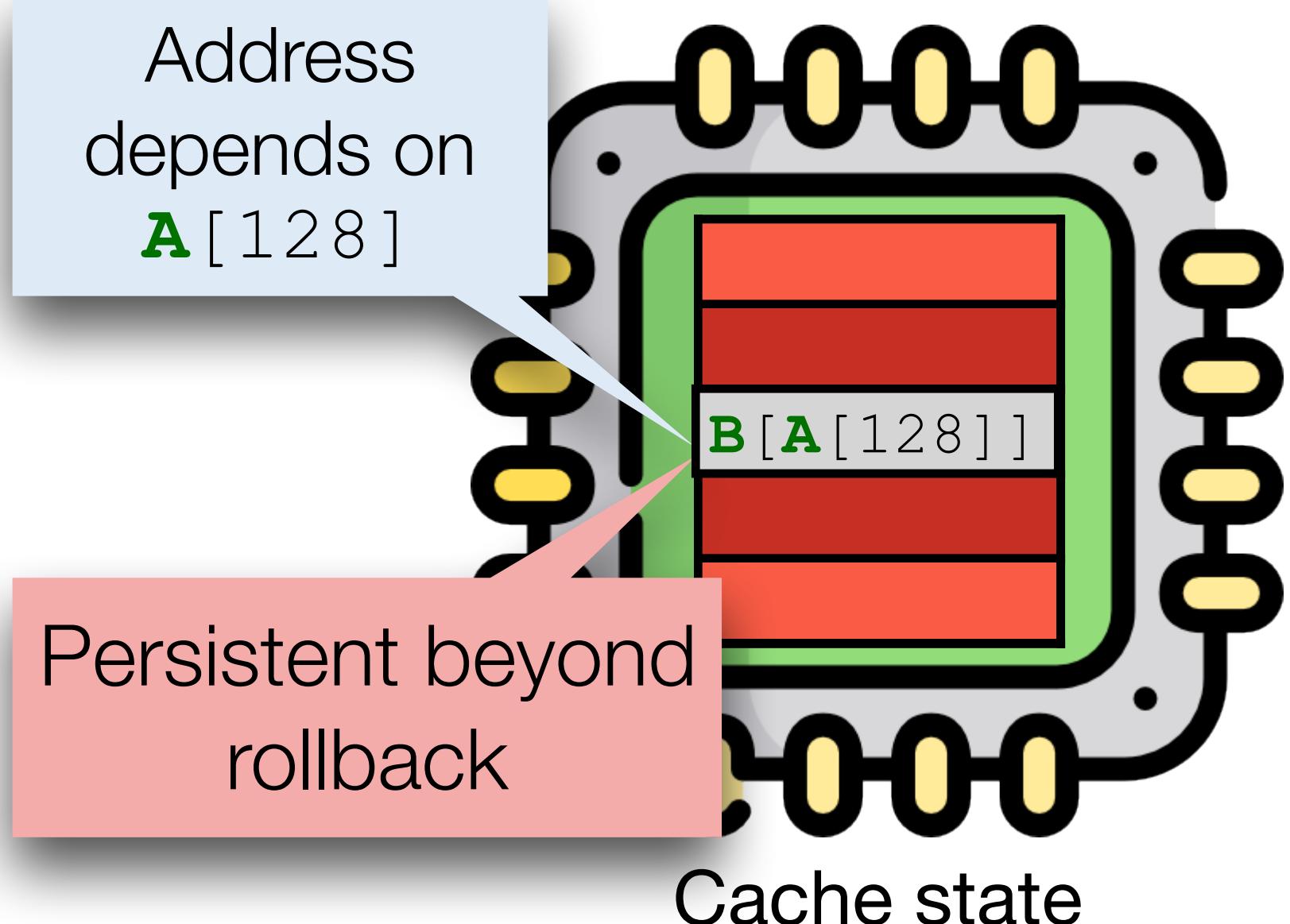


Spectre V1

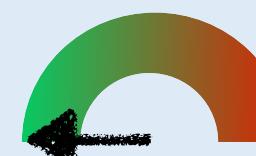
```
A_size=16  
B[B[0] B[1] ... B[A[128]]]  
  
void f(int x)  
if (x < A_size)  
y = B[A[x]]
```



What is in **A**[128]?



1a) Training



f(0); f(1); f(2); ...

1b) Prepare cache

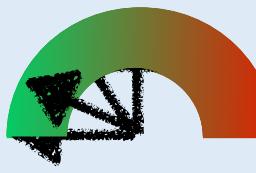
2) Run f(128)

3) Extract from cache

2. Speculative non-interference

Generalizing the Spectre V1 example

1a) Training



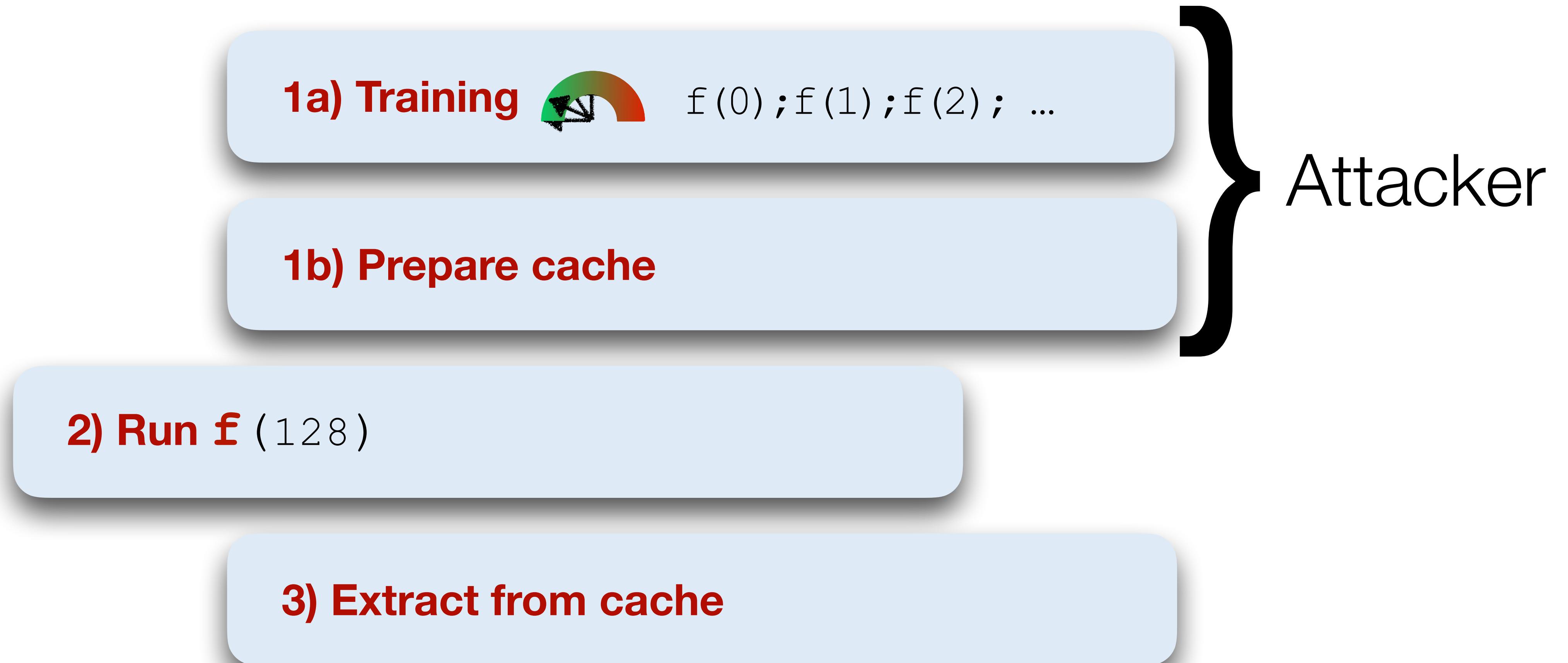
$f(0); f(1); f(2); \dots$

1b) Prepare cache

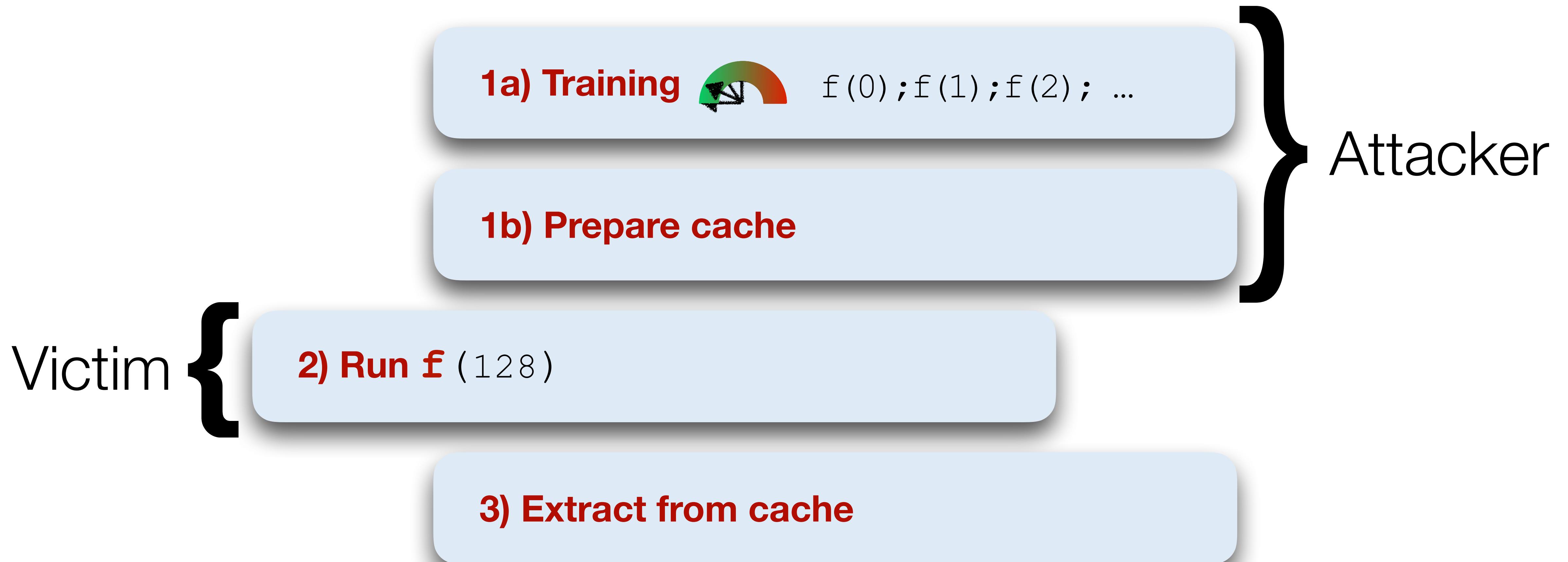
2) Run $f(128)$

3) Extract from cache

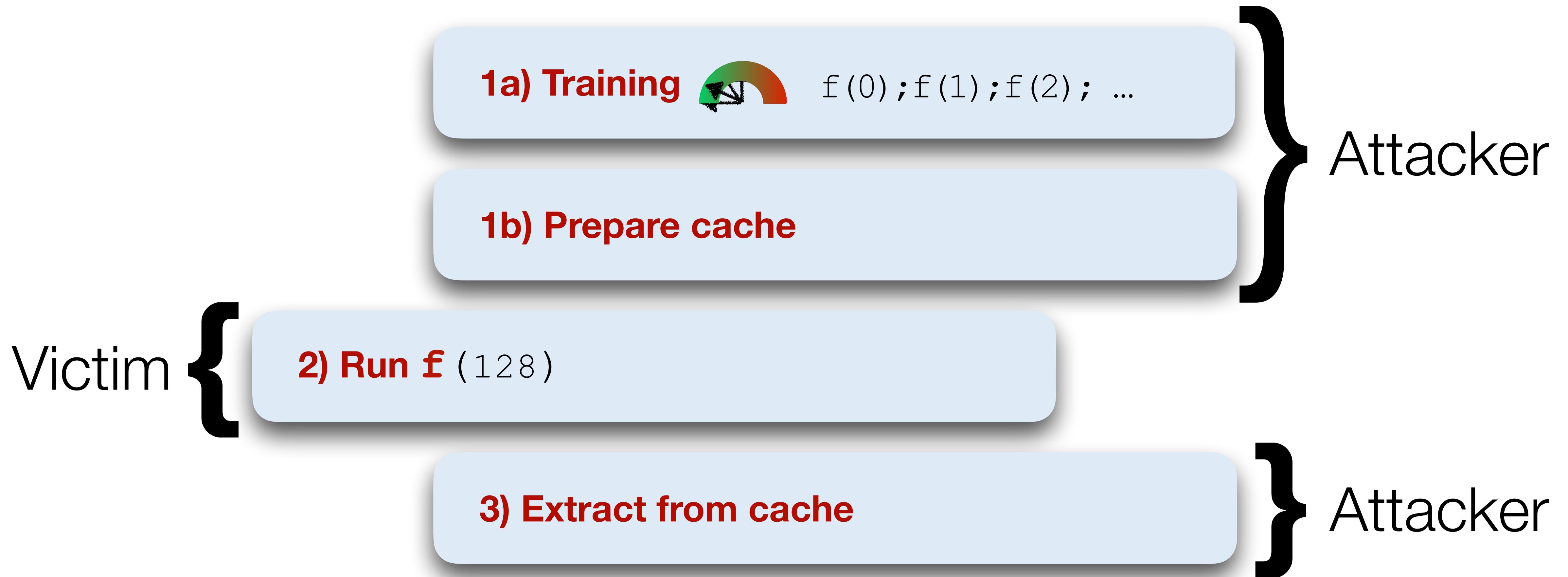
Generalizing the Spectre V1 example



Generalizing the Spectre V1 example



Generalizing the Spectre V1 example



Generalizing the Spectre V1 example

Victim {

2) Leaks information into microarchitectural state

3) Extracts information from microarchitecture

1) Prepares microarchitectural state

}

Attacker

}

Attacker

Speculative non-interference

Speculative non-interference

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

Speculative non-interference

Informally:

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

Leakage of **P** in
non-speculative
execution

?
≡

Leakage of **P** in
speculative
execution

Speculative non-interference

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

Informally:

Leakage of P in
non-speculative
execution

?
≡

Leakage of P in
speculative
execution

More formally:

Speculative non-interference

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

Informally:

Leakage of P in
non-speculative
execution

?
≡

Leakage of P in
speculative
execution

For all program states s and s' :

More formally:

Speculative non-interference

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

Informally:

Leakage of P in
non-speculative
execution

?
≡

Leakage of P in
speculative
execution

More formally:

For all program states s and s' :

$$P_{\text{non-spec}}(s) = P_{\text{non-spec}}(s')$$

Speculative non-interference

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

Informally:

Leakage of P in
non-speculative
execution

?
≡

Leakage of P in
speculative
execution

More formally:

For all program states s and s' :

$$P_{\text{non-spec}}(s) = P_{\text{non-spec}}(s')$$

$$\Rightarrow P_{\text{spec}}(s) = P_{\text{spec}}(s')$$

Speculative non-interference

Extended
with policies

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

Informally:

Leakage of P in
non-speculative
execution

?
≡

Leakage of P in
speculative
execution

More formally:

For all program states s and s' :

$$\begin{aligned} P_{\text{non-spec}}(s) &= P_{\text{non-spec}}(s') \\ \Rightarrow P_{\text{spec}}(s) &= P_{\text{spec}}(s') \end{aligned}$$

How to capture leakage into microarchitectural state?

How to capture leakage into microarchitectural state?

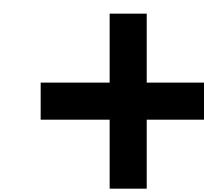
Non-speculative
semantics

Speculative
semantics

How to capture leakage into microarchitectural state?

Non-speculative
semantics

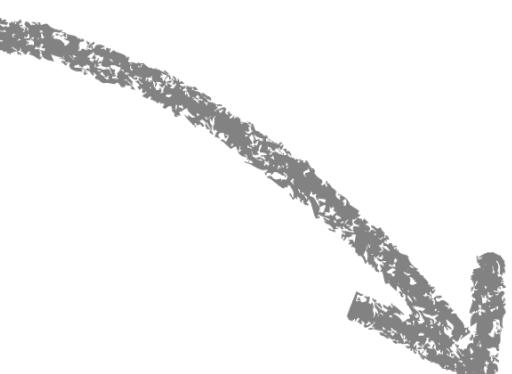
Speculative
semantics



Attacker/Observer
model

μ Assembly

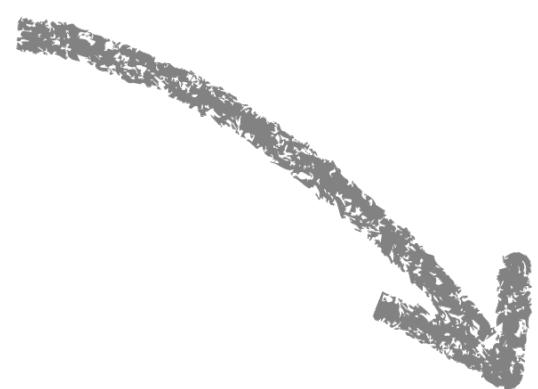
```
if (x < A_size)
    y = B[A[x]]
```



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

μ Assembly + Non-speculative semantics

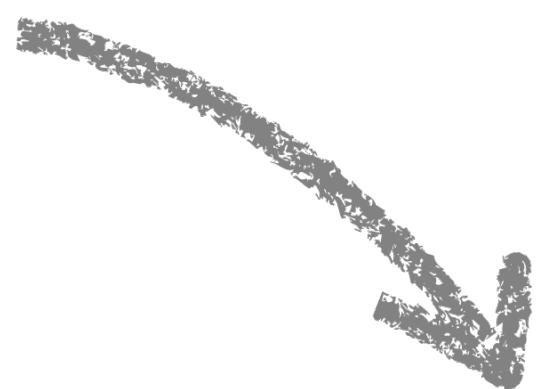
```
if (x < A_size)  
y = B[A[x]]
```



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

μ Assembly + Non-speculative semantics

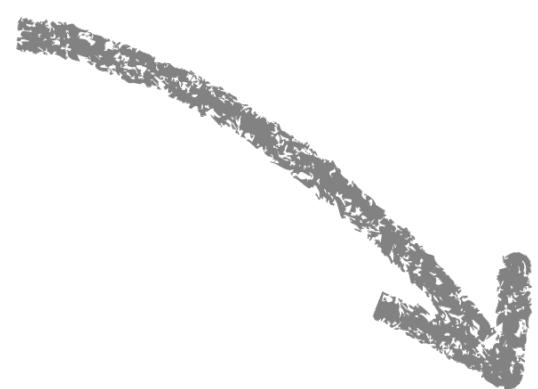
```
if (x < A_size)  
y = B[A[x]]
```



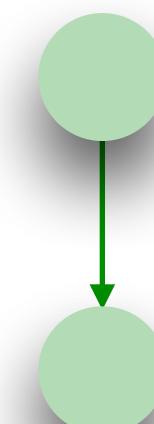
```
rax <- A_size  
rcx <- x  
jmp rcx≥rax, END  
L1: load rax, A + rcx  
     load rax, B + rax  
END:
```

μ Assembly + Non-speculative semantics

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

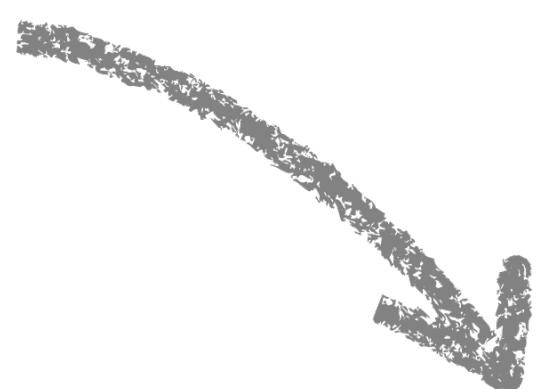


```
rax <- A_size  
rcx <- x  
jmp rcx≥rax, END  
L1: load rax, A + rcx  
      load rax, B + rax  
END:
```



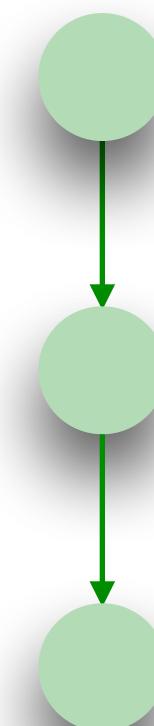
μ Assembly + Non-speculative semantics

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



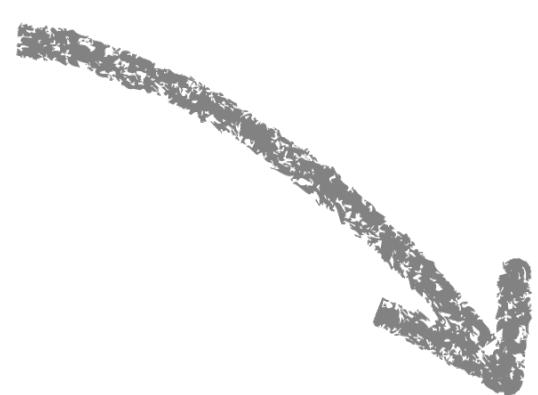
```
rax <- A_size  
rcx <- x
```

```
jmp rcx≥rax, END  
L1: load rax, A + rcx  
     load rax, B + rax  
END:
```



μ Assembly + Non-speculative semantics

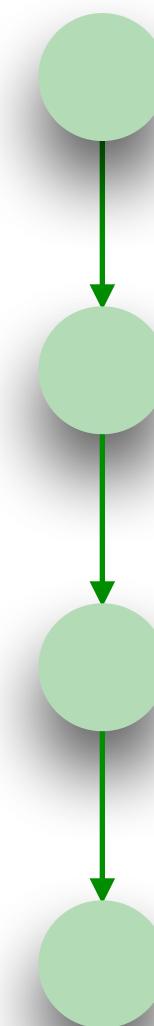
```
if (x < A_size)  
    y = B[A[x]]
```



```
rax <- A_size  
rcx <- x  
jmp rcx≥rax, END
```

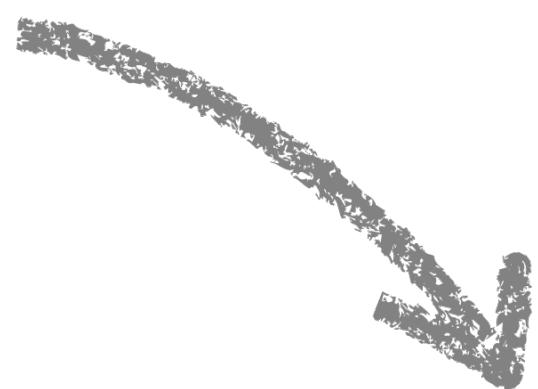
```
L1: load rax, A + rcx  
     load rax, B + rax
```

END:

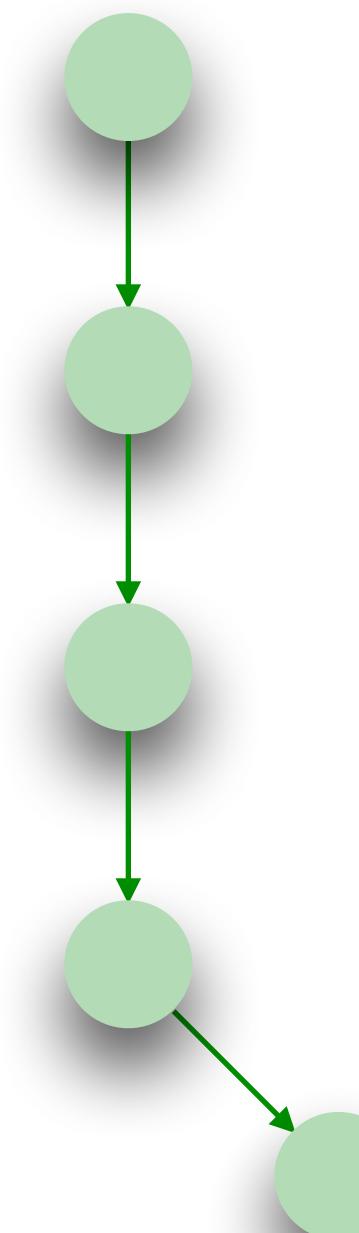


μ Assembly + Non-speculative semantics

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



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



Non-speculative semantics: Inference Rules

Expression evaluation

$$\llbracket n \rrbracket(a) = n \quad \llbracket x \rrbracket(a) = a(x) \quad \llbracket \ominus e \rrbracket(a) = \ominus \llbracket e \rrbracket(a) \quad \llbracket e_1 \otimes e_2 \rrbracket(a) = \llbracket e_1 \rrbracket(a) \otimes \llbracket e_2 \rrbracket(a)$$

Instruction evaluation

SKIP

$$\frac{p(a(\mathbf{pc})) = \mathbf{skip}}{\langle m, a \rangle \rightarrow \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1] \rangle}$$

BARRIER

$$\frac{p(a(\mathbf{pc})) = \mathbf{spbarr}}{\langle m, a \rangle \rightarrow \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1] \rangle}$$

ASSIGN

$$\frac{p(a(\mathbf{pc})) = x \leftarrow e \quad x \neq \mathbf{pc}}{\langle m, a \rangle \rightarrow \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1, x \mapsto \llbracket e \rrbracket(a)] \rangle}$$

CONDITIONAL UPDATE-SAT

$$\frac{p(a(\mathbf{pc})) = x \xleftarrow{e'} e \quad \llbracket e' \rrbracket(a) = 0 \quad x \neq \mathbf{pc}}{\langle m, a \rangle \rightarrow \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1, x \mapsto \llbracket e \rrbracket(a)] \rangle}$$

CONDITIONAL UPDATE-UNSAT

$$\frac{p(a(\mathbf{pc})) = x \xleftarrow{e'} e \quad \llbracket e' \rrbracket(a) \neq 0 \quad x \neq \mathbf{pc}}{\langle m, a \rangle \rightarrow \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1] \rangle}$$

TERMINATE

$$\frac{p(a(\mathbf{pc})) = \perp}{\langle m, a \rangle \rightarrow \langle m, a[\mathbf{pc} \mapsto \perp] \rangle}$$

LOAD

$$\frac{p(a(\mathbf{pc})) = \mathbf{load} \ x, e \quad x \neq \mathbf{pc} \quad n = \llbracket e \rrbracket(a)}{\langle m, a \rangle \xrightarrow{\text{load } n} \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1, x \mapsto m(n)] \rangle}$$

STORE

$$\frac{p(a(\mathbf{pc})) = \mathbf{store} \ x, e \quad n = \llbracket e \rrbracket(a)}{\langle m, a \rangle \xrightarrow{\text{store } n} \langle m[n \mapsto a(x)], a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1] \rangle}$$

BEQZ-SAT

$$\frac{p(a(\mathbf{pc})) = \mathbf{beqz} \ x, \ell \quad a(x) = 0}{\langle m, a \rangle \xrightarrow{\mathbf{pc} \ \ell} \langle m, a[\mathbf{pc} \mapsto \ell] \rangle}$$

BEQZ-UNSAT

$$\frac{p(a(\mathbf{pc})) = \mathbf{beqz} \ x, \ell \quad a(x) \neq 0}{\langle m, a \rangle \xrightarrow{\mathbf{pc} \ a(\mathbf{pc})+1} \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1] \rangle}$$

JMP

$$\frac{p(a(\mathbf{pc})) = \mathbf{jmp} \ e \quad \ell = \llbracket e \rrbracket(a)}{\langle m, a \rangle \xrightarrow{\mathbf{pc} \ \ell} \langle m, a[\mathbf{pc} \mapsto \ell] \rangle}$$

Speculative semantics

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

Speculative semantics

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

Starts **speculative transactions** upon branches

END:

Speculative semantics

```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
```

L1: load rax, **A** + rcx
load rax, **B** + rax
END:

Starts **speculative transactions** upon branches

Committed upon correct speculation

Speculative semantics

```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
```

L1: load rax, **A** + rcx
load rax, **B** + rax

END:

Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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

Starts **speculative transactions** upon branches

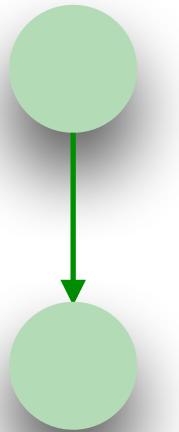
Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
```

L1: load rax, **A** + rcx
load rax, **B** + rax
END:



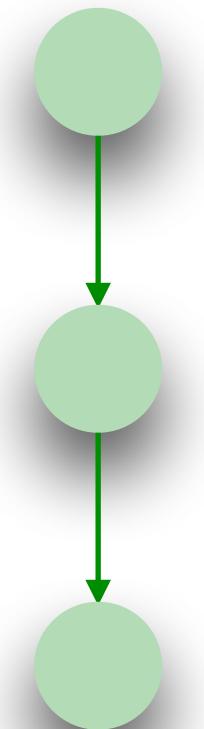
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



Starts **speculative transactions** upon branches

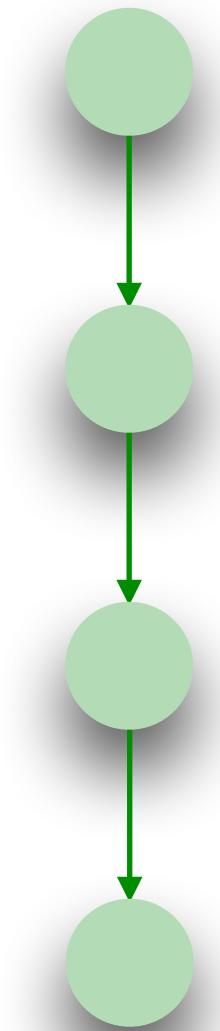
Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

```
rax <- A_size  
rcx <- x  
jmp rcx≥rax, END
```

L1: load rax, A + rcx
load rax, B + rax
END:



Starts **speculative transactions** upon branches

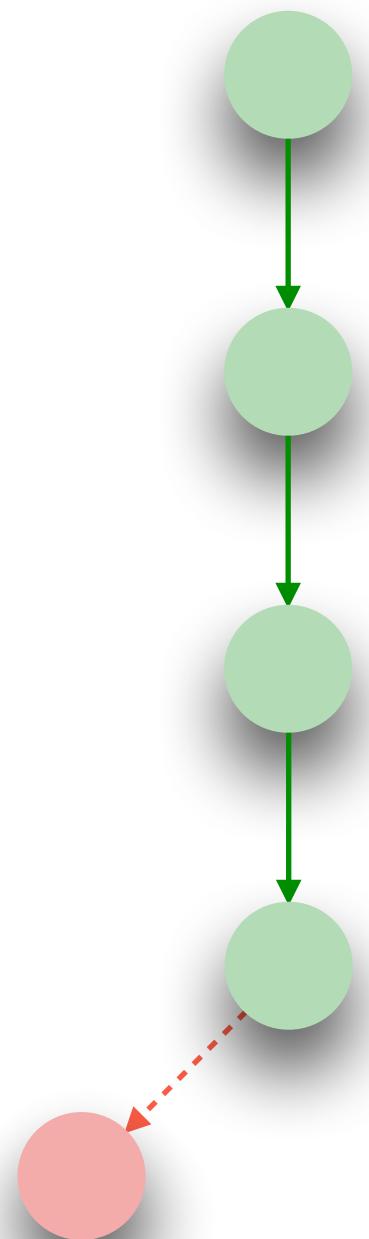
Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

```
rax <- A_size  
rcx <- x  
jmp rcx≥rax, END
```

```
L1: load rax, A + rcx  
load rax, B + rax  
END:
```



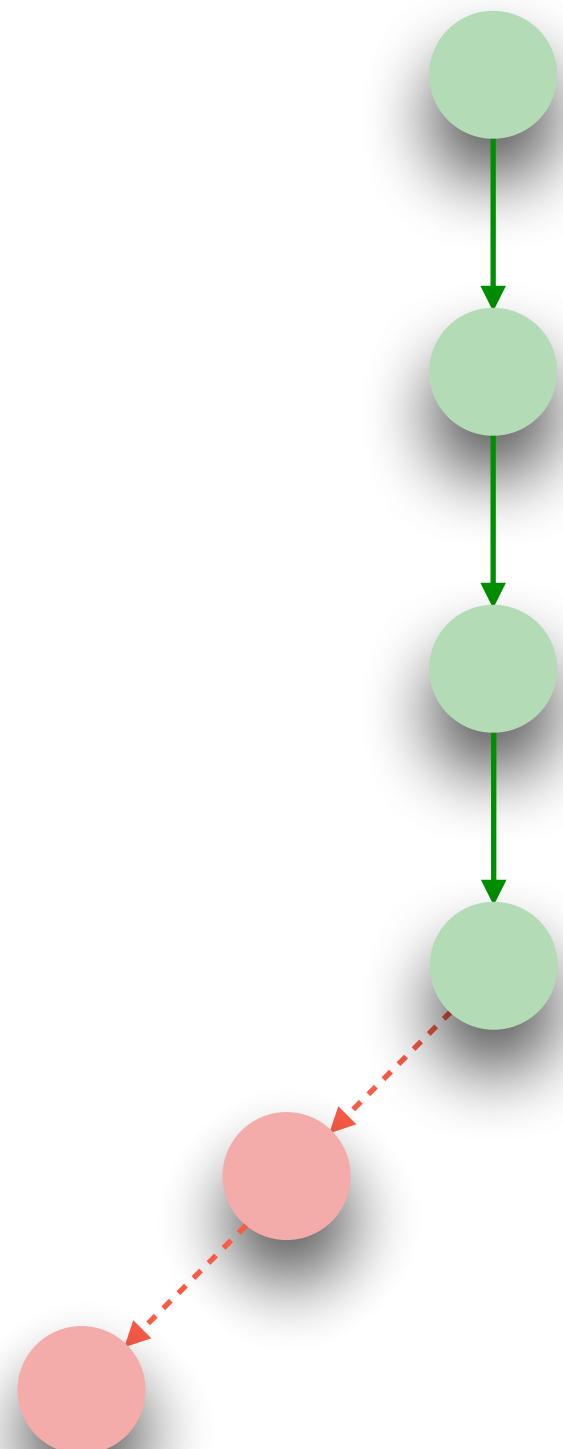
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



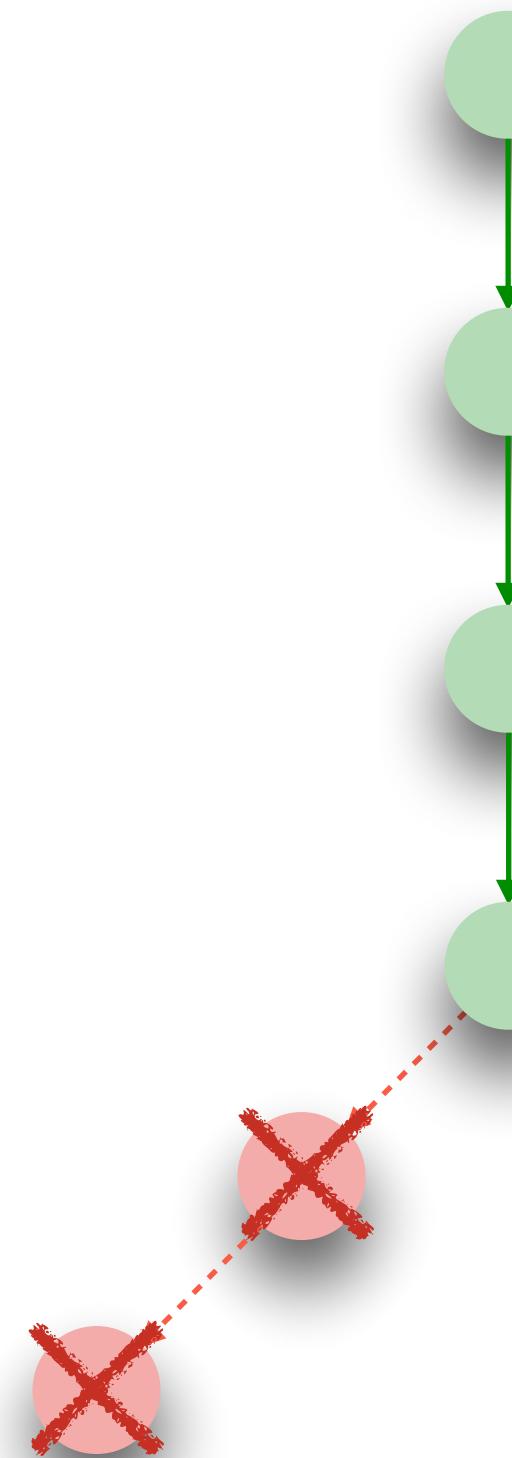
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



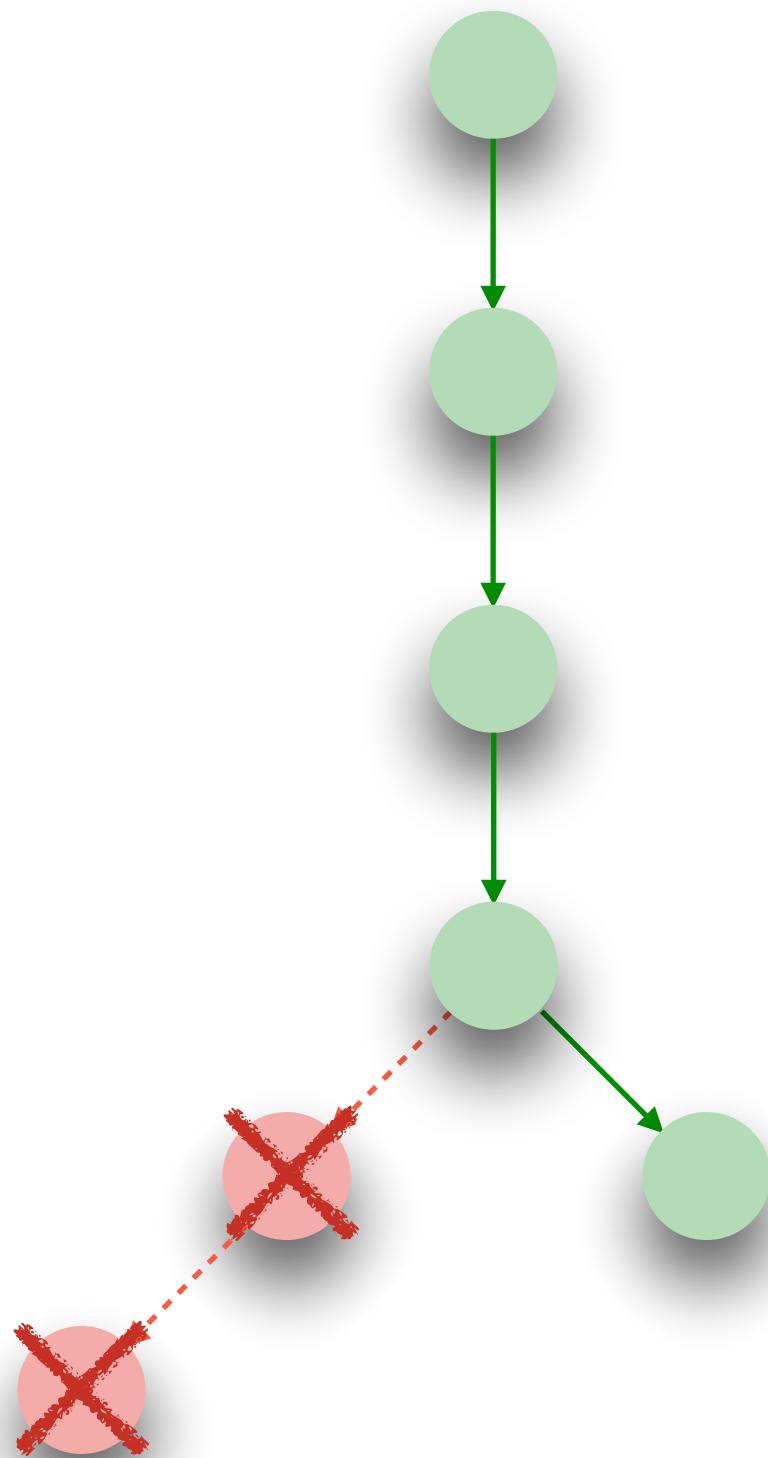
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



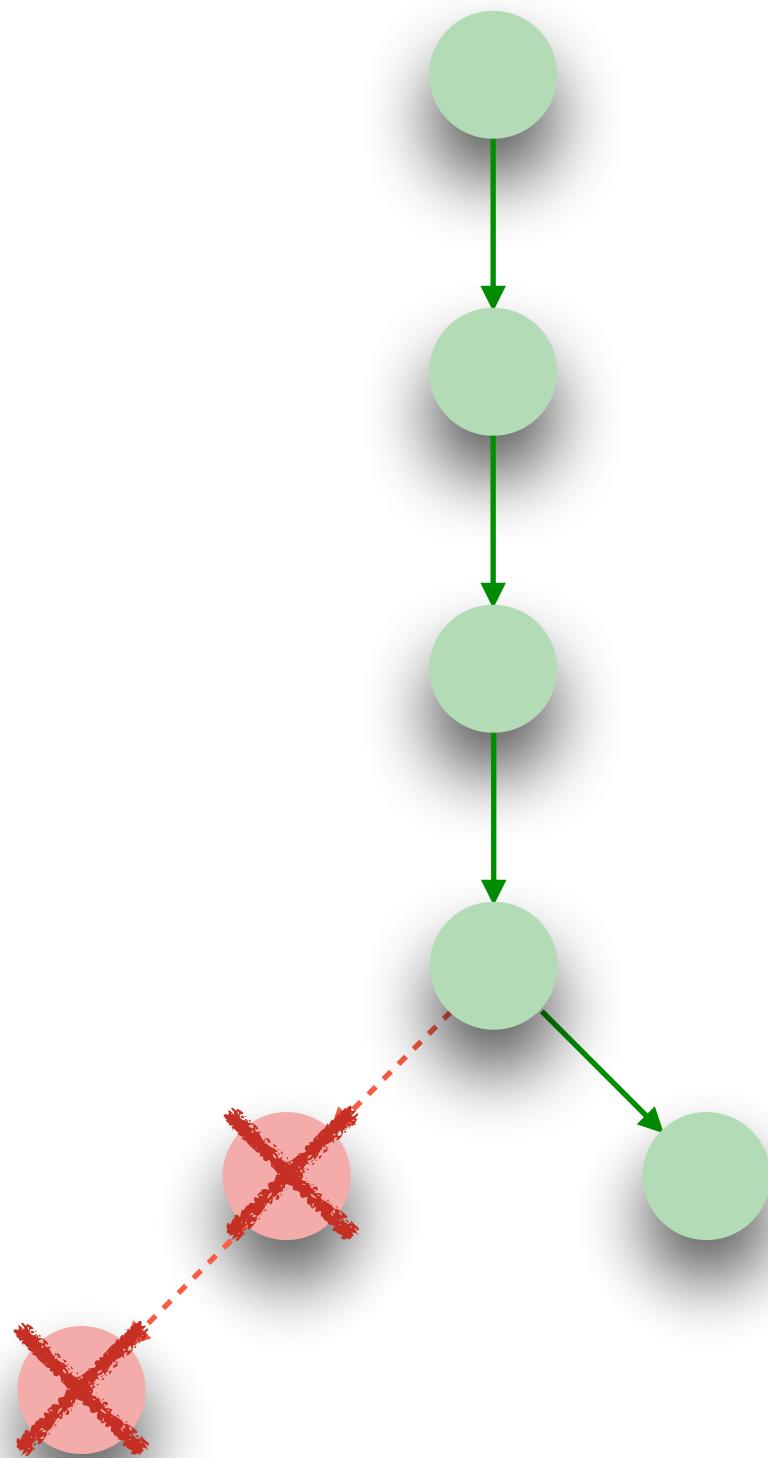
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



Starts **speculative transactions** upon branches

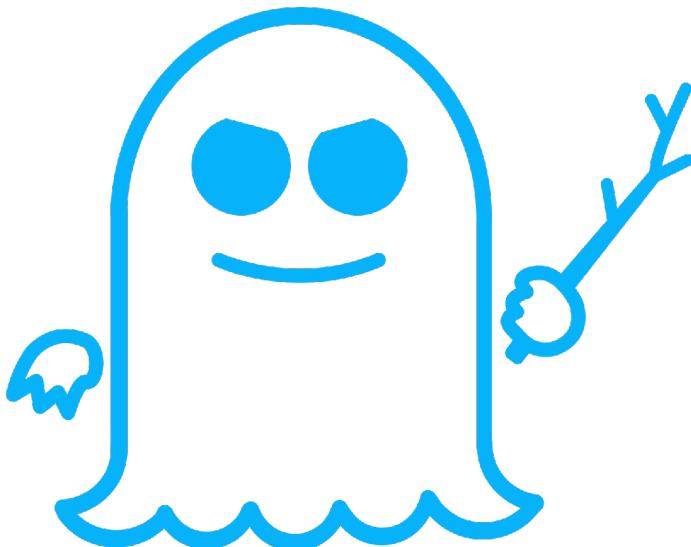
Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle O determines branch prediction + length of speculative window

Observer model: Leakage into μarchitectural state

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

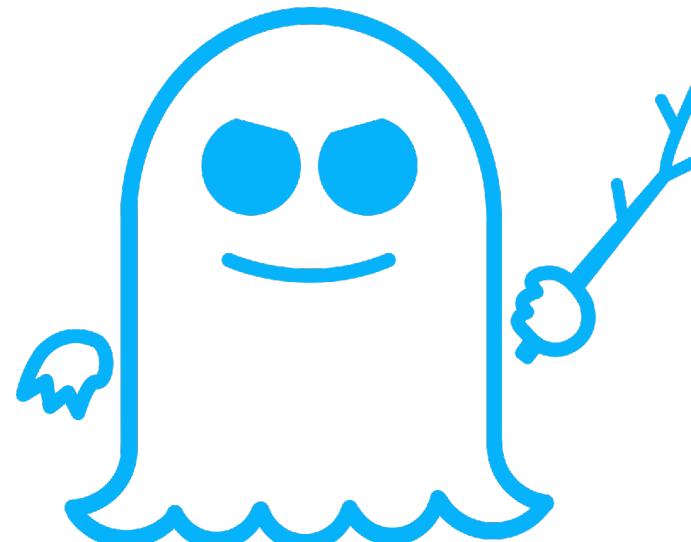


Observer model: Leakage into μarchitectural state

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

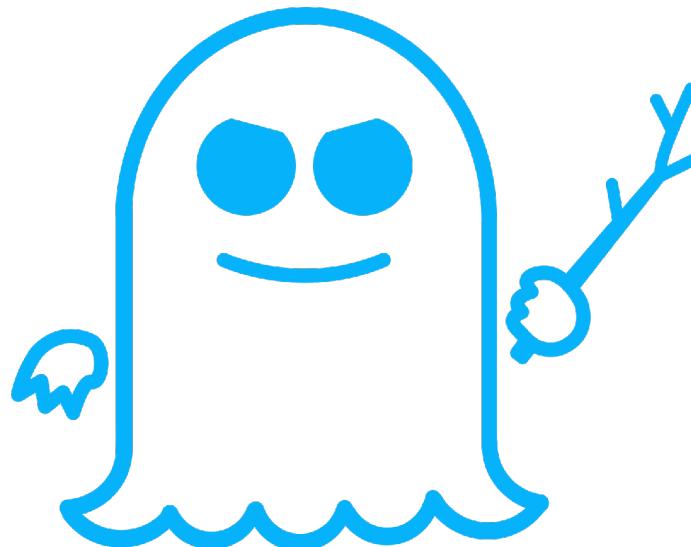
Attacker can observe:

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



Observer model: Leakage into μarchitectural state

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



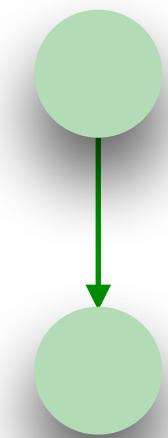
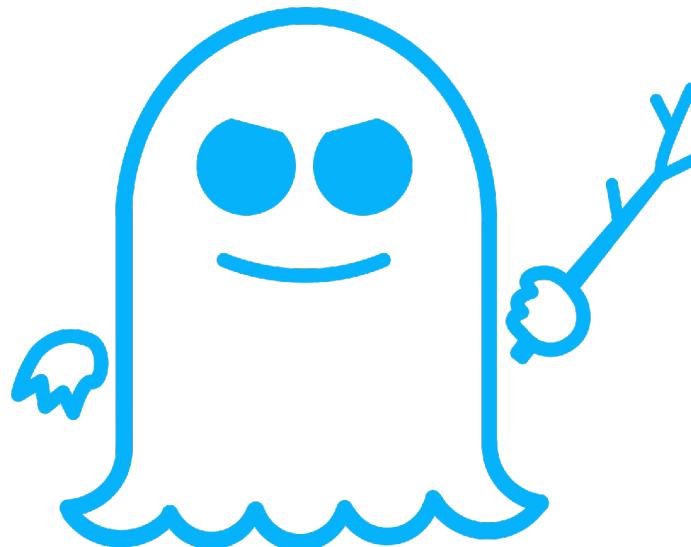
Attacker can observe:

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

Observer model: Leakage into μarchitectural state

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

END:

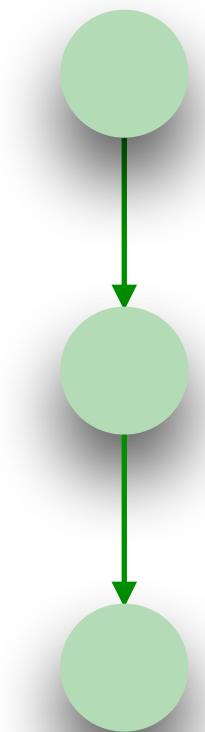


Attacker can observe:

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

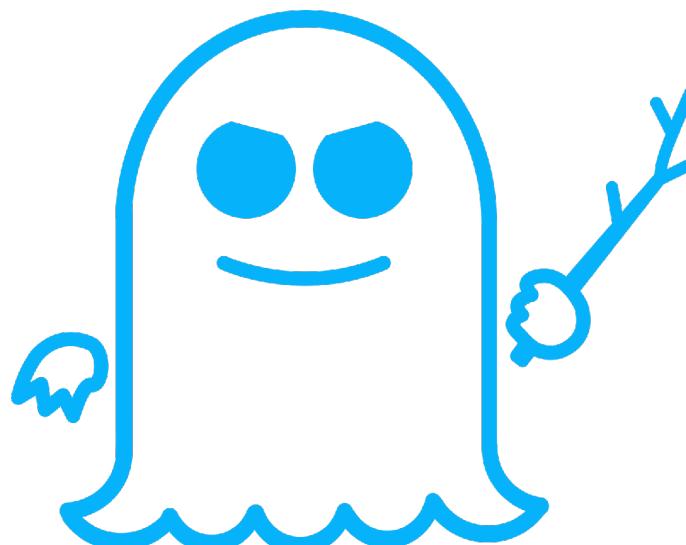
Observer model: Leakage into μarchitectural state

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



Attacker can observe:

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



Observer model: Leakage into μarchitectural state

```
rax <- A_size
```

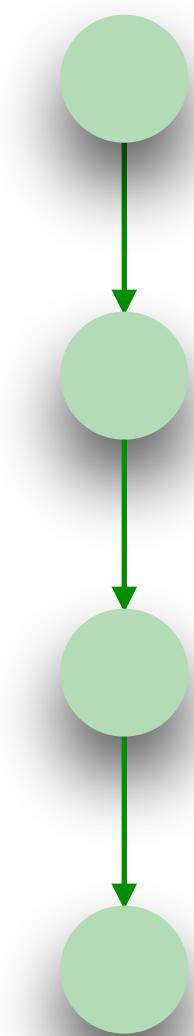
```
rcx <- x
```

```
jmp rcx≥rax, END
```

L1: load rax, A + rcx

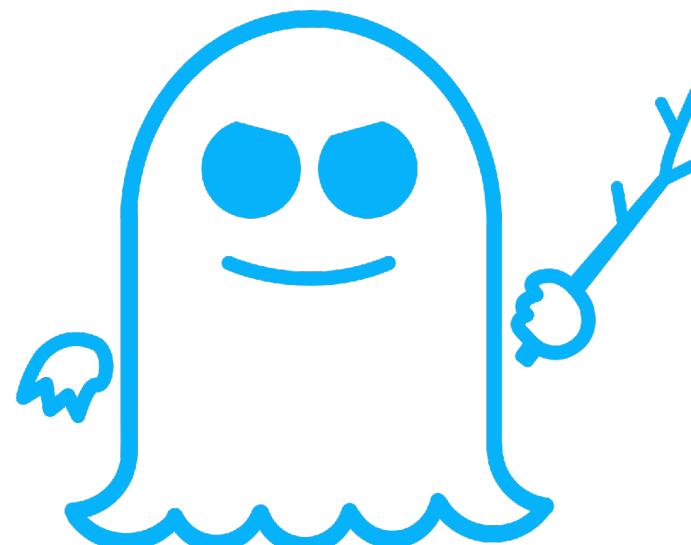
load rax, B + rax

END:



Attacker can observe:

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



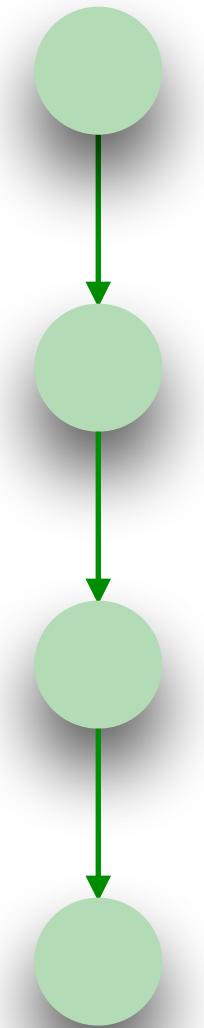
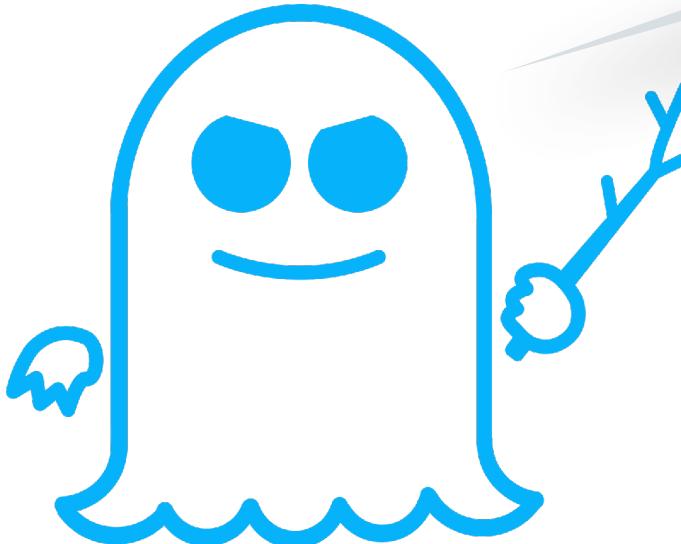
Observer model: Leakage into μarchitectural state

```
rax <- A_size  
rcx <- x  
jmp rcx≥rax, END
```

```
L1: load rax, A + rcx  
load rax, B + rax
```

END:

start;
pc L1

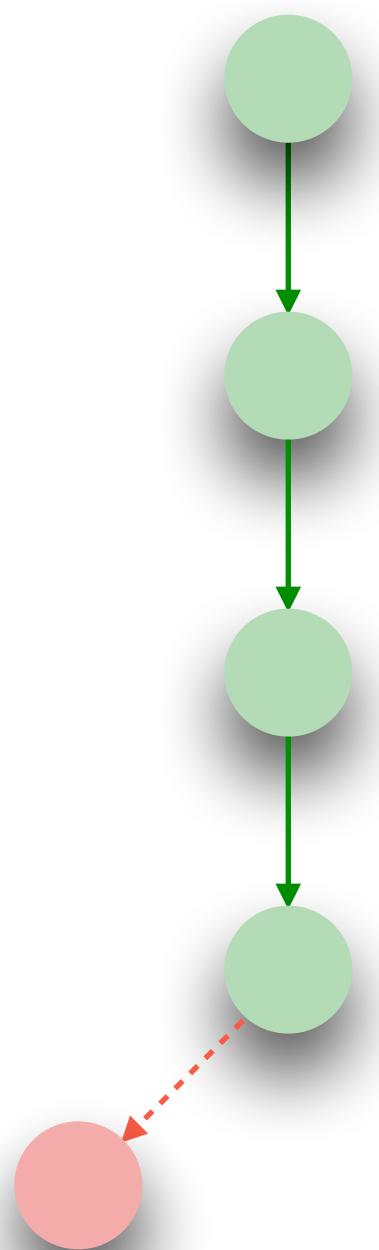
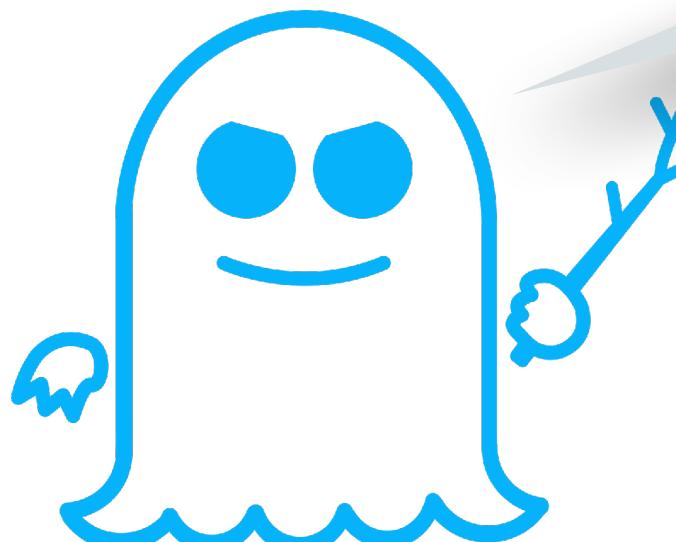


Attacker can observe:

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

Observer model: Leakage into μarchitectural state

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

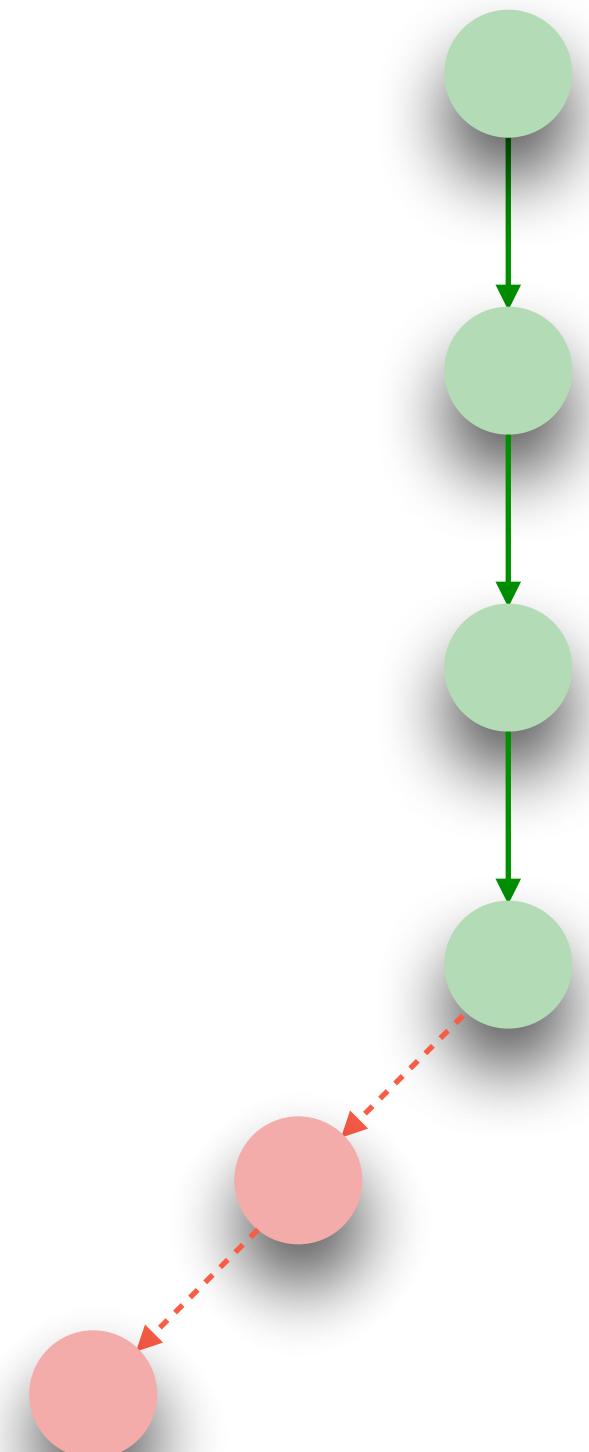
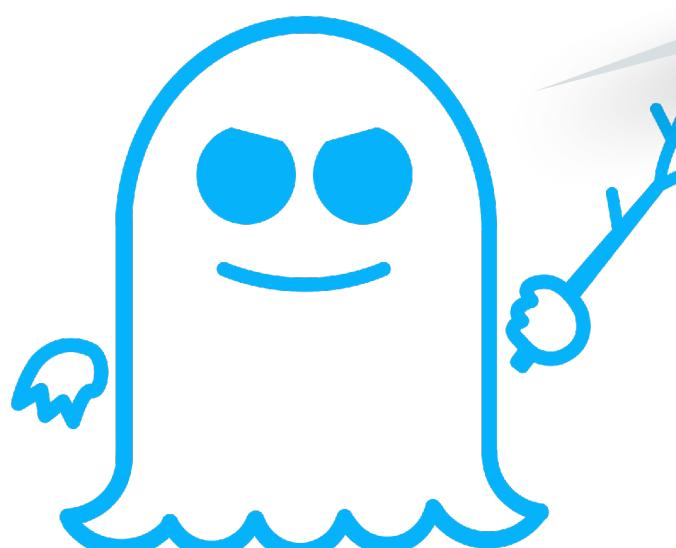


Attacker can observe:

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

Observer model: Leakage into μarchitectural state

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

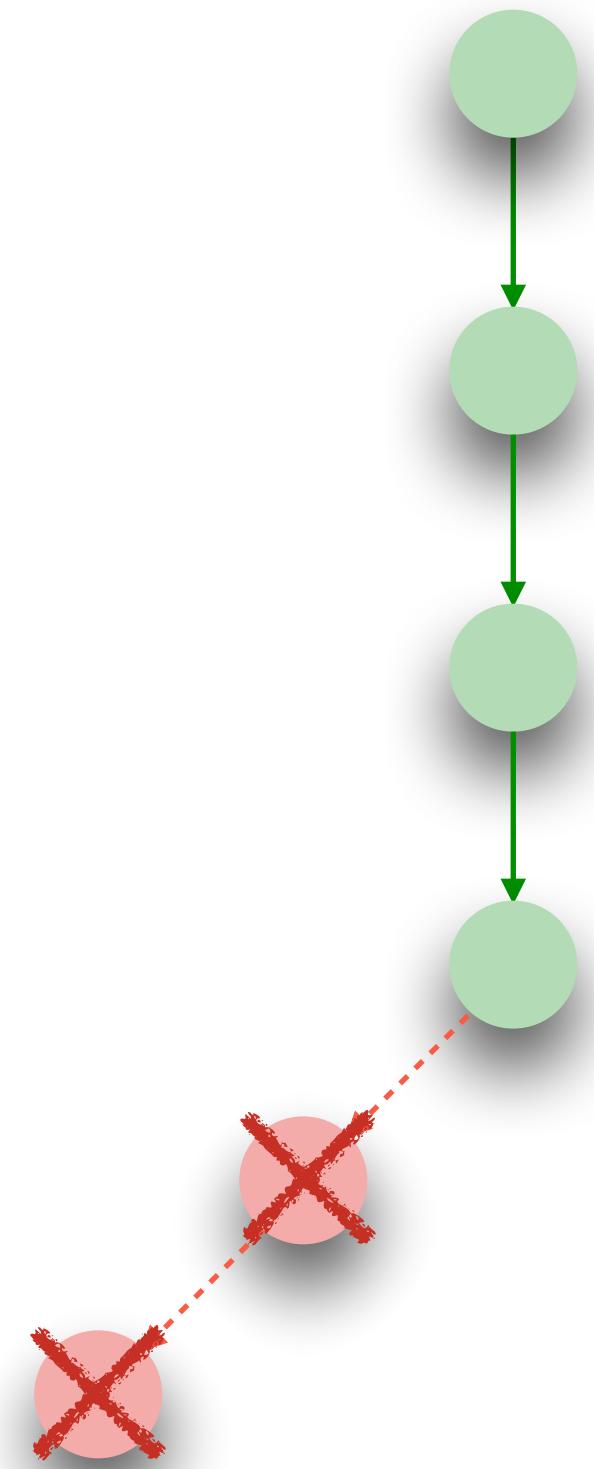
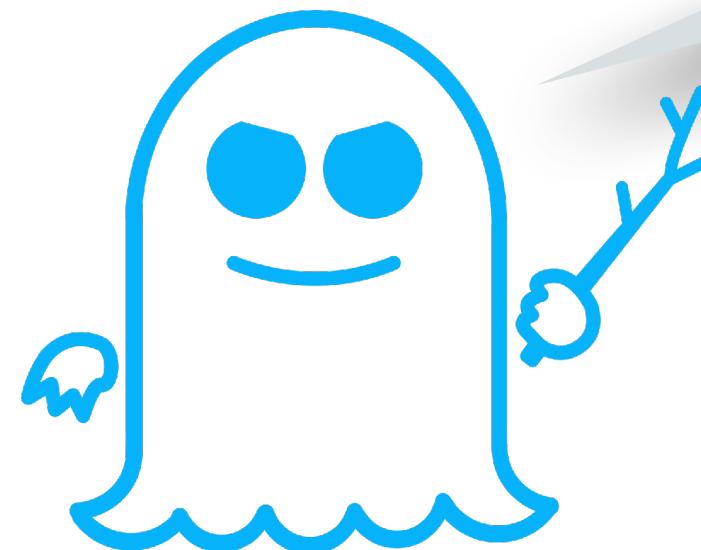


Attacker can observe:

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

Observer model: Leakage into μarchitectural state

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



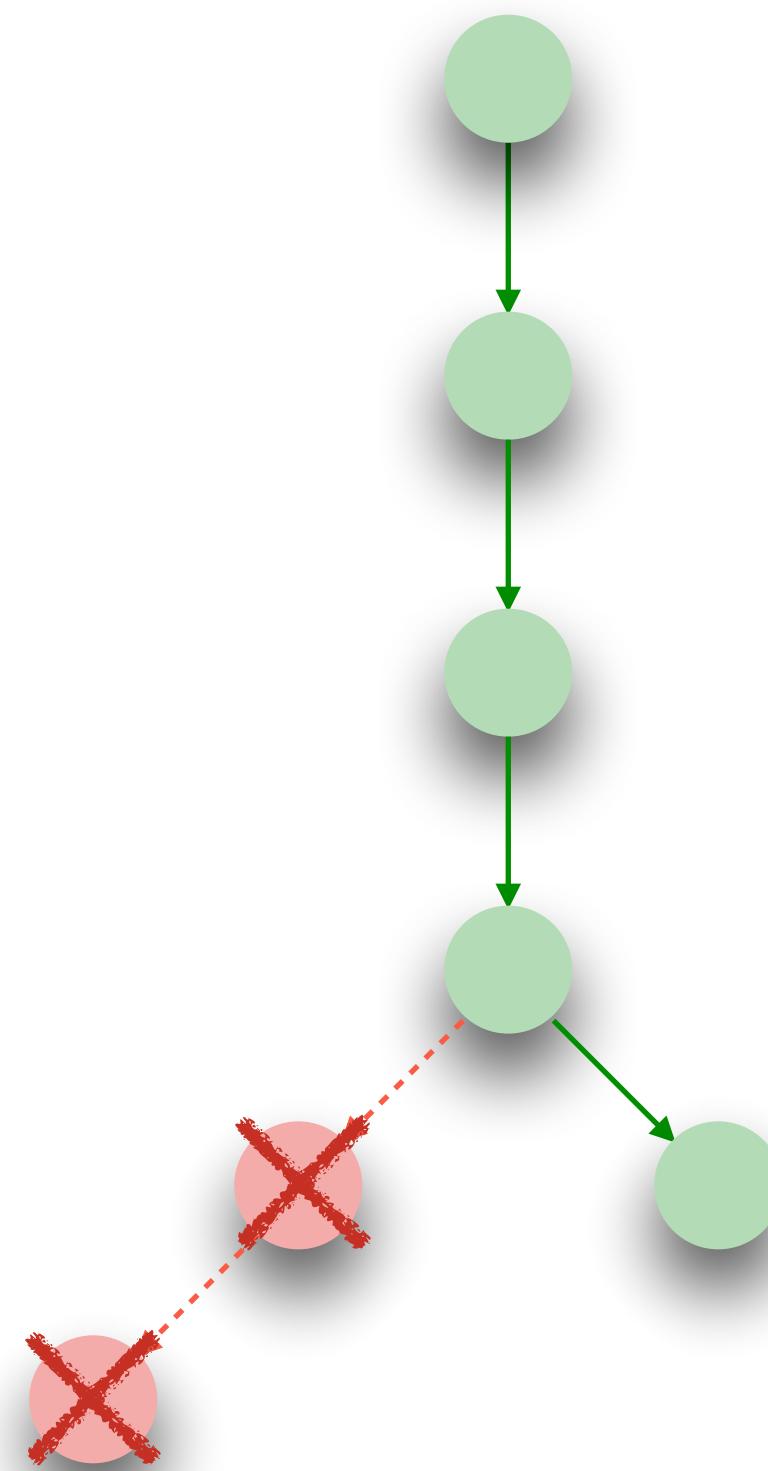
Attacker can observe:

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

Observer model: Leakage into μarchitectural state

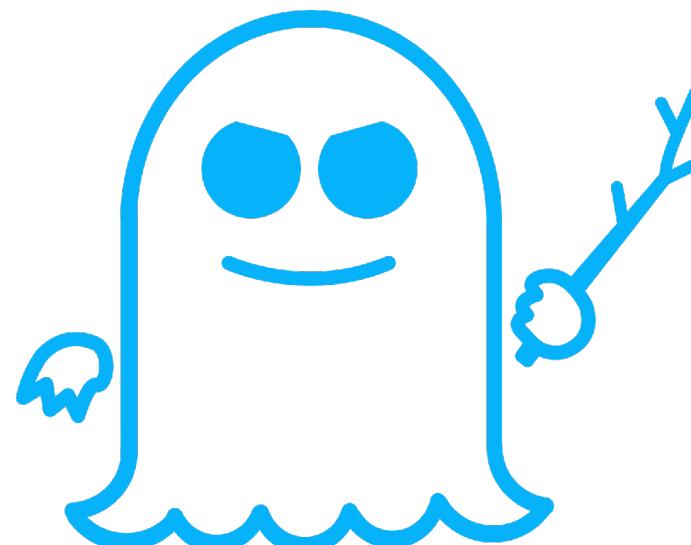
```
rax <- A_size  
rcx <- x  
jmp rcx≥rax, END  
L1: load rax, A + rcx  
     load rax, B + rax
```

END:



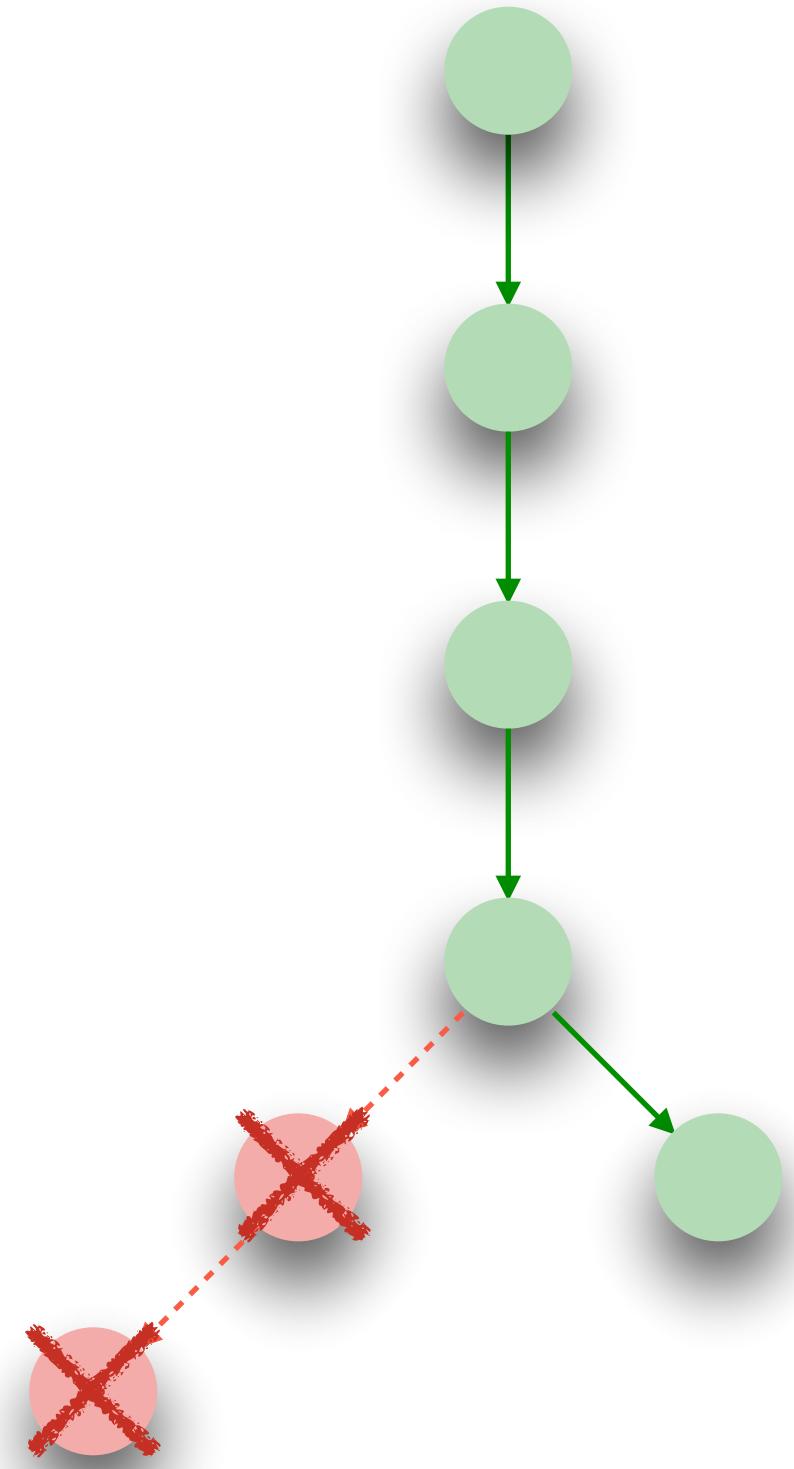
Attacker can observe:

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



Observer model: Leakage into μarchitectural state

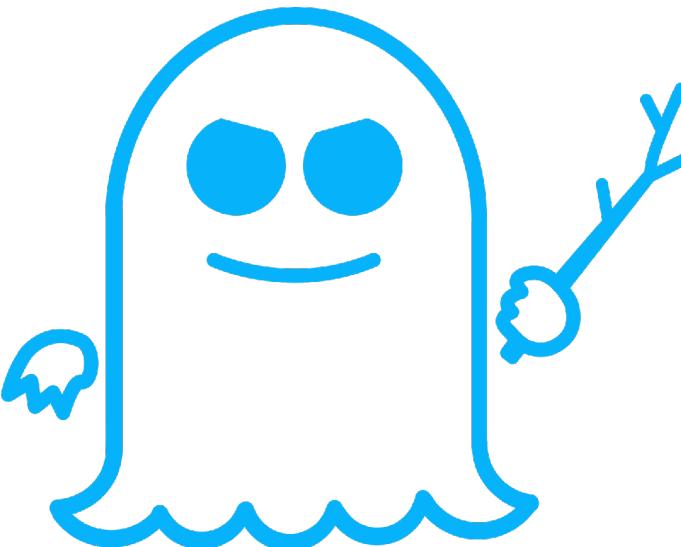
```
rax <- A_size  
rcx <- x  
jmp rcx≥rax, END  
L1: load rax, A + rcx  
     load rax, B + rax  
END:
```



Attacker can observe:

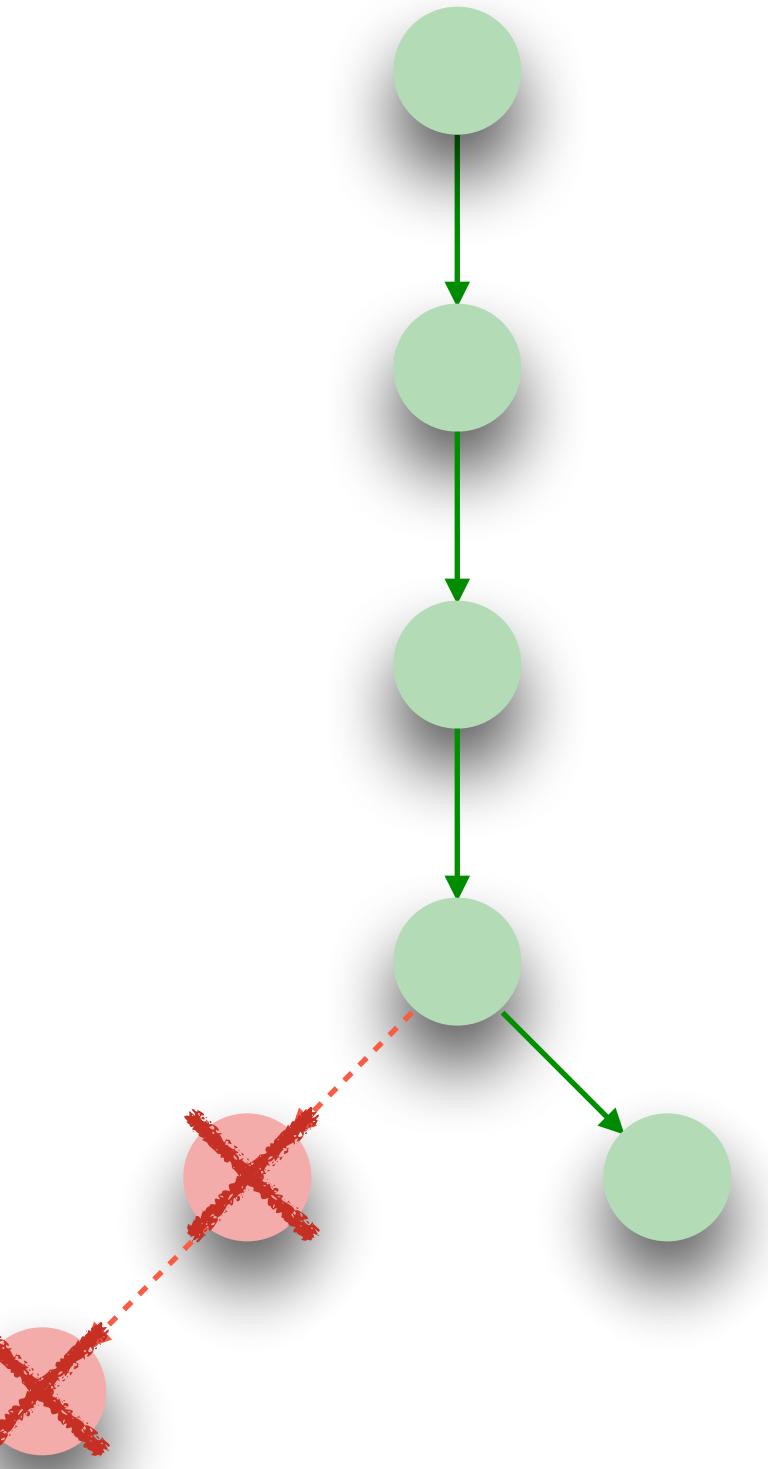
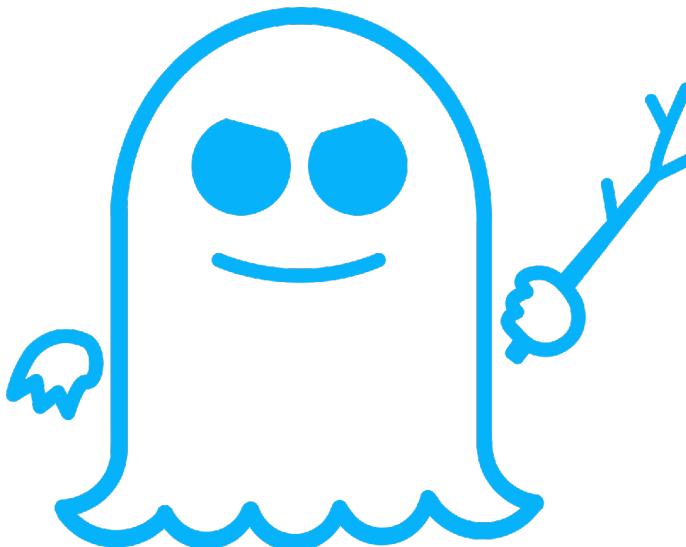
- locations of *memory accesses*
- *branch/jump* targets
- *start/end* speculative execution

Inspired by “constant-time”
programming requirements



Observer model: Leakage into μarchitectural state

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



Attacker can observe:

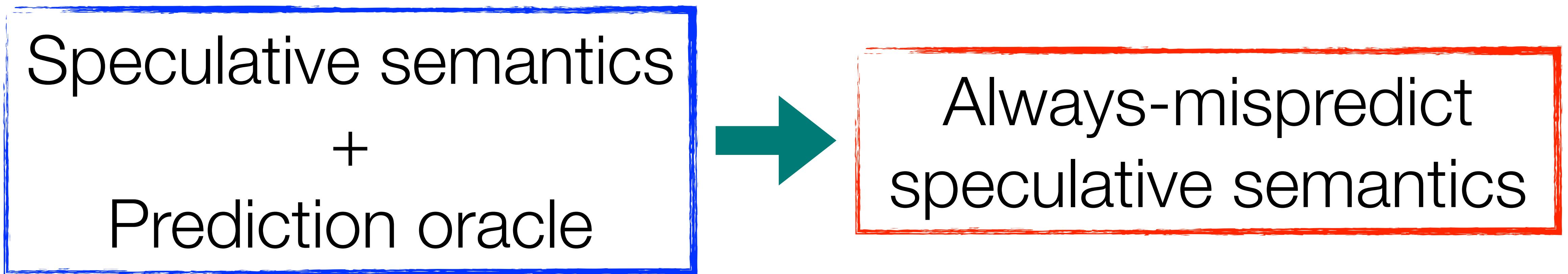
- locations of *memory accesses*
- *branch/jump* targets
- *start/end* speculative execution

Inspired by “constant-time” programming requirements

No need for detailed model of memory hierarchy:

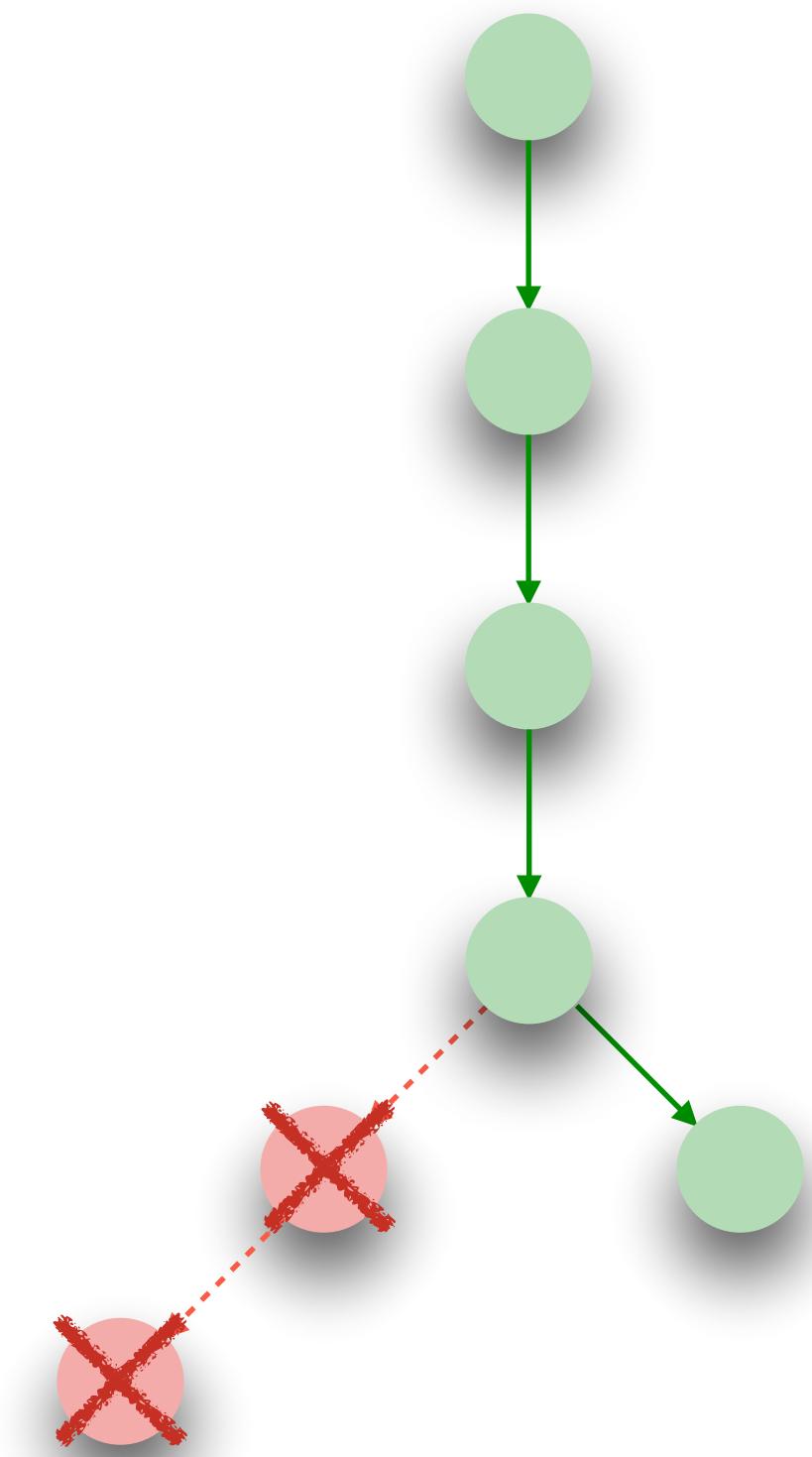
- possibly pessimistic
- more robust

Reasoning about arbitrary prediction oracles



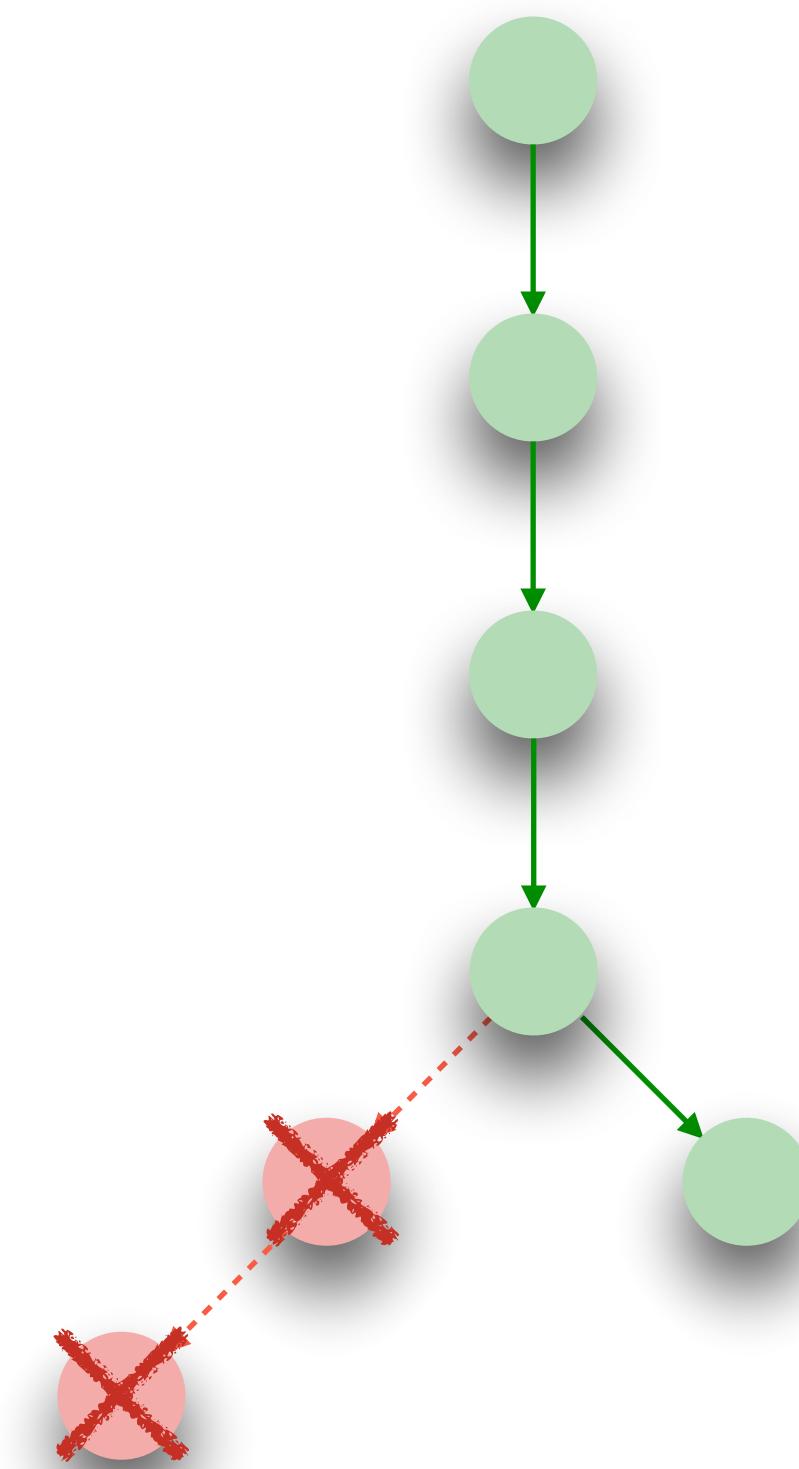
Always-mispredict speculative semantics

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



Always-mispredict speculative semantics

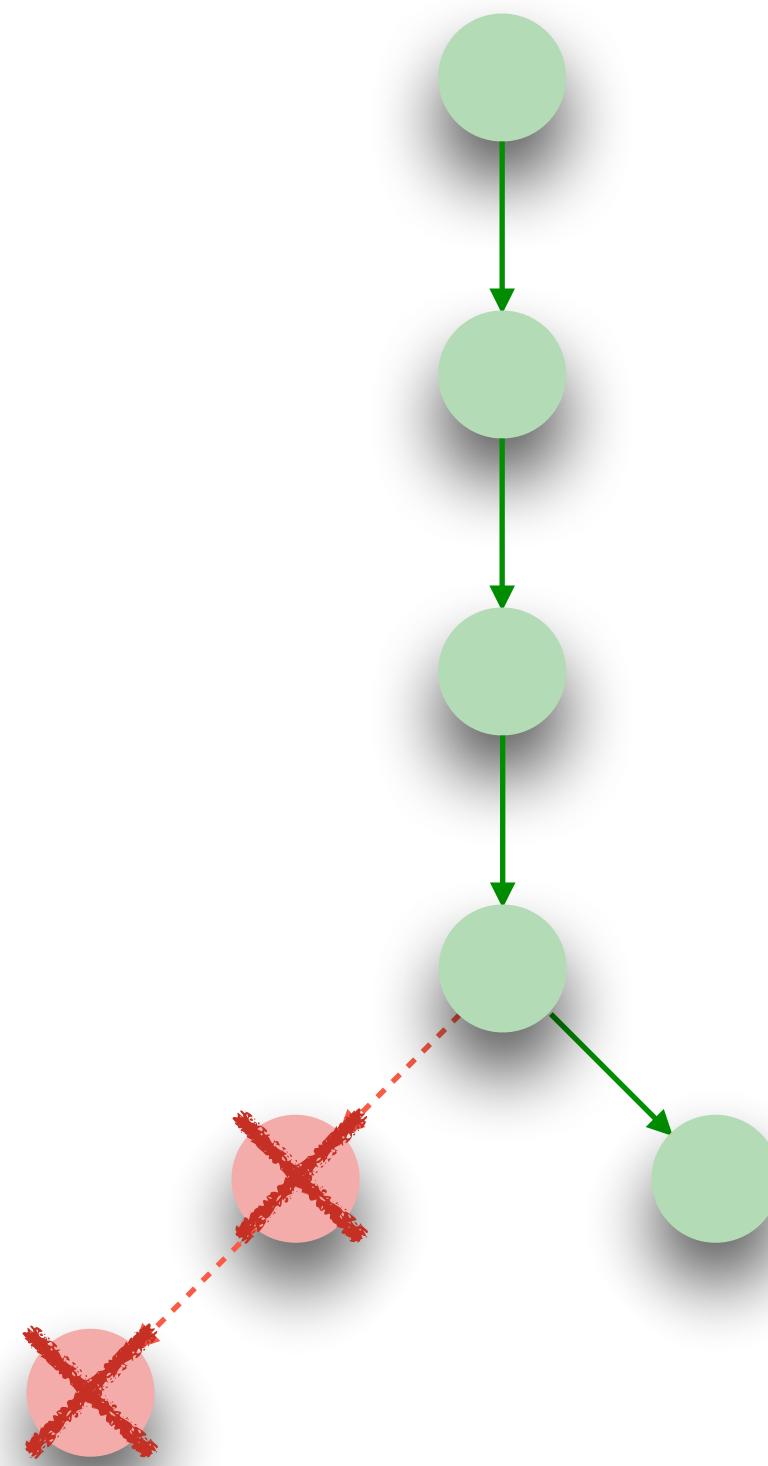
```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Always mispredict branch instructions' outcomes

Always-mispredict speculative semantics

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

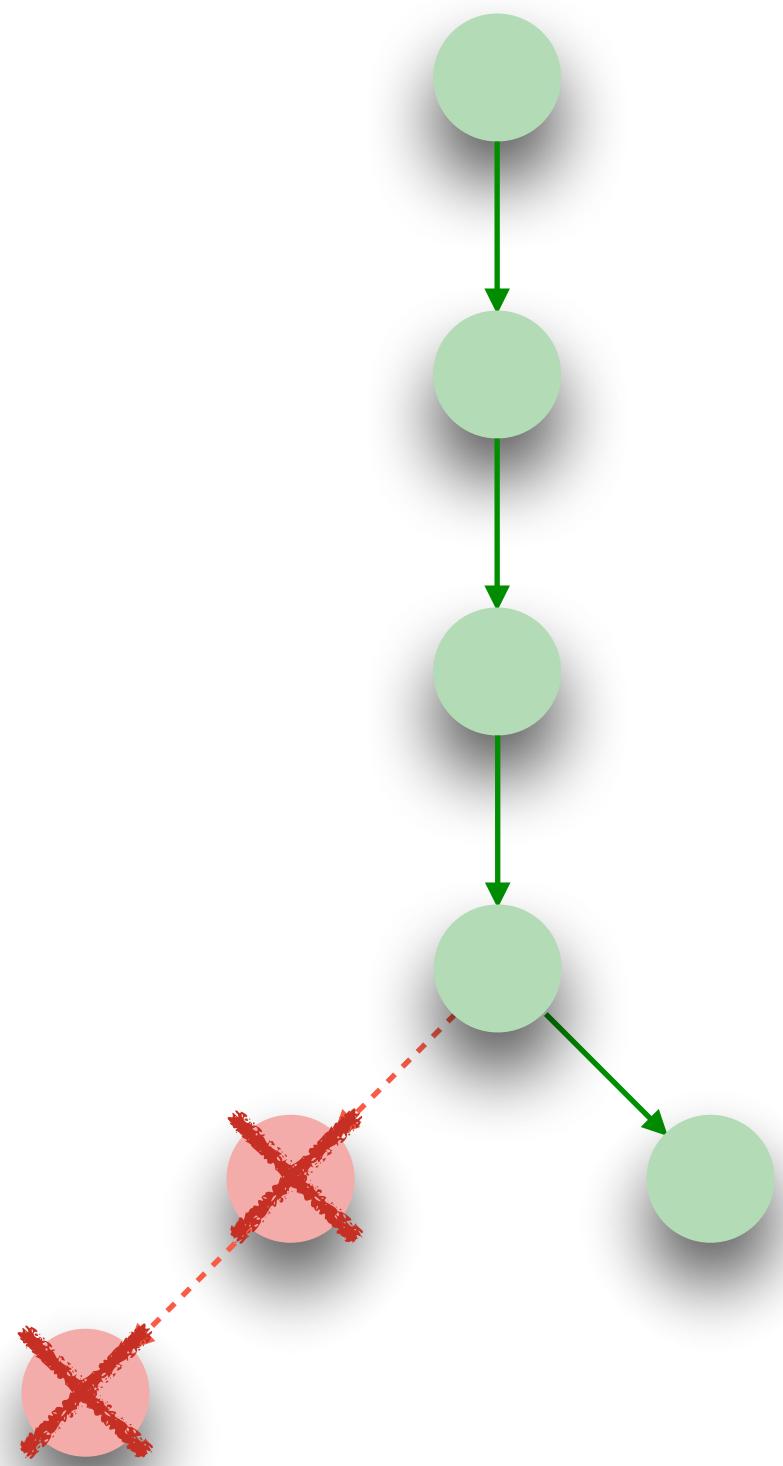


Always mispredict branch instructions' outcomes

Fixed speculative window

Always-mispredict speculative semantics

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



Always mispredict branch instructions' outcomes

Fixed speculative window

Rollback of every transaction

Always-mispredict speculative semantics: Inference Rules

SE-NoBranch

$$p(\sigma(\mathbf{pc})) \neq \mathbf{beqz} \ x, \ell \quad \sigma \xrightarrow{s} \sigma' \quad enabled'(s)$$

$$s' = \begin{cases} decr'(s) & \text{if } p(\sigma(\mathbf{pc})) \neq \mathbf{spbarr} \\ zeroes'(s) & \text{otherwise} \end{cases}$$

$$\langle ctr, \sigma, s \rangle \xrightarrow{s} \langle ctr, \sigma', s' \rangle$$

SE-Rollback

$$\frac{\sigma' \xrightarrow{s} \sigma''}{\langle ctr, \sigma, s \cdot \langle \sigma', id, 0, \ell \rangle \rangle \xrightarrow{s} \langle ctr, \sigma'', s \rangle}$$

$$\xrightarrow{s} \langle ctr + 1, \sigma[\mathbf{pc} \mapsto \ell], s' \rangle$$

SE-Branch-SYMB

$$p(\sigma(\mathbf{pc})) = \mathbf{beqz} \ x, \ell'' \quad enabled'(s)$$

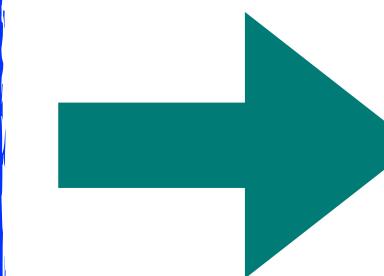
$$\sigma \xrightarrow{s} \sigma' \quad \ell = \begin{cases} \sigma(\mathbf{pc}) + 1 & \text{if } \ell' \neq \sigma(\mathbf{pc}) + 1 \\ \ell'' & \text{if } \ell' = \sigma(\mathbf{pc}) + 1 \end{cases}$$

$$s' = s \cdot \langle \sigma, ctr, min(w, wndw(s) - 1), \ell \rangle \quad id = ctr$$

$$\langle ctr, \sigma, s \rangle \xrightarrow{s} \langle ctr + 1, \sigma[\mathbf{pc} \mapsto \ell], s' \rangle$$

Always-mispredict leaks maximally

Speculative semantics
+
Prediction oracle



Always-mispredict
speculative semantics

For all program states s and s' :

$$P_{\text{spec}}(s) = P_{\text{spec}}(s')$$

$$\Leftrightarrow \forall O: P_{\text{spec},O}(s) = P_{\text{spec},O}(s')$$

Recap: Speculative non-interference

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

For all program states s and s' :

$$\begin{aligned} P_{\text{non-spec}}(s) &= P_{\text{non-spec}}(s') \\ \Rightarrow P_{\text{spec}}(s) &= P_{\text{spec}}(s') \end{aligned}$$

Speculative non-interference: Example

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

Speculative non-interference: Example

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



Speculative non-interference: Example

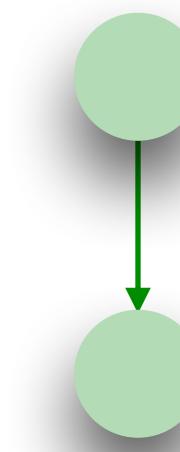
```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
L1: load rax, A + rcx
      load rax, B + rax
END:
```

x=128
A_size=16
A[128]=0

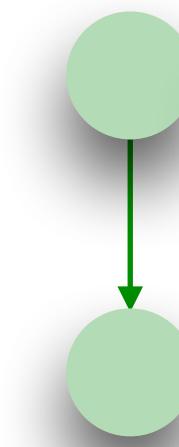
x=128
A_size=16
A[128]=1

Speculative non-interference: Example

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



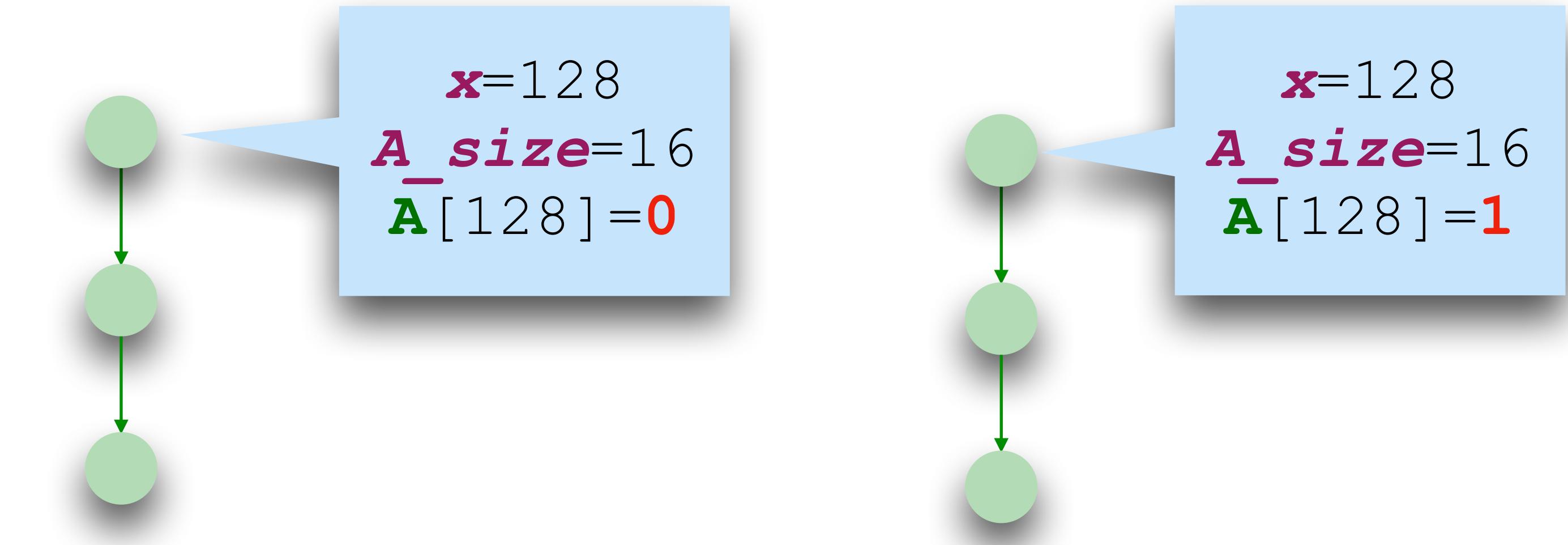
x=128
A_size=16
A[128]=0



x=128
A_size=16
A[128]=1

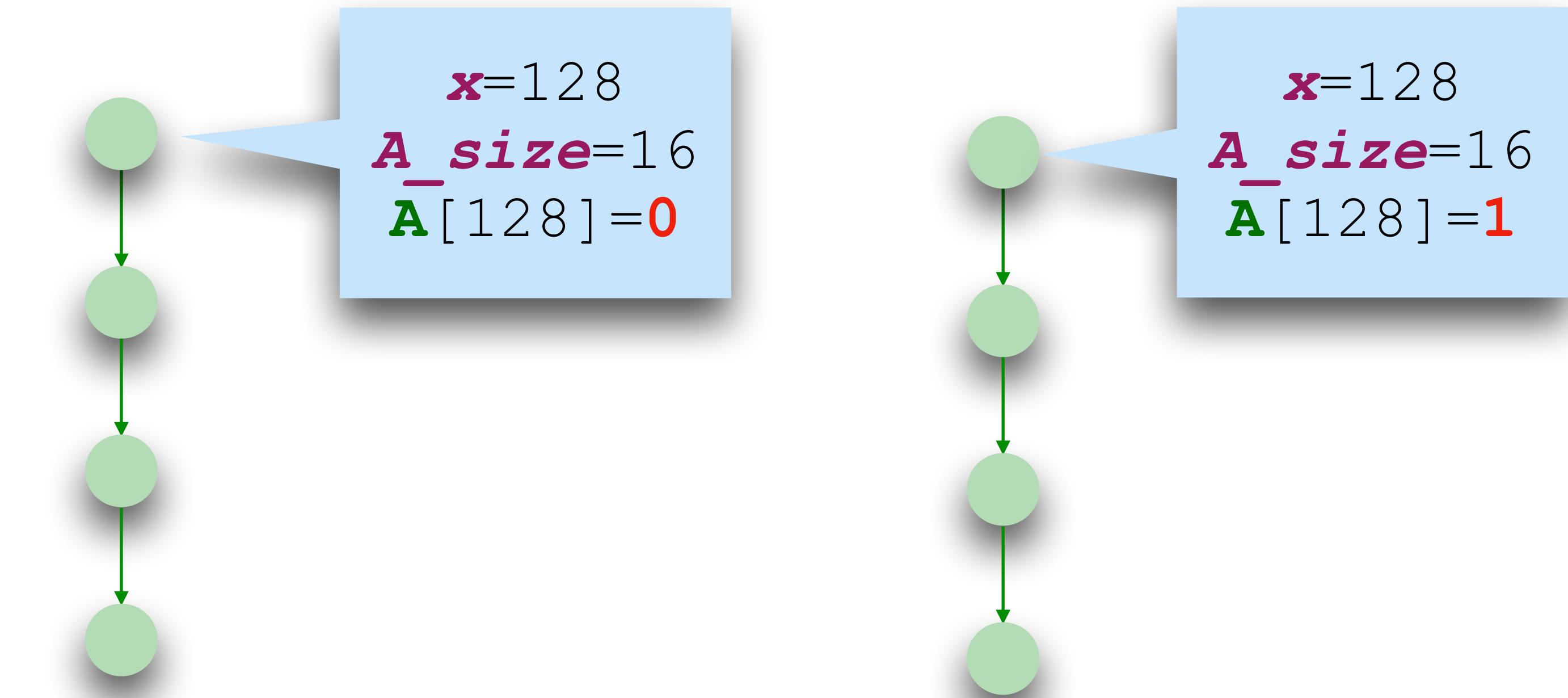
Speculative non-interference: Example

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



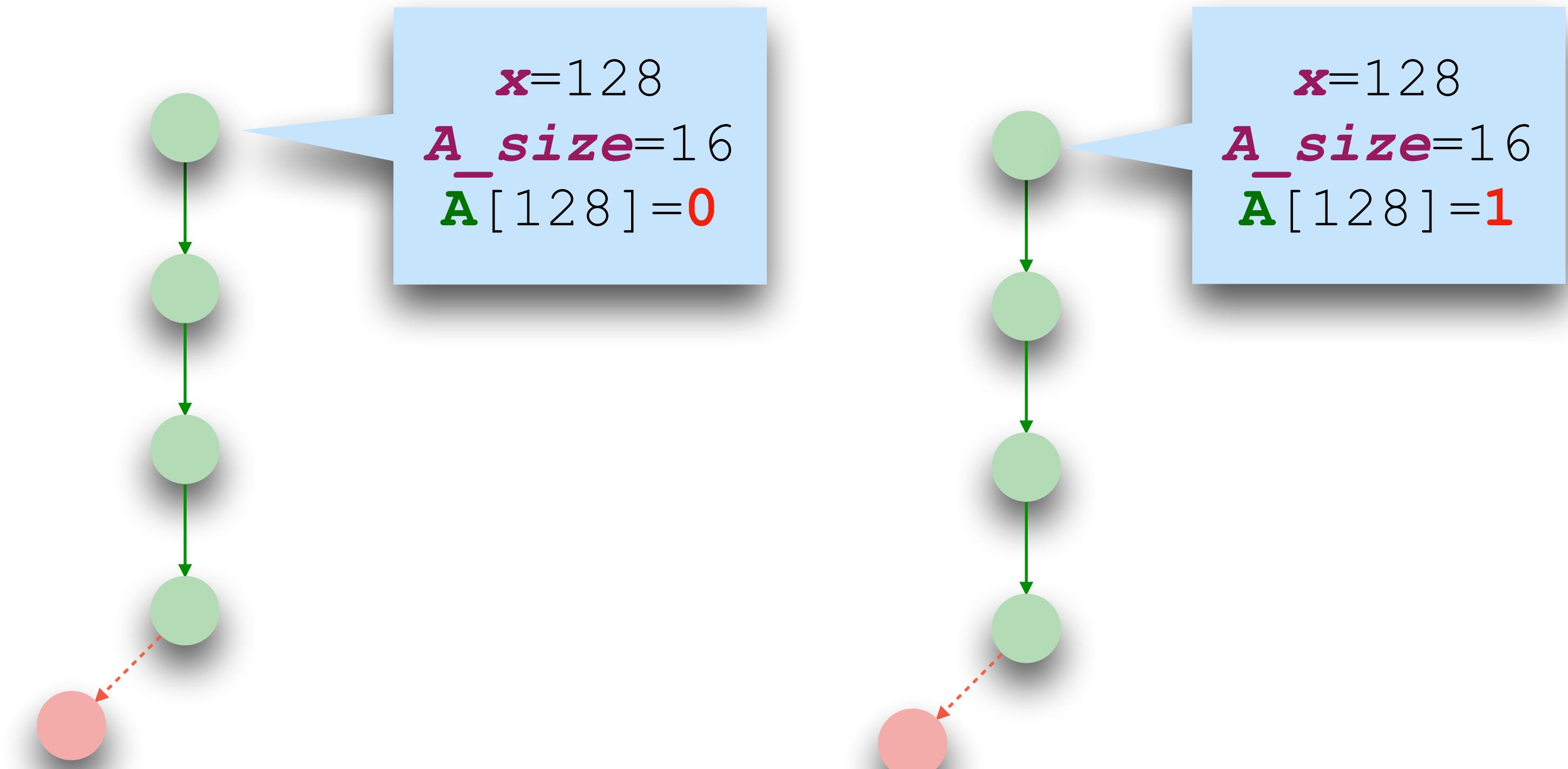
Speculative non-interference: Example

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



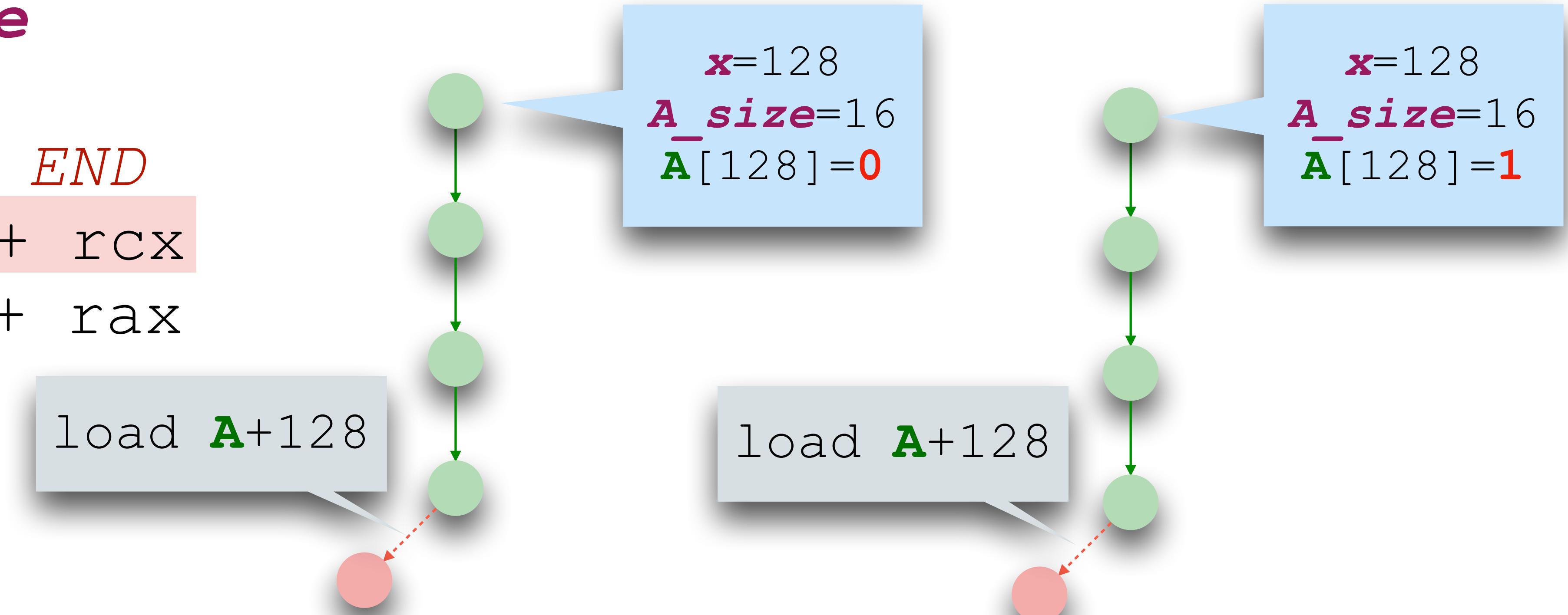
Speculative non-interference: Example

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



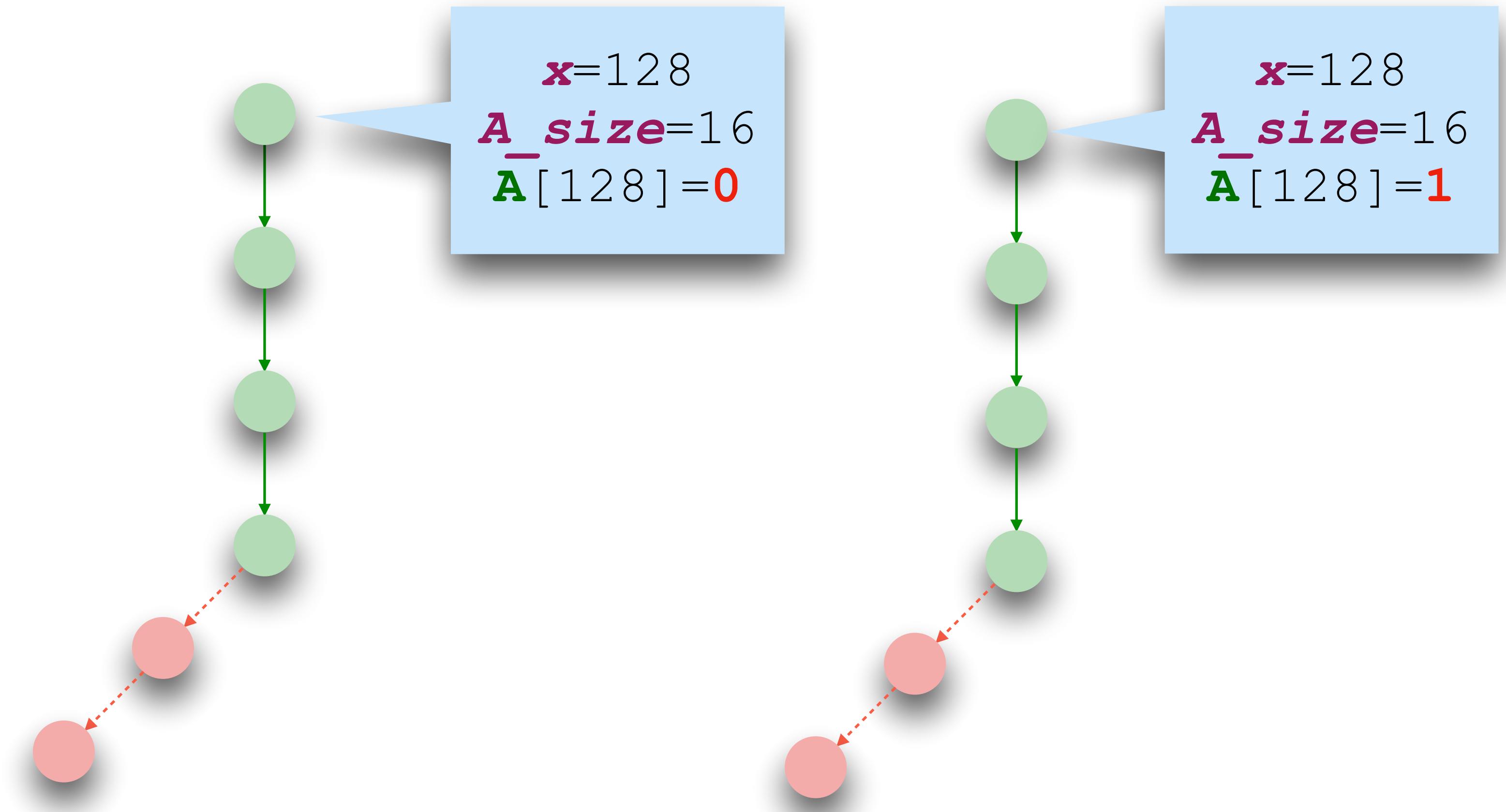
Speculative non-interference: Example

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



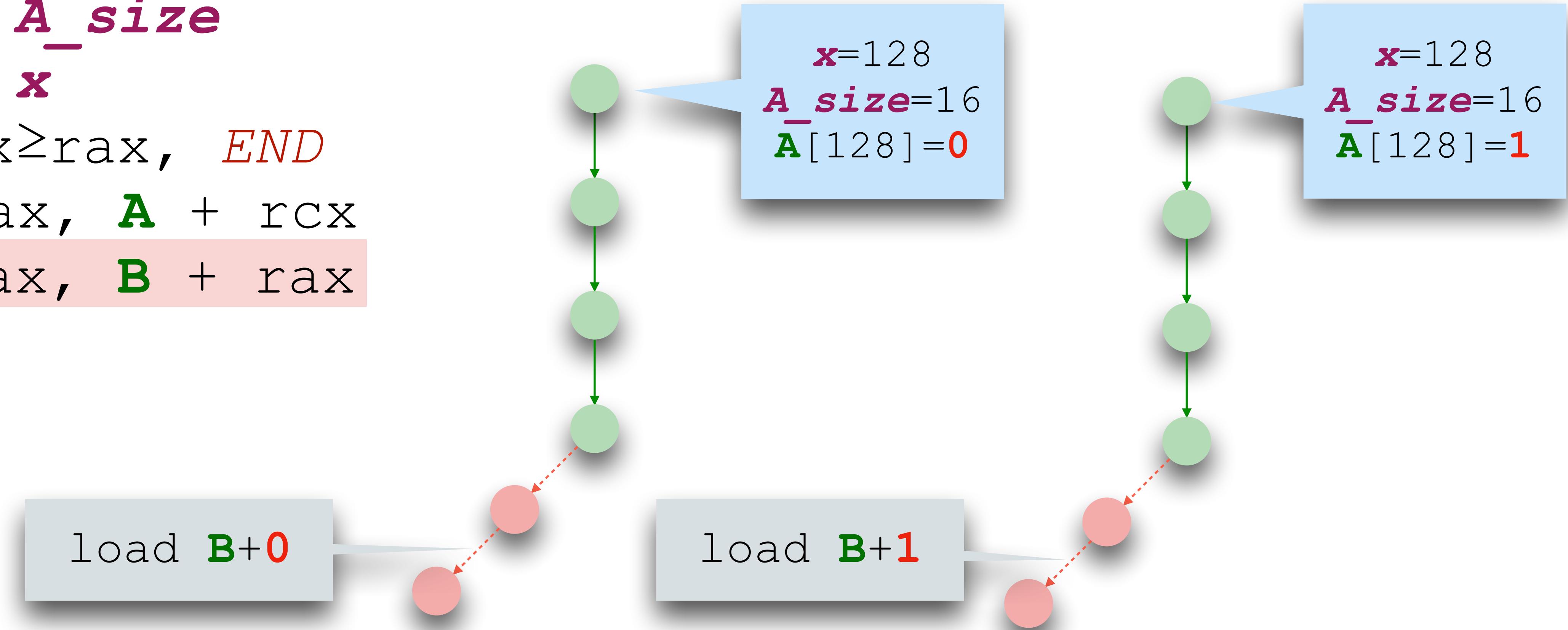
Speculative non-interference: Example

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



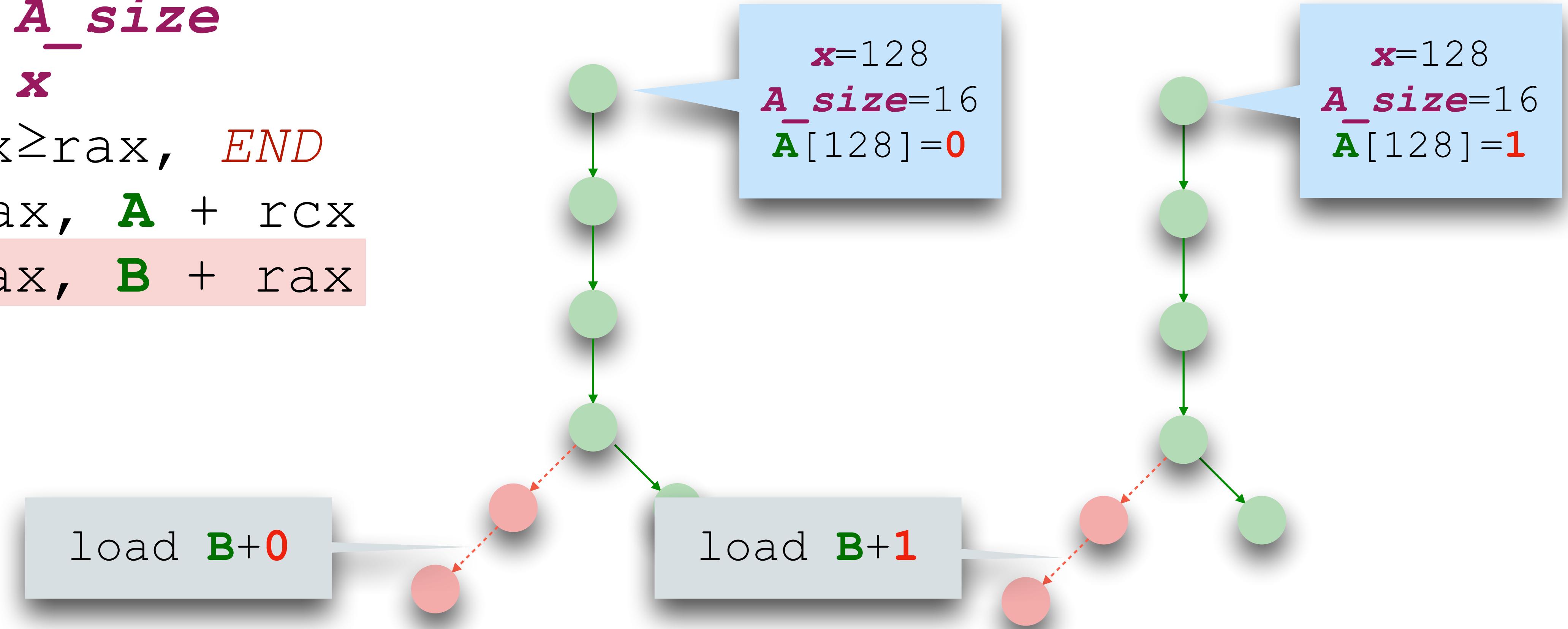
Speculative non-interference: Example

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



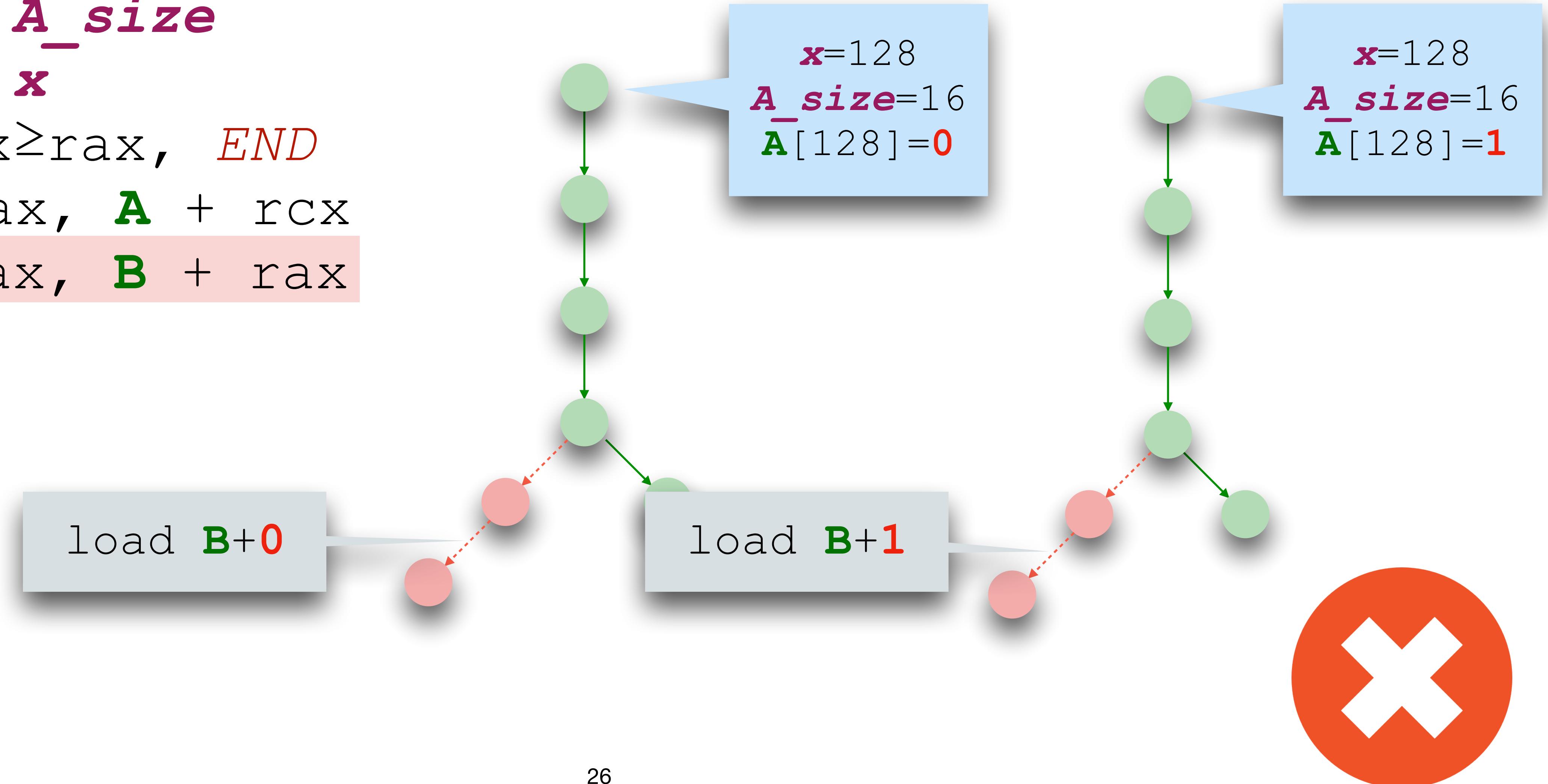
Speculative non-interference: Example

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



Speculative non-interference: Example

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



3. Spectector: Detecting speculative leaks

Spectector: Detecting speculative leaks



Spectector: Detecting speculative leaks



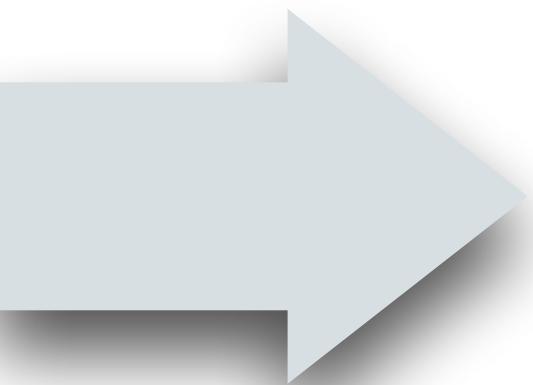
```
rax <- A_size
rcx <- x
jmp rcx>=rax, END
L1:   load rax, A + rcx
        load rax, B + rax
END:
```

Spectector: Detecting speculative leaks



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

Symbolic
execution

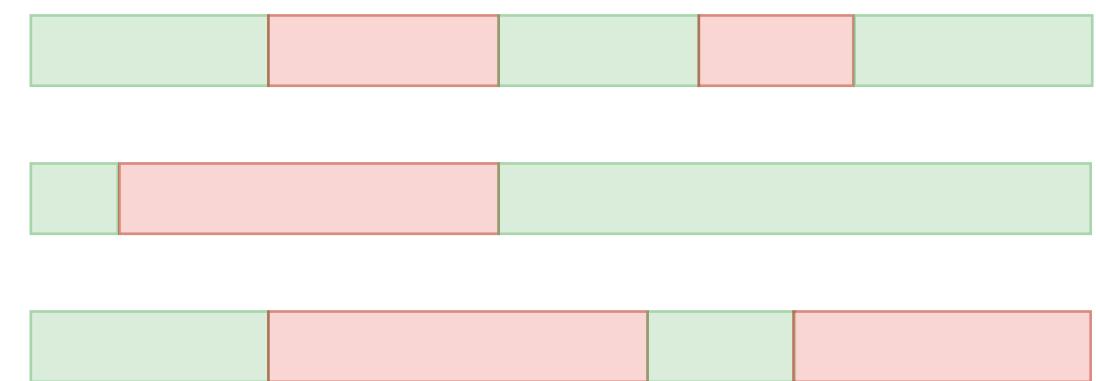


Spectector: Detecting speculative leaks



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

Symbolic
execution

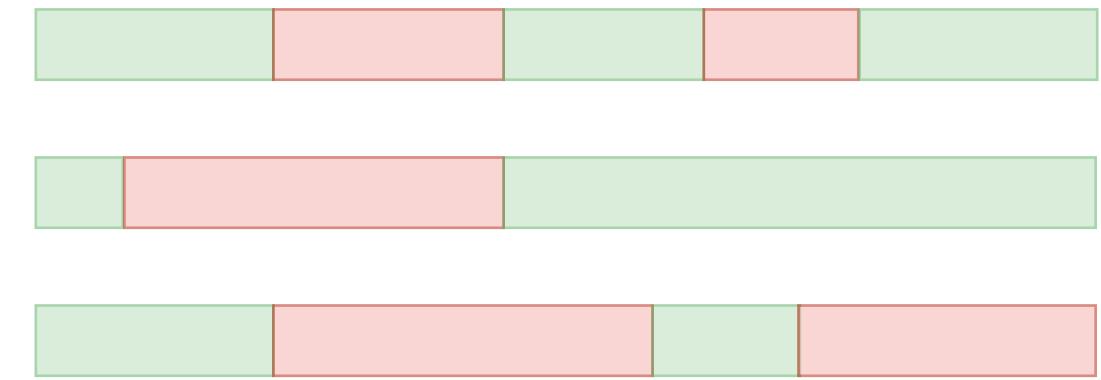
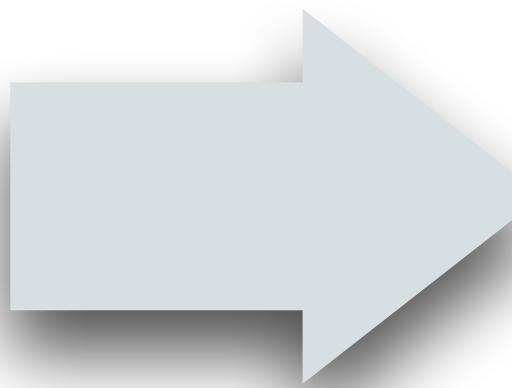


Spectector: Detecting speculative leaks

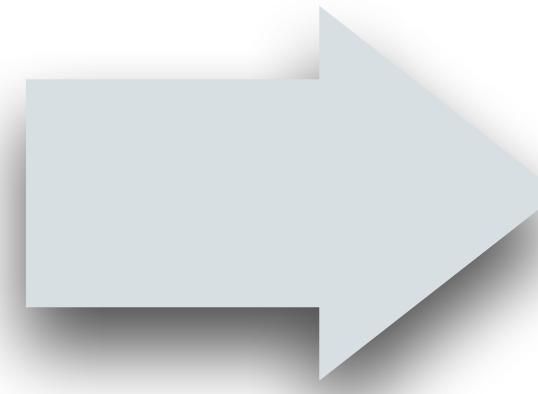


```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
L1: load rax, A + rcx
      load rax, B + rax
END:
```

Symbolic
execution



Detect leaks

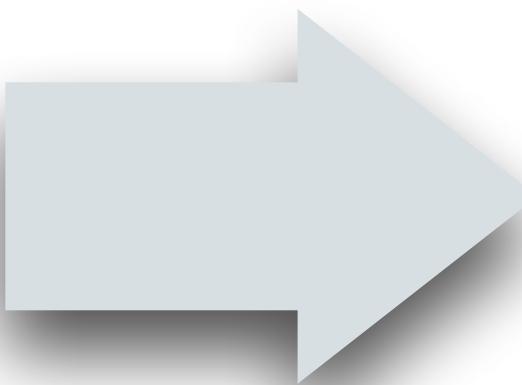


Spectector: Detecting speculative leaks

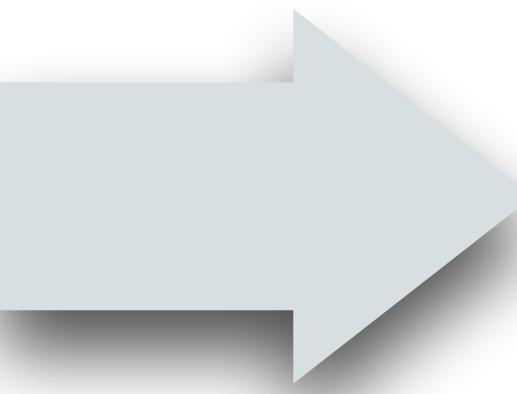


```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
L1: load rax, A + rcx
      load rax, B + rax
END:
```

Symbolic
execution



Detect leaks

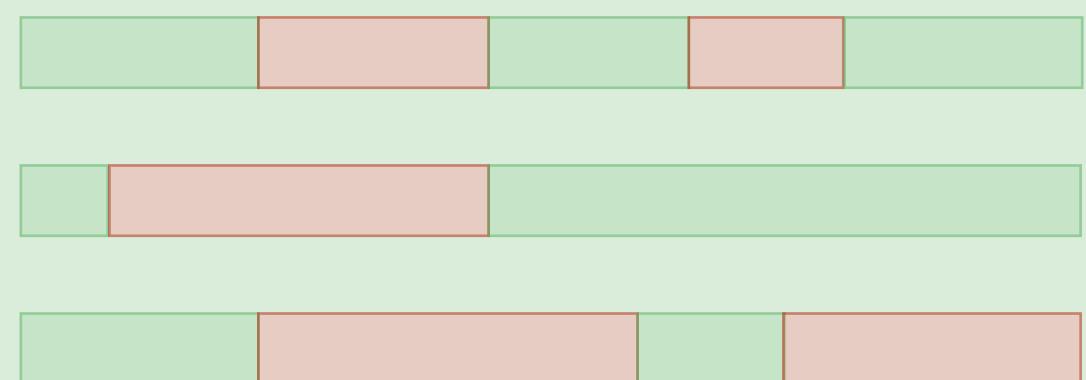


Spectector: Detecting speculative leaks

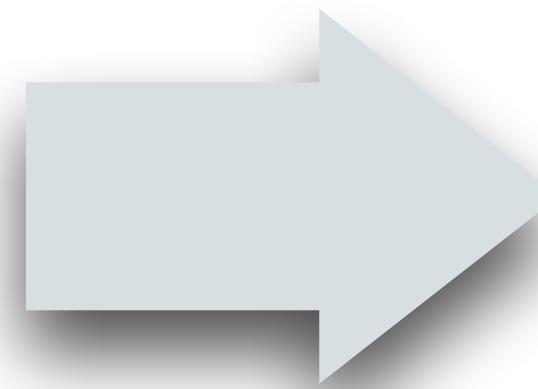


```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
L1: load rax, A + rcx
      load rax, B + rax
END:
```

Symbolic
execution



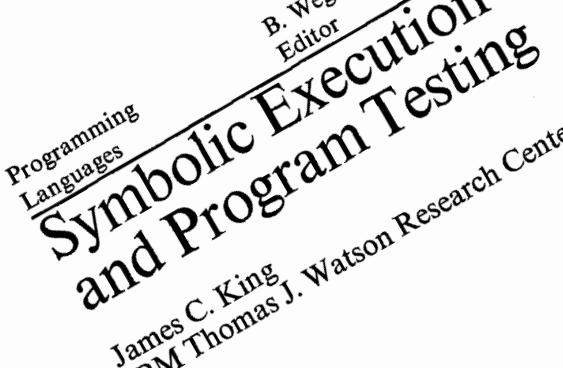
Detect leaks



Symbolic execution

Symbolic execution

- Program analysis technique



Symbolic Execution and Program Testing

Programming Languages

B. Wegbreit
Editor

James C. King
IBM Thomas J. Watson Research Center

Copyright © 1976, Association for Computing Machinery, Inc.
General permission is granted to republish, but not for profit, all or part
of this material is granted provided that ACM's copyright notice
is given and that reference is made to the publication, to its date
of issue, and to the fact that reprinting privileges were granted
by permission of the Association for Computing Machinery.
Author's address: IBM Thomas J. Watson Research Center,
P.O. Box 218, Yorktown Heights, N.Y. 10598.

Ex

1. Introduction

The large-scale production of reliable programs is one of the fundamental requirements for applying computers to today's challenging problems. Several techniques are used in practice; others are the focus of current research. The work reported in this paper is directed at assuring that a program meets its requirements even when formal specifications are not given. The current technology in this area is basically a testing technology. That is, some small sample of the data that a program is expected to handle is presented to the program. If the program is judged to produce correct results for the sample, it is assumed to be correct. Much current work [11] focuses on the question of how to choose this sample.

Recent work on proving the correctness of programs via model analysis [5] shows great promise and appears to be a valuable technique for producing reliable programs for routine use. Fundamental theory to practice are not yet well developed, however. Model checking can be used to verify programs, a problem that has been studied for many years.

Section
Testing
Research Center

When formal specification techniques are used in the development process, the question of how to choose a sample, it is assumed to be correct. Much current research [11] focuses on the question of how to choose a sample.

Recent work on proving the correctness of programs by formal analysis [5] shows great promise and appears to be the ultimate technique for producing reliable programs. However, the practical accomplishment of this area fall short of a tool for routine use. Fundamental problems in reducing the theory to practice are not likely to be solved in the immediate future. While testing can be used as an alternative to program proving, the results of program testing and program proving are not necessarily comparable. The sample is still in doubt, and the program may be modified during the testing process.

Recent work on program testing [1] focuses on sample, pre-sample, and post-sample, showing that formal analysis [5] shows to be the ultimate technique for programs. However, the practical problems in reducing the theory to practice likely to be solved in the immediate future. Program testing and program proving can be considered as extreme alternatives. While testing, a programmer can be assured that sample test runs, a correct by carefully checking the results. The programmatic execution for inputs not in the sample is still in doubt, alternately, in program proving the program actually proves that the program meets its specification at all. To do this he gives a formal proof procedure to show that the specification is consistent. This method hinges on the care and construction of the proof steps, as well as the extremes. This paper describes a technique for program proving. From

as over the Internet. Additional issues arise due to the fact that the standard EFFIGY which provides symbolic program testing and debugging is also an interpreted programming language. It includes no standard debugging features, the ability to manage an environment, or a program verifier. A brief discussion of the relationship between symbolic execution and program proving is also included.

Key Words and Phrases: symbolic execution, program testing, program debugging, program proving, program verification, symbolic interpretation

CR Categories: 4.13, 5.21, 5.24

both construction and machine-destruction etc. This paper describes these two extremes. Frequency-hanced testing techniques on a set of sample programs are executed for a set of normal symbolic execution results. The number of normal results against the number of either formal or informal results. The class of programs for which the program's behavior is acceptable is defined by the user.

Experiments with a symbolic evaluation system

of classes of inputs, a result may be equivalent to test cases. These results can be programmed's expectations for correctly or informally. A class of inputs characterized by each symbolic function is determined by the dependence of the program flow on its inputs. If the control flow of a program is completely independent of the input variables, a single symbolic execution will suffice to check all possible executions of the program. If the control program is dependent on the inputs, one must do a case analysis. Often the set of input

as to
1)

Communications
of
the ACM

July 1976
Volume 19
Number 7

2

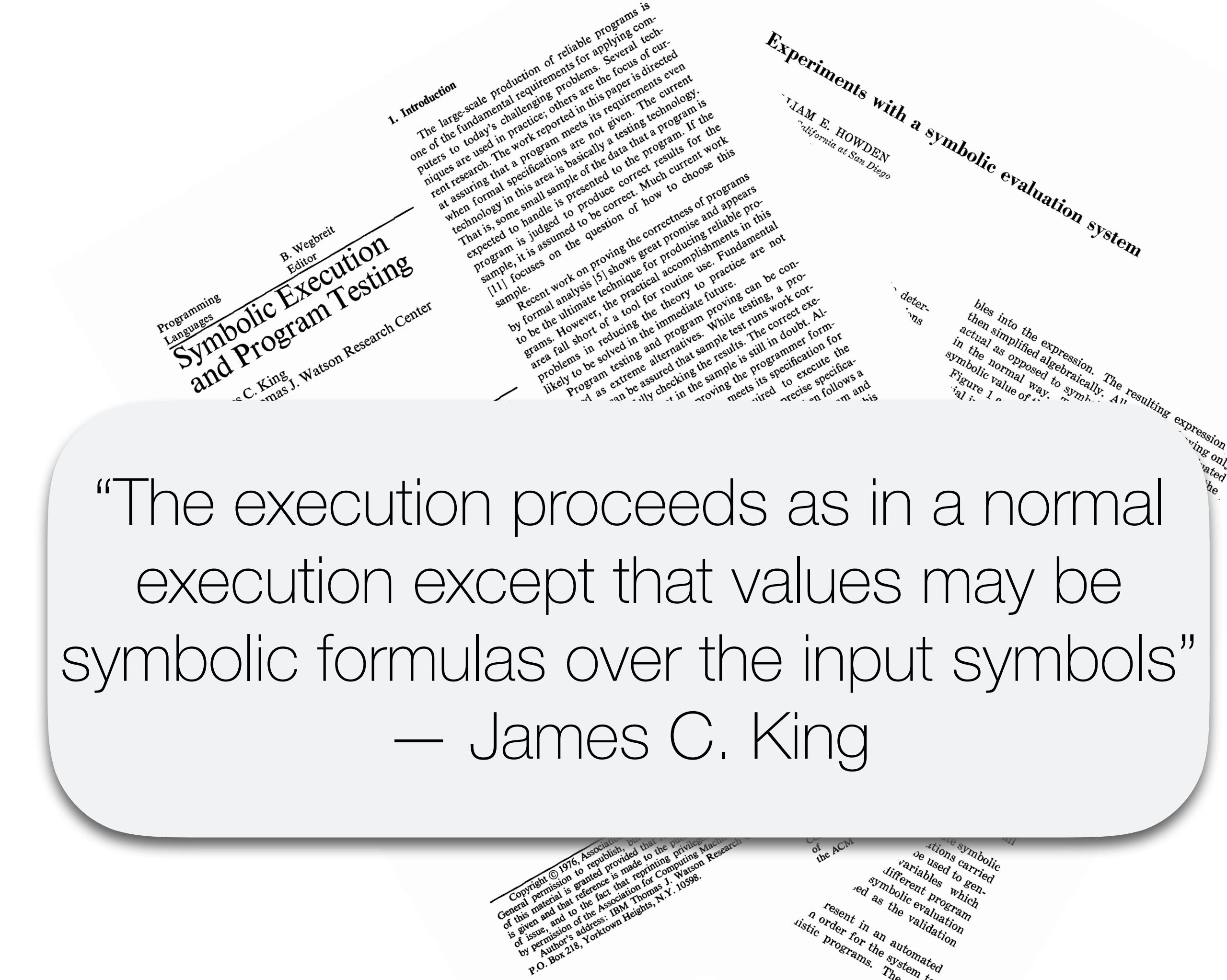
ng
ease
ection
ormula
t for all

ate symbolic
ations carried
be used to gen-
variables which
different program
symbolic evaluation
ed as the validation

resent in an automated
n order for the sys-
stistic programs

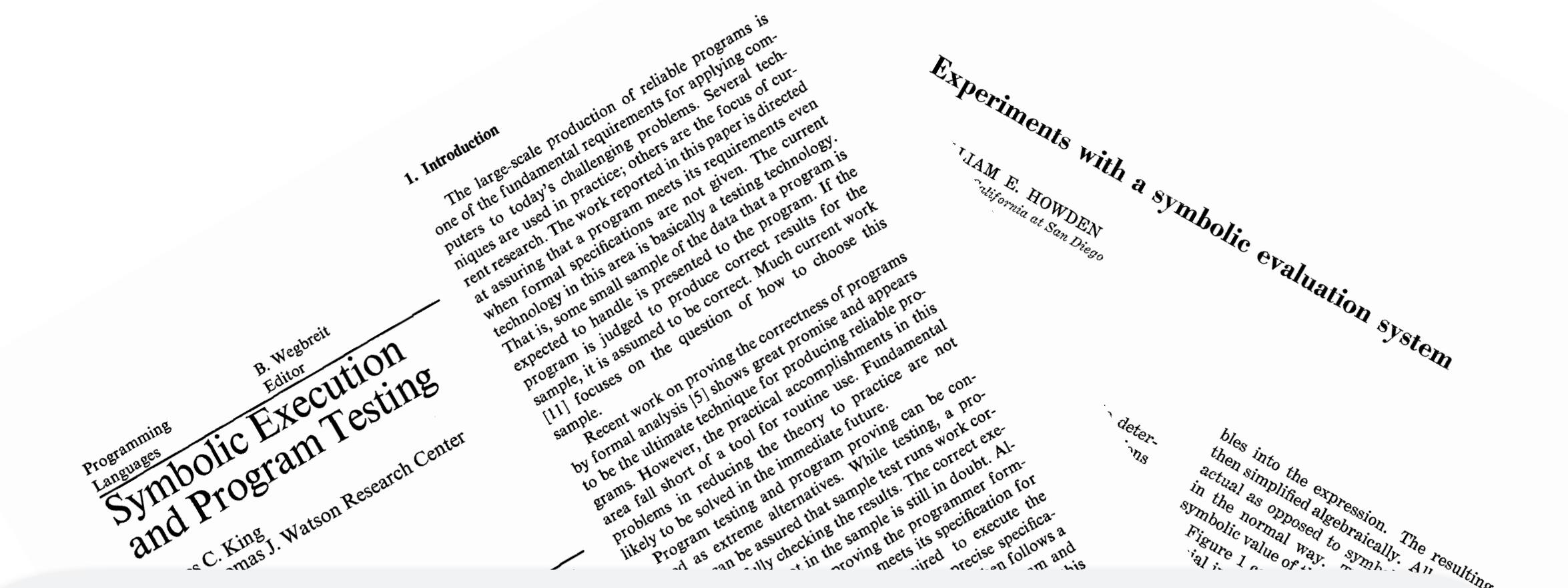
Symbolic execution

- Program analysis technique



Symbolic execution

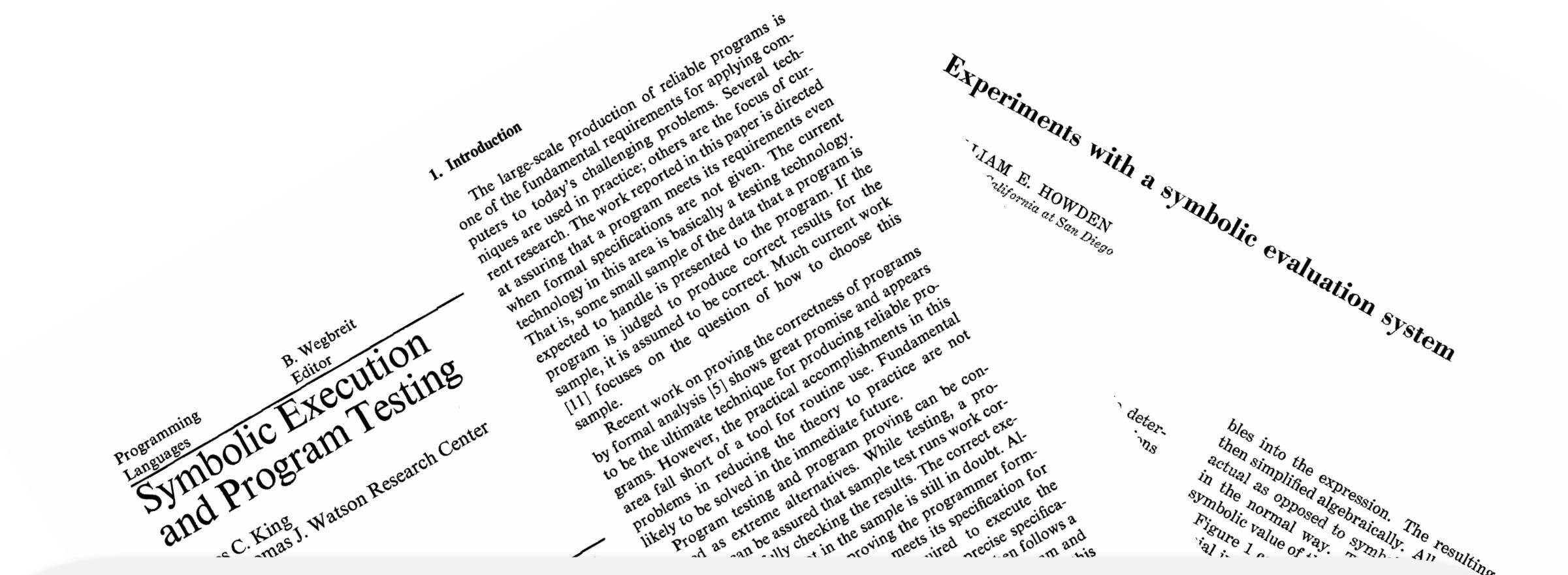
- Program analysis technique
- Execute programs over symbolic values



“The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols”
— James C. King

Symbolic execution

- Program analysis technique
- Execute programs over symbolic values
 - Explore all paths, each with its own path constraint



“The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols”
— James C. King

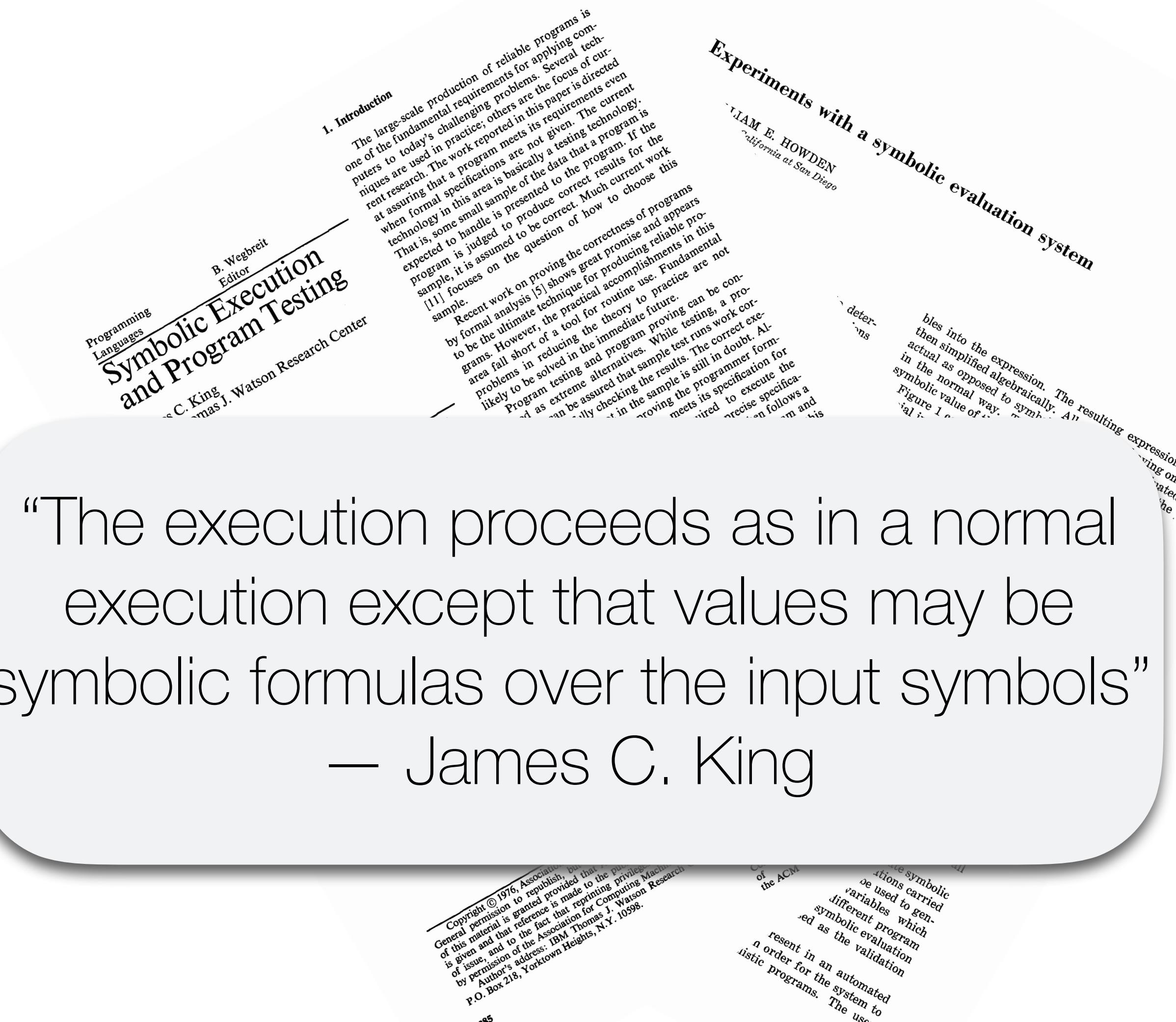
Copyright © 1976 Association
of Computing Machinery, Inc.
General permission to republish
is given and to the extent that
the original source is given,
by permission of the author or
by permission of the Association for
Computing Machinery.
P.O. Box 218, Yorktown Heights, N.Y. 10598.

385

Symbolic execution

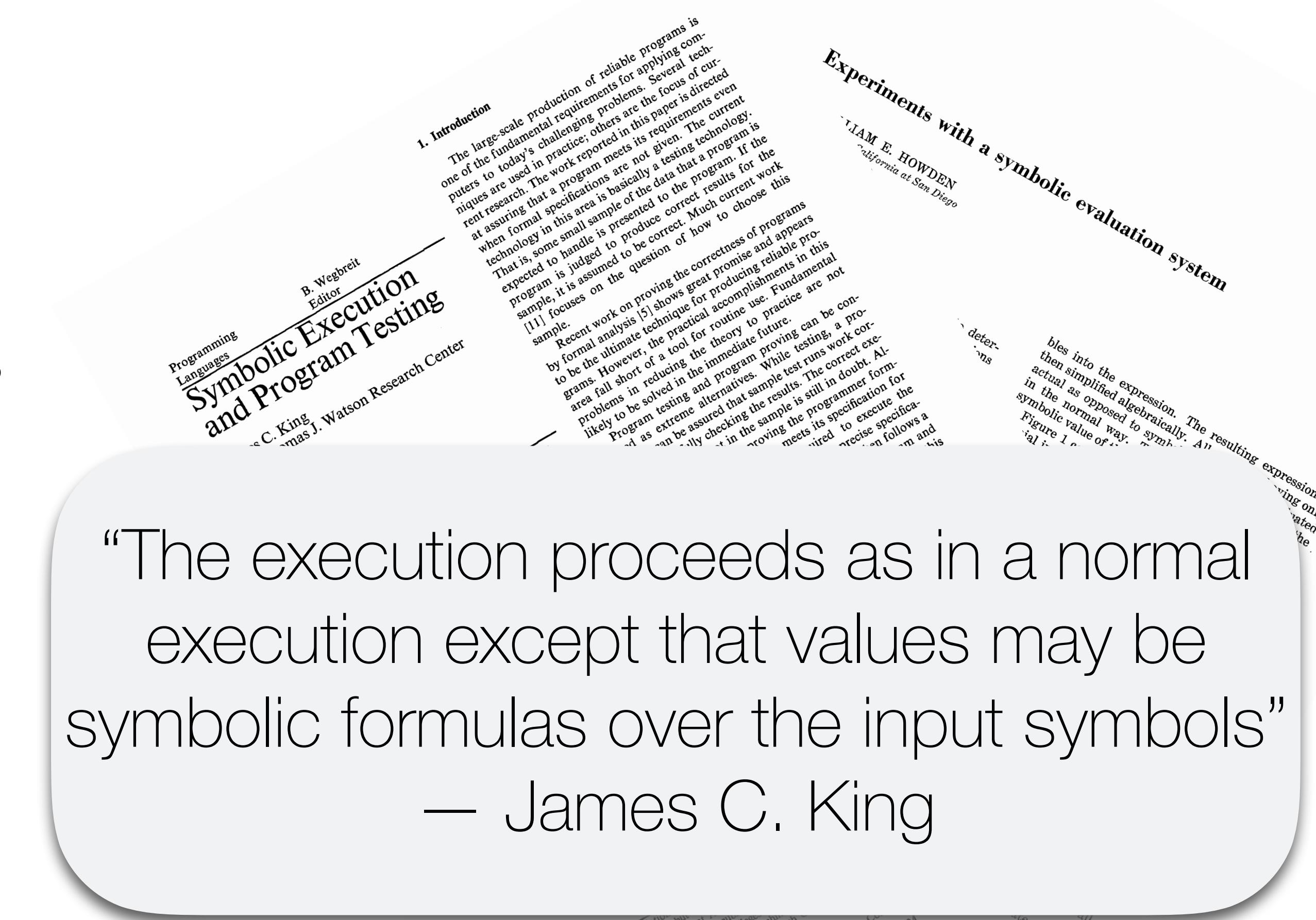
- Program analysis technique
 - Execute programs over symbolic values
 - Explore all paths,
each with its own path constraint
 - Each path represents all possible
executions satisfying the constraints

“The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols”



Symbolic execution

- Program analysis technique
- Execute programs over symbolic values
 - Explore all paths, each with its own path constraint
 - Each path represents all possible executions satisfying the constraints
 - Branch and jump instructions: fork paths and update path constraint



Copyright © 1976 Association
of Computing Machinery, Inc.
General permission to republish
is given and to the extent that
the material is granted provided
that credit is given and to the
fact that reprinting is
by permission of the Association for Computing
Machinery. P.O. Box 218, Yorktown Heights, N.Y. 10598.

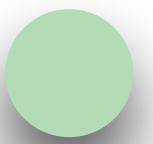
385

Symbolic execution

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

Symbolic execution

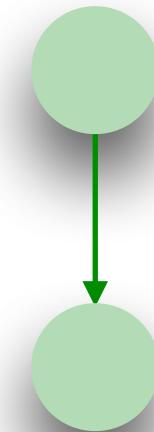
```
rax <- A_size           true  
rcx <- x  
jmp rcx≥rax, END  
L1: load rax, A + rcx  
      load rax, B + rax  
END:
```



Symbolic execution

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

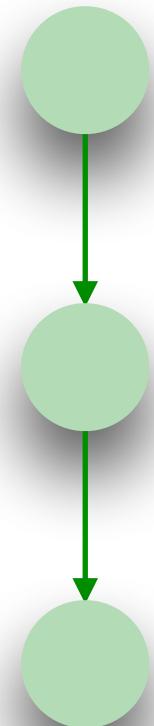
true



Symbolic execution

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

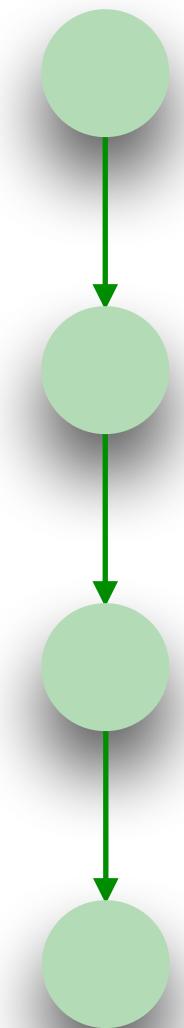
true



Symbolic execution

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

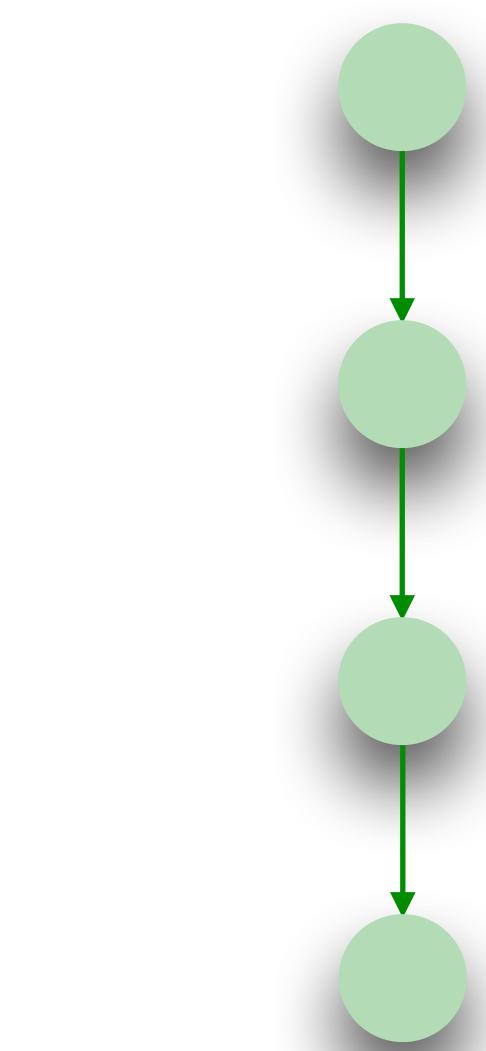
true



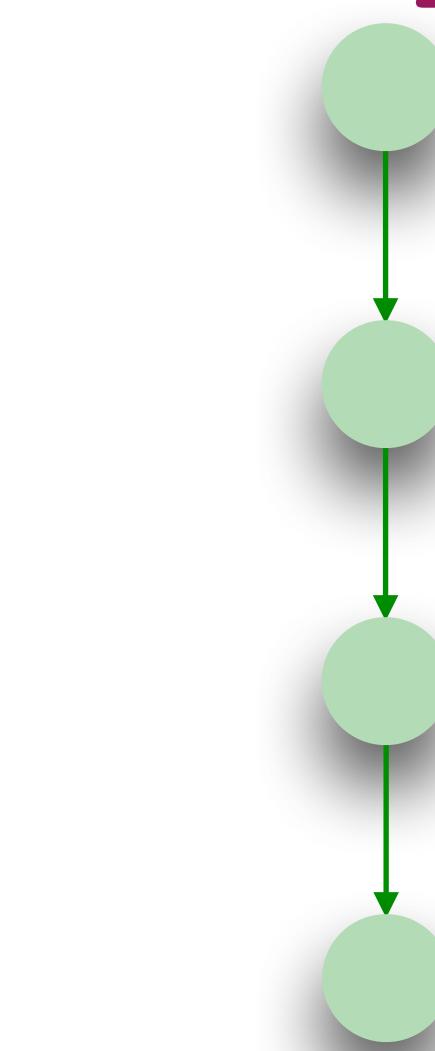
Symbolic execution

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

$x \geq A_size$

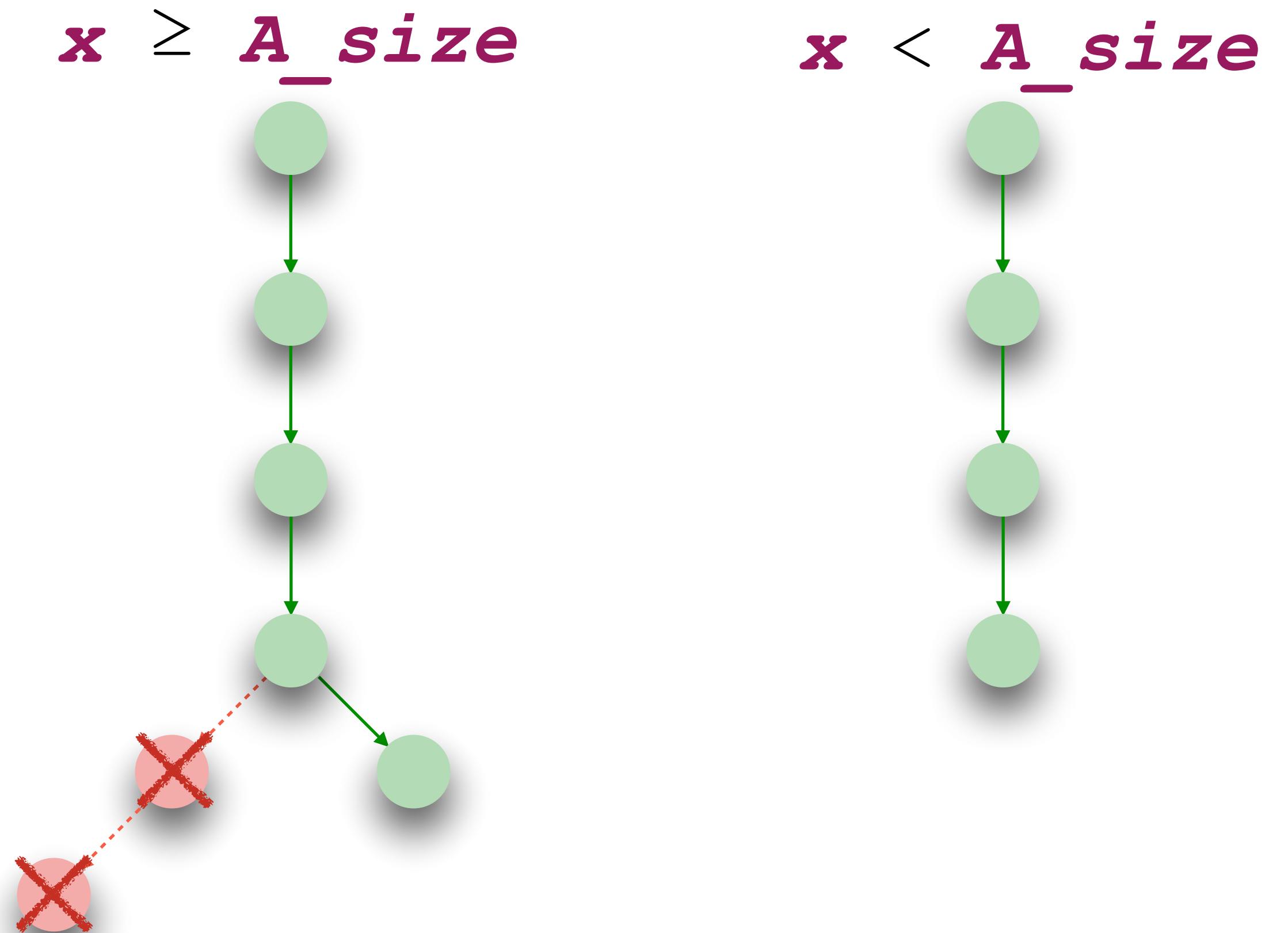


$x < A_size$



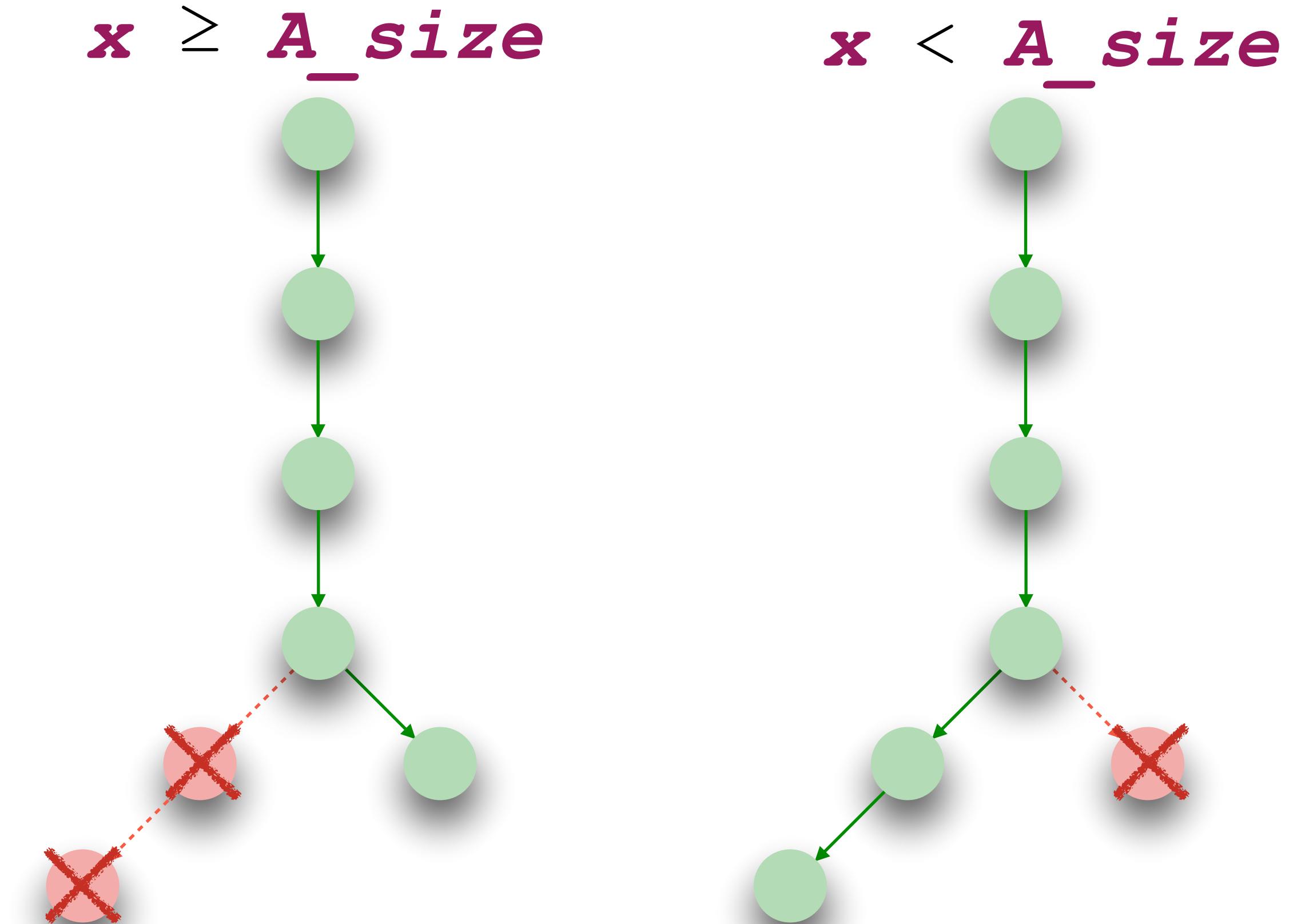
Symbolic execution

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



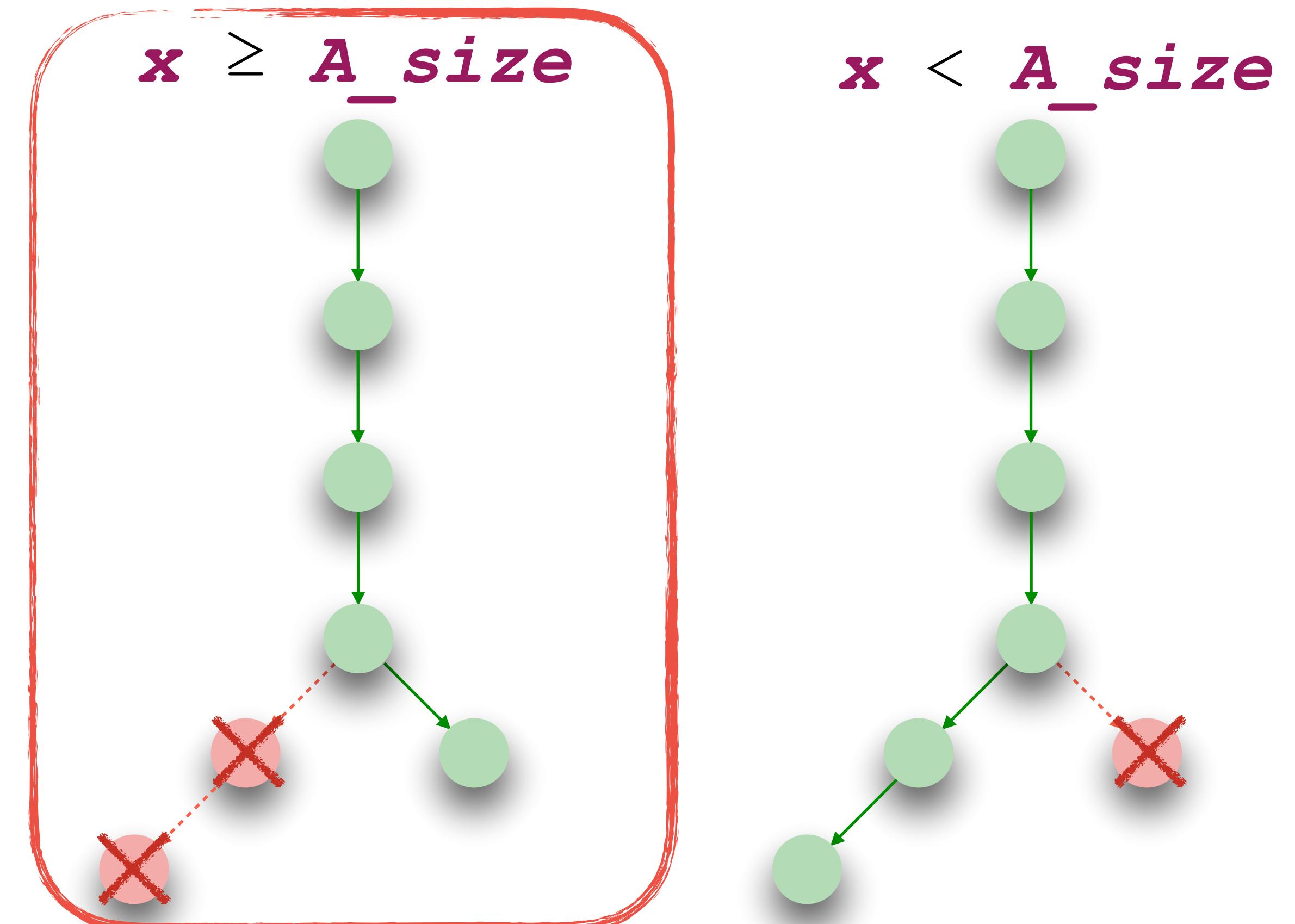
Symbolic execution

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



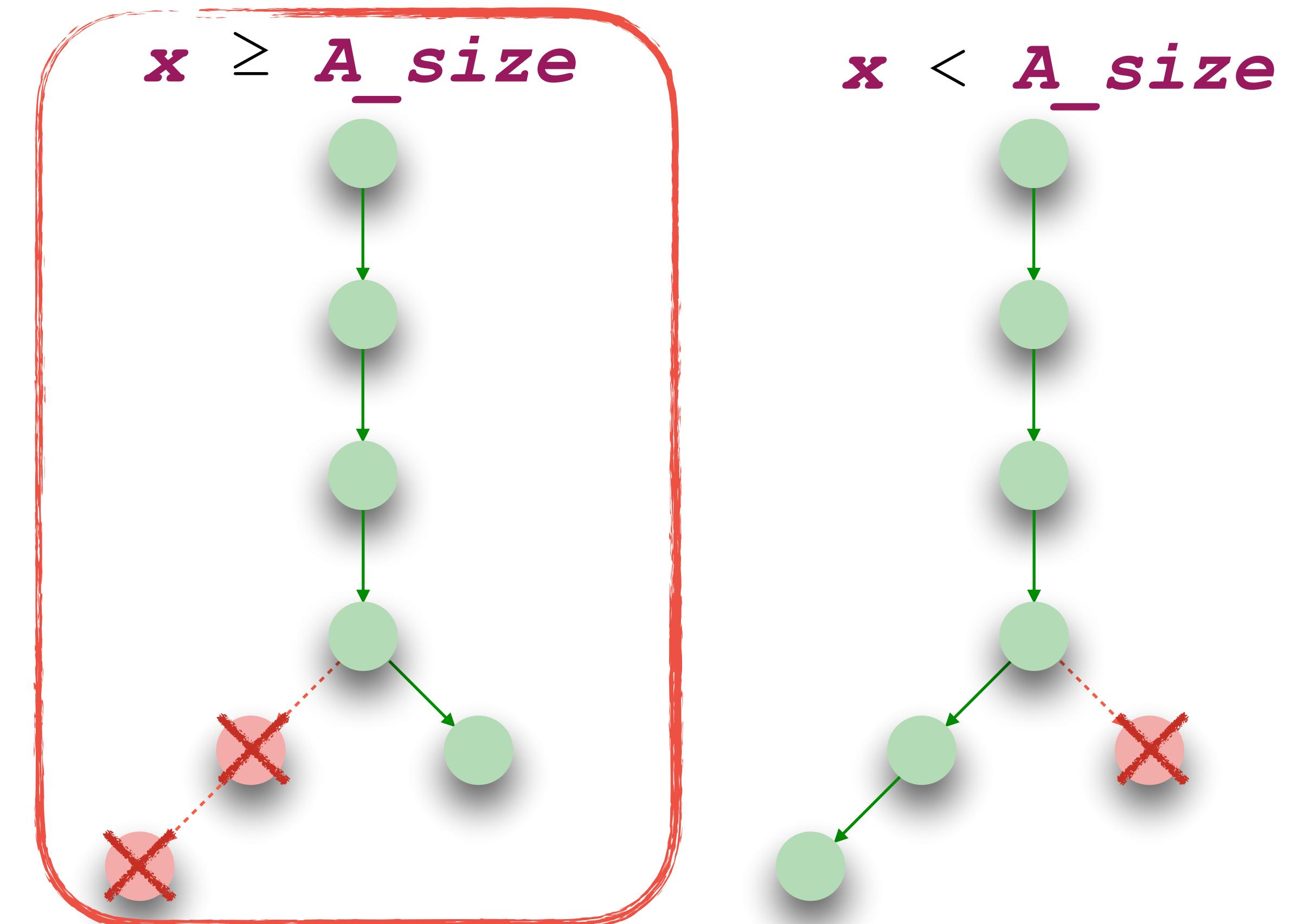
Symbolic execution

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



Symbolic execution

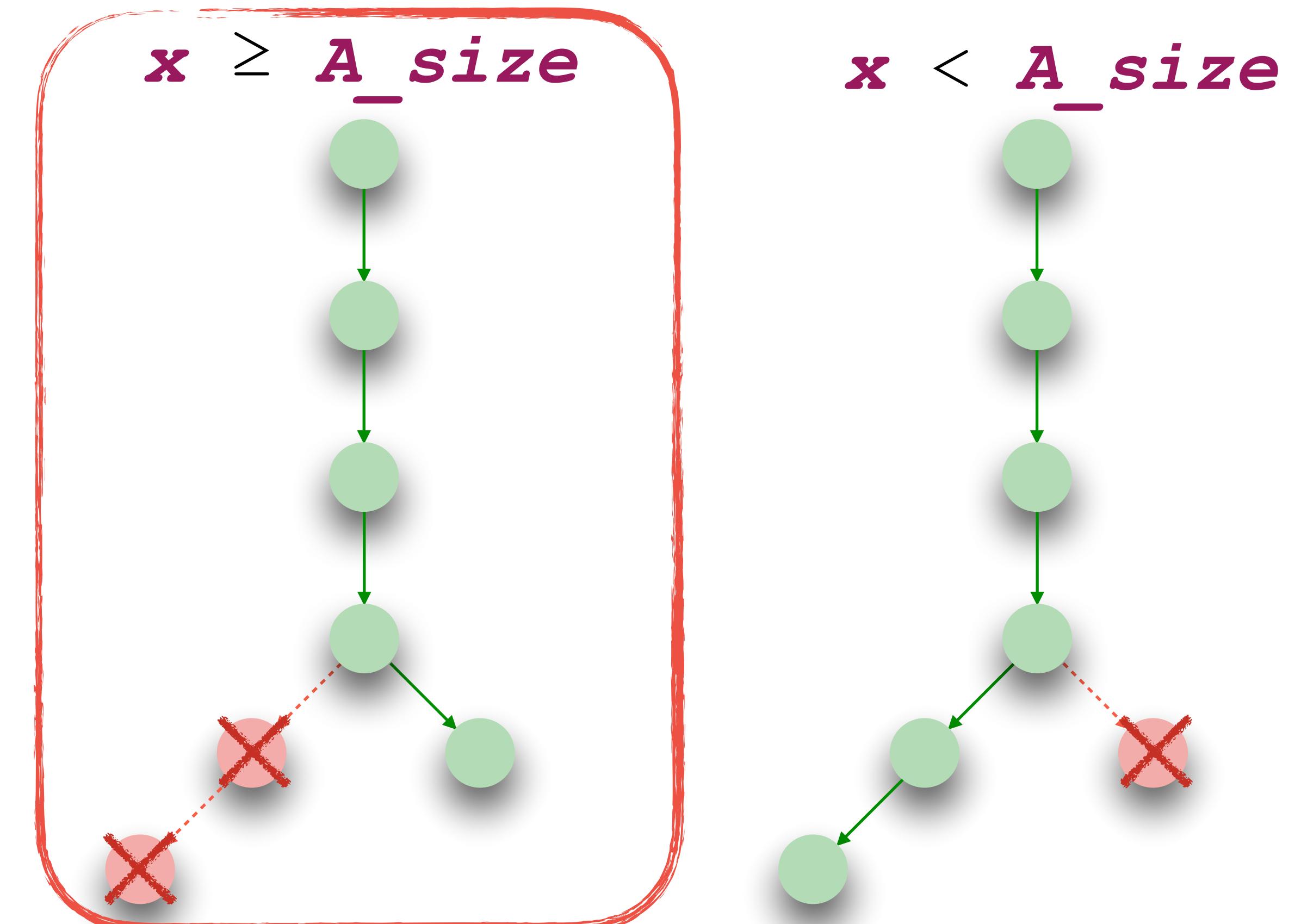
```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



```
start; pc L1; load A+x; load B+A[x]; rollback; pc END
```

Symbolic execution

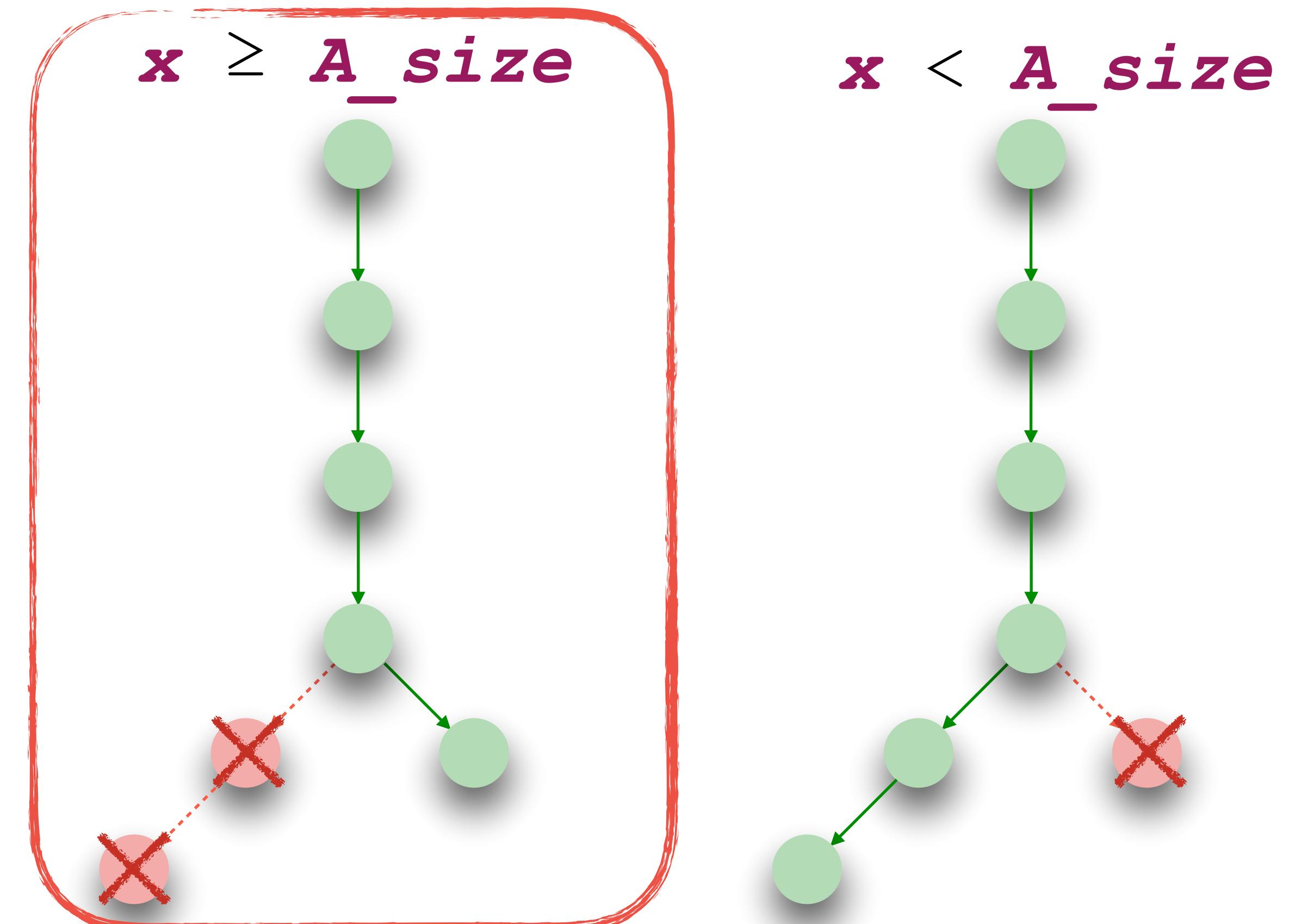
```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



```
start; pc L1; load A+x; load B+A[x]; rollback; pc END
```

Symbolic execution

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



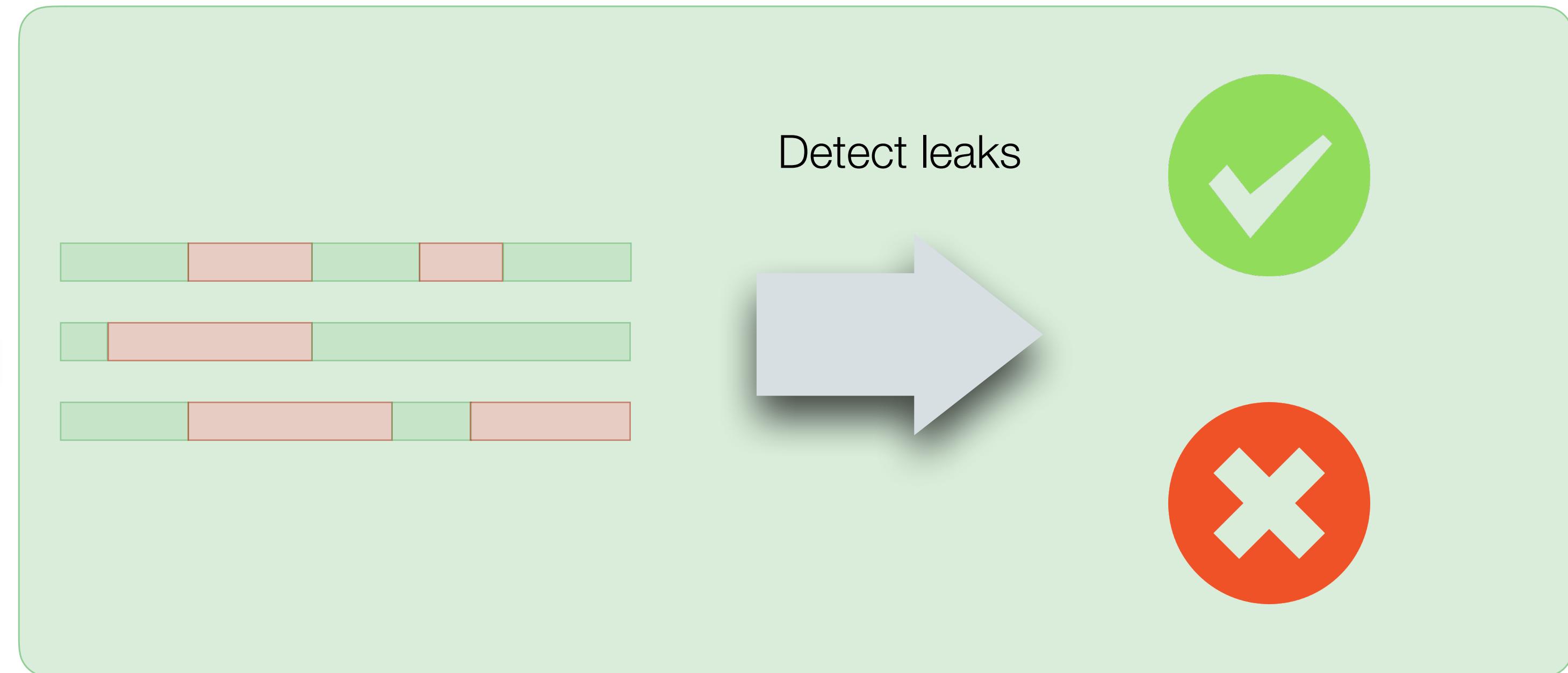
```
start; pc L1; load A+x; load B+A[x]; rollback; pc END
```

Detecting speculative leaks



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

Symbolic
execution



Detecting speculative leaks



For each $\tau \in \text{sym-traces}(P)$

if $\text{MemLeak}(\tau)$ then

return INSECURE

if $\text{CtrlLeak}(\tau)$ then

return INSECURE

return SECURE

L1:

END:

```
rax <- A  
rcx <- x  
jmp rcx≥rax  
load rax,  
load rax,
```



Detecting speculative leaks



For each $\tau \in \text{sym-traces}(P)$

if $\text{MemLeak}(\tau)$ then

return INSECURE

if $\text{CtrlLeak}(\tau)$ then

return INSECURE

return SECURE

```
rax <- A  
rcx <- x  
jmp rcx≥rax  
load rax,  
load rax,
```

L1:

END:



Memory leaks



Speculative memory accesses **must** depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

Memory leaks



Speculative memory accesses **must** depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

τ



Memory leaks



Speculative memory accesses **must** depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

τ



$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

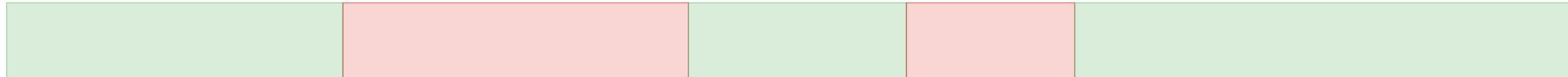
Memory leaks



Speculative memory accesses **must** depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

τ



$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

s_1

s_2

Memory leaks



Speculative memory accesses **must** depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

τ



$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$

$$s_1 \models \varphi$$

$$s_2 \models \varphi$$

Memory leaks



Speculative memory accesses **must** depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

τ

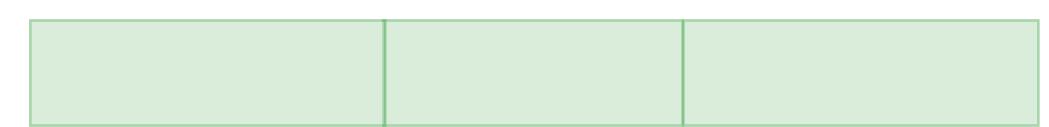


$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$

$s_1 \models \varphi$



$s_2 \models \varphi$



Memory leaks



Speculative memory accesses **must** depend only on

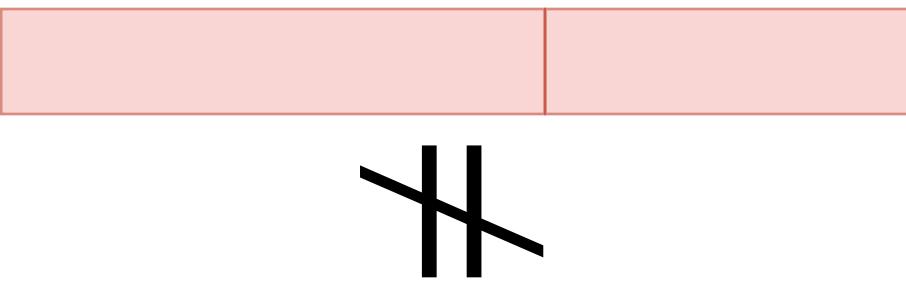
- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

τ



$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$

$s_1 \models \varphi$

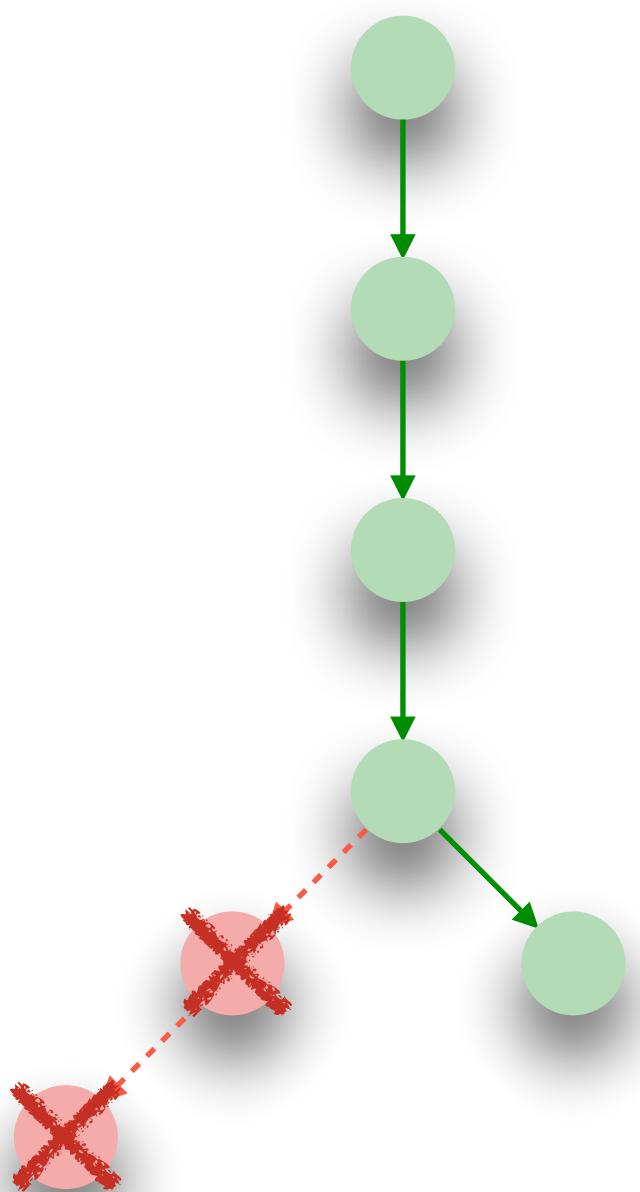


$s_2 \models \varphi$

Memory leaks

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

END:



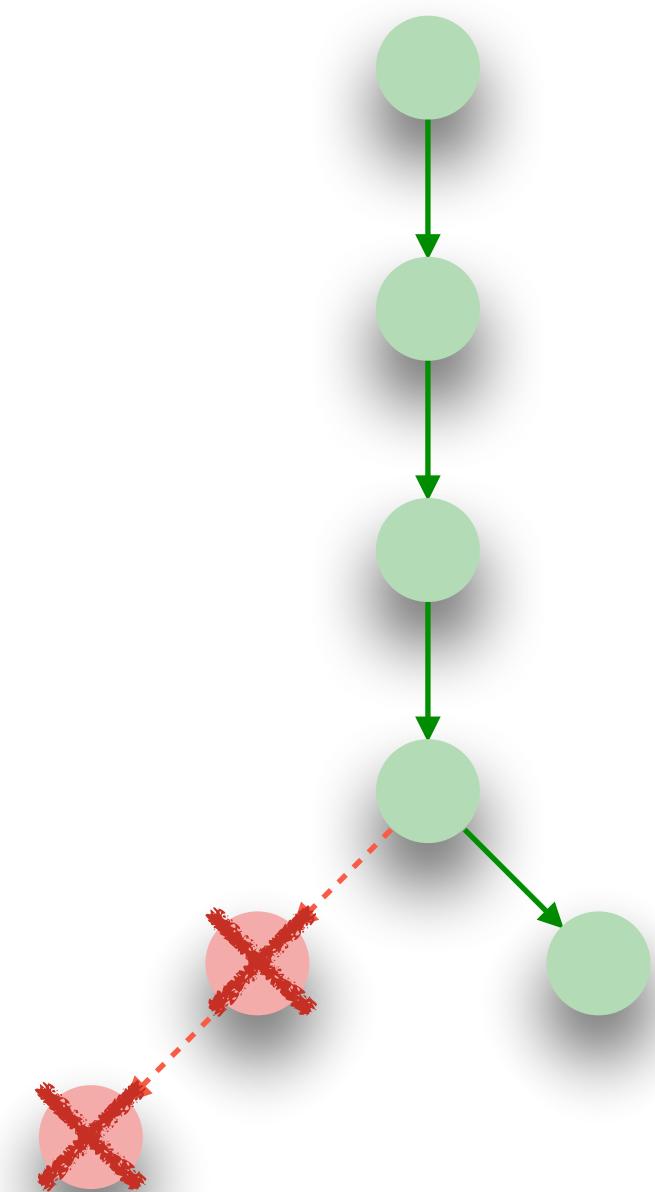
Policy
x, A_size, A, B
are public

τ = start; pc *L1*; load **A+x**; load **B+A**[**x**]; rollback; pc *END*

Memory leaks

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

END:



Policy
x, A_size, A, B
are public

τ = start; pc *L1*; load **A+x**; load **B+A** [**x**] ; rollback; pc *END*

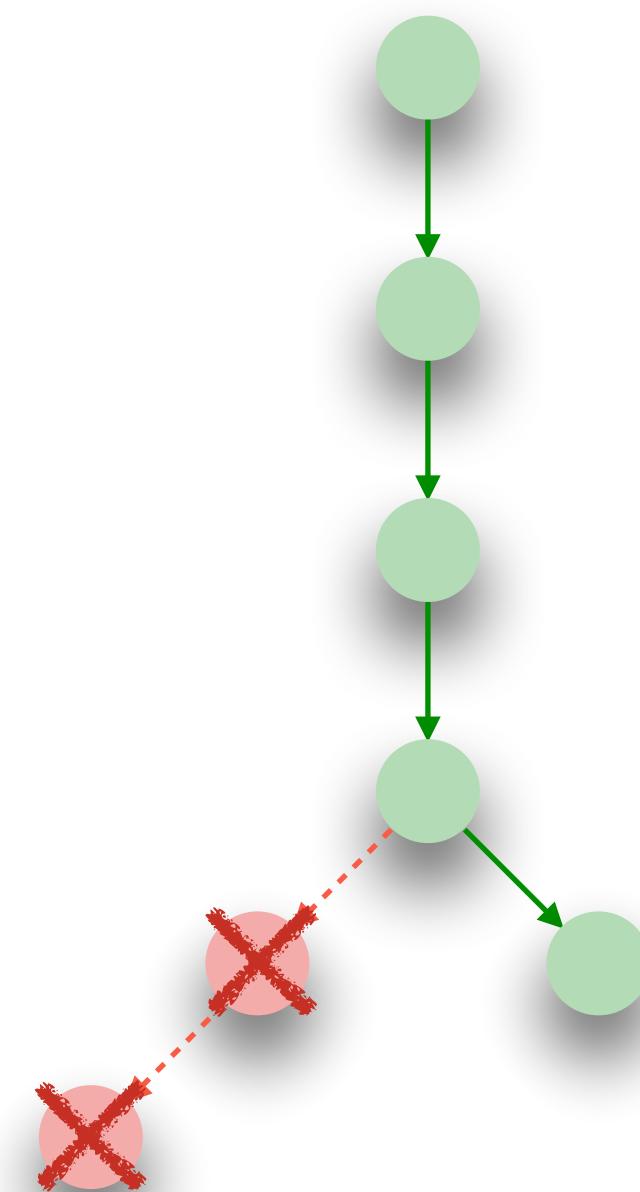
$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$



Memory leaks

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

END:



Policy
x, A_size, A, B
are public

$\tau = \text{start; pc } L1; \text{ load } \mathbf{A+x}; \text{ load } \mathbf{B+A}[\mathbf{x}]; \text{ rollback; pc } END$

$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

s_1

s_2

Memory leaks

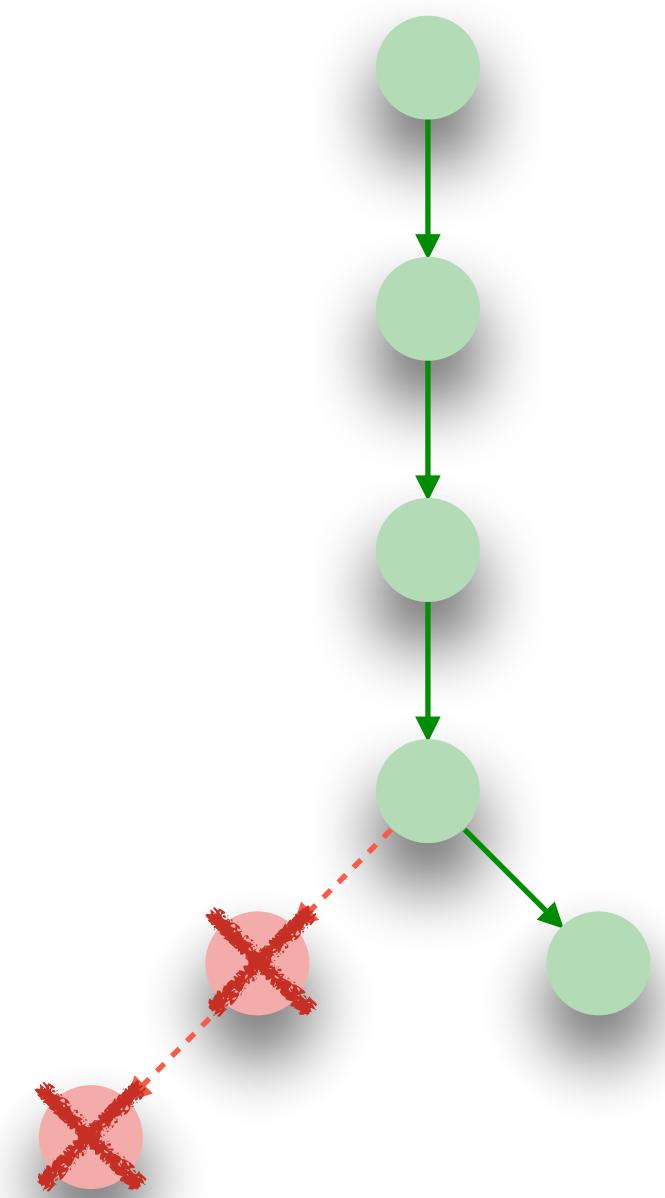


```

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

```

END:



Policy
 $\mathbf{x}, \mathbf{A_size}, \mathbf{A}, \mathbf{B}$
are public

$\tau = \text{start; pc } L1; \text{ load } \mathbf{A+x}; \text{ load } \mathbf{B+A}[\mathbf{x}]; \text{ rollback; pc } END$

$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

s_1

s_2

$$\mathbf{x}_1=\mathbf{x}_2 \wedge \mathbf{A_size}_1=\mathbf{A_size}_2 \wedge \mathbf{A}_1=\mathbf{A}_2 \wedge \mathbf{B}_1=\mathbf{B}_2$$

Memory leaks

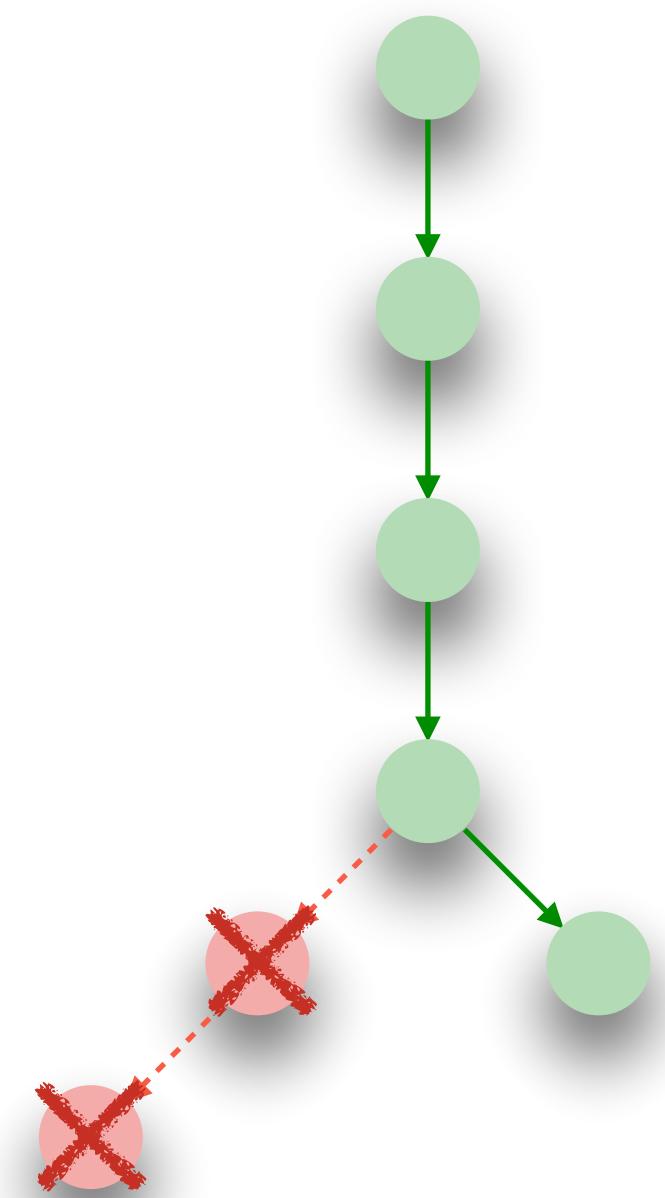


```

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

```

END:



Policy
 $\mathbf{x}, \mathbf{A_size}, \mathbf{A}, \mathbf{B}$
are public

$\tau = \text{start; pc } L1; \text{ load } \mathbf{A+x}; \text{ load } \mathbf{B+A}[\mathbf{x}]; \text{ rollback; pc } END$

$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

$S_1 \models \mathbf{x}_1 \geq \mathbf{A_size}_1$

$S_2 \models \mathbf{x}_2 \geq \mathbf{A_size}_2$

$$\mathbf{x}_1 = \mathbf{x}_2 \wedge \mathbf{A_size}_1 = \mathbf{A_size}_2 \wedge \mathbf{A}_1 = \mathbf{A}_2 \wedge \mathbf{B}_1 = \mathbf{B}_2$$

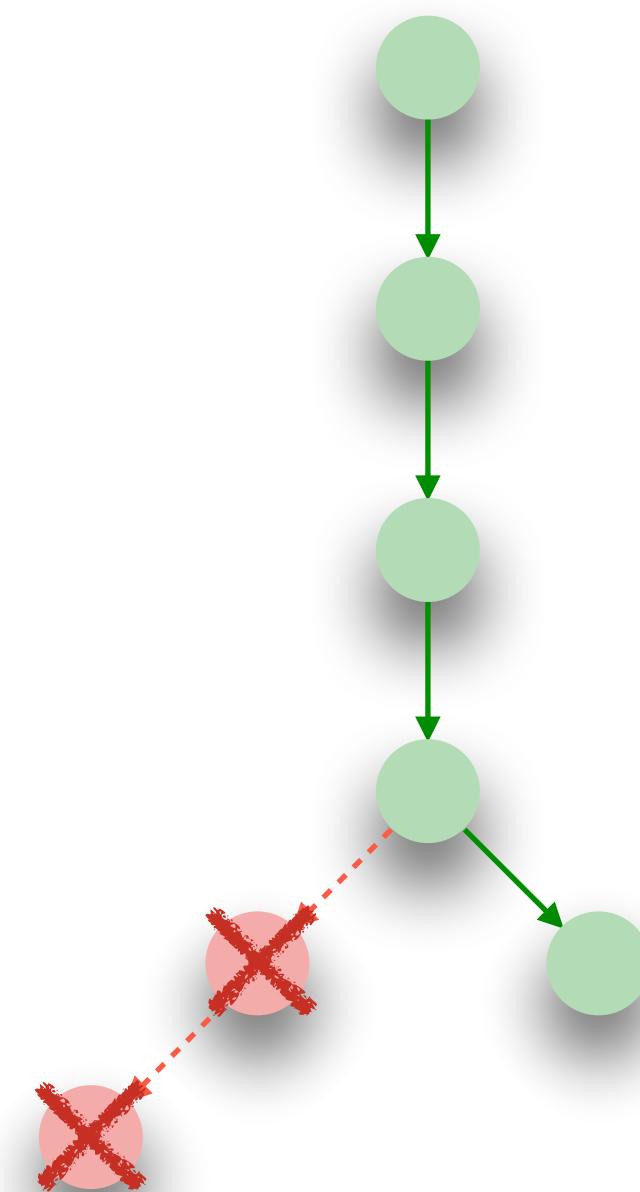
Memory leaks

```

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

```

END:



Policy
 x, A_size, A, B
are public

$\tau = \text{start; pc } L1; \text{ load } A+x; \text{ load } B+A[x]; \text{ rollback; pc } END$

$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

$S_1 \models x_1 \geq A_size_1$

pc *END*

||

$S_2 \models x_2 \geq A_size_2$

pc *END*

$x_1 = x_2 \wedge A_size_1 = A_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

Always true!

Memory leaks

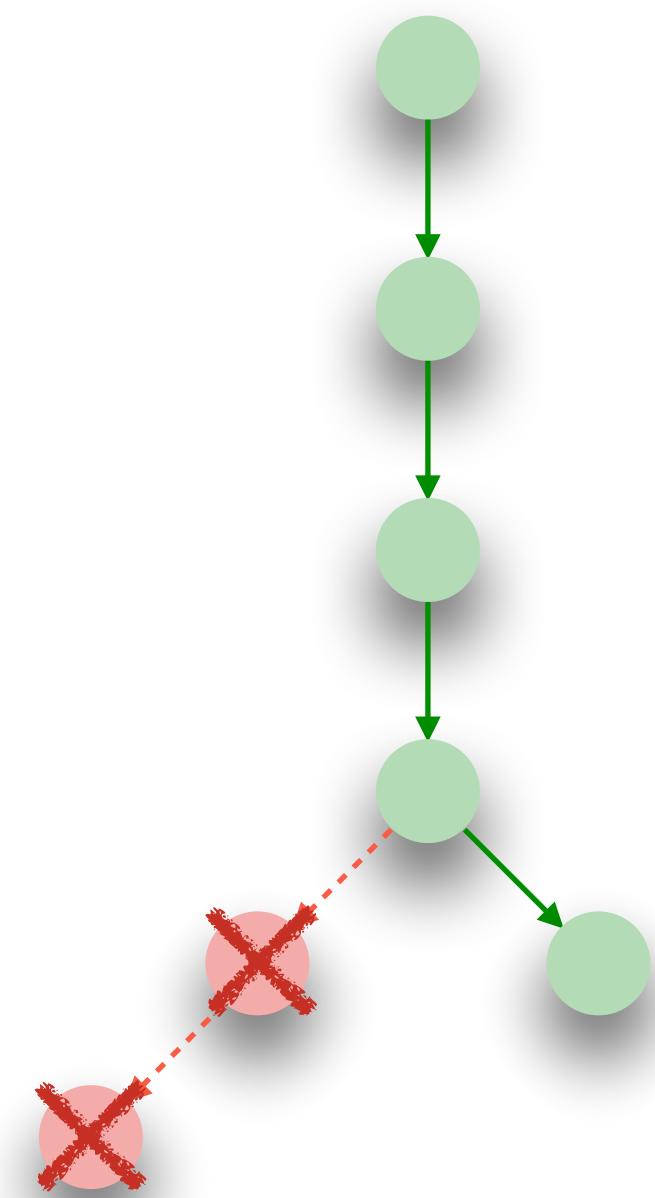


```

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

```

END:



Policy
 x, A_size, A, B
 are public

$\tau = \text{start; pc } L1; \text{ load } A+x; \text{ load } B+A[x]; \text{ rollback; pc } END$

$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$

$S_1 \models x_1 \geq A_size_1$

pc *END*

||

$S_2 \models x_2 \geq A_size_2$

pc *END*

$x_1 = x_2 \wedge A_size_1 = A_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

Always true!

Memory leaks

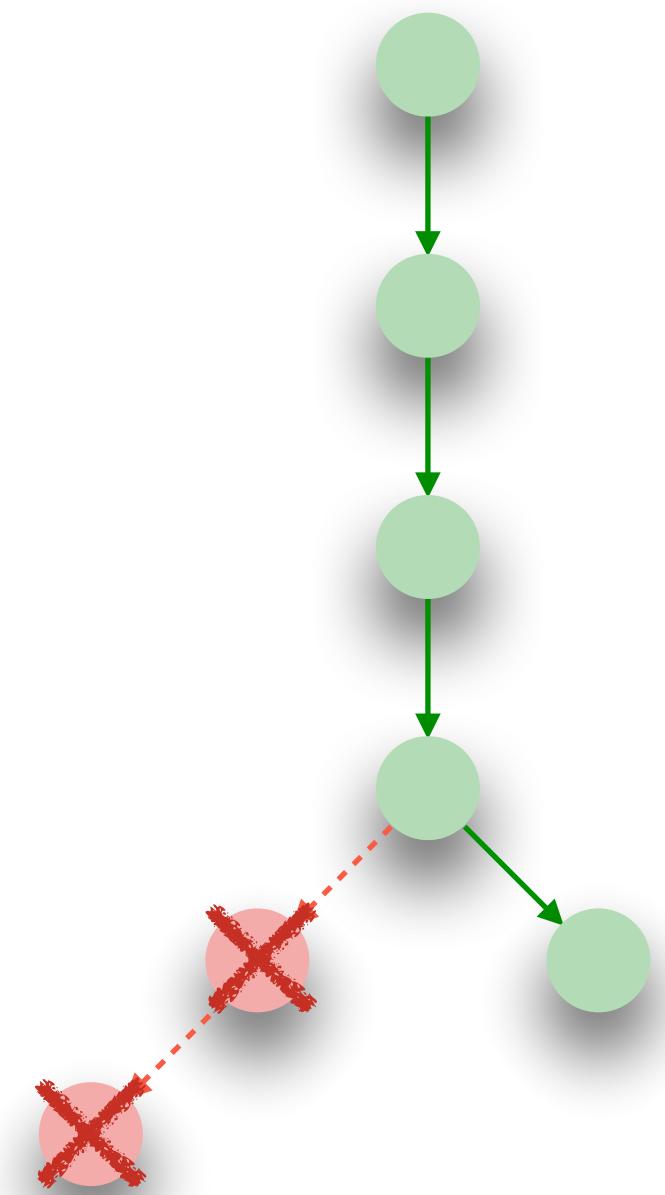


```

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

```

END:



Policy
 x, A_size, A, B
 are public

$\tau = \boxed{\text{start; pc } L1; \text{ load } A+x; \text{ load } B+A[x]; \text{ rollback; pc } END}$

$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \boxed{\neg obsEqv(\tau|_{spec})}$

$S_1 \models x_1 \geq A_size_1$

pc *END*

A_1+x_1

$S_2 \models x_2 \geq A_size_2$

pc *END*

A_2+x_2

$x_1=x_2 \wedge A_size_1=A_size_2 \wedge A_1=A_2 \wedge B_1=B_2$

Always true!

Memory leaks

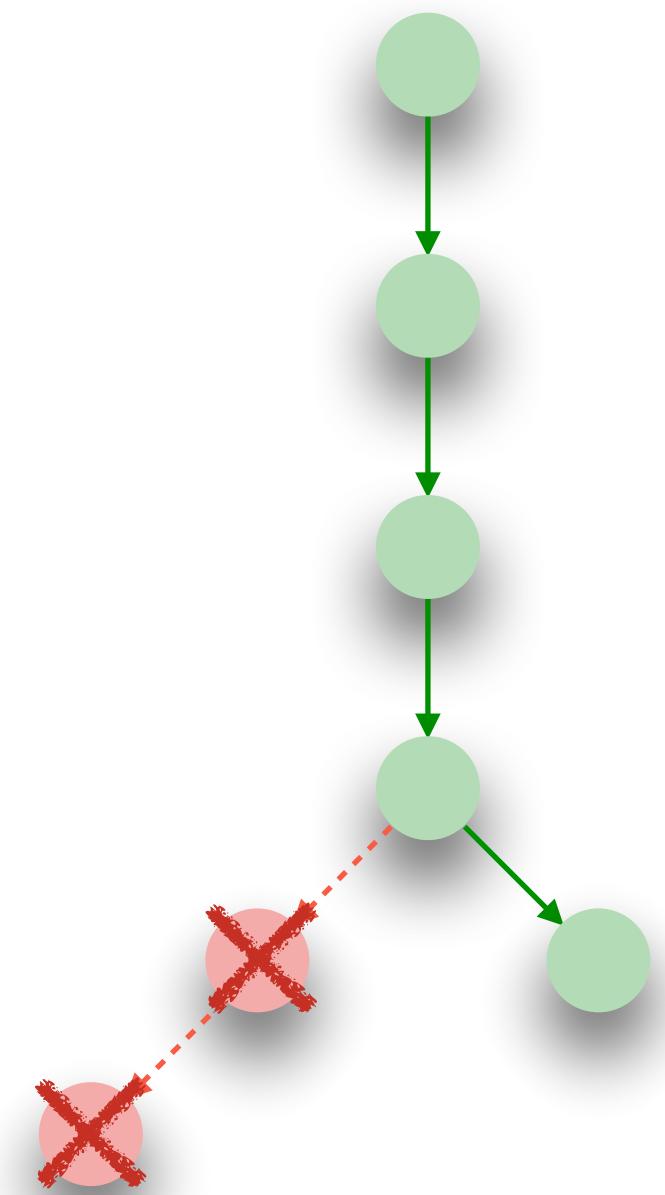


```

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

```

END:



Policy
 $\mathbf{x}, \mathbf{A_size}, \mathbf{A}, \mathbf{B}$
are public

$\tau = \text{start; pc } L1; \text{ load } \mathbf{A+x}; \text{ load } \mathbf{B+A[x]}; \text{ rollback; pc } END$

$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$

$S_1 \models \mathbf{x}_1 \geq \mathbf{A_size}_1$

pc *END*

$\mathbf{A}_1 + \mathbf{x}_1$

∨

$S_2 \models \mathbf{x}_2 \geq \mathbf{A_size}_2$

pc *END*

$\mathbf{A}_2 + \mathbf{x}_2$

$\mathbf{x}_1 = \mathbf{x}_2 \wedge \mathbf{A_size}_1 = \mathbf{A_size}_2 \wedge \mathbf{A}_1 = \mathbf{A}_2 \wedge \mathbf{B}_1 = \mathbf{B}_2$

Always true!

Memory leaks

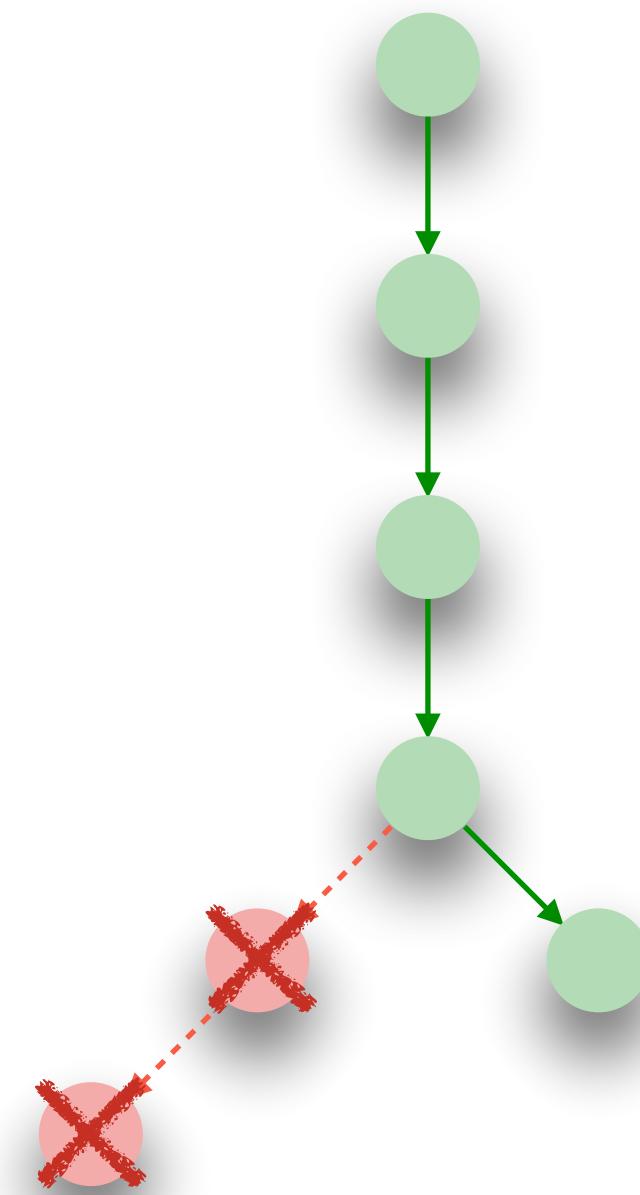


```

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

```

END:



Policy
 $\mathbf{x}, \mathbf{A_size}, \mathbf{A}, \mathbf{B}$
are public

$\tau = \boxed{\text{start; pc } L1; \text{ load } \mathbf{A+x}; \text{ load } \mathbf{B+A[x]}; \text{ rollback;}} \quad \boxed{\text{pc } END}$

$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \boxed{\neg obsEqv(\tau|_{spec})}$$

$$S_1 \models \mathbf{x}_1 \geq \mathbf{A_size}_1$$

pc *END*

$\mathbf{A}_1 + \mathbf{x}_1$

$\mathbf{B}_1 + \mathbf{A}_1 [\mathbf{x}_1]$

$$S_2 \models \mathbf{x}_2 \geq \mathbf{A_size}_2$$

pc *END*

$\mathbf{A}_2 + \mathbf{x}_2$

$\mathbf{B}_2 + \mathbf{A}_2 [\mathbf{x}_2]$

$\mathbf{x}_1 = \mathbf{x}_2 \wedge \mathbf{A_size}_1 = \mathbf{A_size}_2 \wedge \mathbf{A}_1 = \mathbf{A}_2 \wedge \mathbf{B}_1 = \mathbf{B}_2$

Always true!

Memory leaks

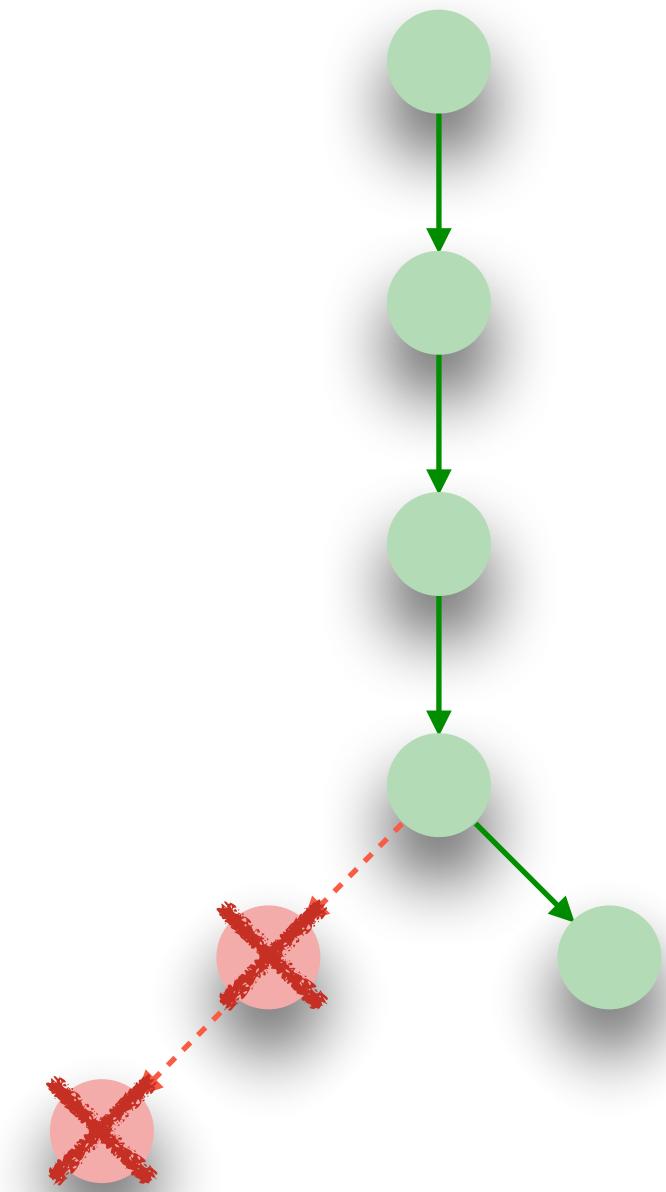


```

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

```

END:



Policy
 $\mathbf{x}, \mathbf{A_size}, \mathbf{A}, \mathbf{B}$
are public

$\tau = \text{start; pc } L1; \text{ load } \mathbf{A+x}; \text{ load } \mathbf{B+A [x]}; \text{ rollback; pc } END$

$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

$$S_1 \models \mathbf{x}_1 \geq \mathbf{A_size}_1$$

pc *END*

$\mathbf{A}_1 + \mathbf{x}_1$

$\mathbf{B}_1 + \mathbf{A}_1 [\mathbf{x}_1]$

$$S_2 \models \mathbf{x}_2 \geq \mathbf{A_size}_2$$

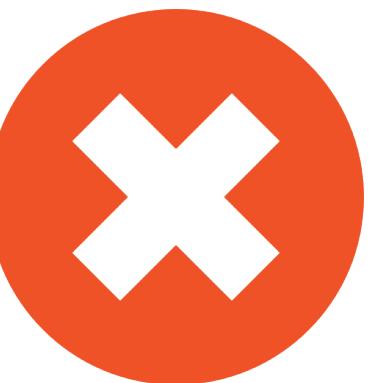
pc *END*

$\mathbf{A}_2 + \mathbf{x}_2$

$\mathbf{B}_2 + \mathbf{A}_2 [\mathbf{x}_2]$

$\mathbf{x}_1 = \mathbf{x}_2 \wedge \mathbf{A_size}_1 = \mathbf{A_size}_2 \wedge \mathbf{A}_1 = \mathbf{A}_2 \wedge \mathbf{B}_1 = \mathbf{B}_2$

Always true!



Experimental results

Ex.	VISUAL C++				ICC				CLANG					
	UNP		FEN		UNP		FEN		UNP		FEN		SLH	
	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•	•	o	o	•	•	o	o	•	•	•	•
02	o	o	•	•	o	o	•	•	o	o	•	•	•	•
03	o	o	•	o	o	o	•	•	o	o	•	•	•	•
04	o	o	o	o	o	o	•	•	o	o	•	•	•	•
05	o	o	•	o	o	o	•	•	o	o	•	•	•	•
06	o	o	o	o	o	o	•	•	o	o	•	•	•	•
07	o	o	o	o	o	o	•	•	o	o	•	•	•	•
08	o	•	o	•	o	•	•	•	o	•	•	•	•	•
09	o	o	o	o	o	o	•	•	o	o	•	•	•	•
10	o	o	o	o	o	o	•	•	o	o	•	•	•	o
11	o	o	o	o	o	o	•	•	o	o	•	•	•	•
12	o	o	o	o	o	o	•	•	o	o	•	•	•	•
13	o	o	o	o	o	o	•	•	o	o	•	•	•	•
14	o	o	o	o	o	o	•	•	o	o	•	•	•	•
15	o	o	o	o	o	o	•	•	o	o	•	•	o	•

Experimental results

15 Spectre variants from
Paul Kocher

Variant	ICC				CLANG			
	UNP		FEN		UNP		FEN	
	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•	•	o	o	•	•
02	o	o	•	•	o	o	•	•
03	o	o	•	o	o	•	o	•
04	o	o	o	o	o	•	o	•
05	o	o	•	o	o	•	o	•
06	o	o	o	o	o	•	o	•
07	o	o	o	o	o	•	o	•
08	o	•	o	•	o	•	o	•
09	o	o	o	o	o	•	o	•
10	o	o	o	o	o	•	o	•
11	o	o	o	o	o	•	o	•
12	o	o	o	o	o	•	o	•
13	o	o	o	o	o	•	o	•
14	o	o	o	o	o	•	o	•
15	o	o	o	o	o	•	o	•

Experimental results

15 Spectre variants from
Paul Kocher

Line	Variant	ICC				CLANG			
		UNP		FEN		UNP		FEN	
		-00	-02	-00	-02	-00	-02	-00	-02
01	if (x < A_size)	•	○	•	○	○	○	•	•
02	y = B[A[x] * 512]	○	○	○	○	○	○	●	●
03		○	○	○	○	○	○	●	●
04		○	○	○	○	○	○	●	●
05		○	○	●	○	○	○	●	●
06		○	○	○	○	○	○	●	●
07		○	○	○	○	○	○	●	●
08		○	●	○	●	○	●	●	●
09		○	○	○	○	○	○	●	●
10		○	○	○	○	○	○	●	○
11		○	○	○	○	●	○	●	●
12		○	○	○	○	●	○	●	●
13		○	○	○	○	●	○	●	●
14		○	○	○	○	●	○	●	●
15		○	○	○	○	●	○	●	●

Experimental results

15 Spectre variants from
Paul Kocher

Variant	ICC				CLANG			
	UNP		FEN		UNP		FEN	
	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•	•	o	o	•	•
02	o	o	•	•	o	o	•	•
03	o	o	•	o	o	•	•	•
04	o							
05	o	y = B[A[x < A_size ? (x+1) : 0] * 512]						
06	o	o	o	o	o	o	o	o
07	o	o	o	o	o	o	o	o
08	o	•	o	•	o	•	•	•
09	o	o	o	o	o	o	o	o
10	o	o	o	o	o	o	o	o
11	o	o	o	o	o	o	o	o
12	o	o	o	o	o	o	o	o
13	o	o	o	o	o	o	o	o
14	o	o	o	o	o	o	o	o
15	o	o	o	o	o	o	o	o

Experimental results

15 Spectre variants from
Paul Kocher

Variant	ICC				CLANG			
	UNP		FEN		UNP		FEN	
	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•	•	o	o	•	•
02	o	o	•	•	o	o	•	•
03	o	o	•	o	o	•	o	•
04	o	o	o	o	o	•	o	•
05	o	o	•	o	o	•	o	•
06	o	o	o	o	o	o	•	•
07	o	o	o	o	o	o	•	•
08	o	•	o	•	o	o	•	•
09	o	o	o	o	o	o	•	•
10	o	o	o	o	o	o	o	o
11	o	o	o	o	o	•	o	•
12	o	o	o	o	o	•	o	•
13	o	o	o	o	o	•	o	•
14	o	o	o	o	o	•	o	•
15	o	o	o	o	o	•	o	•

```
if (x < A_size)
    if (A[x]==k)
        y = B[0]
```

Experimental results

Ex.	VISUAL C++				ICC				CLANG					
	UNP		FEN		UNP		FEN		UNP		FEN		SLH	
	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•	•	o	o	•	•	o	o	•	•	•	•
02	o	o	•	•	o	o	•	•	o	o	•	•	•	•
03	o	o	•	o	o	o	•	•	o	o	•	•	•	•
04	o	o	o	o	o	o	•	•	o	o	•	•	•	•
05	o	o	•	o	o	o	•	•	o	o	•	•	•	•
06	o	o	o	o	o	o	•	•	o	o	•	•	•	•
07	o	o	o	o	o	o	•	•	o	o	•	•	•	•
08	o	•	o	•	o	•	•	•	o	•	•	•	•	•
09	o	o	o	o	o	o	•	•	o	o	•	•	•	•
10	o	o	o	o	o	o	•	•	o	o	•	•	•	o
11	o	o	o	o	o	o	•	•	o	o	•	•	•	•
12	o	o	o	o	o	o	•	•	o	o	•	•	•	•
13	o	o	o	o	o	o	•	•	o	o	•	•	•	•
14	o	o	o	o	o	o	•	•	o	o	•	•	•	•
15	o	o	o	o	o	o	•	•	o	o	•	•	o	•

Experimental results

Ex.	VISUAL C++				ICC				CLANG				
	UNP		FEN		UNP		FEN		UNP		FEN		SLH
	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00
01	o	o	•	•	o	o	•	•	o	o	•	•	•
02	o	o	•	•	o	o	•	•	o	o	•	•	•
03	o	o	•	o	o	o	•	•	o	o	•	•	•
04	o	o	o	o	o	o	•	•	o	o	•	•	•
05	o	o	•	o	o	o	•	•	o	o	•	•	•
06	o	o	o	o	o	o	•	•	o	o	•	•	•
07	o	o	o	o	o	o	•	•	o	o	•	•	•
08	o	•	o	•	o	•	•	•	o	•	•	•	•
09	o	o	o	o	o	o	•	•	o	o	•	•	•
10	o	o	o	o	o	o	•	•	o	o	•	•	o
11	o	o	o	o	o	o	•	•	o	o	•	•	•
12	o	o	o	o	o	o	•	•	o	o	•	•	•
13	o	o	o	o	o	o	•	•	o	o	•	•	•
14	o	o	o	o	o	o	•	•	o	o	•	•	•
15	o	o	o	o	o	o	•	•	o	o	•	•	o

Experimental results

No countermeasures

Ex.	VISUAL C++				ICC				CLANG					
	UNP	FEN	UNP	FEN	UNP	FEN	UNP	FEN	SLH					
	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•	•	o	o	•	•	o	o	•	•	•	•
02	o	o	•	•	o	o	•	•	o	o	•	•	•	•
03	o	o	•	o	o	o	•	•	o	o	•	•	•	•
04	o	o	o	o	o	o	•	•	o	o	•	•	•	•
05	o	o	•	o	o	o	•	•	o	o	•	•	•	•
06	o	o	o	o	o	o	•	•	o	o	•	•	•	•
07	o	o	o	o	o	o	•	•	o	o	•	•	•	•
08	o	•	o	•	o	•	•	•	o	•	•	•	•	•
09	o	o	o	o	o	o	•	•	o	o	•	•	•	•
10	o	o	o	o	o	o	•	•	o	o	•	•	•	o
11	o	o	o	o	o	o	•	•	o	o	•	•	•	•
12	o	o	o	o	o	o	•	•	o	o	•	•	•	•
13	o	o	o	o	o	o	•	•	o	o	•	•	•	•
14	o	o	o	o	o	o	•	•	o	o	•	•	•	•
15	o	o	o	o	o	o	•	•	o	o	•	•	o	•

Experimental results

Automated insertion of fences

Ex.	VISUAL C++				ICC				CLANG				
	UNP		FEN		UNP		FEN		UNP		FEN		SLH
	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00
01	o	o	•	•	o	o	•	•	o	o	•	•	•
02	o	o	•	•	o	o	•	•	o	o	•	•	•
03	o	o	•	o	o	o	•	•	o	o	•	•	•
04	o	o	o	o	o	o	•	•	o	o	•	•	•
05	o	o	•	o	o	o	•	•	o	o	•	•	•
06	o	o	o	o	o	o	•	•	o	o	•	•	•
07	o	o	o	o	o	o	•	•	o	o	•	•	•
08	o	•	o	•	o	•	•	•	o	•	•	•	•
09	o	o	o	o	o	o	•	•	o	o	•	•	•
10	o	o	o	o	o	o	•	•	o	o	•	•	o
11	o	o	o	o	o	o	•	•	o	o	•	•	•
12	o	o	o	o	o	o	•	•	o	o	•	•	•
13	o	o	o	o	o	o	•	•	o	o	•	•	•
14	o	o	o	o	o	o	•	•	o	o	•	•	•
15	o	o	o	o	o	o	•	•	o	o	•	•	•

Experimental results

Speculative load
hardening

Ex.	VISUAL C++				ICC				CLANG				SLH	
	UNP		FEN		UNP		FEN		UNP		FEN		SLH	
	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•	•	o	o	•	•	o	o	•	•	•	•
02	o	o	•	•	o	o	•	•	o	o	•	•	•	•
03	o	o	•	o	o	o	•	•	o	o	•	•	•	•
04	o	o	o	o	o	o	•	•	o	o	•	•	•	•
05	o	o	•	o	o	o	•	•	o	o	•	•	•	•
06	o	o	o	o	o	o	•	•	o	o	•	•	•	•
07	o	o	o	o	o	o	•	•	o	o	•	•	•	•
08	o	•	o	•	o	•	•	•	o	•	•	•	•	•
09	o	o	o	o	o	o	•	•	o	o	•	•	•	•
10	o	o	o	o	o	o	•	•	o	o	•	•	•	o
11	o	o	o	o	o	o	•	•	o	o	•	•	•	•
12	o	o	o	o	o	o	•	•	o	o	•	•	•	•
13	o	o	o	o	o	o	•	•	o	o	•	•	•	•
14	o	o	o	o	o	o	•	•	o	o	•	•	•	•
15	o	o	o	o	o	o	•	•	o	o	•	•	o	•

Experimental results

Ex.	VISUAL C++				ICC				CLANG				SLH	
	UNP		FEN		UNP		FEN		UNP		FEN			
	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•	•	o	o	•	•	o	o	•	•	•	•
02	o	o	•	•	o	o	•	•	o	o	•	•	•	•
03	o	o	•	o	o	o	•	•	o	o	•	•	•	•
04	o	o	o	o	o	o	•	•	o	o	•	•	•	•
05	o	o	•	o	o	o	•	•	o	o	•	•	•	•
06	o	o	o	o	o	o	•	•	o	o	•	•	•	•
07	o	o	o	o	o	o	•	•	o	o	•	•	•	•
08	o	•	o	•	o	•	•	•	o	•	•	•	•	•
09	o	o	o	o	o	o	•	•	o	o	•	•	•	•
10	o	o	o	o	o	o	•	•	o	o	•	•	•	o
11	o	o	o	o	o	o	•	•	o	o	•	•	•	•
12	o	o	o	o	o	o	•	•	o	o	•	•	•	•
13	o	o	o	o	o	o	•	•	o	o	•	•	•	•
14	o	o	o	o	o	o	•	•	o	o	•	•	•	•
15	o	o	o	o	o	o	•	•	o	o	•	•	o	•

Experimental results

Ex.	VISUAL C++				ICC				CLANG					
	UNP		FEN		UNP		FEN		UNP		FEN		SLH	
	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•	•	o	o	•	•	o	o	•	•	•	•
02	o	o	•	•	o	o	•	•	o	o	•	•	•	•
03	o	o	•	o	o	o	•	•	o	o	•	•	•	•
04	o	o	o	o	o	o	•	•	o	o	•	•	•	•
05	o	o	•	o	o	o	•	•	o	o	•	•	•	•
06	o	o	o	o	o	o	•	•	o	o	•	•	•	•
07	o	o	o	o	o	o	•	•	o	o	•	•	•	•
08	o	•	o	•	o	•	•	•	o	•	•	•	•	•
09	o	o	o	o	o	o	•	•	o	o	•	•	•	•
10	o	o	o	o	o	o	•	•	o	o	•	•	•	o
11	o	o	o	o	o	o	•	•	o	o	•	•	•	•
12	o	o	o	o	o	o	•	•	o	o	•	•	•	•
13	o	o	o	o	o	o	•	•	o	o	•	•	•	•
14	o	o	o	o	o	o	•	•	o	o	•	•	•	•
15	o	o	o	o	o	o	•	•	o	o	•	•	o	•

Experimental results

Ex.	VISUAL C++		ICC		CLANG		SLH	
	UNP	FEN	UND	FEN	UND	FEN		
	-00	-01	-01	-01	-01	-01	-00	-02
01	o						•	•
02	o						•	•
03	o						•	•
04	o						•	•
05	o						•	•
06	o						•	•
07	o						•	•
08	o						•	•
09	o						•	•
10	o						•	o
11	o						•	•
12	o						•	•
13	o	o	o	o	•	o	•	•
14	o	o	o	o	•	o	•	•
15	o	o	o	o	•	o	•	•

Summary

- Leaks in all unprotected programs (except example #08 with optimizations)
- Confirm all vulnerabilities in VCC pointed out by Paul Kocher
- Programs with fences (ICC and Clang) are secure
 - But: Unnecessary fences
- Programs with SLH are secure except #10 and #15

Experimental results

Ex.	VISUAL C++				ICC				CLANG					
	UNP		FEN		UNP		FEN		UNP		FEN		SLH	
	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02	-00	-02
01	o	o	•								•	•	•	•
02	o	o	•								•	•	•	•
03	o	o	•								•	•	•	•
04	o	o	o								•	•	•	•
05	o	o	•								•	•	•	•
06	o	o	o								•	•	•	•
07	o	o	o								•	•	•	•
08	o	•	o								•	•	•	•
09	o	o	o								•	•	•	•
10	o	o	o								•	•	•	o
11	o	o	o	o	o	o	•	•	o	o	•	•	•	•
12	o	o	o	o	o	o	•	•	o	o	•	•	•	•
13	o	o	o	o	o	o	•	•	o	o	•	•	•	•
14	o	o	o	o	o	o	•	•	o	o	•	•	•	•
15	o	o	o	o	o	o	•	•	o	o	•	•	o	•

Performance

- Programs ~20-200 lines of assembly code
- Analysis terminates in less than 30 sec
- Except for example #05 (< 2 min)

4. Challenges

Scalable analysis

Goal:

Analysis of large, security-critical applications:

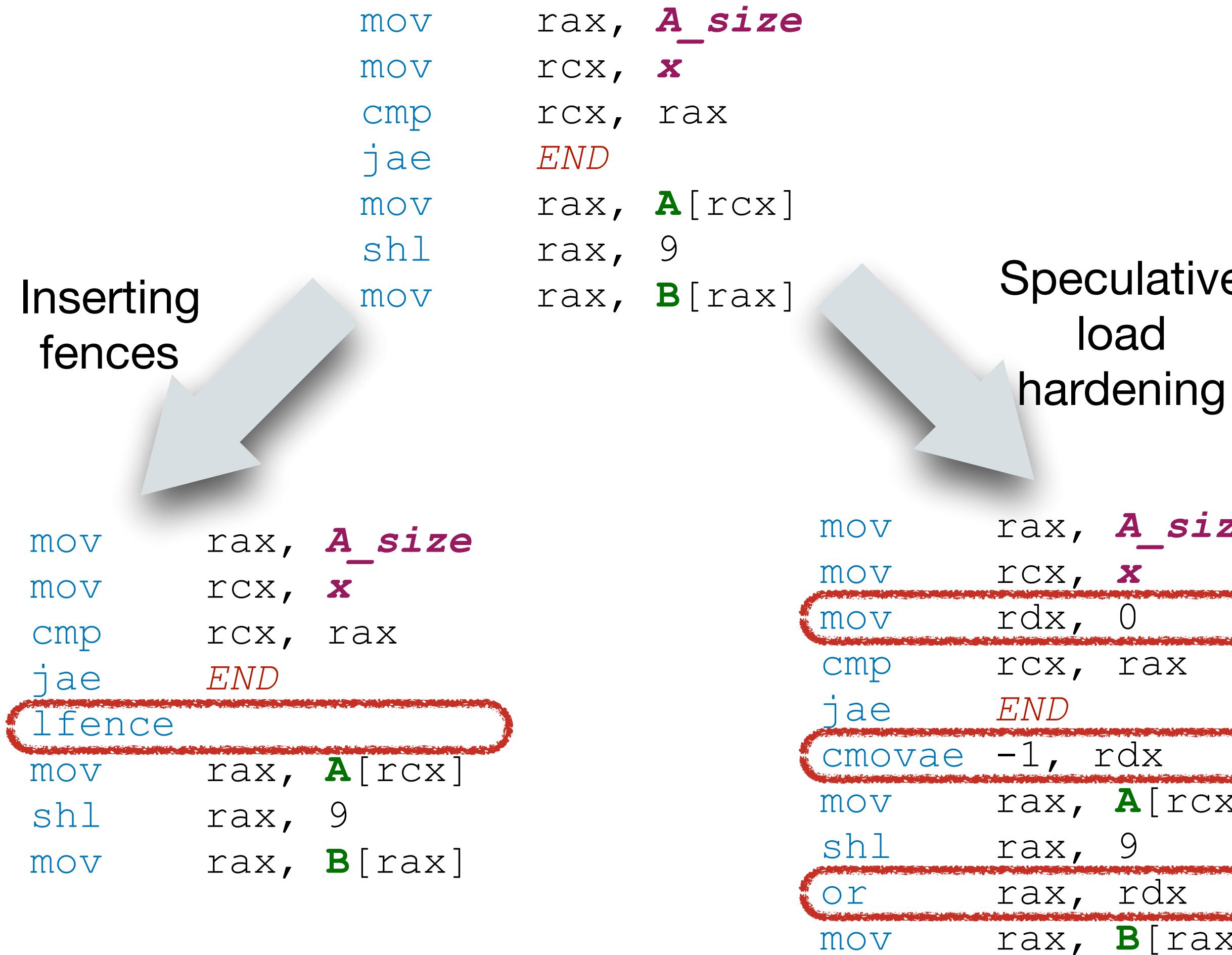
- Intel SGX SDK
- Xen hypervisor
- microkernels



Need: Scalable analysis of speculative non-interference

- Exploit “locality” of speculative execution
- Develop scalable abstractions

Verifying compiler-level countermeasures



How can we **verify** such countermeasures?

A sound HW/SW security contract

Instruction-set architecture: too weak for security guarantees

Microarchitecture: not available publicly, and too detailed for analysis

A sound HW/SW security contract

Instruction-set architecture: to weak for security guarantees

HW/SW security contract

Microarchitecture: not available publicly, and too detailed for analysis

Find out more in the paper:
<https://arxiv.org/abs/1812.08639>

To appear in: *IEEE Symposium on Security & Privacy, 2020*

Find out more in the paper:
<https://arxiv.org/abs/1812.08639>

To appear in: *IEEE Symposium on Security & Privacy, 2020*

I am looking for PhD students and postdocs!

Find out more in the paper:
<https://arxiv.org/abs/1812.08639>

To appear in: *IEEE Symposium on Security & Privacy, 2020*

I am looking for PhD students and postdocs!

Thank you for your attention!

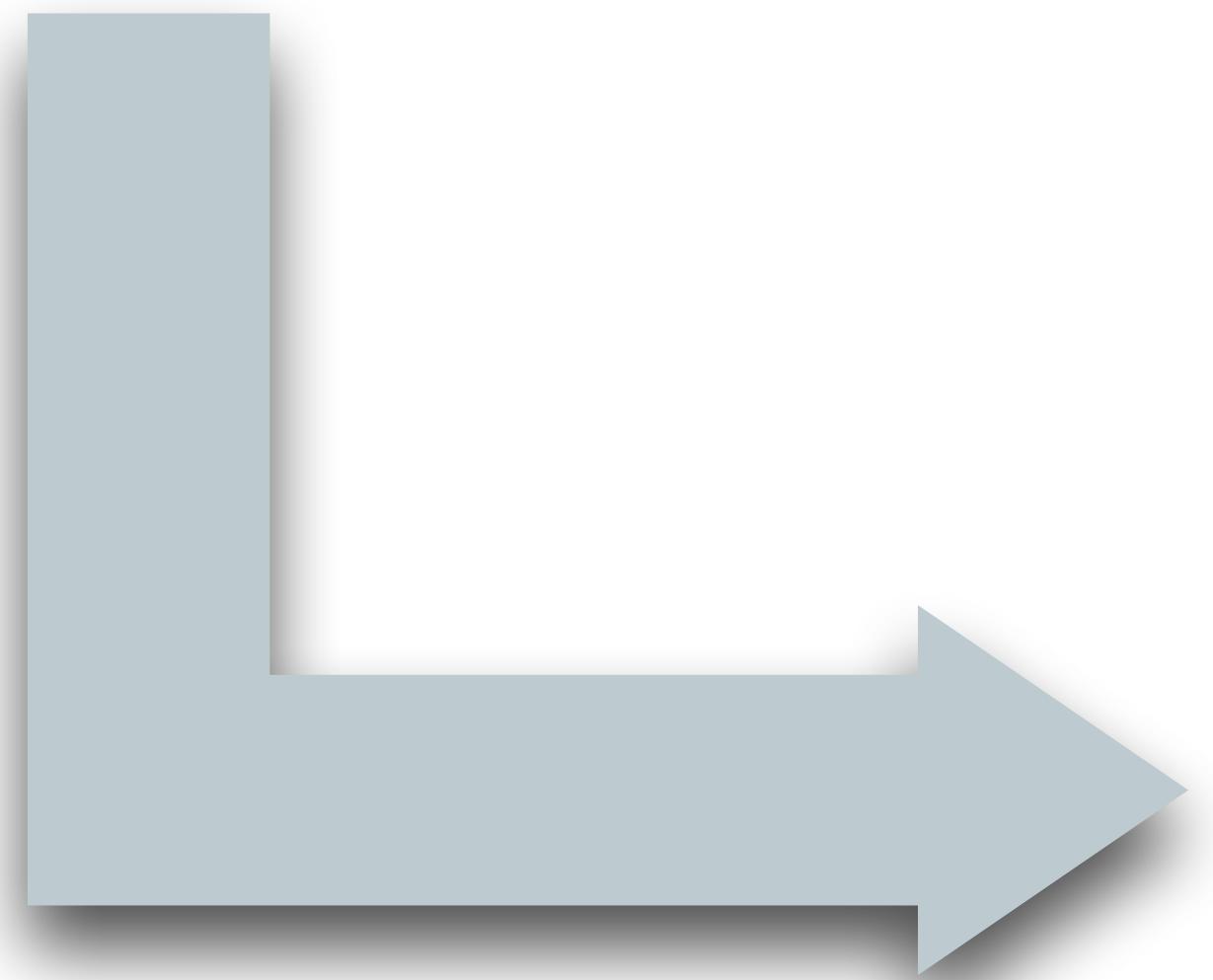
Backup

Example #01 - SLH

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

Example #01 - SLH

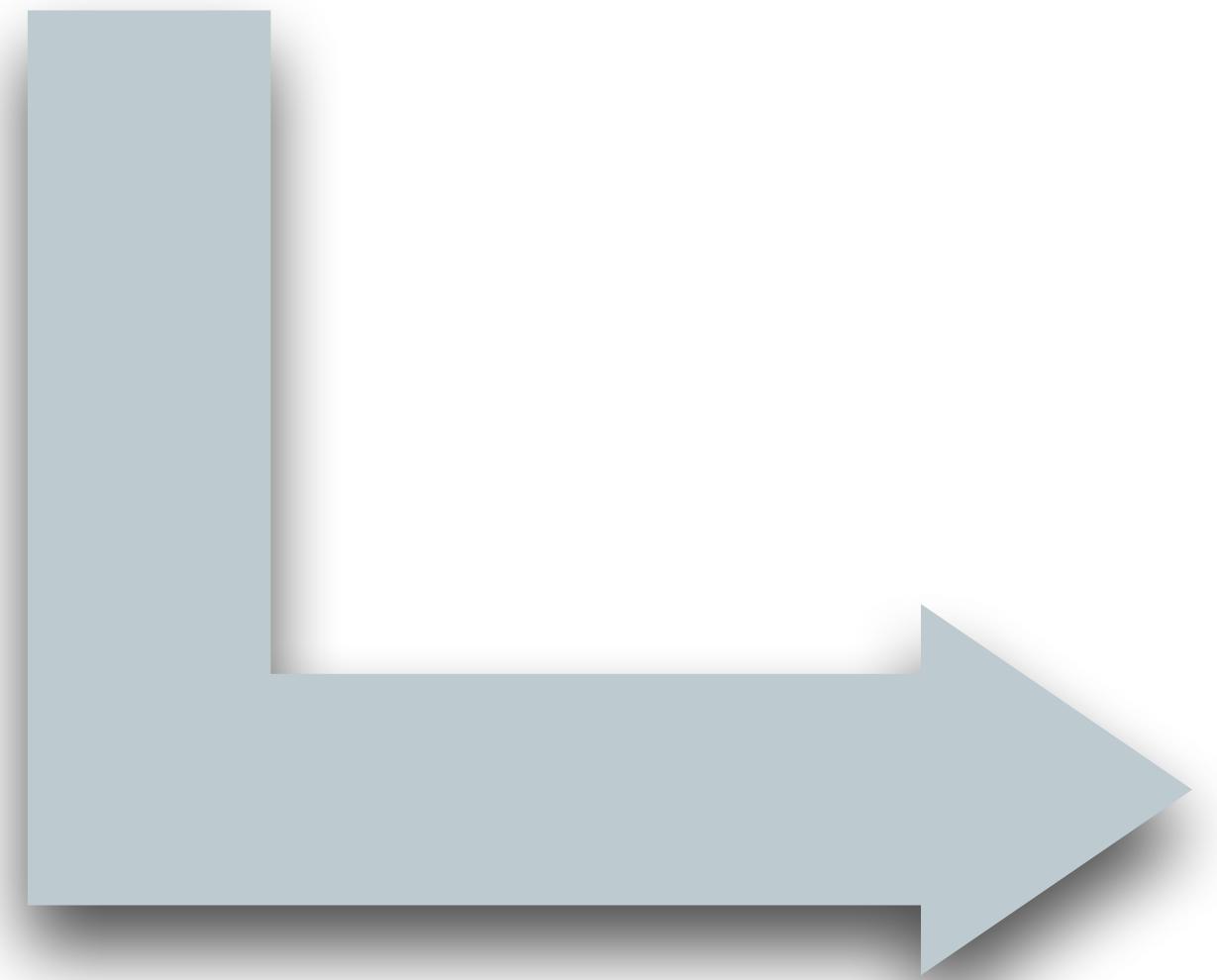
```
if (x < A_size)
    y = B[A[x] * 512]
```



mov	rax, A_size
mov	rcx, x
mov	rdx, 0
cmp	rcx, rax
jae	END
cmoveae	-1, rdx
mov	rax, A[rcx]
shl	rax, 9
or	rax, rdx
mov	rax, B[rax]

Example #01 - SLH

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



rax is -1 whenever $x \geq A_size$
We can prove security

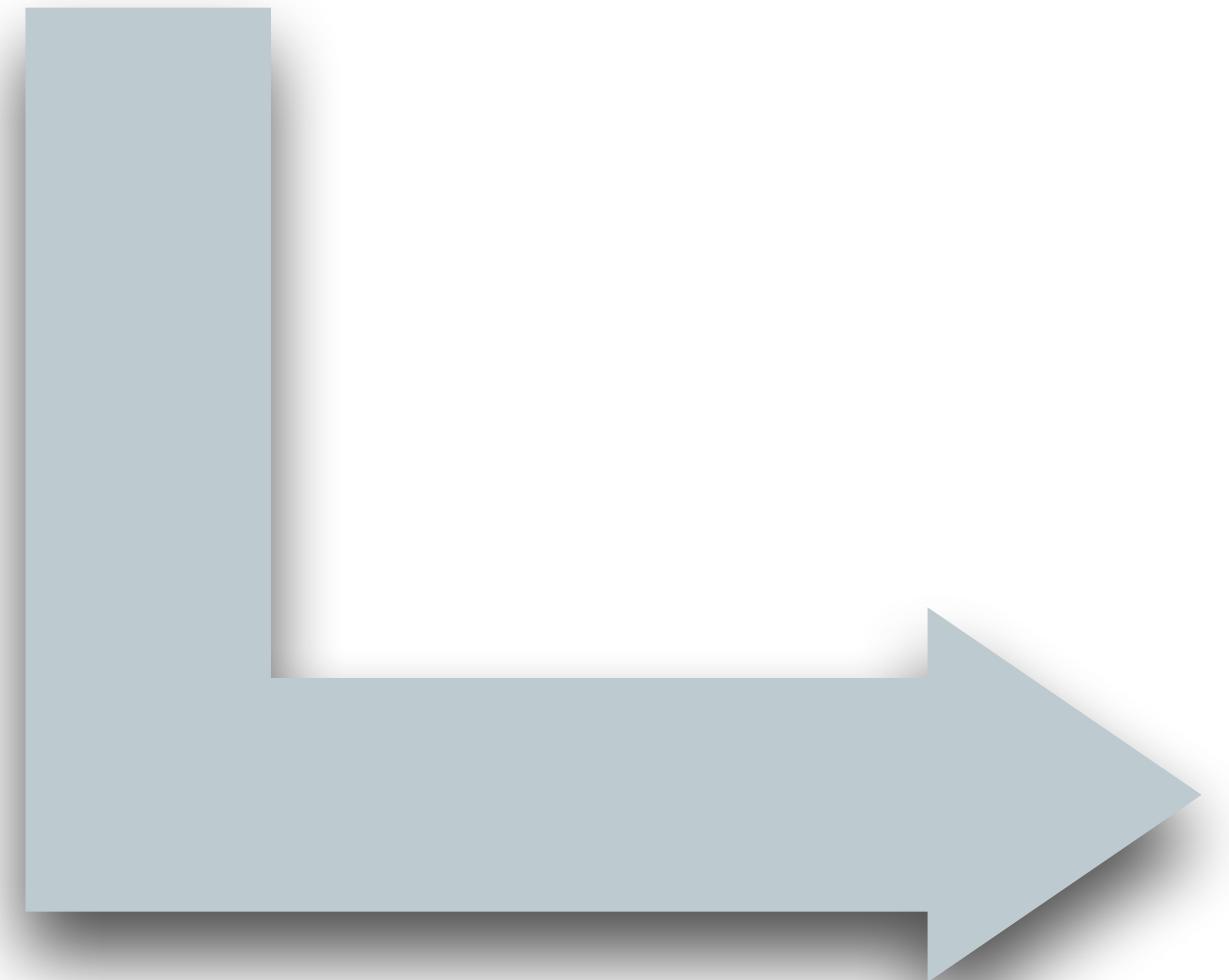
mov	rax, A_size
mov	rcx, x
mov	rdx, 0
cmp	rcx, rax
jae	<i>END</i>
cmoveae	-1, rdx
mov	rax, A[rcx]
shl	rax, 9
or	rax, rdx
mov	rax, B[rax]

Example #10 - SLH

```
if  (x < A_size)
  if  (A[x] == 0)
    y = B[0]
```

Example #10 - SLH

```
if (x < A_size)
    if (A[x]==0)
        y = B[0]
```



mov	rax, A_size
mov	rcx, x
mov	rdx, 0
cmp	rcx, rax
jae	END
cmovae	-1, rdx
mov	rax, A[rcx]
jne	rax, END
cmovne	-1, rdx
mov	rax, [B]

Example #10 - SLH

```
if (x < A_size)
    if (A[x]==0)
        y = B[0]
```

Leaks A[x]==0 via
control-flow
We detect the leak!

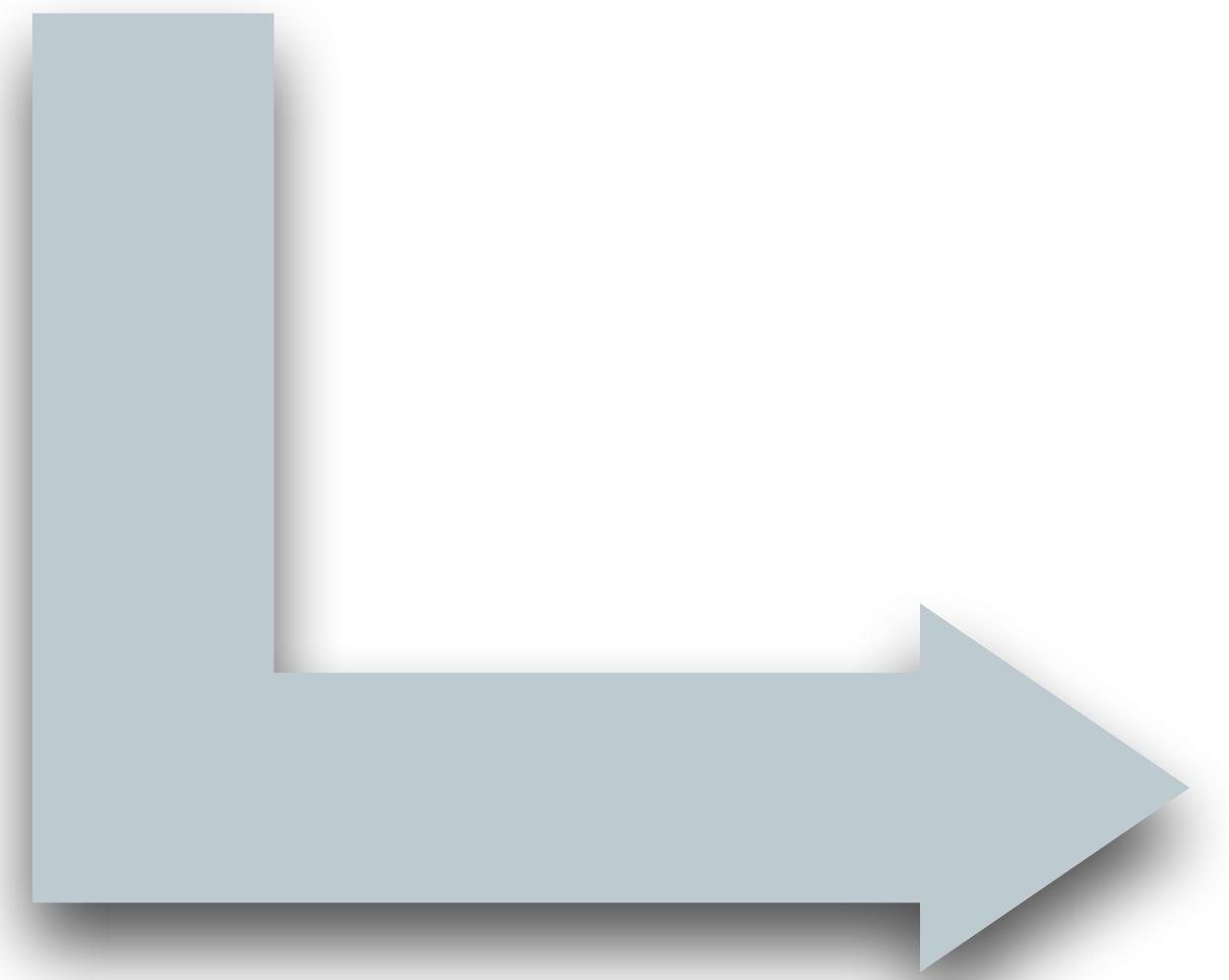
mov	rax, A_size
mov	rcx, x
mov	rdx, 0
cmp	rcx, rax
jae	END
cmovae	-1, rdx
mov	rax, A[rcx]
jne	rax, END
cmovne	-1, rdx
mov	rax, [B]

Example #08 - FEN

```
y = B[A[x < A_size ? (x+1) : 0] * 512]
```

Example #08 - FEN

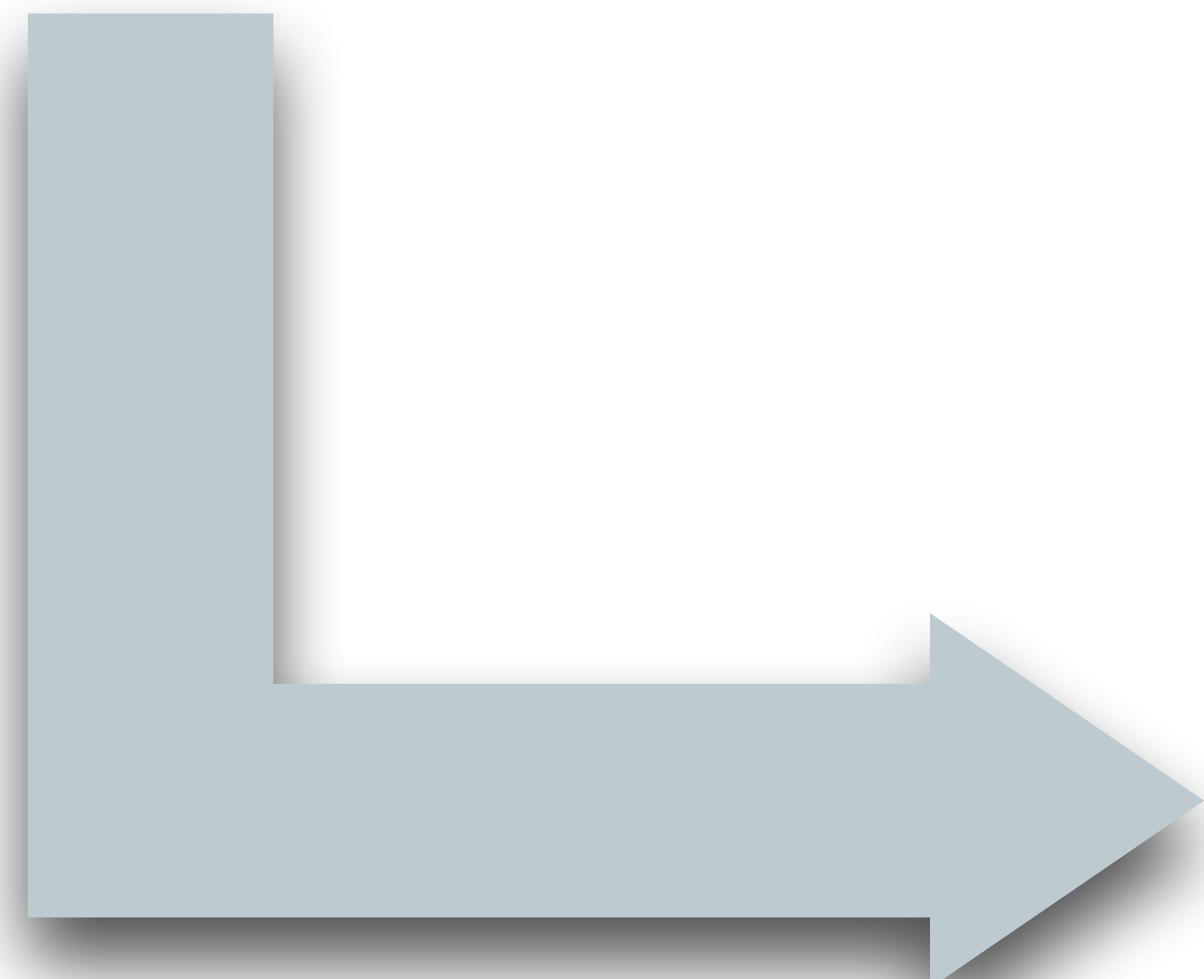
```
y = B[A[x < A_size ? (x+1) : 0] * 512]
```



mov	rax, A_size
mov	rcx, x
lea	rcx, [rcx+1]
xor	rdx, rdx
cmp	rcx, rax
cmoveae	rdx, rcx
mov	rax, A[rdx]
shl	rax, 9
lfence	
mov	rax, B[rax]

Example #08 - FEN

```
y = B[A[x < A_size ? (x+1) : 0] * 512]
```



`lfence` is unnecessary

mov	rax, A_size
mov	rcx, x
lea	rcx, [rcx+1]
xor	rdx, rdx
cmp	rcx, rax
cmovae	rdx, rcx
mov	rax, A[rdx]
shl	rax, 9
lfence	
mov	rax, B[rax]