Spectector: Principled detection of speculative information flows

Marco Guarnieri
IMDEA Software Institute

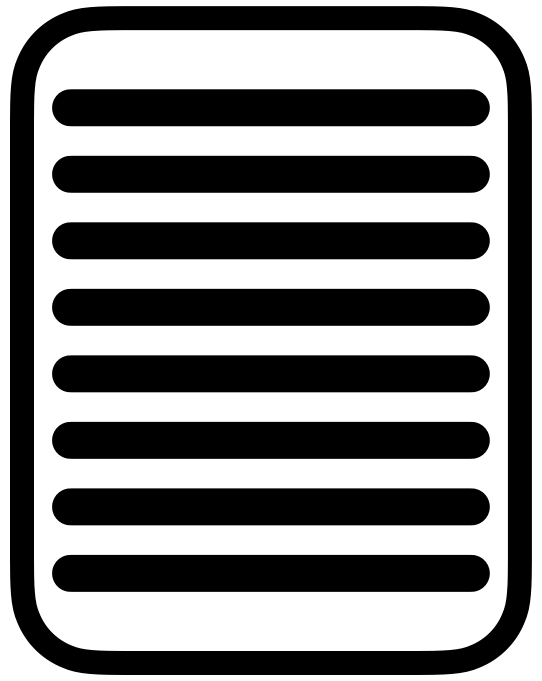
Joint work with
José F. Morales, Andrés Sánchez @ IMDEA Software Institute
Boris Köpf @ Microsoft Research
Jan Reineke @ Saarland University

SPECTRE

Exploits ***speculative execution***

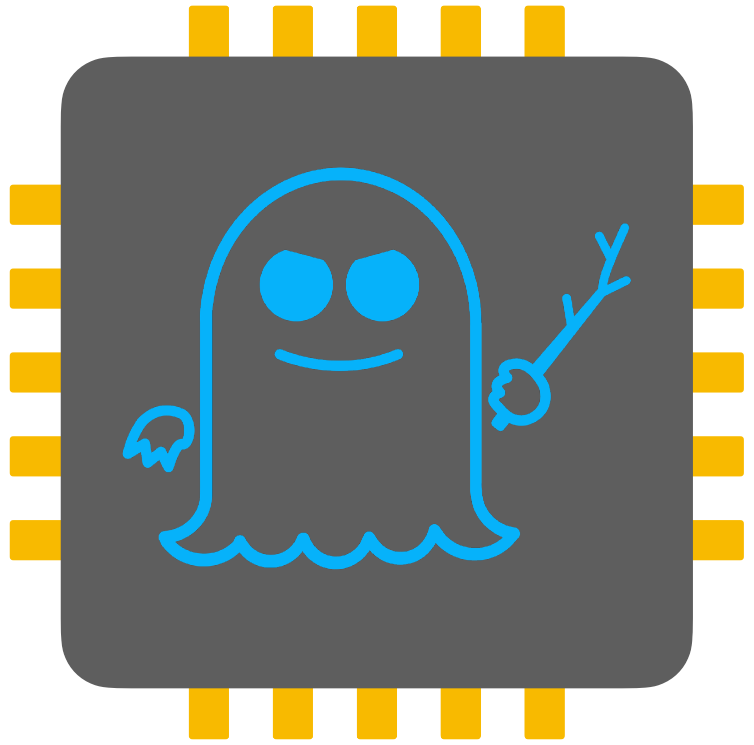
Almost ***all*** modern ***CPUs*** are ***affected***

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



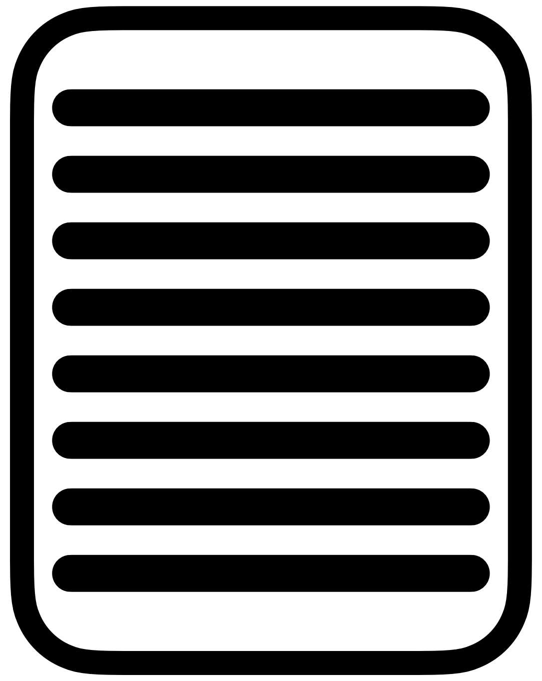
Program

+



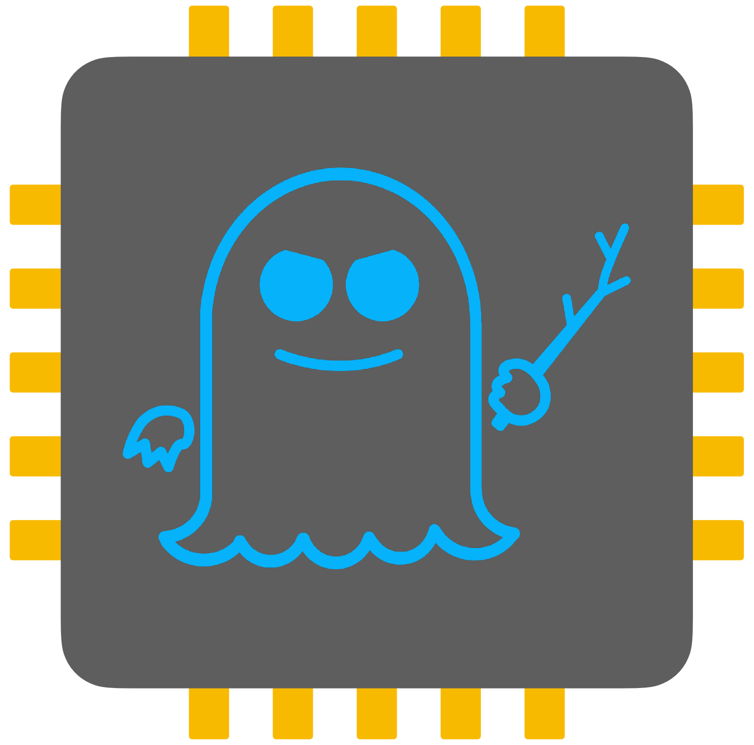
CPU with ***speculative execution***

= Secure?



Program

+



CPU with ***speculative execution***

= Secure?

In this talk..

1. ***Semantic notion*** of ***security*** against ***speculative execution attacks***

2. Analysis to ***detect vulnerability*** or ***prove security***

Speculative execution attacks 101

Speculative execution + branch prediction

Size of array **A**

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

Speculative execution + branch prediction

Size of array **A**

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

Speculative execution + branch prediction

Size of array **A**

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

Branch predictor

Speculative execution + branch prediction

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

Size of array **A**

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

HELP

Branch predictor

Speculative execution + branch prediction

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

Size of array **A**

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

Branch predictor

Speculative execution + branch prediction

Size of array **A**

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

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

Branch predictor

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

- ✓ Architectural (ISA) state
- ✗ Microarchitectural state

Spectre V1

Spectre V1

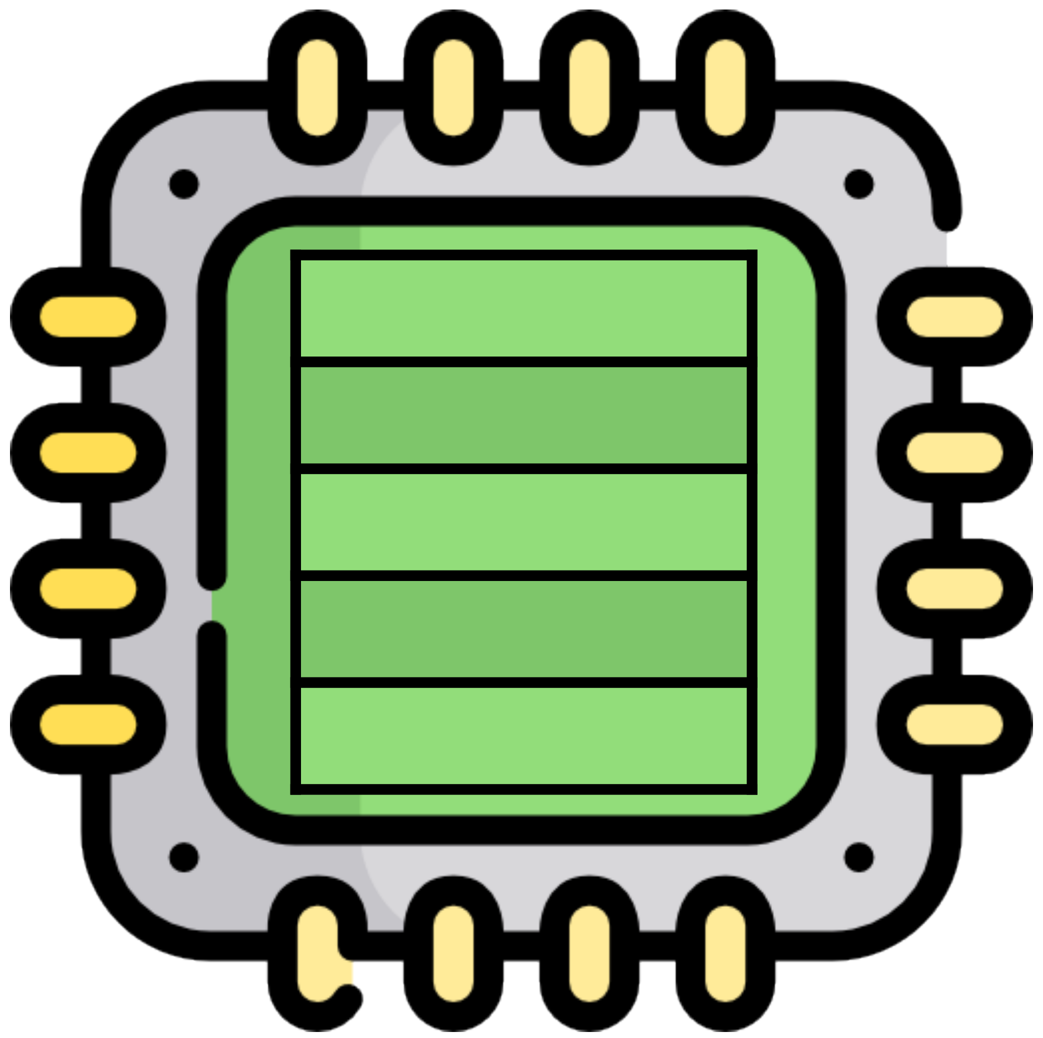
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

Spectre V1

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

Spectre V1

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

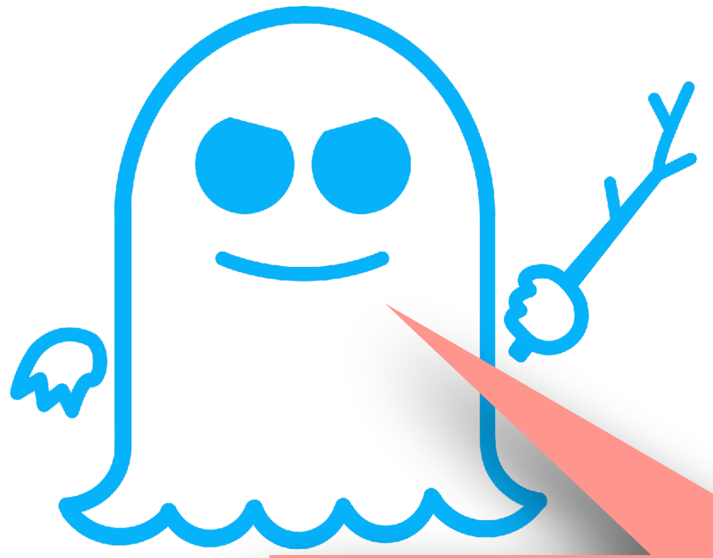


Spectre V1

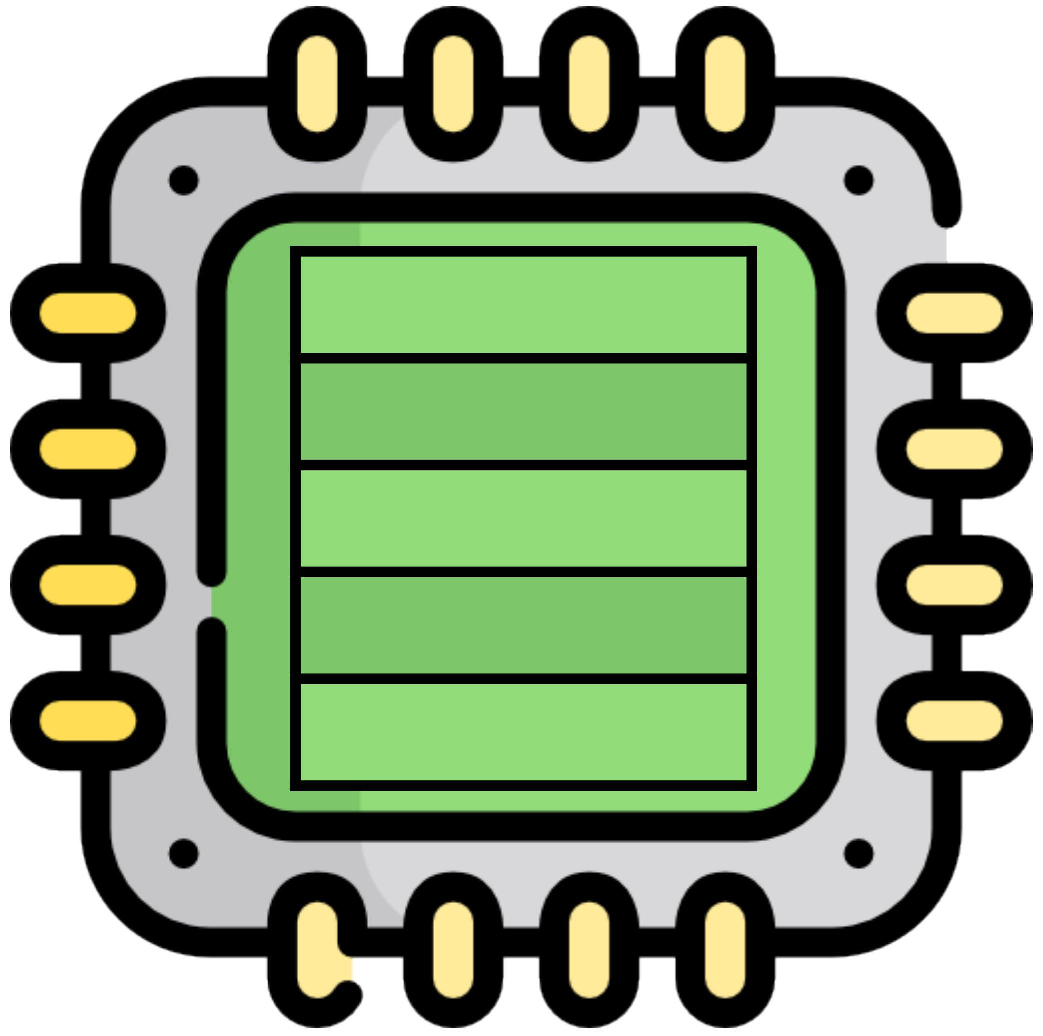
A_size=16

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

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



What is in A[128]?



Spectre V1

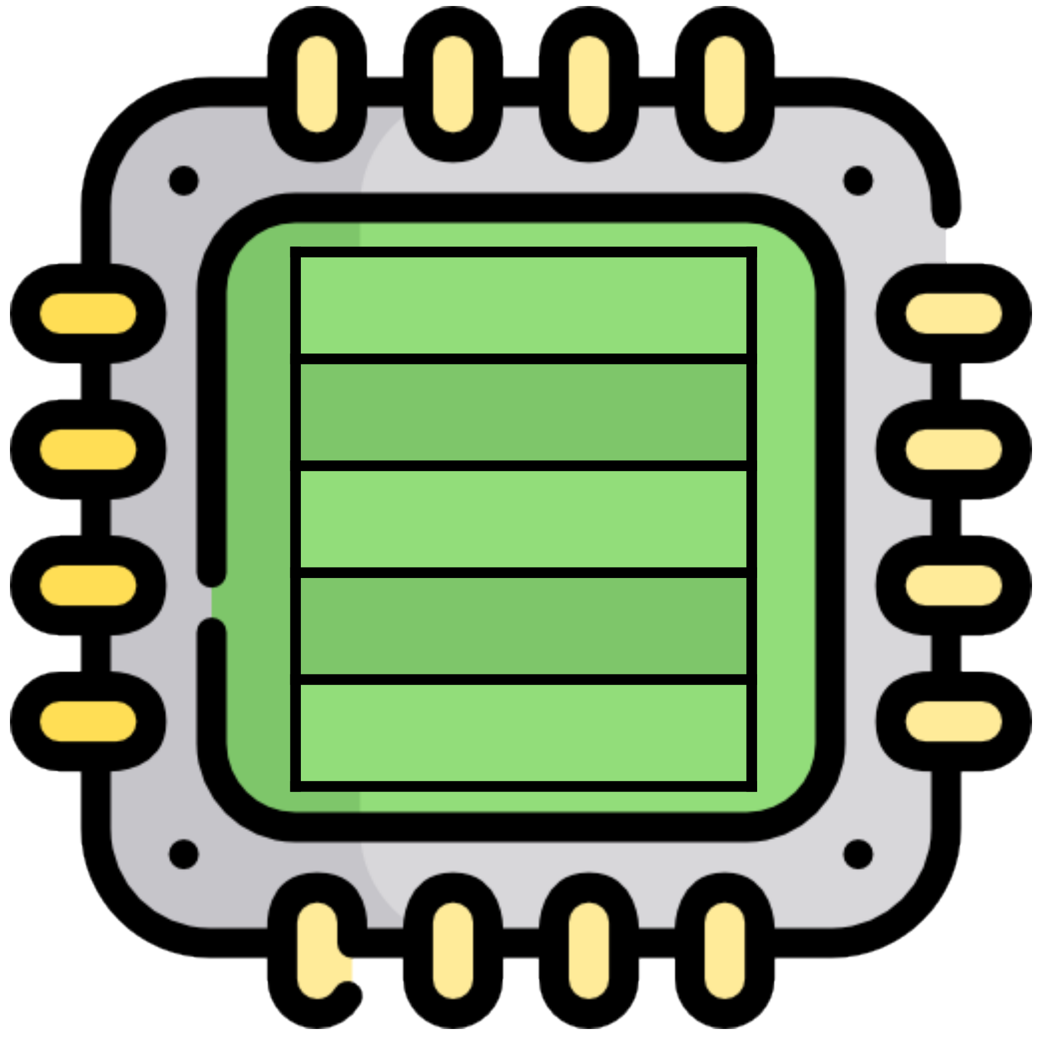
A_size=16

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

What is in A[128]?

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

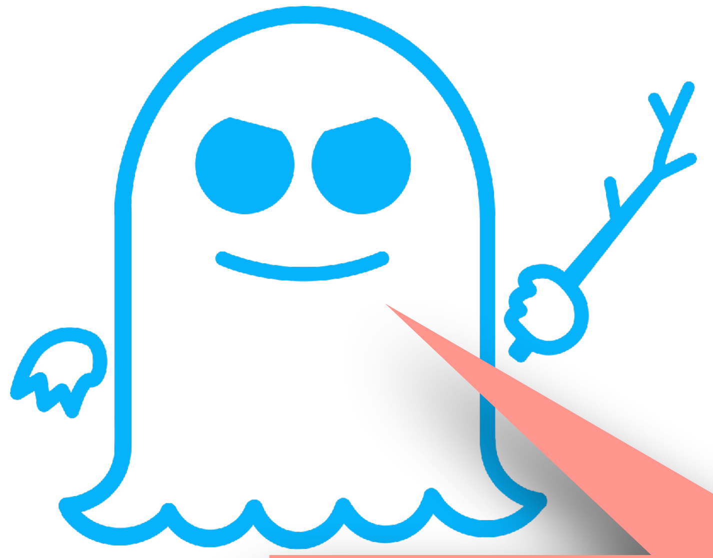
1) Train branch predictor



Spectre V1

A_size=16

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

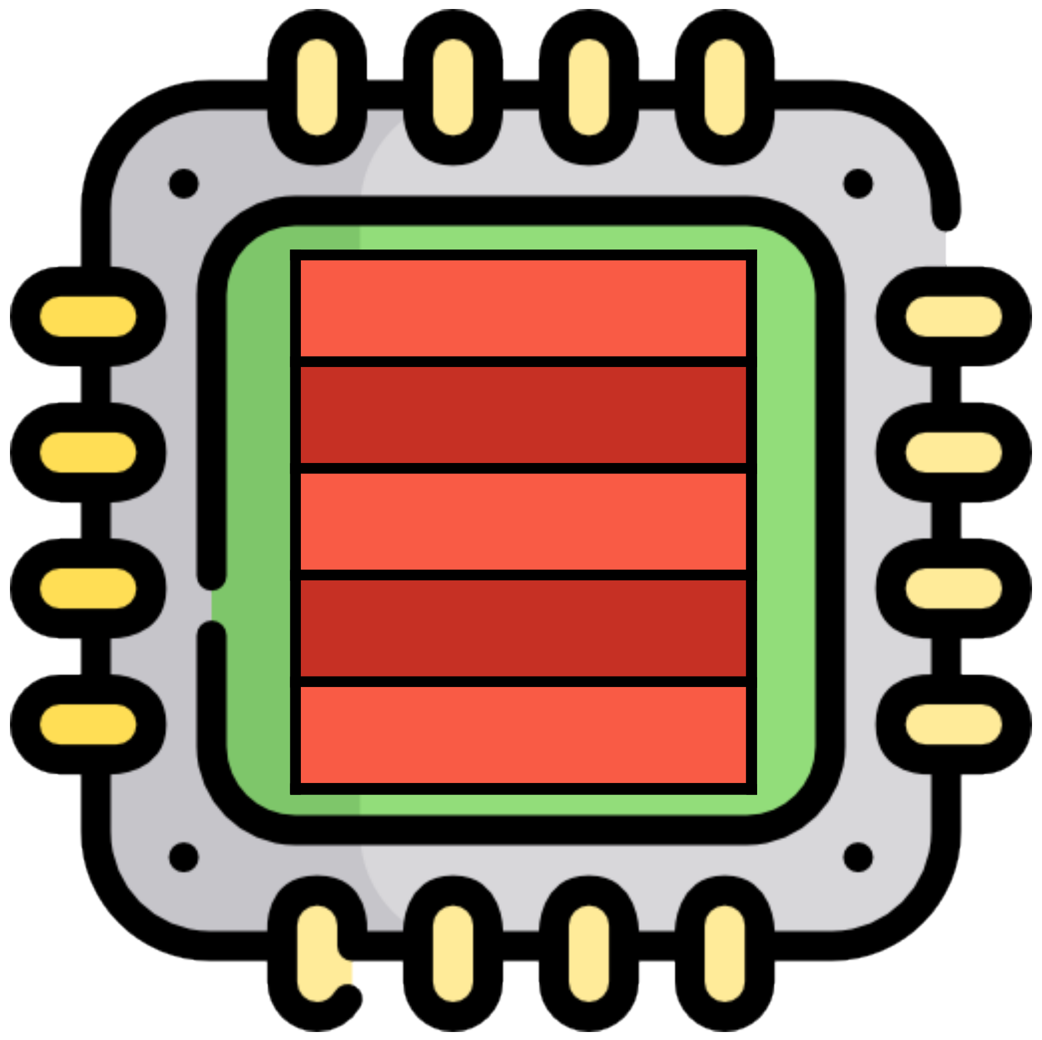


What is in A[128]?

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

1) Train branch predictor

2) Prepare cache

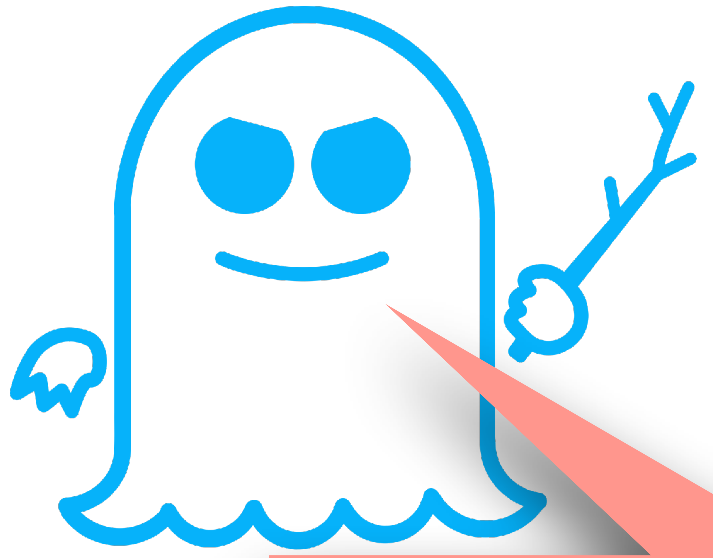


Spectre V1

A_size=16

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

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

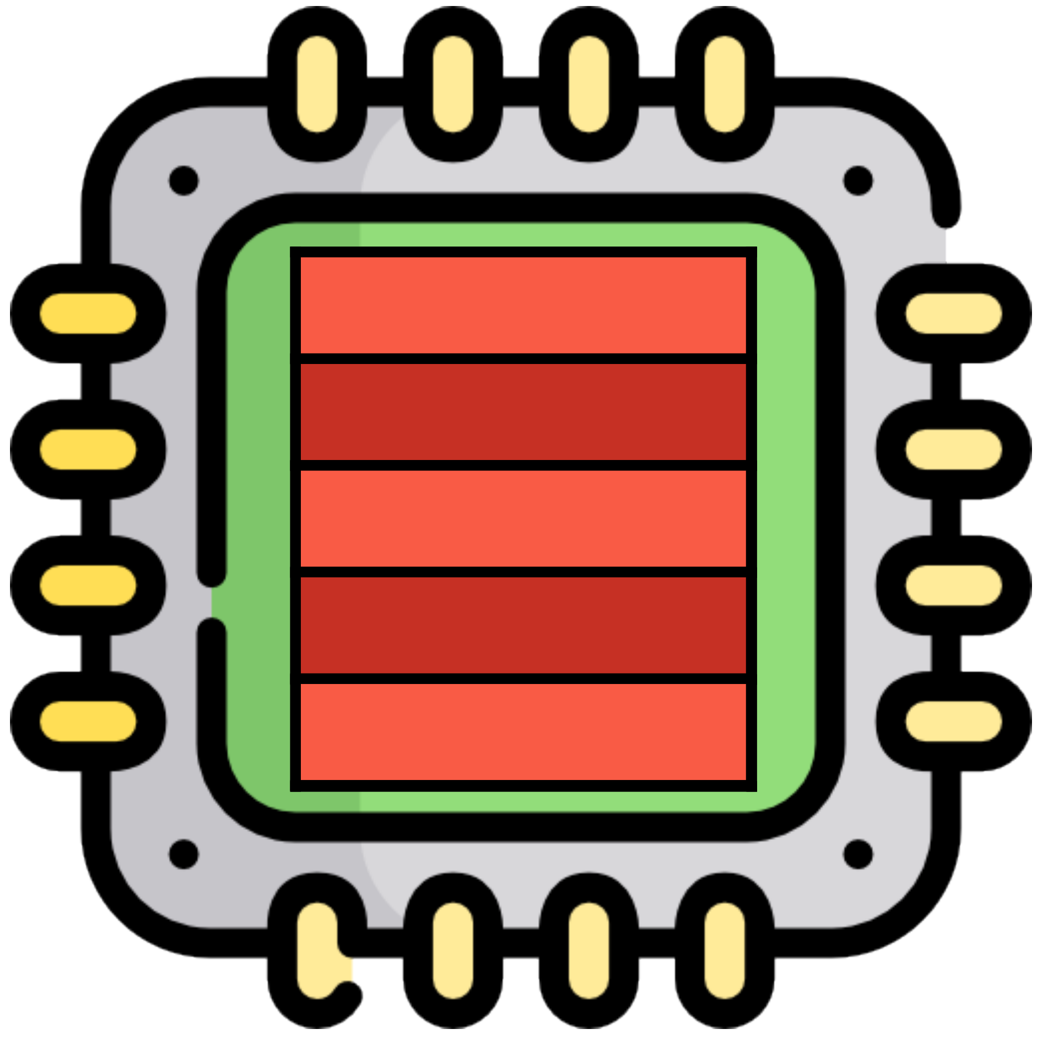


What is in A[128]?

1) Train branch predictor

2) Prepare cache

3) Run with x = 128

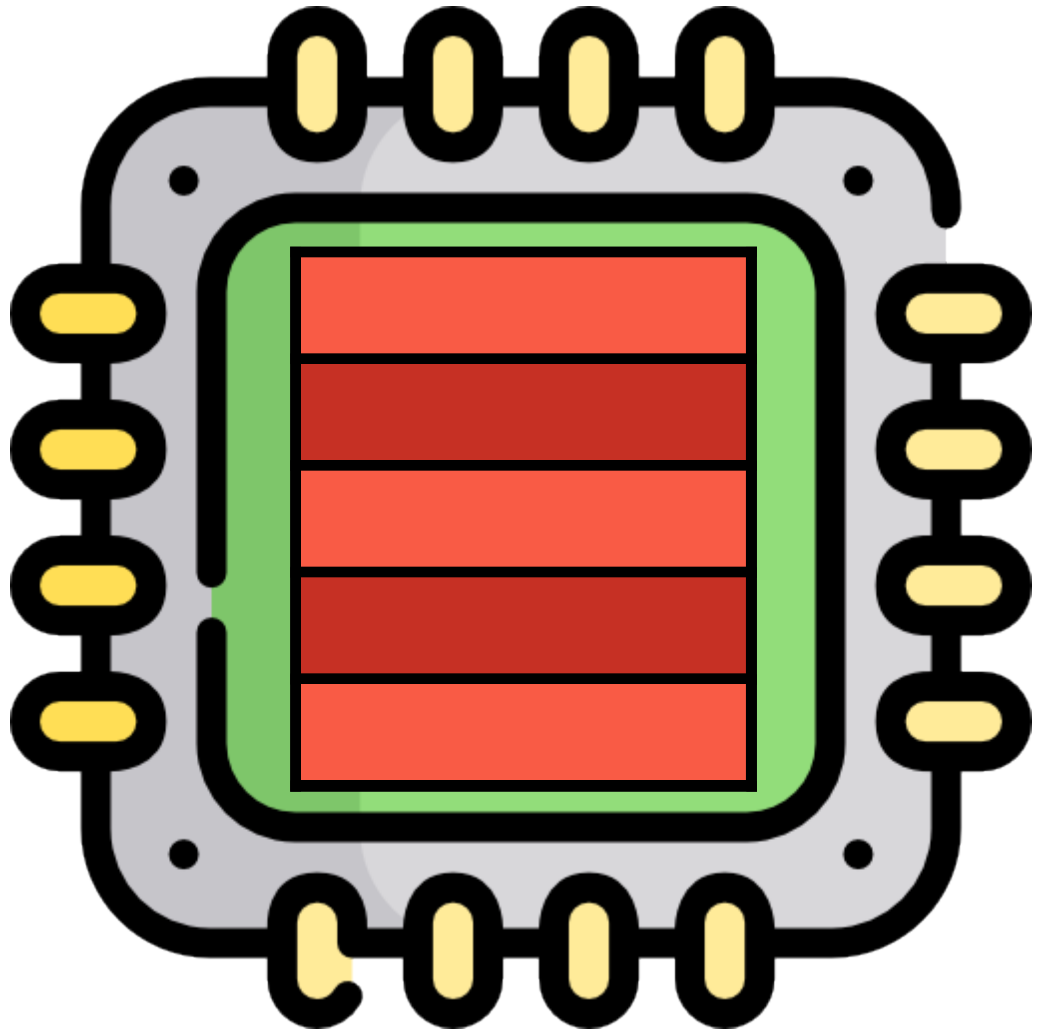


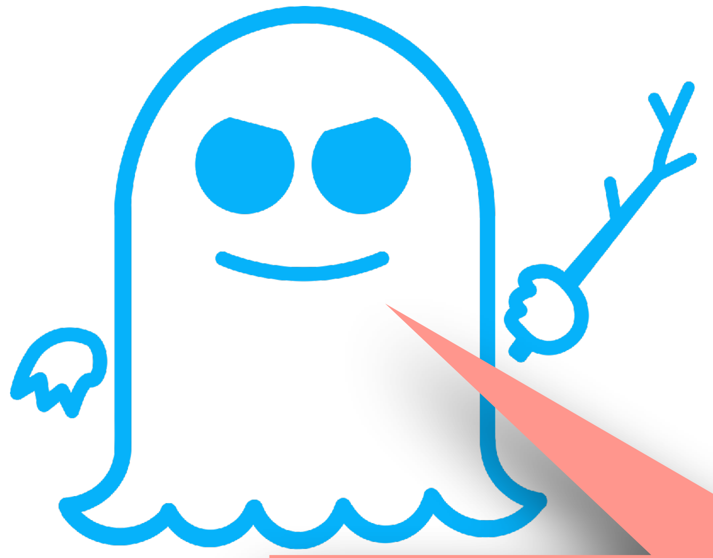
Spectre V1

A_size=16

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

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





What is in A[128]?

1) **Train branch predictor**

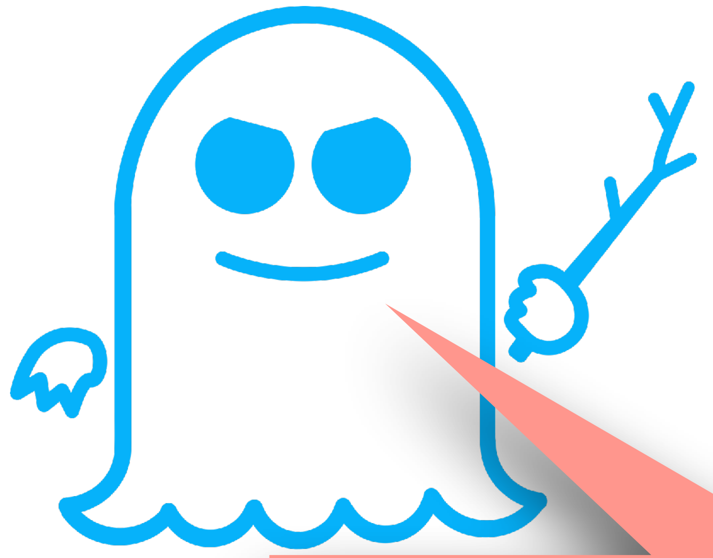
2) **Prepare cache**

3) **Run with** x = 128

Spectre V1

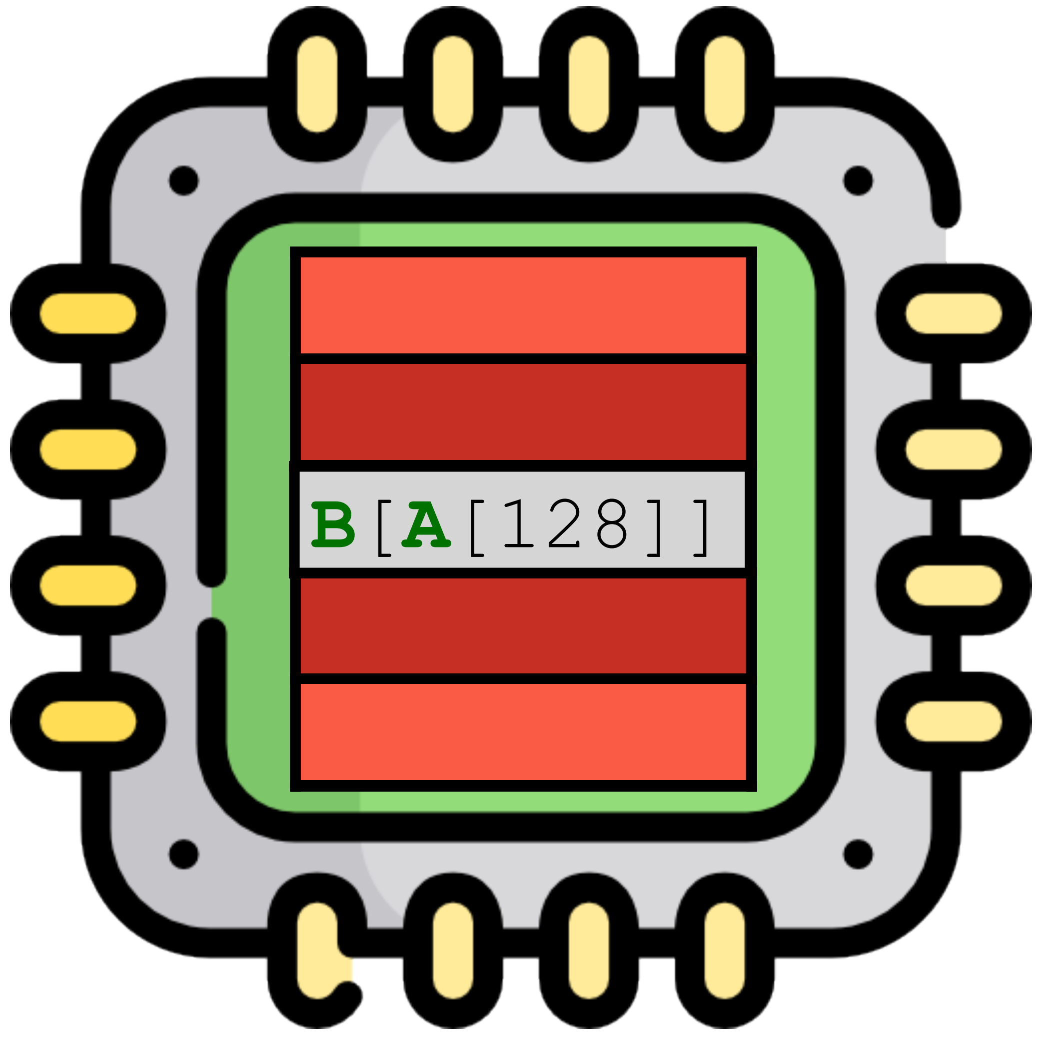
A_size=16

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



What is in A[128]?

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



1) Train branch predictor

2) Prepare cache

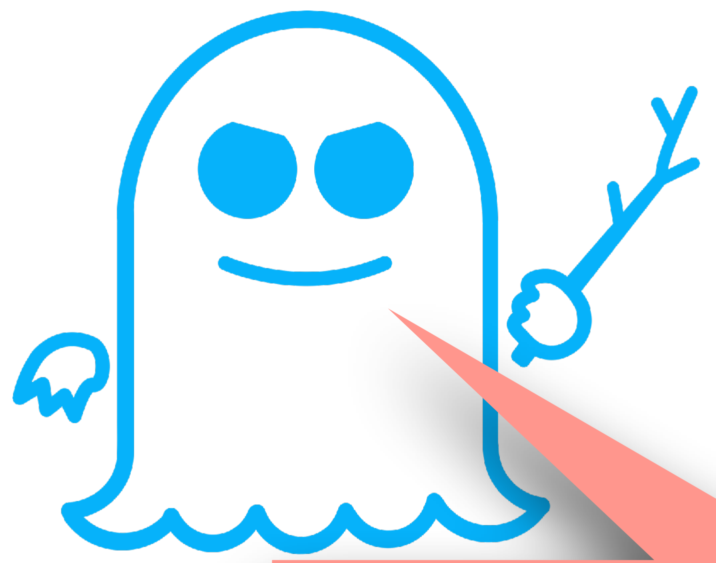
3) Run with x = 128

Spectre V1

A_size=16

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

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



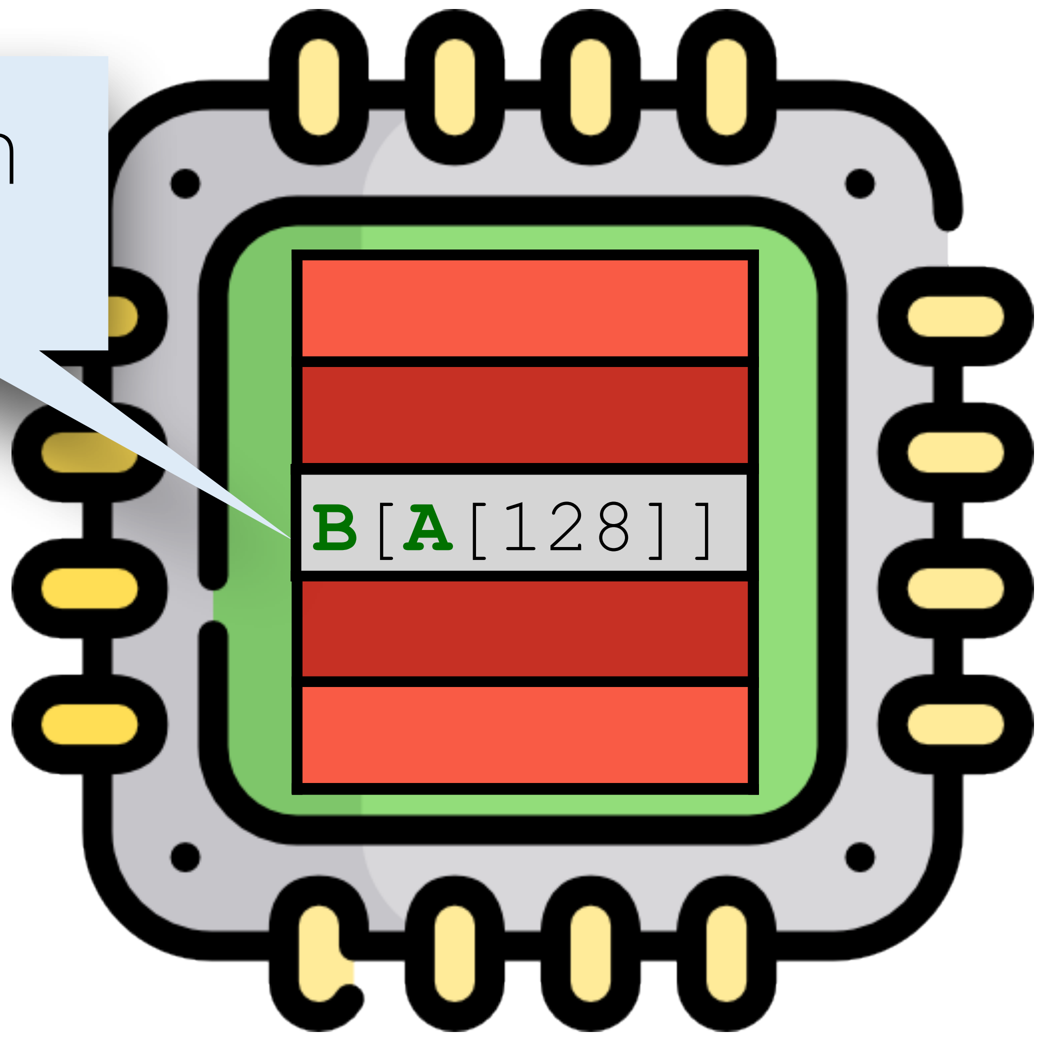
What is in A[128]?

1) **Train branch predictor**

2) **Prepare cache**

3) **Run with** x = 128

Depends on A[128]

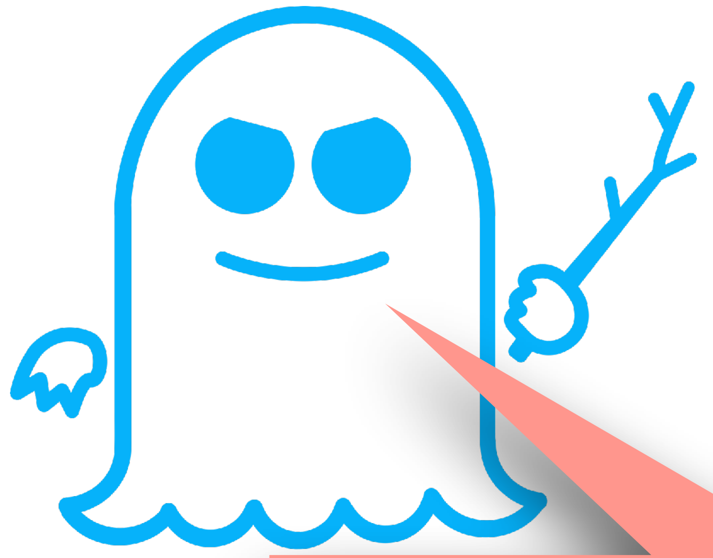


Spectre V1

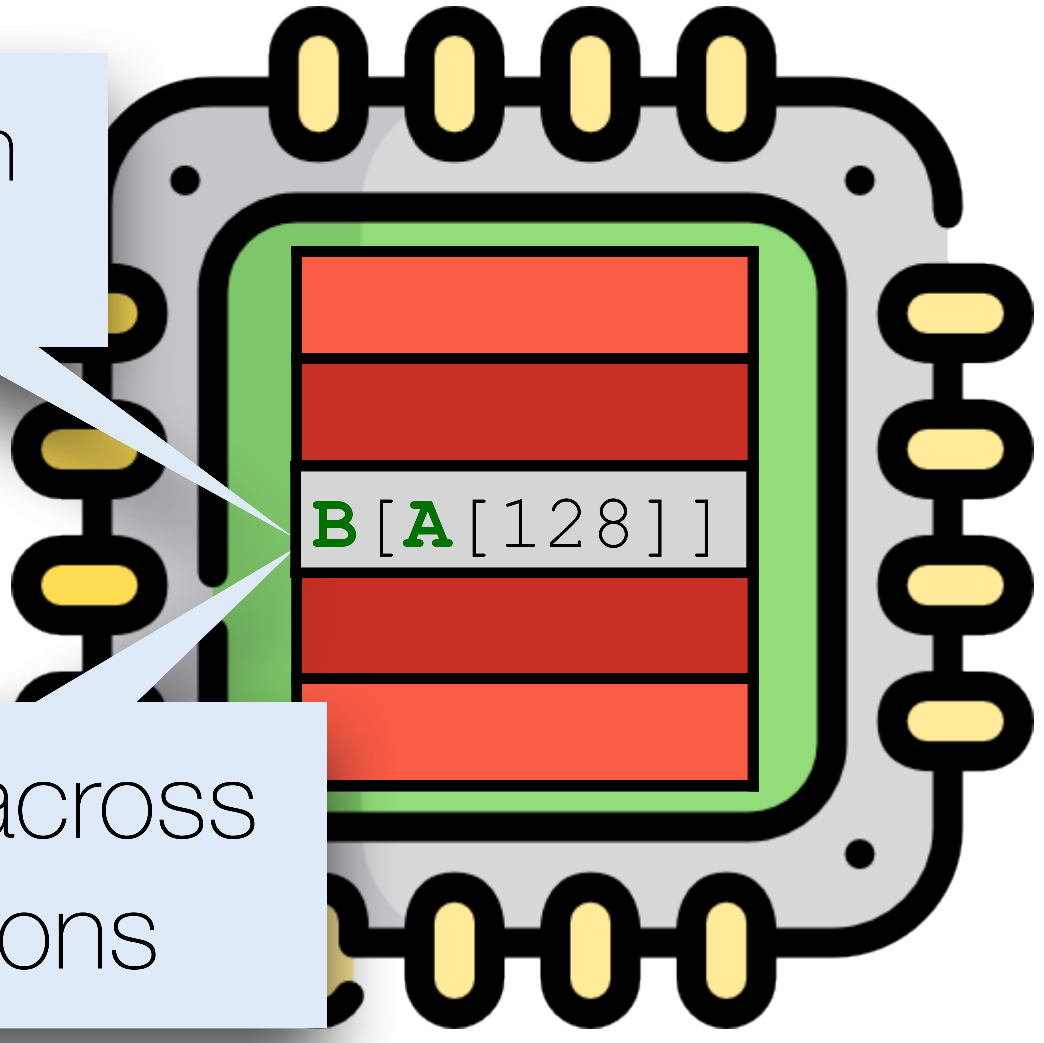
A_size=16

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]?



Depends on A[128]

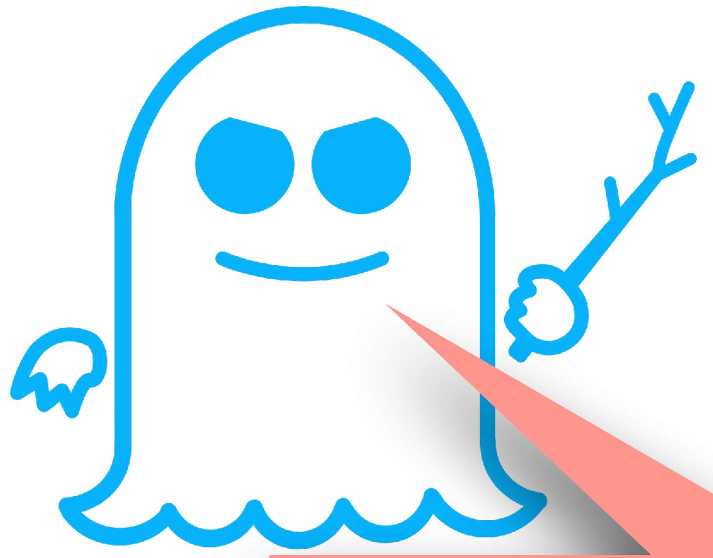
Persistent across speculations

1) Train branch predictor

2) Prepare cache

3) Run with x = 128

Spectre V1

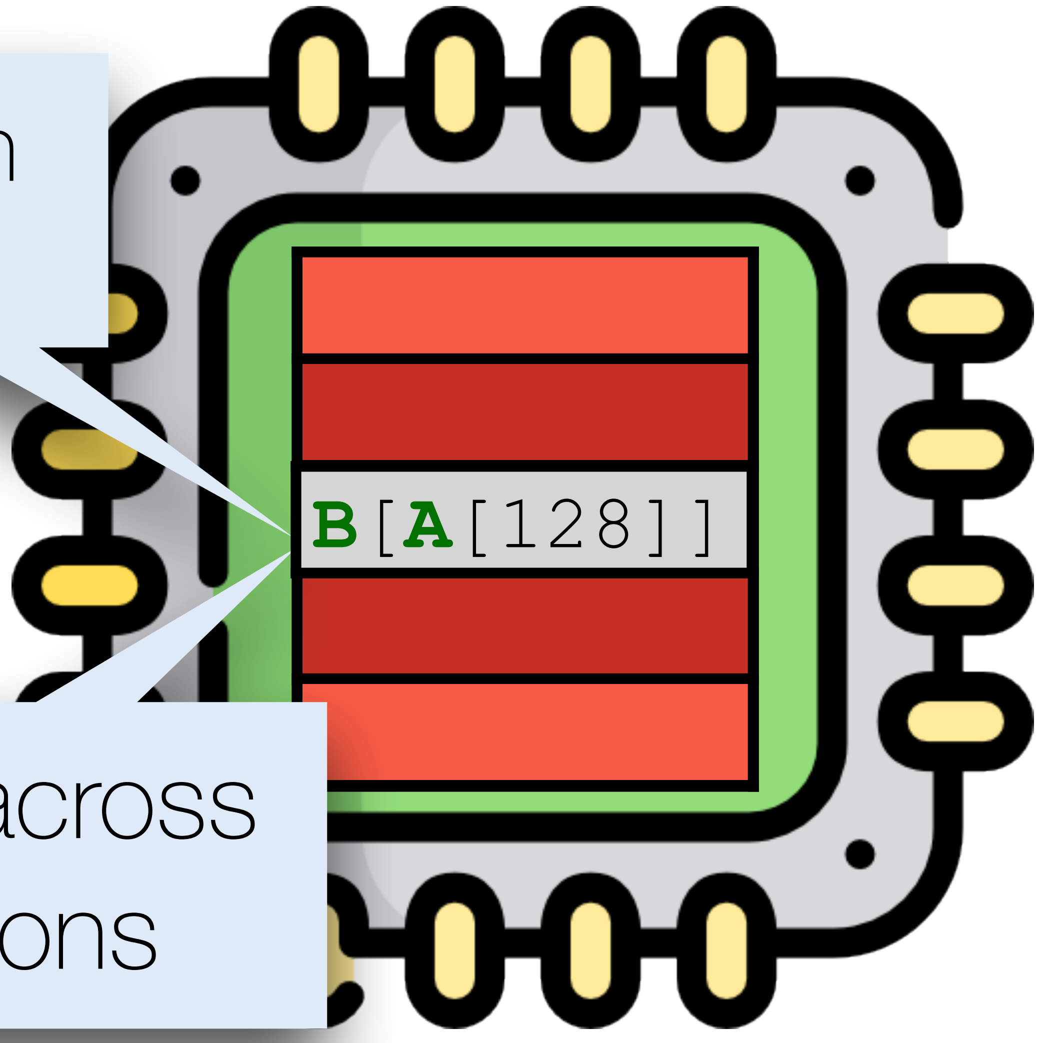


What is in A[128]?

A_size=16

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

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



Depends on A[128]

Persistent across speculations

1) Train branch predictor

2) Prepare cache

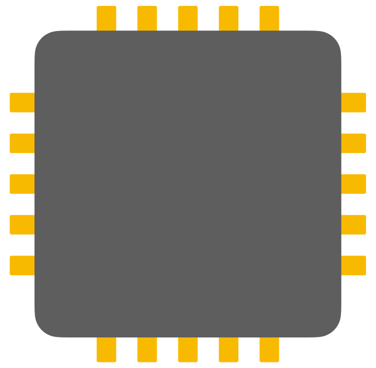
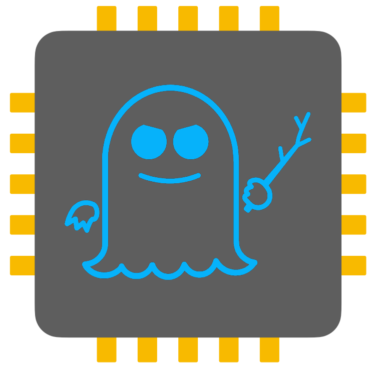
3) Run with x = 128

4) Extract from cache

Speculative non-interference

Speculative non-interference

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

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

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

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

How to capture leakage?

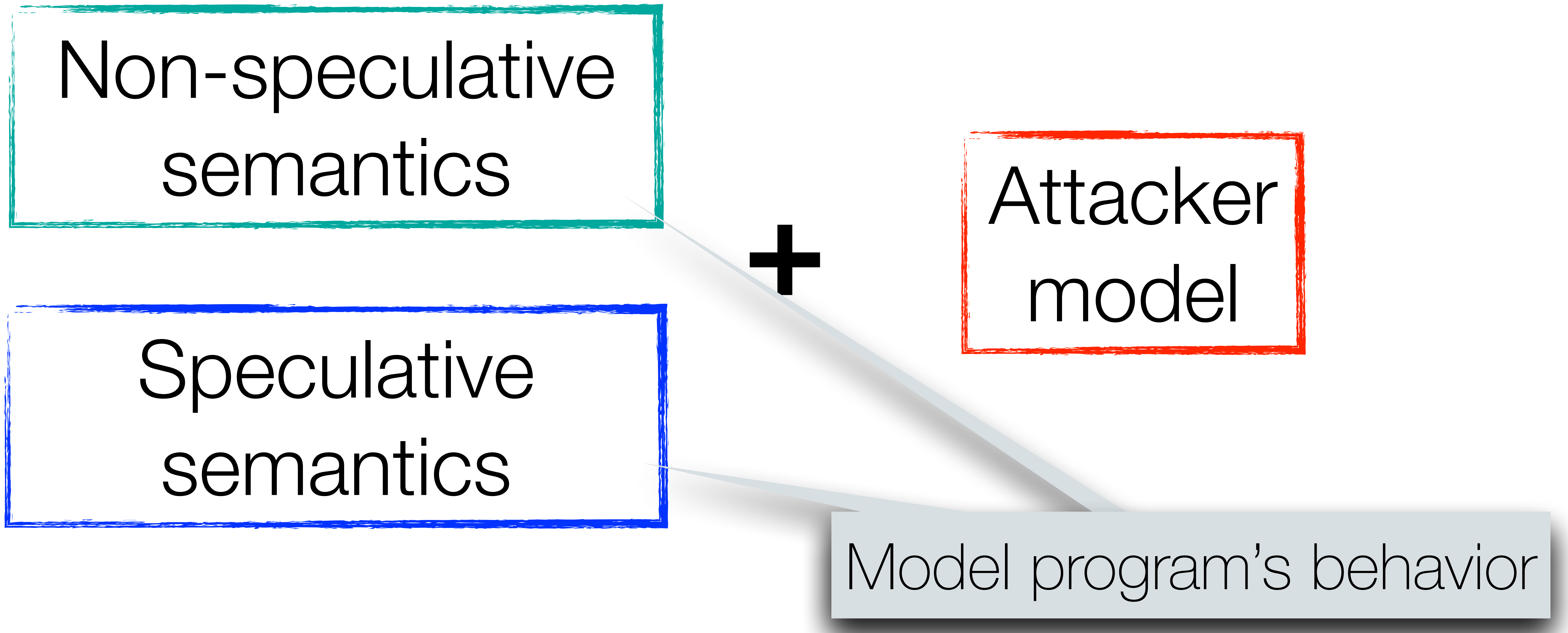
Non-speculative semantics

Speculative semantics

+

Attacker model

How to capture leakage?



How to capture leakage?

Capture attacker's observational power

Non-speculative semantics

Speculative semantics

+

Attacker model

Model program's behavior

How to capture leakage?

Capture attacker's observational power

Non-speculative semantics

\+

Attacker model

Speculative semantics

Model program's behavior

How to capture leakage?

Standard ***in-order*** semantics

Non-speculative semantics

Speculative semantics

\+

Attacker model

Capture attacker's observational power

Model program's behavior

How to capture leakage?

Capture attacker's observational power

Non-speculative semantics

Speculative semantics

+

Attacker model

Model program's behavior

How to capture leakage?

Capture attacker's observational power

Prediction Oracle O :
branch prediction + length of speculative window

+

Attacker model

Speculative semantics

Model program's behavior

How to capture leakage?

Capture attacker's observational power

Prediction Oracle ***O*** :
branch prediction + length of speculative window

Speculative semantics

Starts ***speculative transactions*** upon branch instructions

- ***Committed*** upon correct speculation
- ***Rolled-back*** upon misprediction

How to capture leakage?

Capture attacker's observational power

Non-speculative semantics

Speculative semantics

+

Attacker model

Model program's behavior

How to capture leakage?

Capture attacker's observational power

Non-speculative semantics

Speculative semantics

+

Attacker model

Mo

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

How to capture leakage?

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

How to capture leakage?

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

`x > A_size`

`x < A_size` predicted as satisfied

How to capture leakage?

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

x > A_size

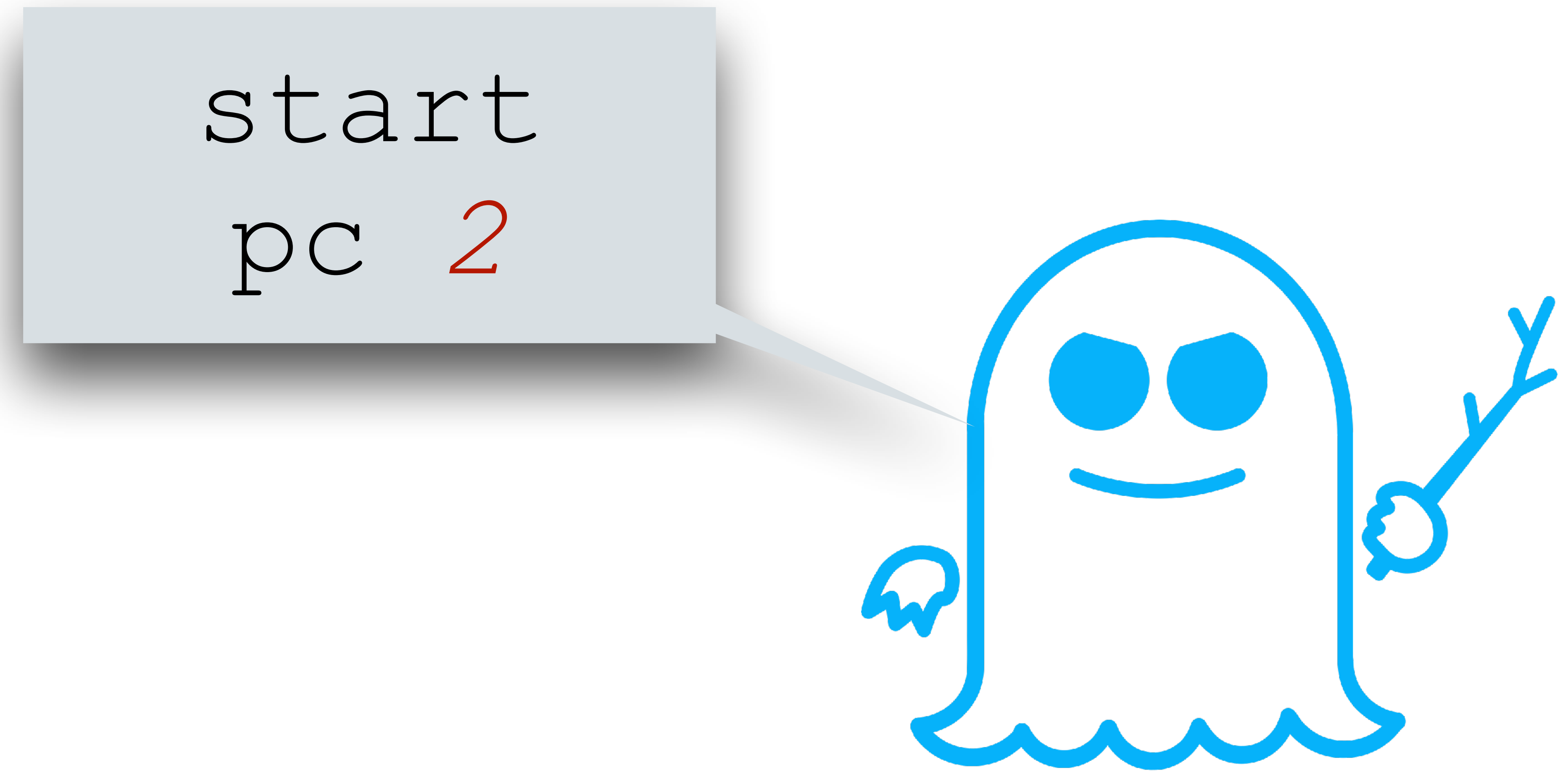
x < A_size predicted as satisfied

How to capture leakage?

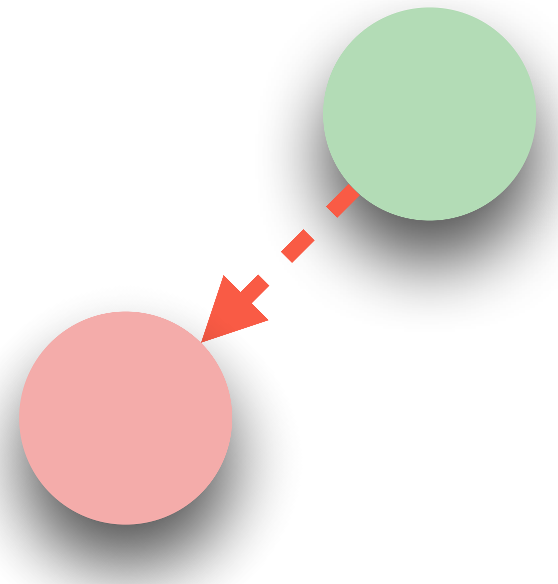
```
1. if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4. end
```

$x > A\_size$

$x < A\_size$ predicted as satisfied



How to capture leakage?

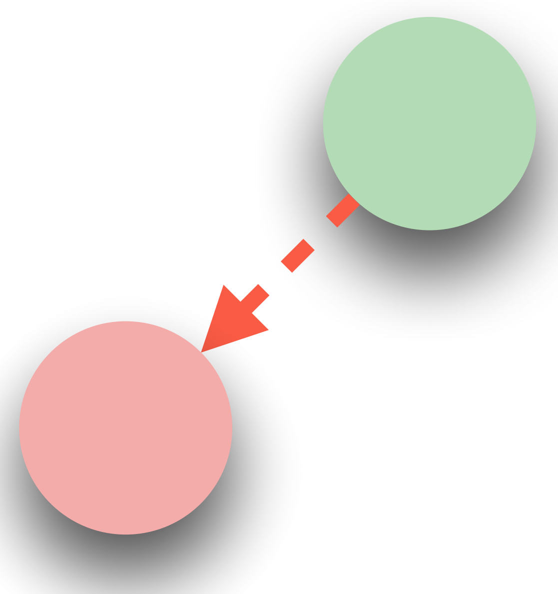


```
if (x < A_size)
    y = A[x]
    z = B[y]
end
```

$x > A\_size$

$x < A\_size$ predicted as satisfied

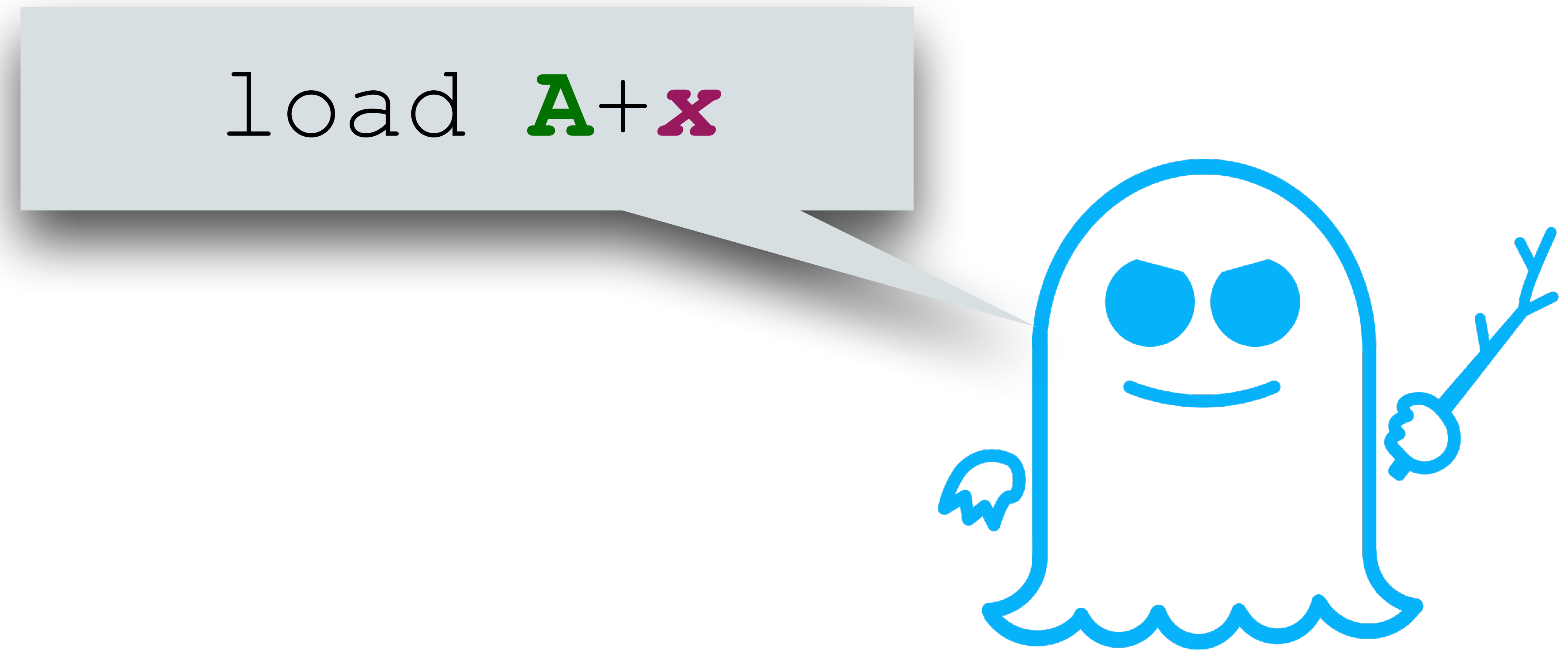
How to capture leakage?



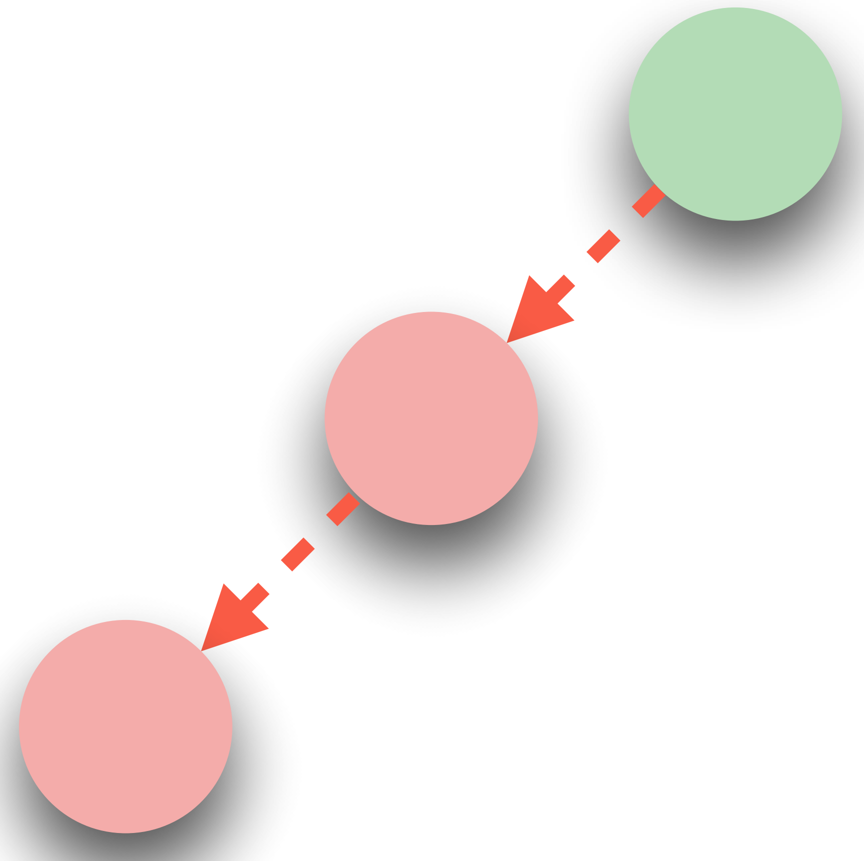
```
1. if (x < A_size)
2.     y = A[x]
3.     z = B[y]
4. end
```

x > A_size

x < A_size predicted as satisfied



How to capture leakage?

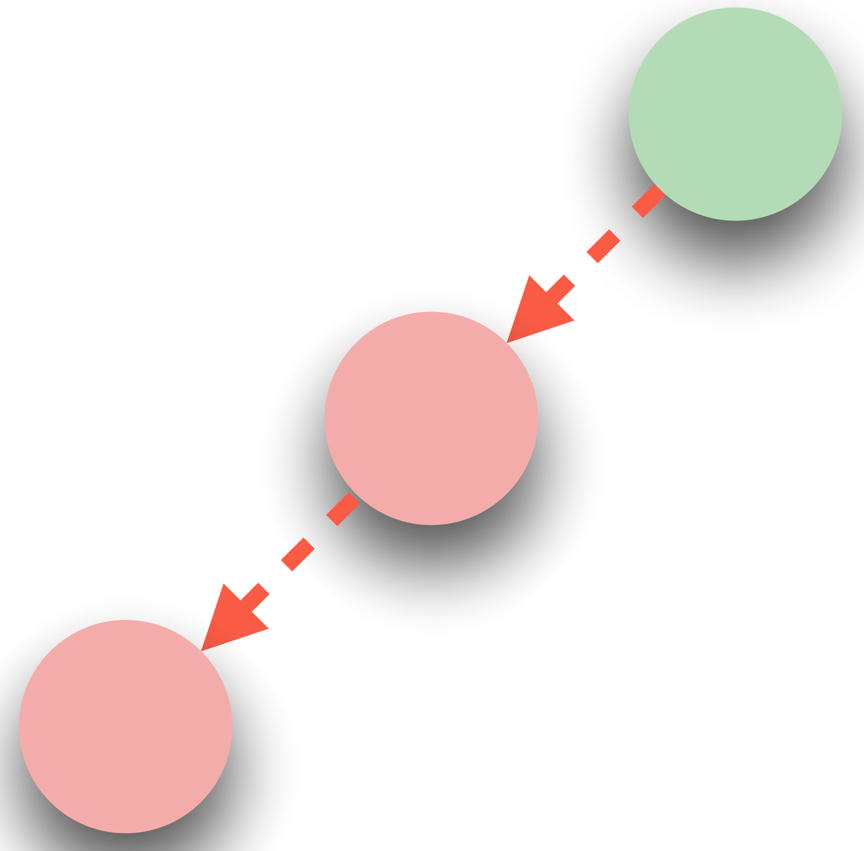


```
if (x < A_size)
    y = A[x]
    z = B[y]
end
```

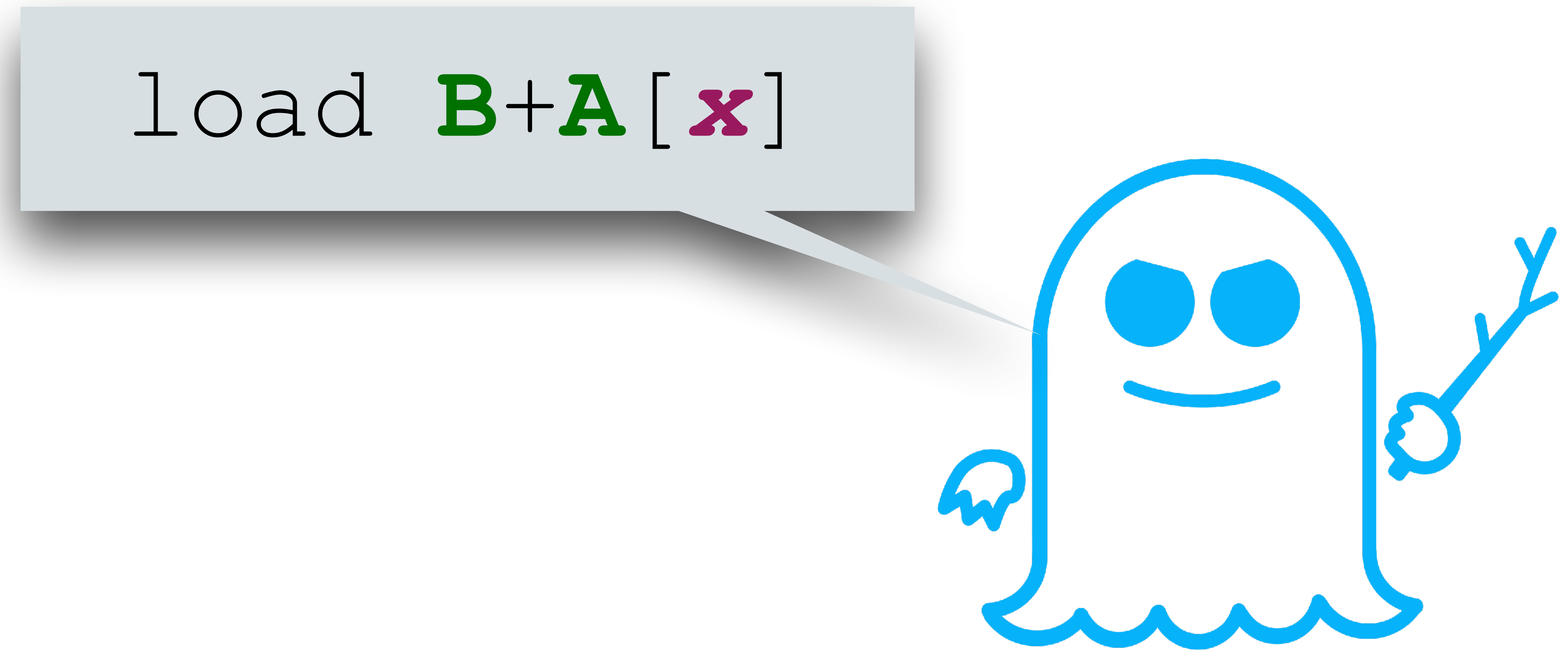
$x > A\_size$

$x < A\_size$ predicted as satisfied

How to capture leakage?



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

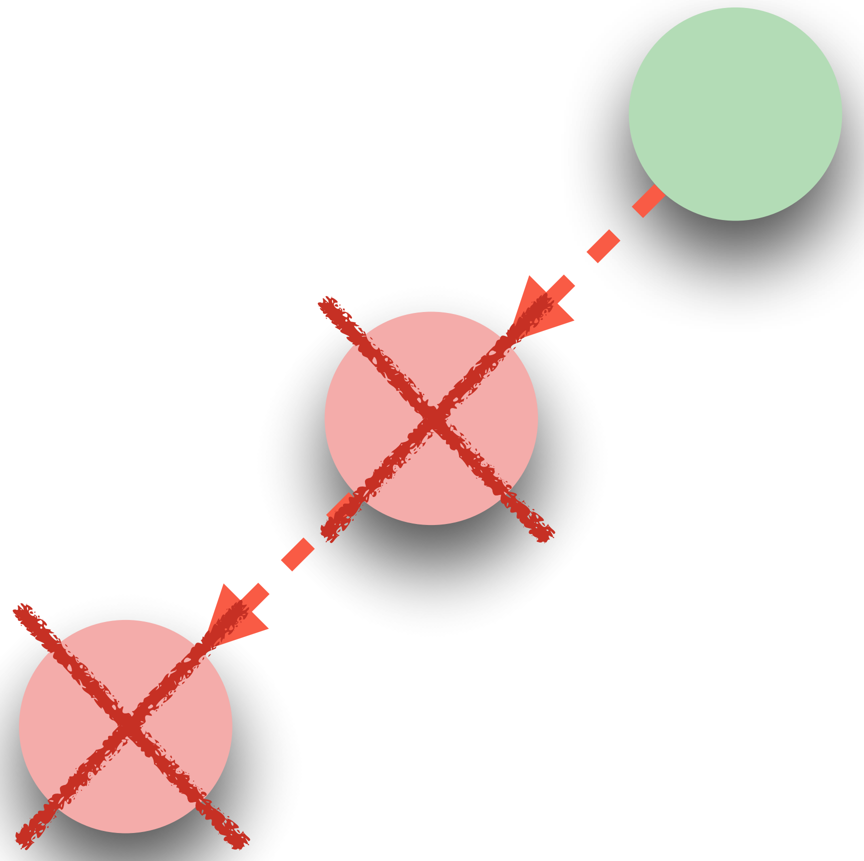


$x > A\_size$

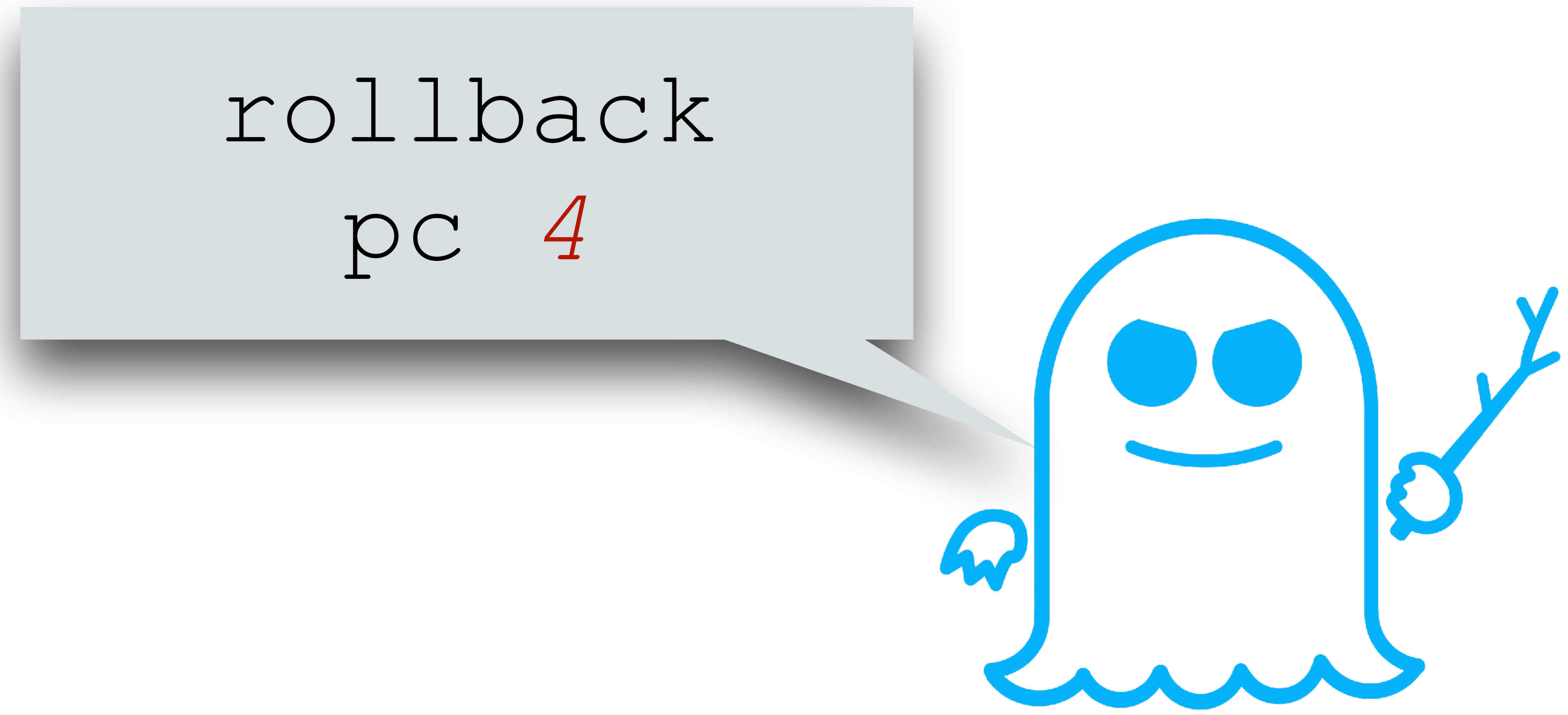
$x < A\_size$ predicted as satisfied

How to capture leakage?

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

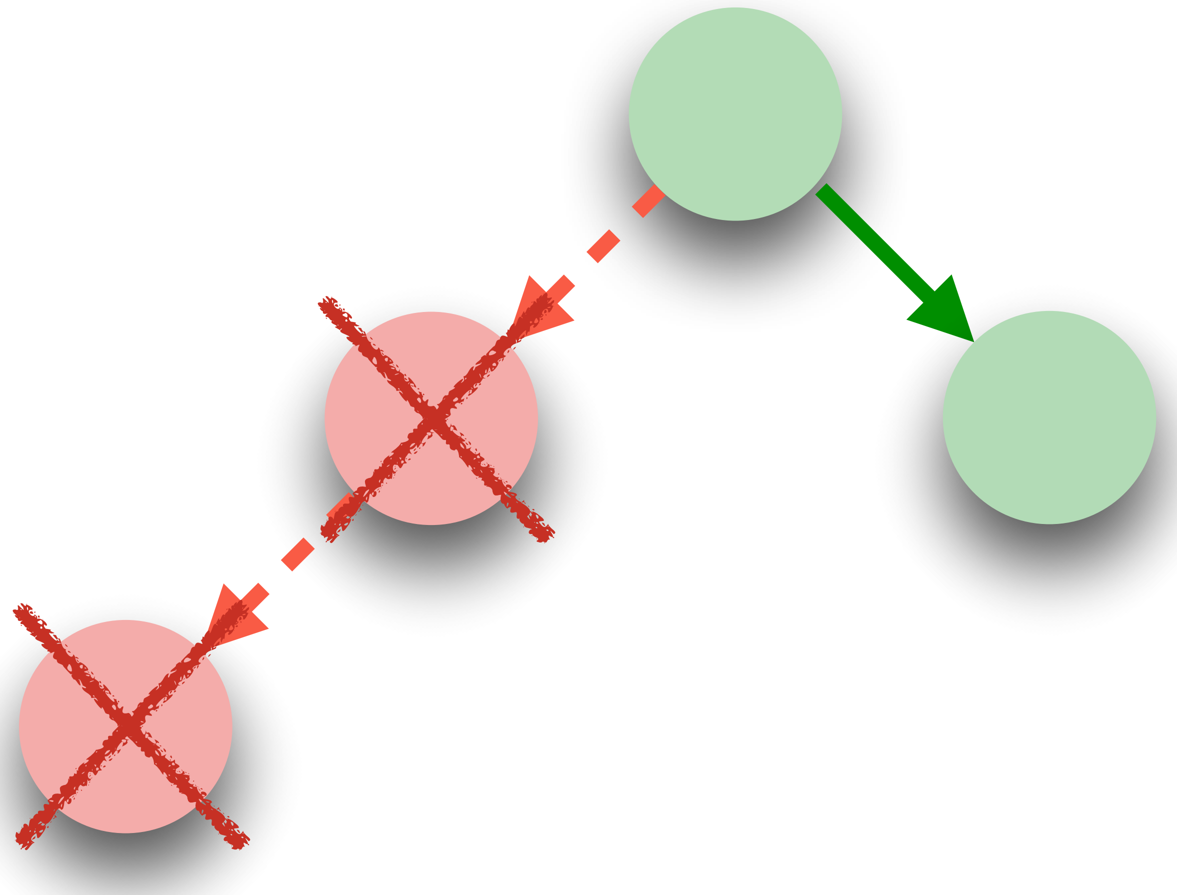


$x > A\_size$



$x < A\_size$ predicted as satisfied

How to capture leakage?



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

x > A_size

x < A_size predicted as satisfied

Speculative non-interference

Formally!

Speculative non-interference

Formally!

Program **P** is **speculatively non-interferent** for prediction oracle ***O*** if

Speculative non-interference

Formally!

Program **P** is **speculatively non-interferent** for prediction oracle ***O*** if

For all program states ***s*** and ***s'***:

Speculative non-interference

Formally!

Program **P** is **speculatively non-interferent** for prediction oracle ***O*** if

For all program states ***s*** and ***s'***:

$$\mathbf{P}_{\texttt{non-spec}}(\boldsymbol{s}) = \mathbf{P}_{\texttt{non-spec}}(\boldsymbol{s'})$$

Speculative non-interference

Formally!

Program $\mathbf{P}$ is **speculatively non-interferent** for prediction oracle $\boldsymbol{O}$ if

For all program states $\boldsymbol{s}$ and $\boldsymbol{s'}$:

$$\mathbf{P}_{\texttt{non-spec}}(\boldsymbol{s}) = \mathbf{P}_{\texttt{non-spec}}(\boldsymbol{s'})$$
$$\Rightarrow \mathbf{P}_{\texttt{spec}}(\boldsymbol{s},\boldsymbol{O}) = \mathbf{P}_{\texttt{spec}}(\boldsymbol{s'},\boldsymbol{O})$$

Speculative non-interference

Formally!

Program $\mathbf{P}$ is **speculatively non-interferent** for prediction oracle ***O*** if

For all program states s and s':

$$\mathbf{P}_{\texttt{non-spec}}(s) = \mathbf{P}_{\texttt{non-spec}}(s') \Rightarrow \mathbf{P}_{\texttt{spec}}(s, O) = \mathbf{P}_{\texttt{spec}}(s', O)$$

See paper for: reasoning about ***arbitrary prediction oracles***

Speculative non-interference

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

Speculative non-interference

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

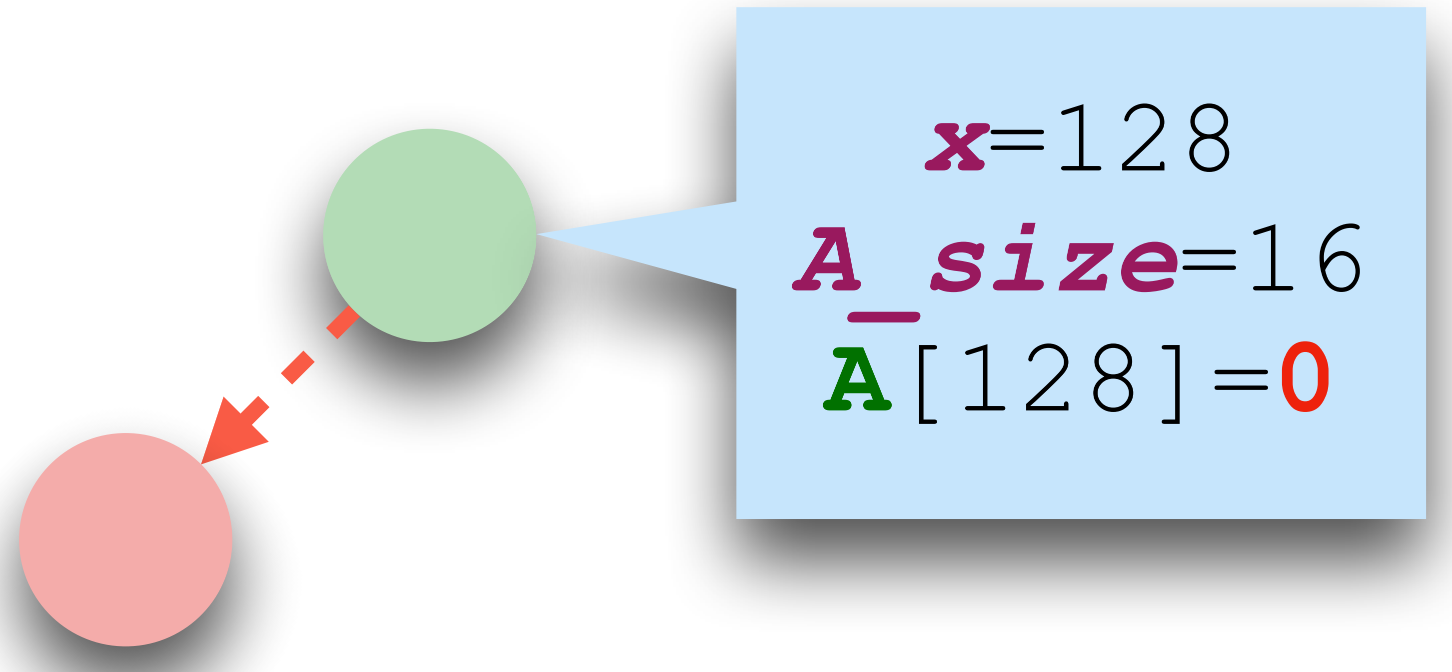
x < A_size predicted as satisfied

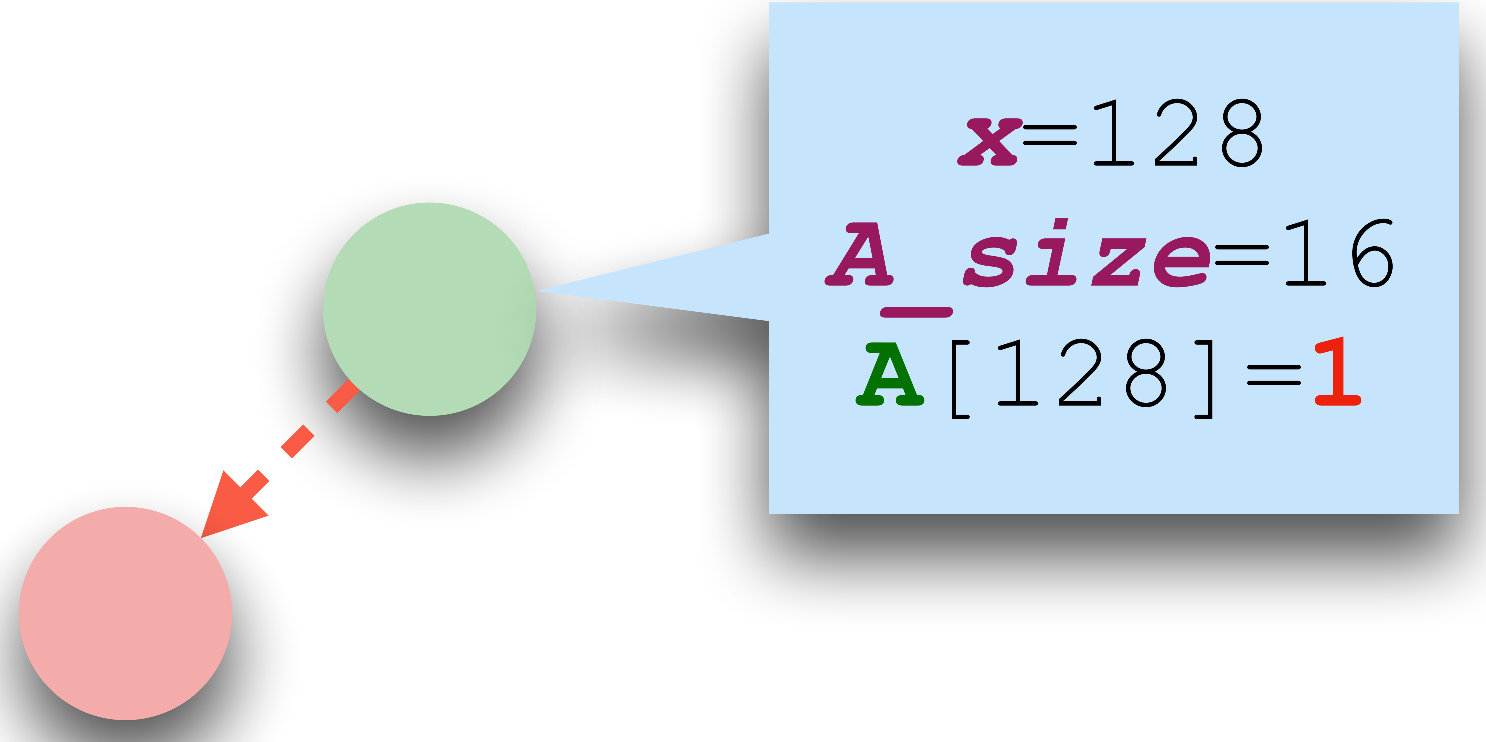
x=128
A_size=16
A[128]=0

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

Speculative non-interference

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



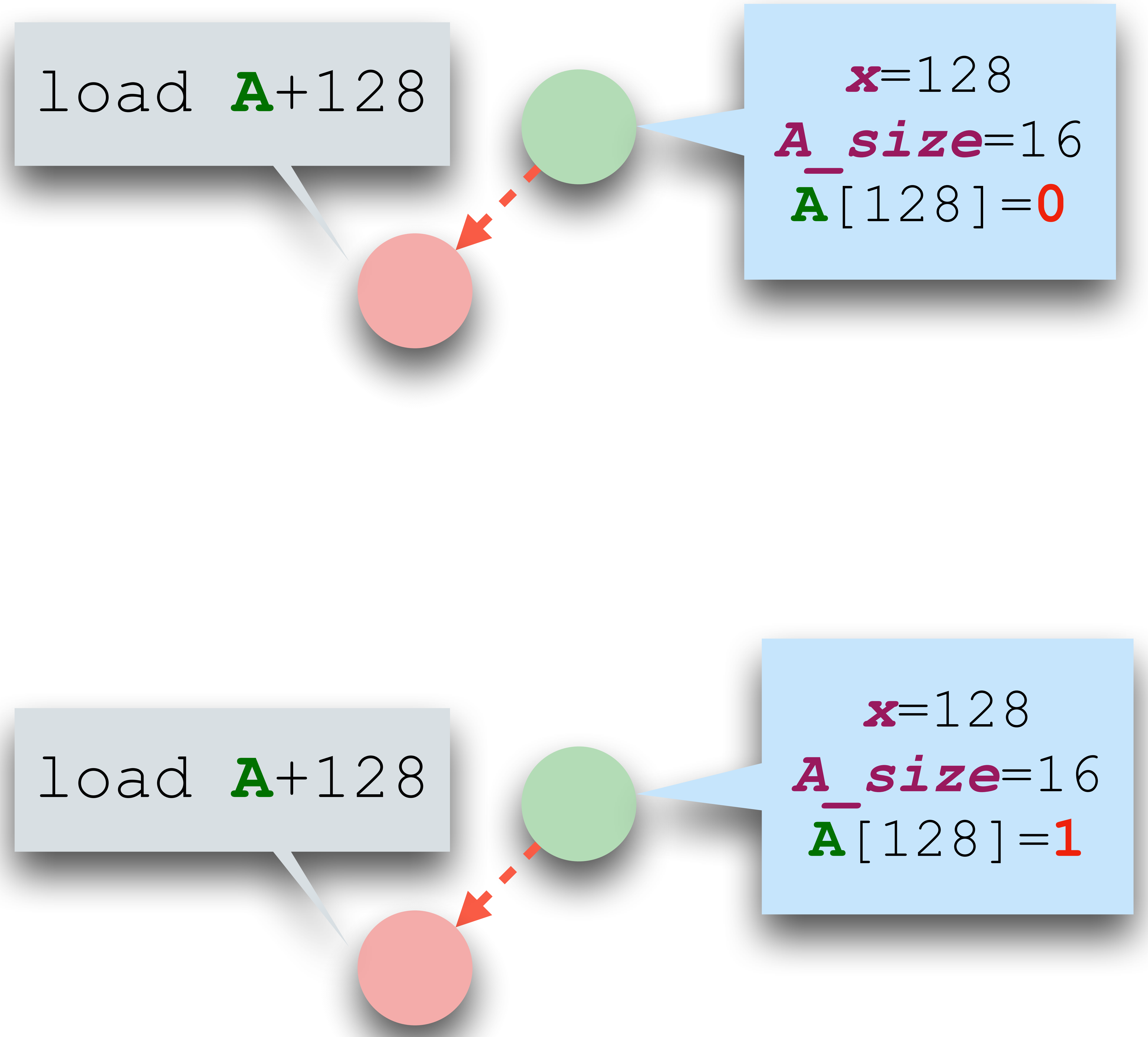


x < A_size predicted as satisfied

Speculative non-interference

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

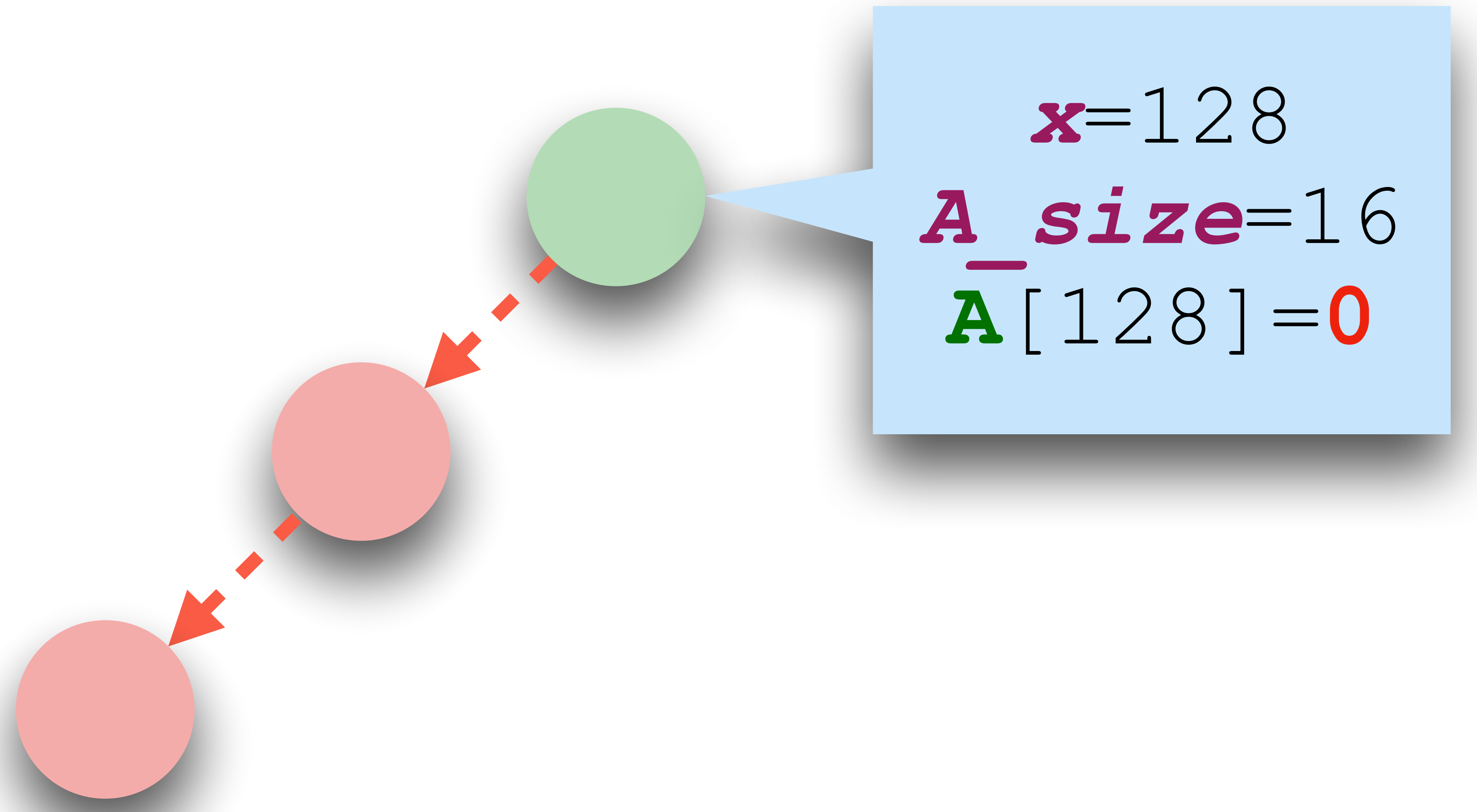
`x < A_size` predicted as satisfied

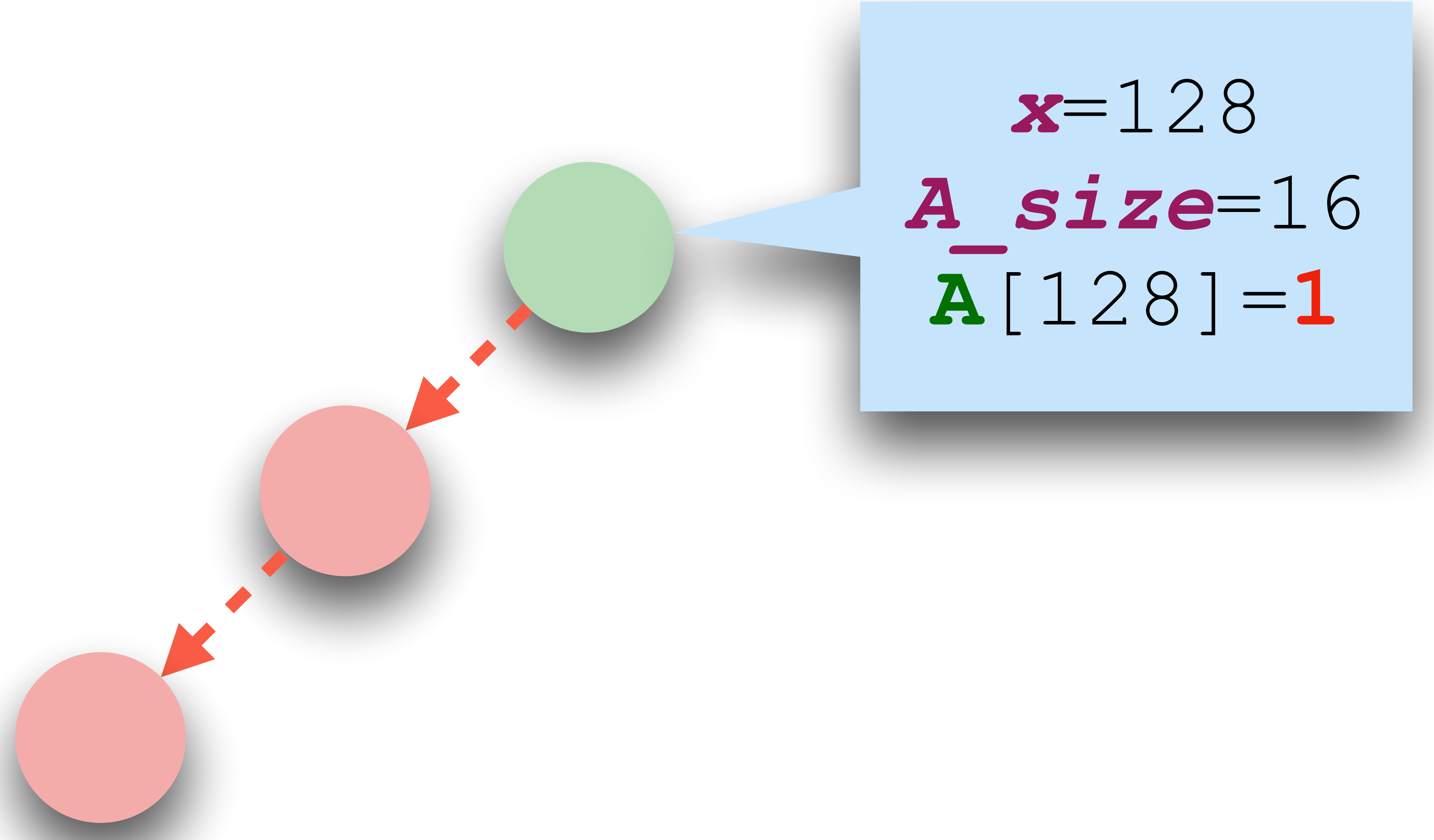


Speculative non-interference

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

x < A_size predicted as satisfied

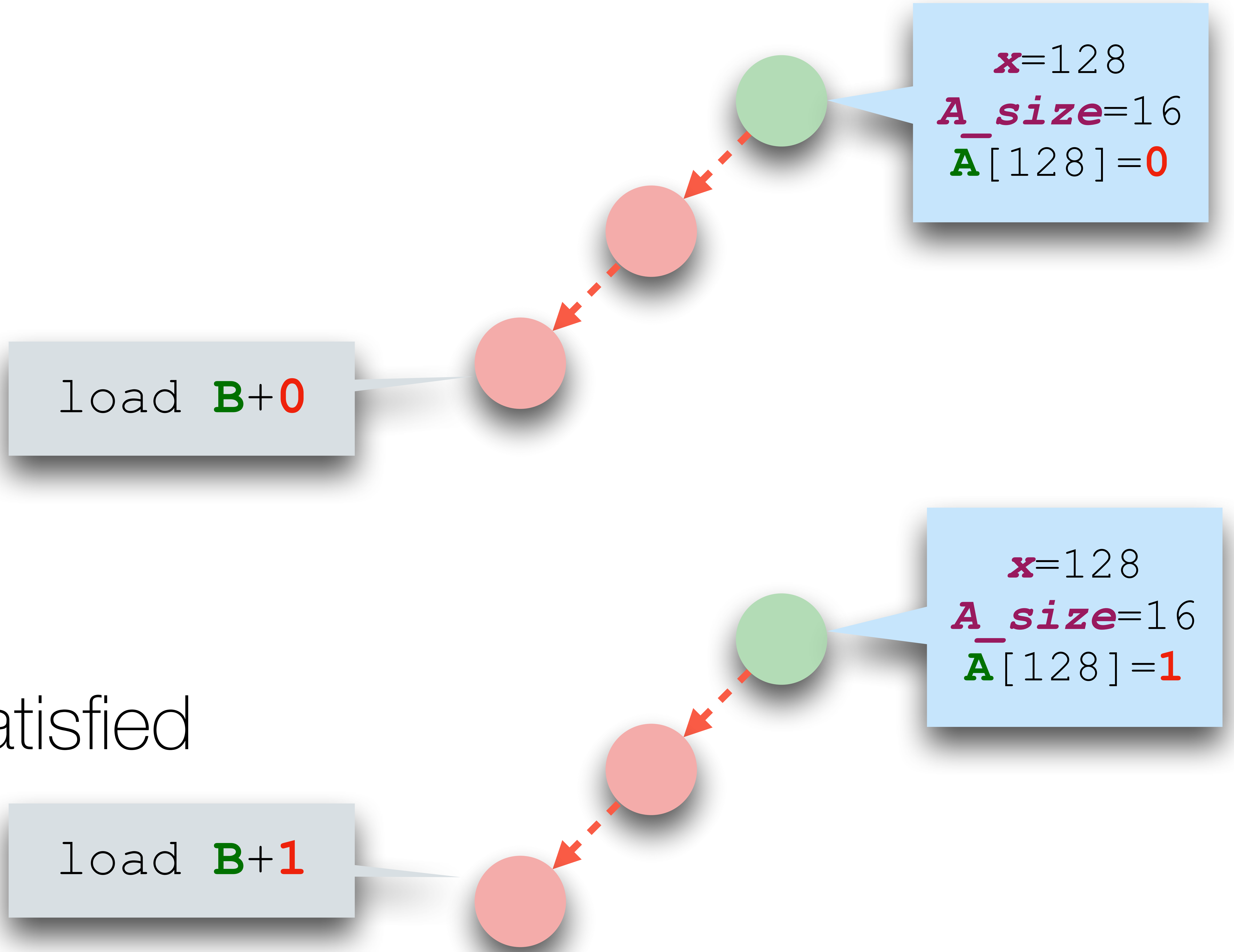




Speculative non-interference

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

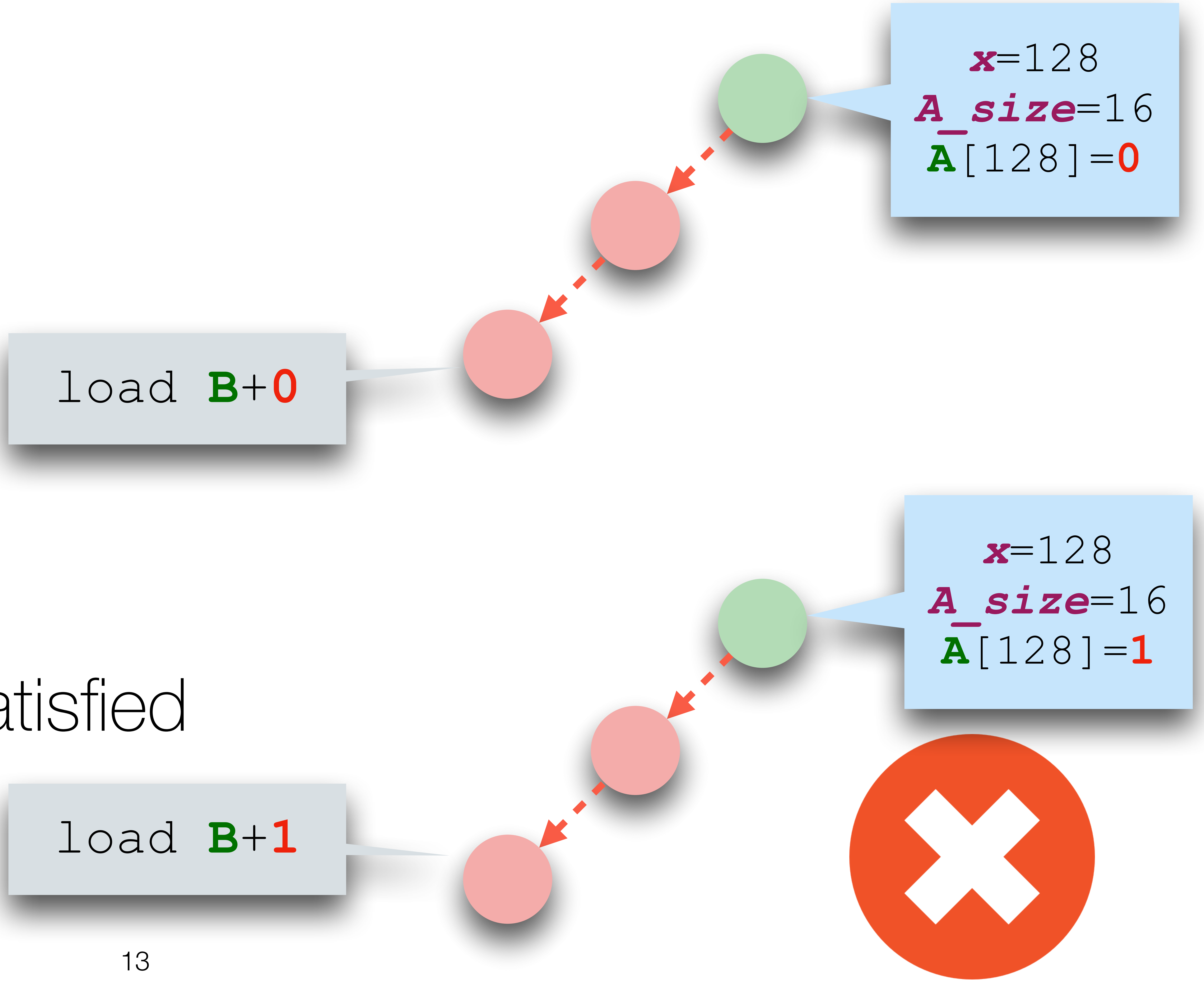
`x < A_size` predicted as satisfied



Speculative non-interference

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

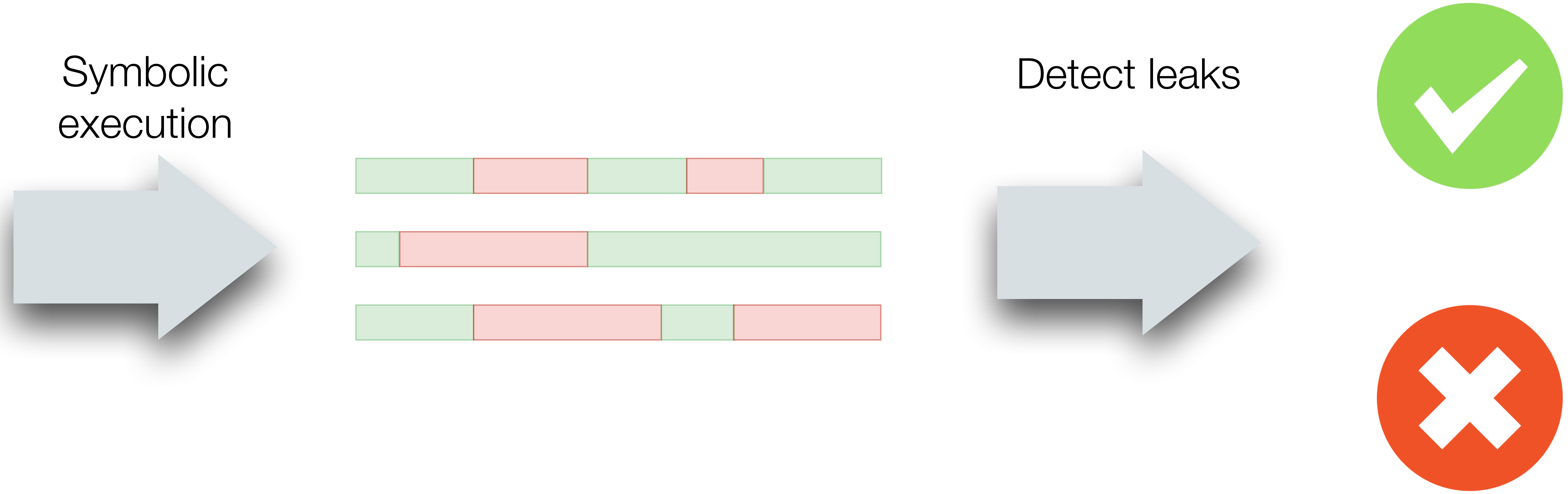
x < A_size predicted as satisfied



Detecting speculative leaks

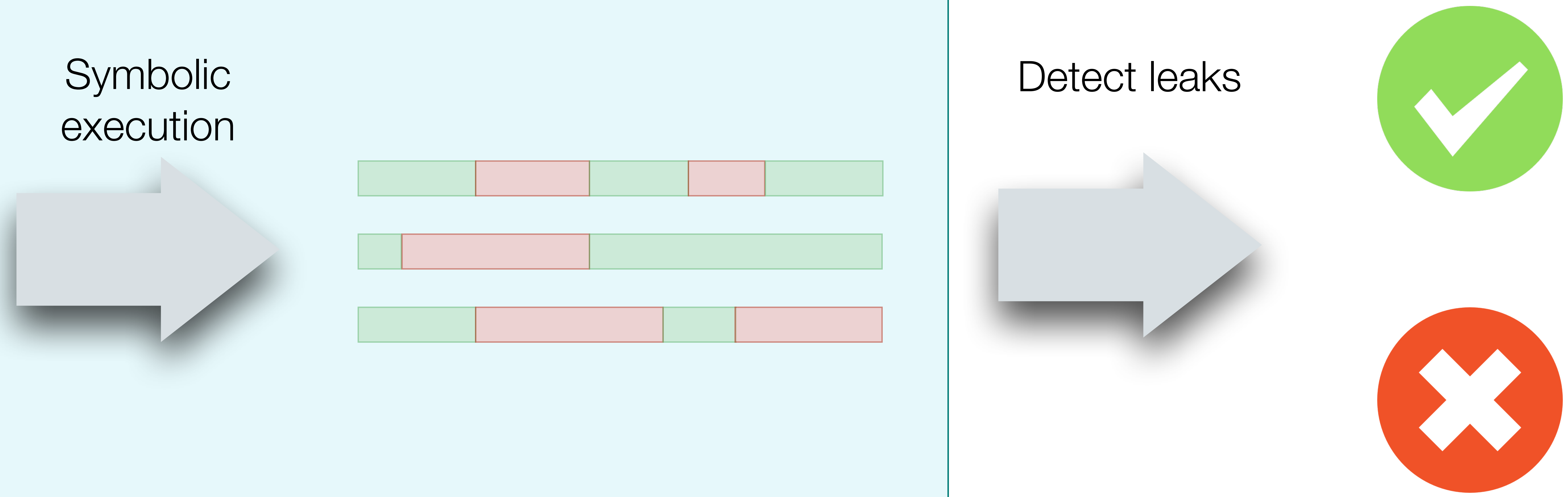
Detecting speculative leaks

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



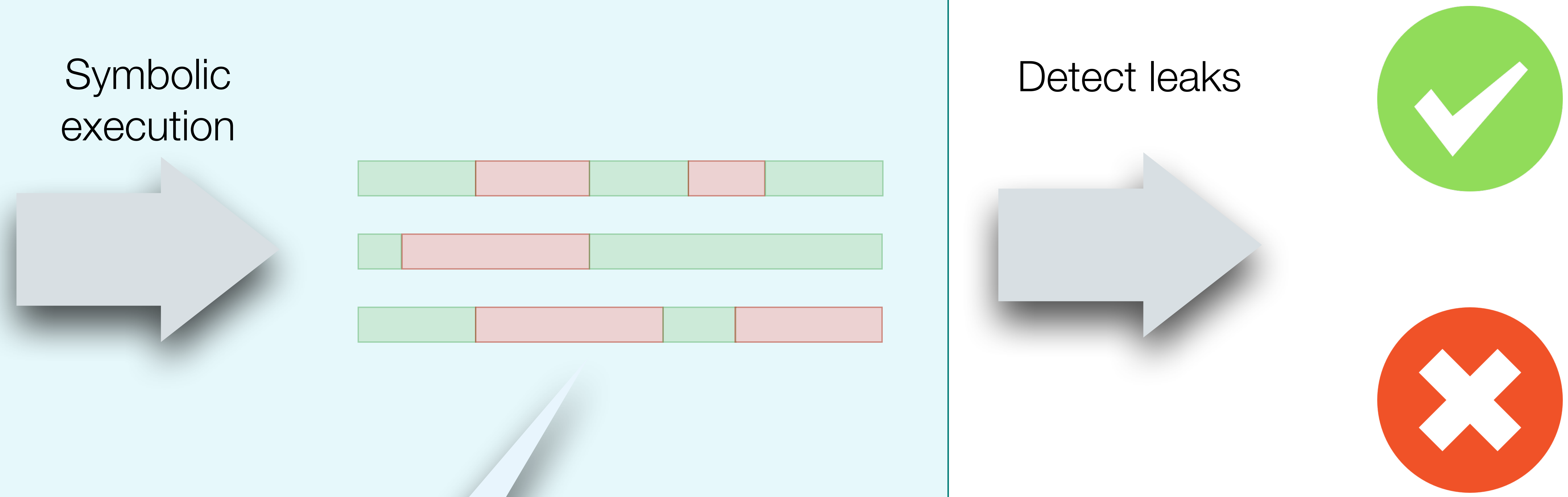
Detecting speculative leaks

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



Detecting speculative leaks

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



Symbolic trace: path condition + observations along the symbolic path

Symbolic execution

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

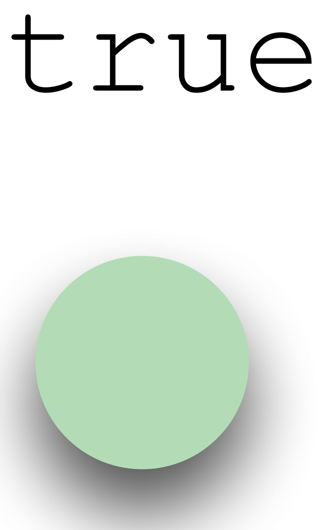
Symbolic execution

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

Always mispredict branch instructions

Symbolic execution

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

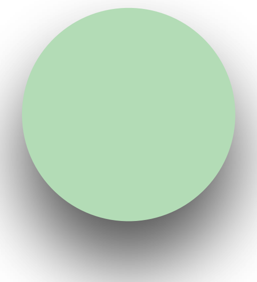


Always mispredict
branch instructions

Symbolic execution

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

true



Always mispredict
branch instructions

Symbolic execution

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

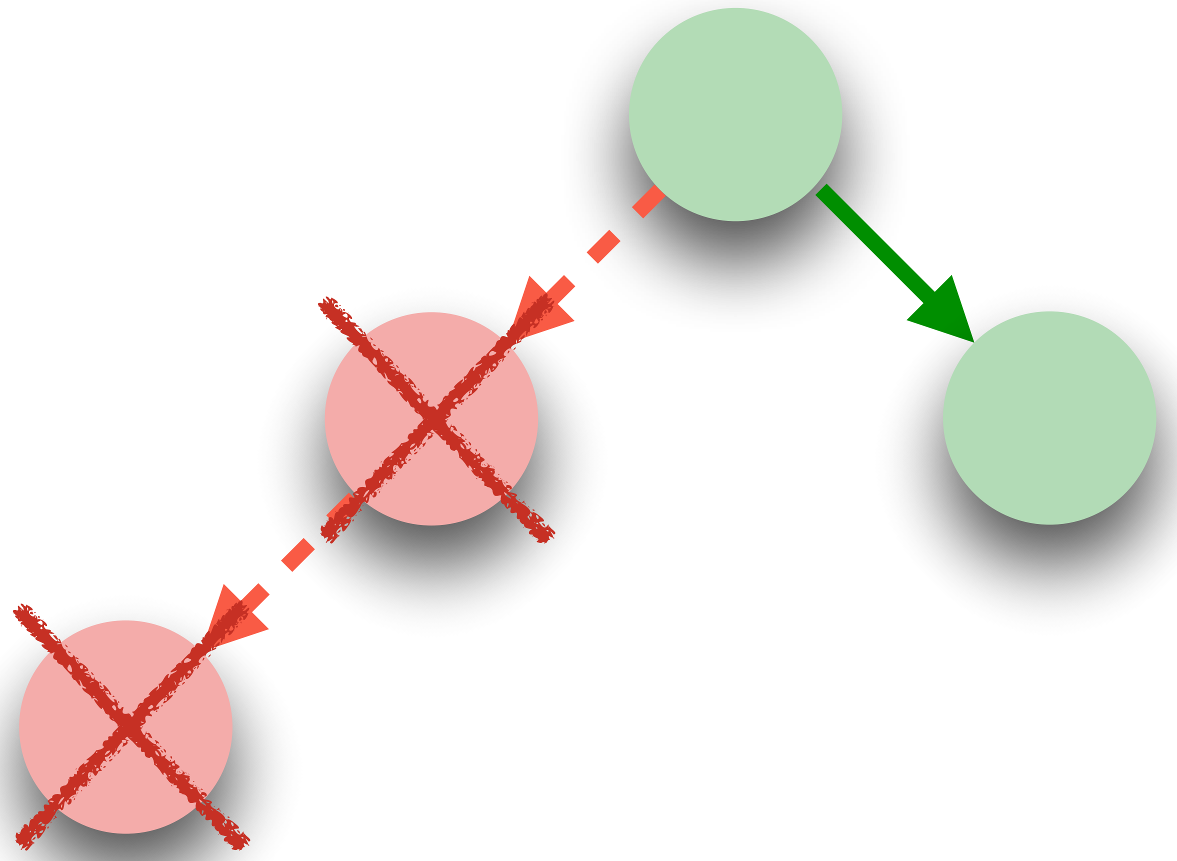
$x \geq A\_size$ $\qquad$ $x < A\_size$

Always mispredict branch instructions

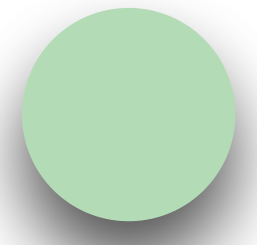
Symbolic execution

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

$x \geq A\_size$



$x < A\_size$



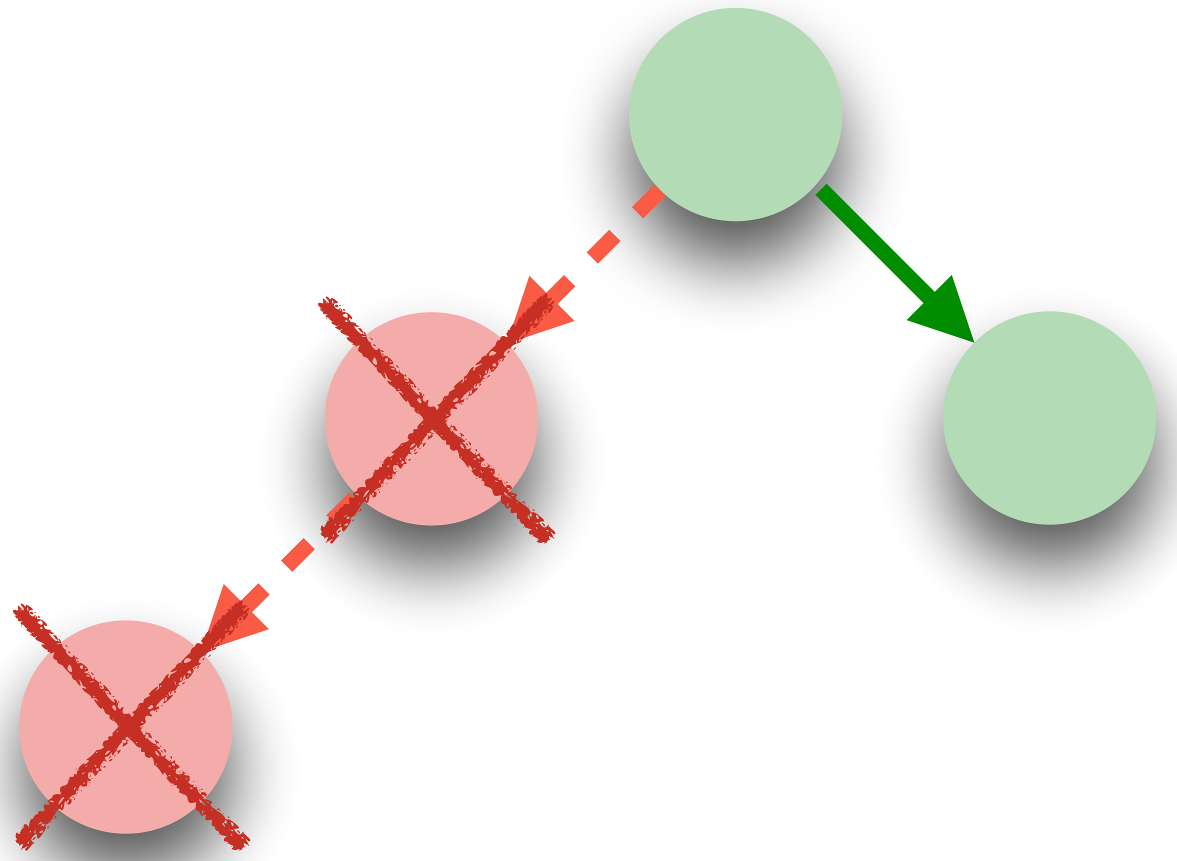
Always mispredict branch instructions

Symbolic execution

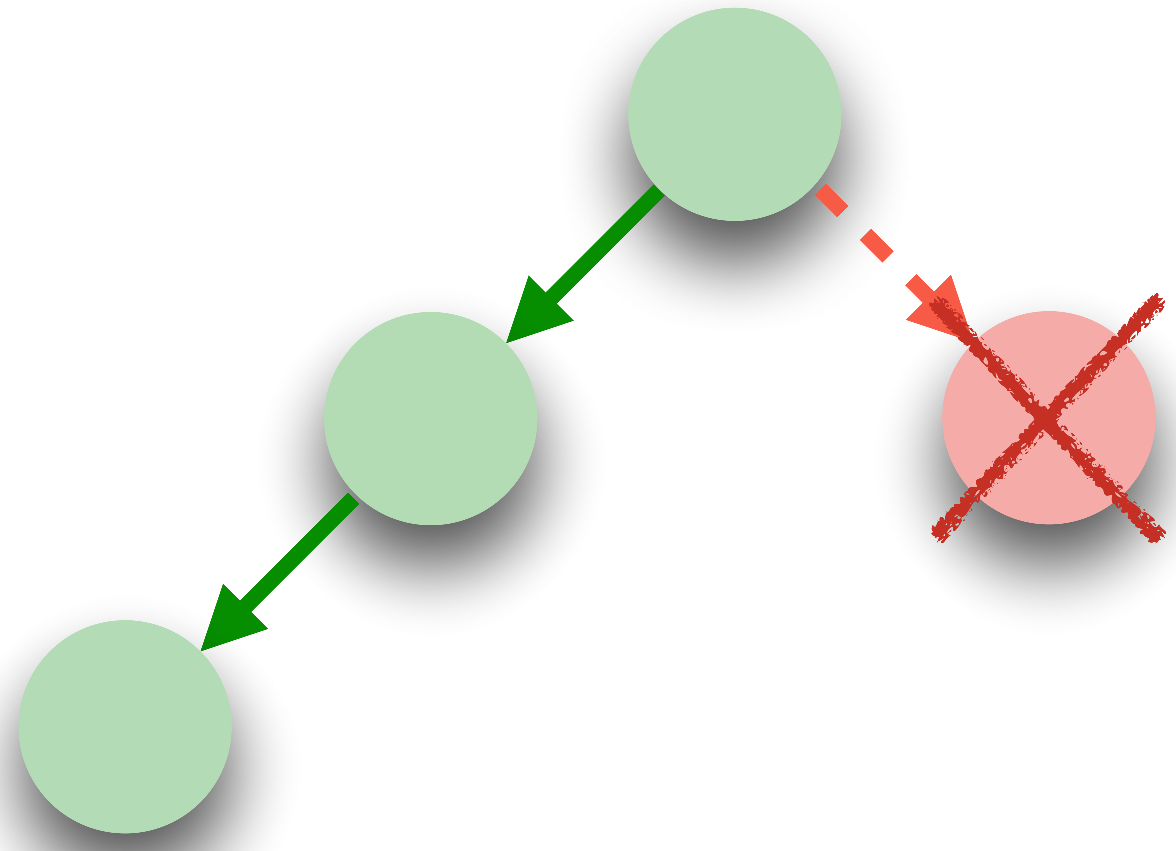
```
if (x < A_size)
    y = A[x]
    z = B[y]
end
```

Always mispredict
branch instructions

$x \geq A\_size$



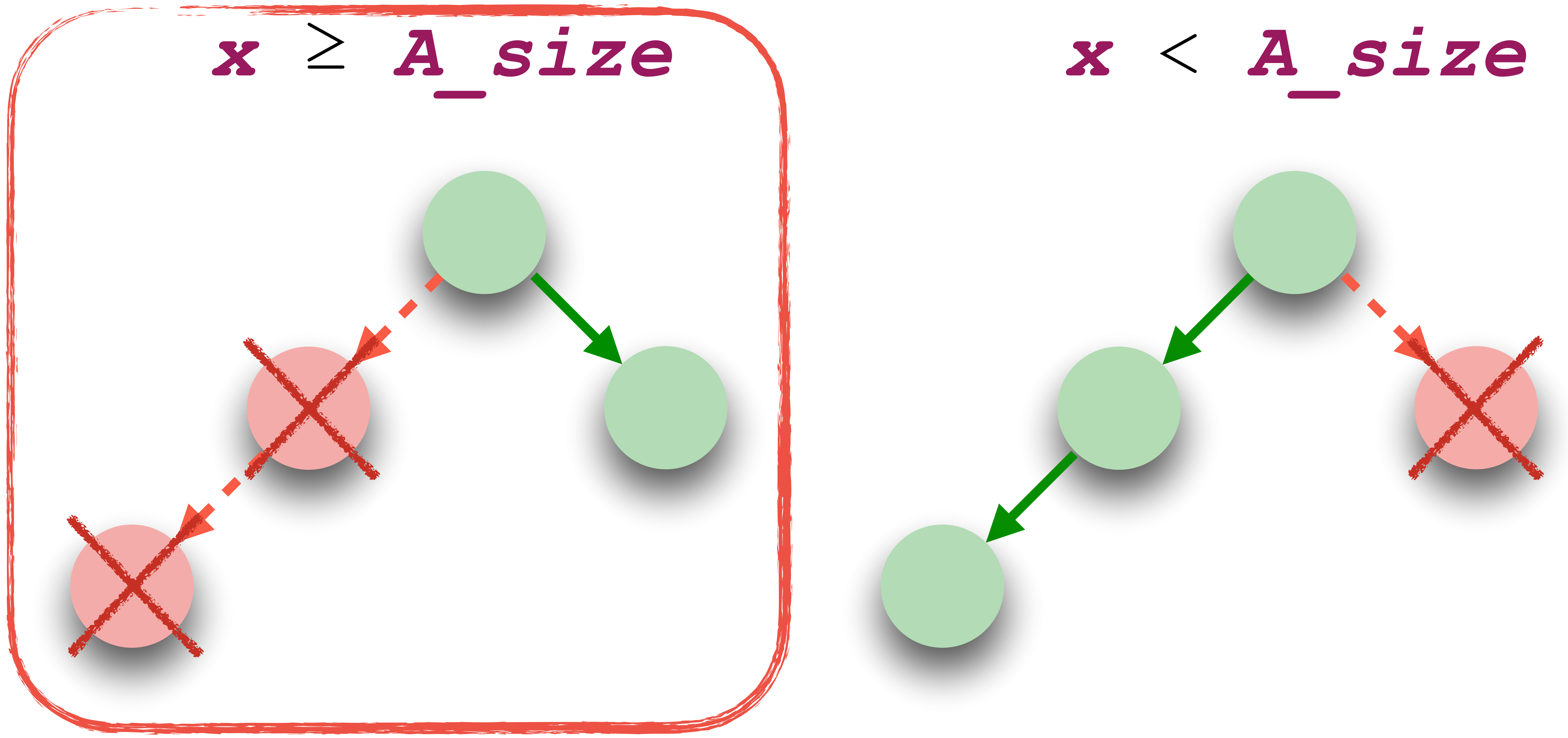
$x < A\_size$



Symbolic execution

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

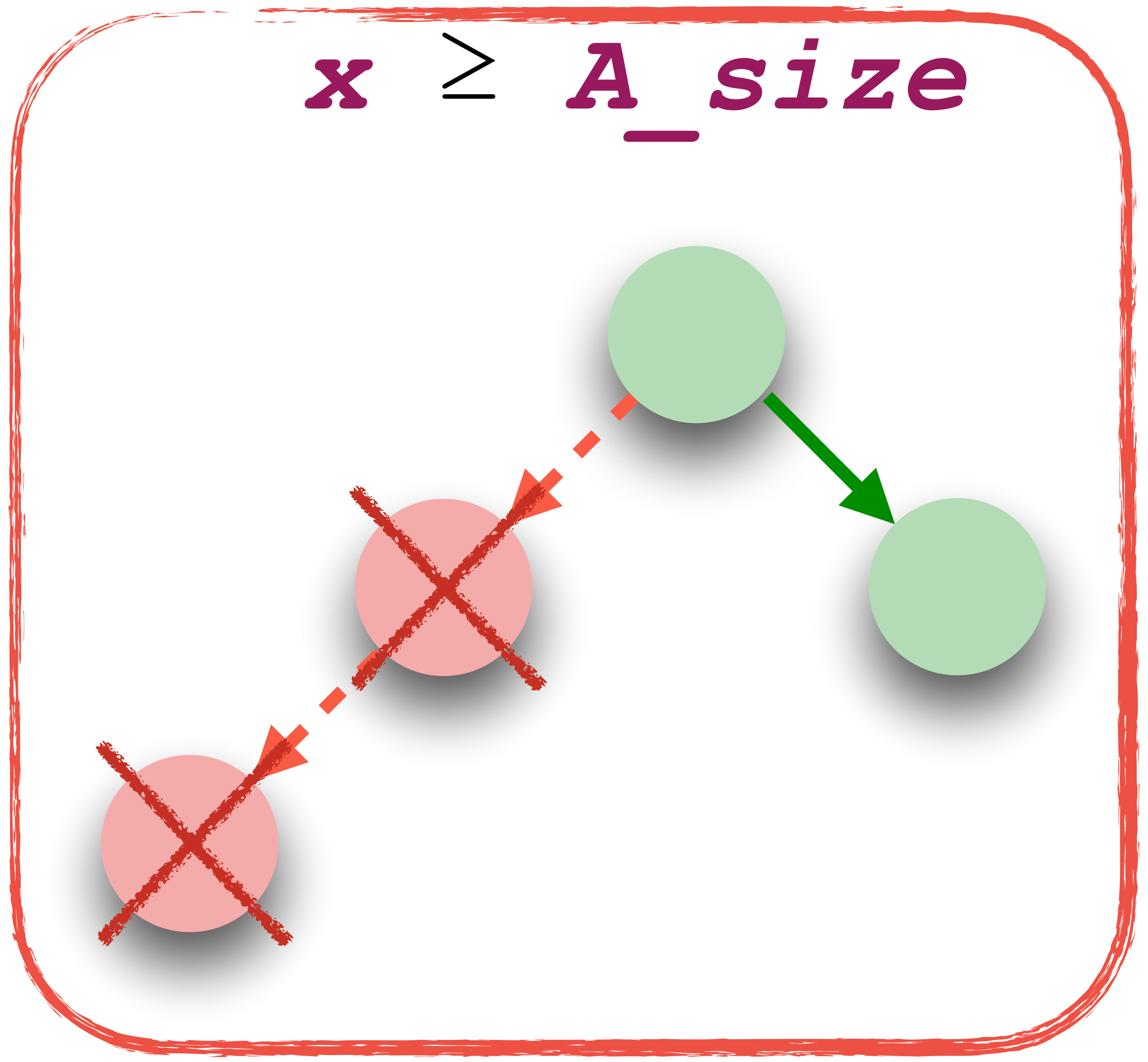
Always mispredict branch instructions

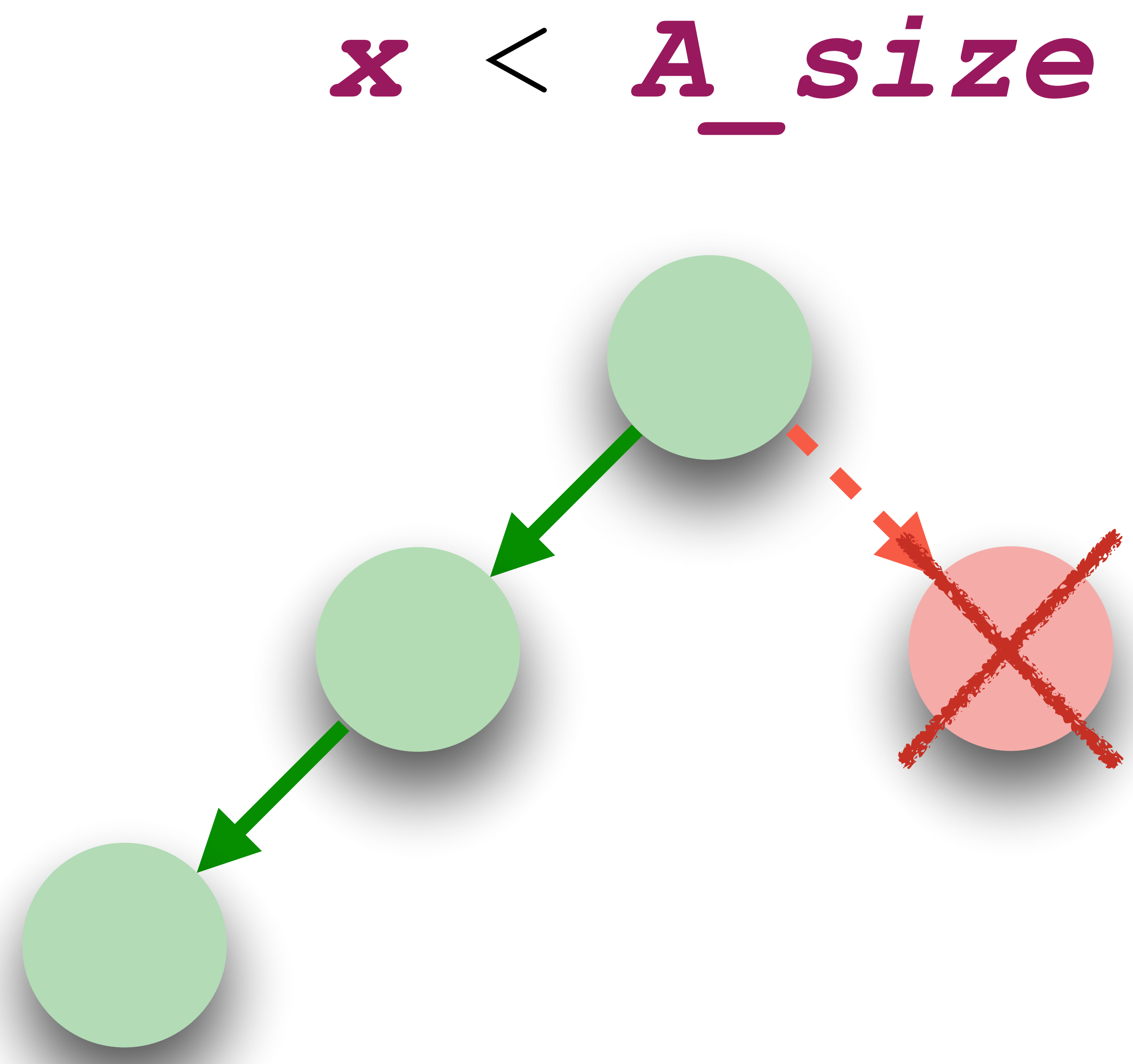


Symbolic execution

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

Always mispredict
branch instructions



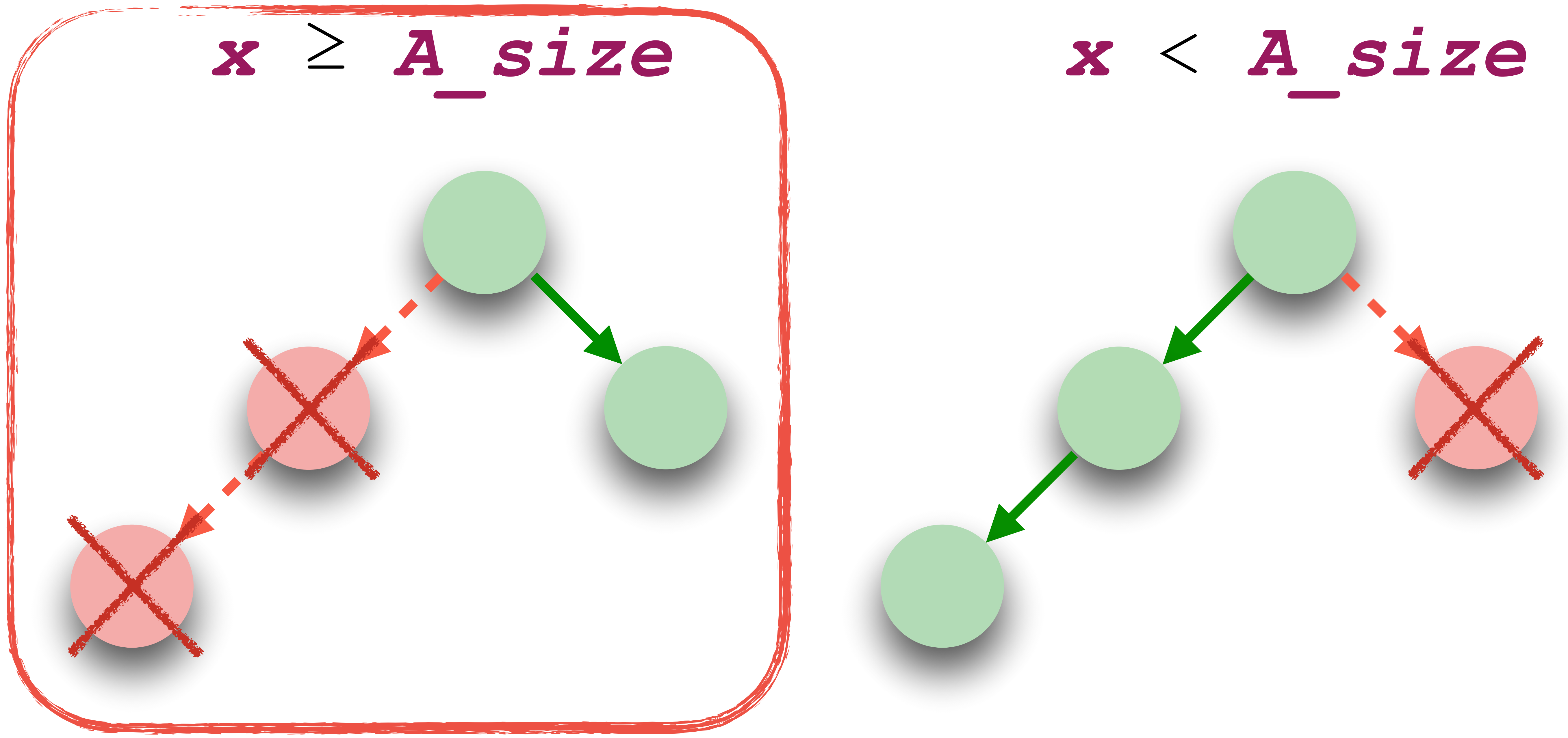


`start pc 2 load A+x load B+A[x] rollback pc 4`

Symbolic execution

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

Always mispredict branch instructions

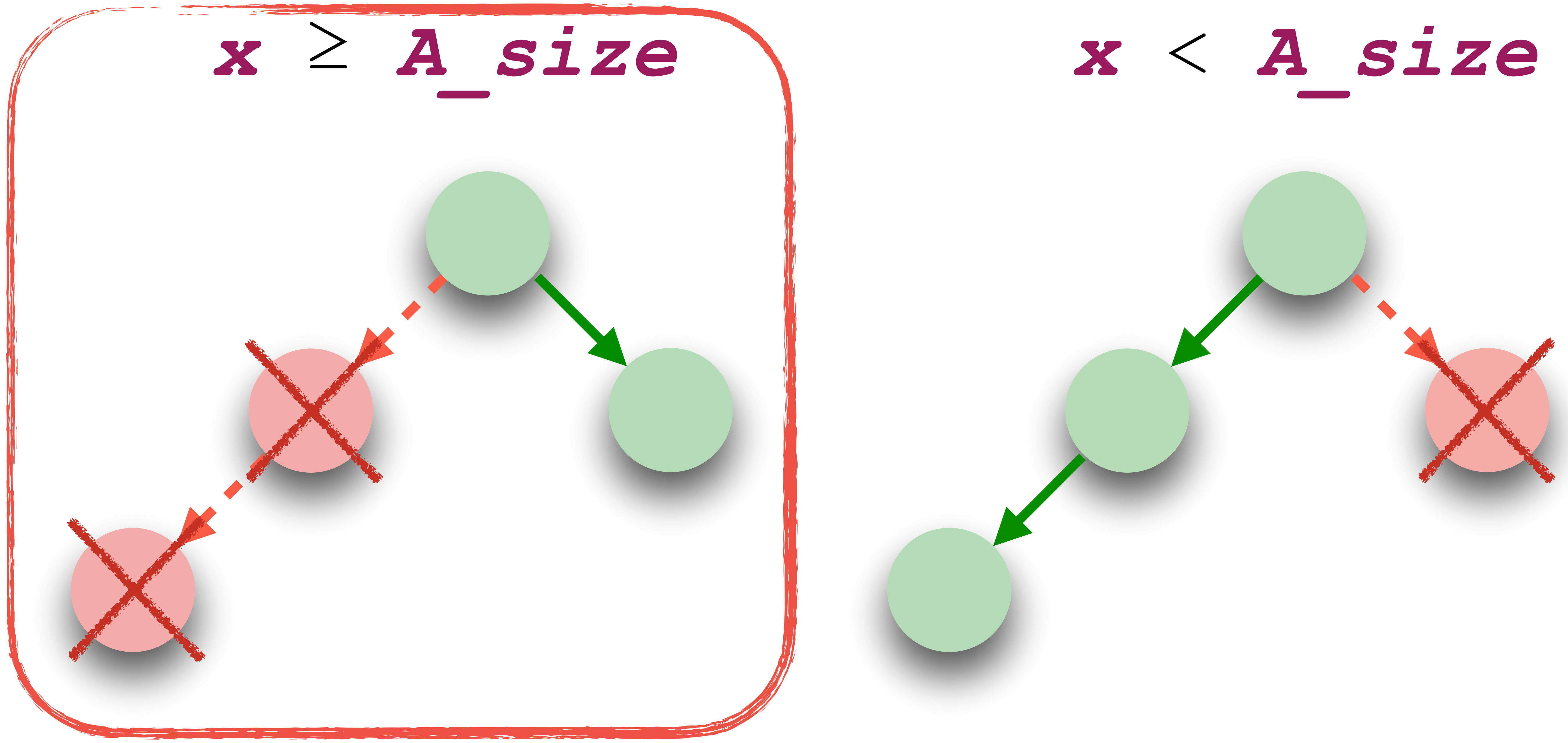


`start pc 2 load A+x load B+A[x] rollback pc 4`

Symbolic execution

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

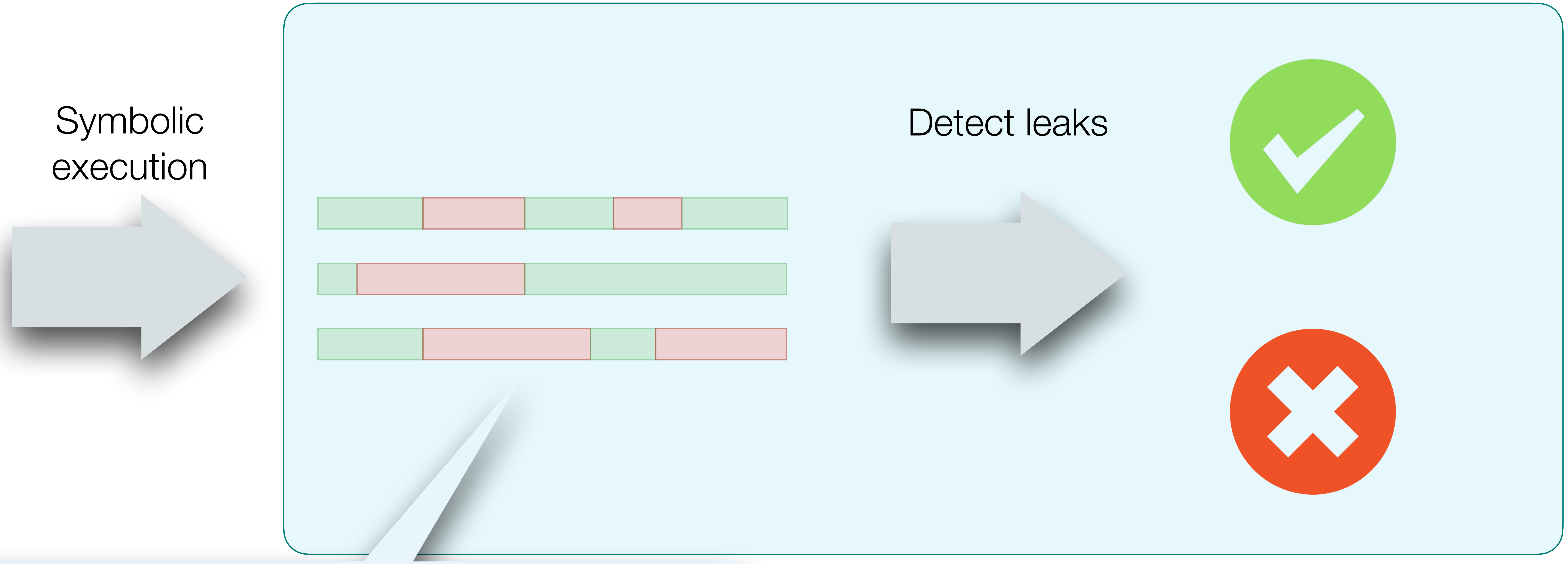
Always mispredict
branch instructions



`start pc 2 load A+x load B+A[x] rollback | pc 4`

Detecting speculative leaks

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



Symbolic trace: path condition + observations along the symbolic path

Detecting speculative leaks

```
      rax
      rcx
      jmp
L1:   loa
      loa
END:
```

For each symbolic trace $\tau \in traces(prg)$

if $MemLeak(\tau)$ **then**

return $INSECURE$

if $CtrlLeak(\tau)$ **then**

return $INSECURE$

return $SECURE$

Detecting speculative leaks

```
      rax
      rcx
      jmp
L1:   loa
      loa
END:
```

For each symbolic trace $\tau \in traces(prg)$

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

return $INSECURE$

if $CtrlLeak(\tau)$ then

return $INSECURE$

return $SECURE$

Memory leaks

Speculative memory accesses ***must be fully determined*** by ***non-speculative*** observations

Memory leaks

Speculative memory accesses ***must be fully determined*** by ***non-speculative*** observations

Memory leaks

Speculative memory accesses ***must be fully determined*** by ***non-speculative*** observations

τ

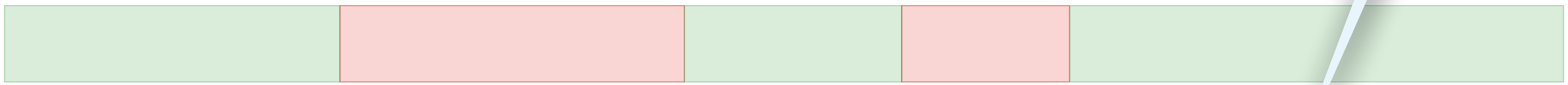
$$pathCnd(\tau) \land obsEqv(\tau|_{non-spec}) \land \neg obsEqv(\tau|_{spec})$$

Memory leaks

Speculative memory accesses ***must be fully determined*** by ***non-speculative*** observations

Check with self-composition

τ



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

Memory leaks

Speculative memory accesses ***must be fully determined*** by ***non-speculative*** observations

Check with self-composition

τ

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

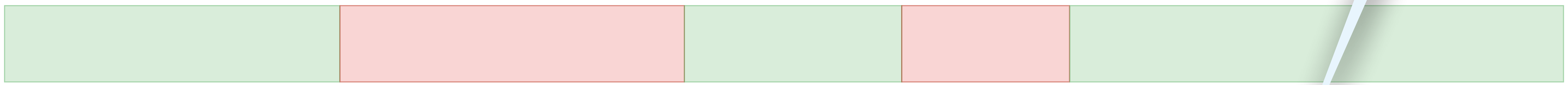
s_1

s_2

Memory leaks

Speculative memory accesses ***must be fully determined*** by ***non-speculative*** observations

Check with self-composition

τ 

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

$$s_1 \models \varphi$$

$$s_2 \models \varphi$$

Memory leaks

Speculative memory accesses ***must be fully determined*** by ***non-speculative*** observations

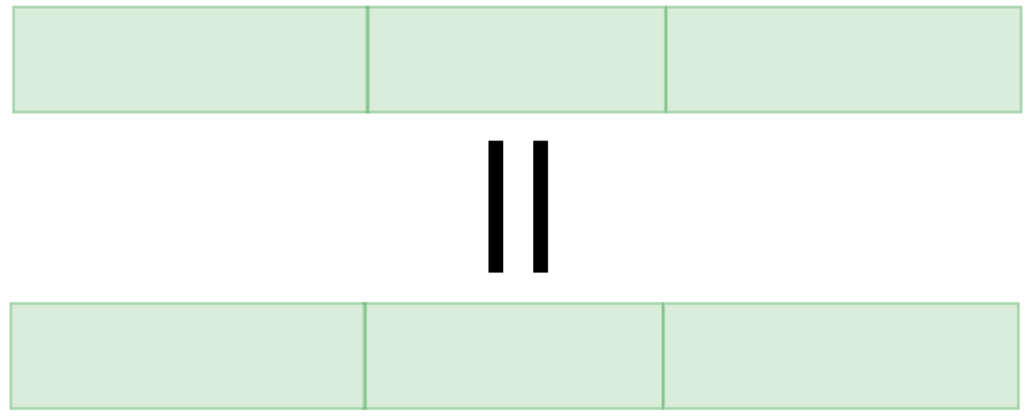
Check with self-composition

τ 

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

$s_1 \models \varphi$

$s_2 \models \varphi$

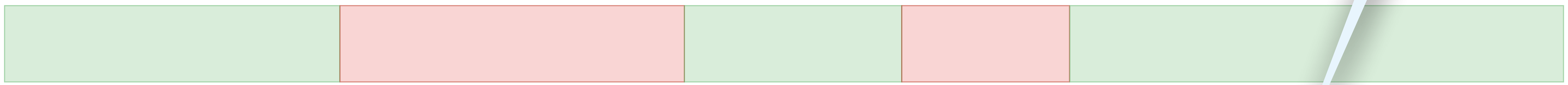


Memory leaks

Speculative memory accesses ***must be fully determined*** by ***non-speculative*** observations

Check with self-composition

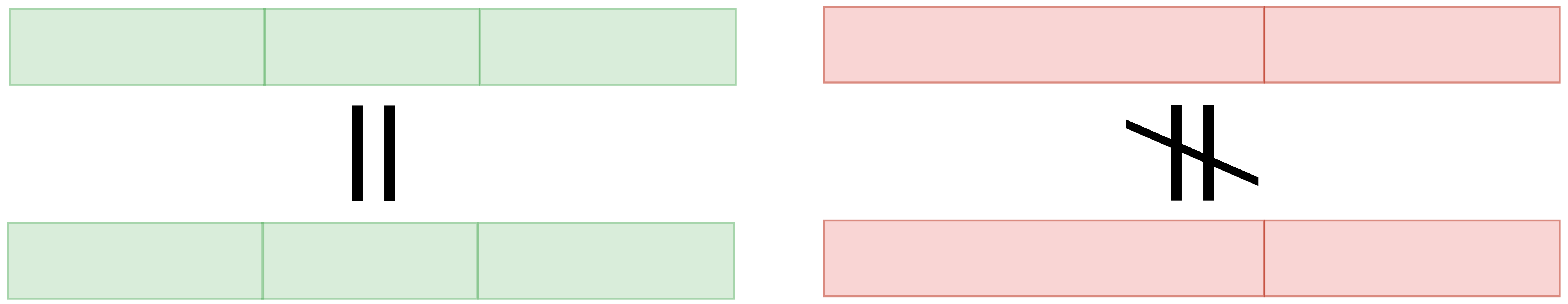
τ



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

$s_1 \models \varphi$

$s_2 \models \varphi$



Spectector + Case studies

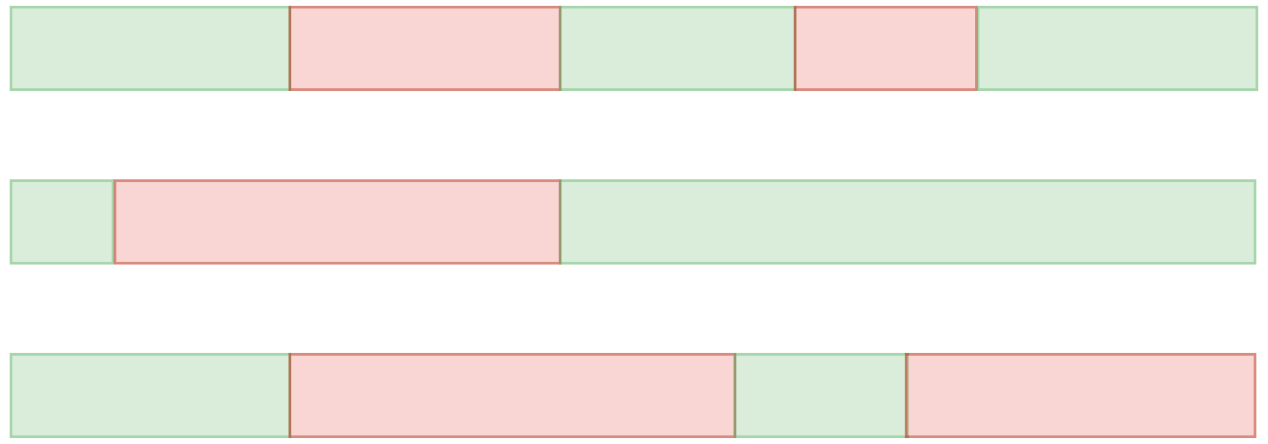
Spectector

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

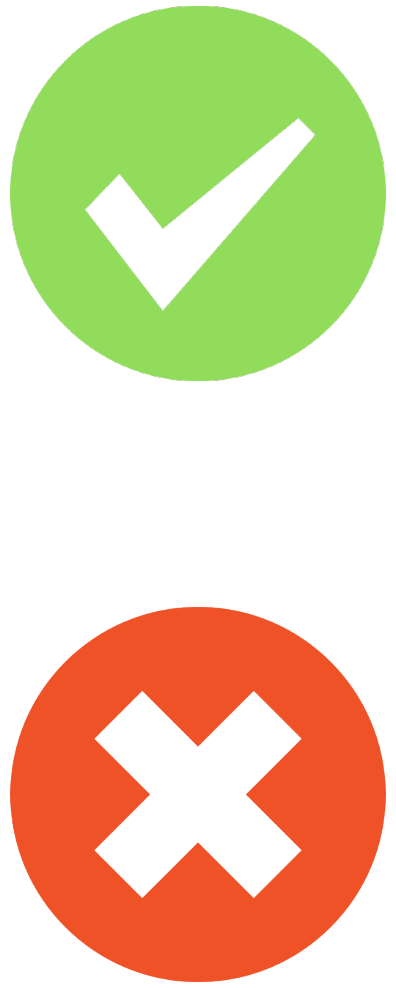
x64 to µASM

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

Symbolic execution



Check for speculative leaks



Spectector

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

x64 to µASM

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

More details

- Built in Ciao Prolog
- **Z3** for symbolic execution and leak detection

Symbolic execution

Check for speculative leaks

Case study: compiler mitigations

Target:

- 15 variants of Spectre V1 by Paul Kocher*
- Compiled with Microsoft Visual C++, Intel ICC, and Clang with different mitigations and optimization levels
- 240 assembly programs of up to 200 instructions each

How:

- Use Spectector to prove security or detect leaks

* Paul Kocher - Spectre Mitigations in Microsoft C/C++ Compiler — https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html

Results

Ex.	VCC						ICC				CLANG					
	UNP		FEN 19.15		FEN 19.20		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	VCC						ICC				CLANG					
	UNP		FEN 19.15		FEN 19.20		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

No countermeasures

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Automated insertion of fences

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Speculative load hardening

Ex.	VCC						ICC				CLANG					
	UNP		FEN 19.15		FEN 19.20		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	`-O0`	`-O2`	`-O0`	`-O2`	`-O0`	`-O2`	`-O0`	`-O2`	`-O0`	`-O2`	`-O0`	`-O2`	`-O0`	`-O2`	`-O0`	`-O2`
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	VCC						ICC				CLANG					
	UNP		FEN 19.15		FEN 19.20		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2												-O2	-O0	-O2
01	○	○												●	●	●
02	○	○												●	●	●
03	○	○												●	●	●
04	○	○												●	●	●
05	○	○												●	●	●
06	○	○												●	●	●
07	○	○												●	●	●
08	○	●												●	●	●
09	○	○												●	●	●
10	○	○												●	●	○
11	○	○												●	●	●
12	○	○												●	●	●
13	○	○												●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

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
  - Unnecessary fences
- Programs with SLH are secure except #10 and #15

Case study: scalability

Target: Xen hypervisors

Main challenges for scalability:

- Policy definition
- ISA coverage
- Path explosion

How:

- Analyze scalability of checking SNI ***relative to*** symbolic execution
    - *24'000* symbolic paths of < *10'000* instructions (from ~ *4'000* functions)

Case study: scalability

Target: Xen hypervisors

Main challenges for scalability:

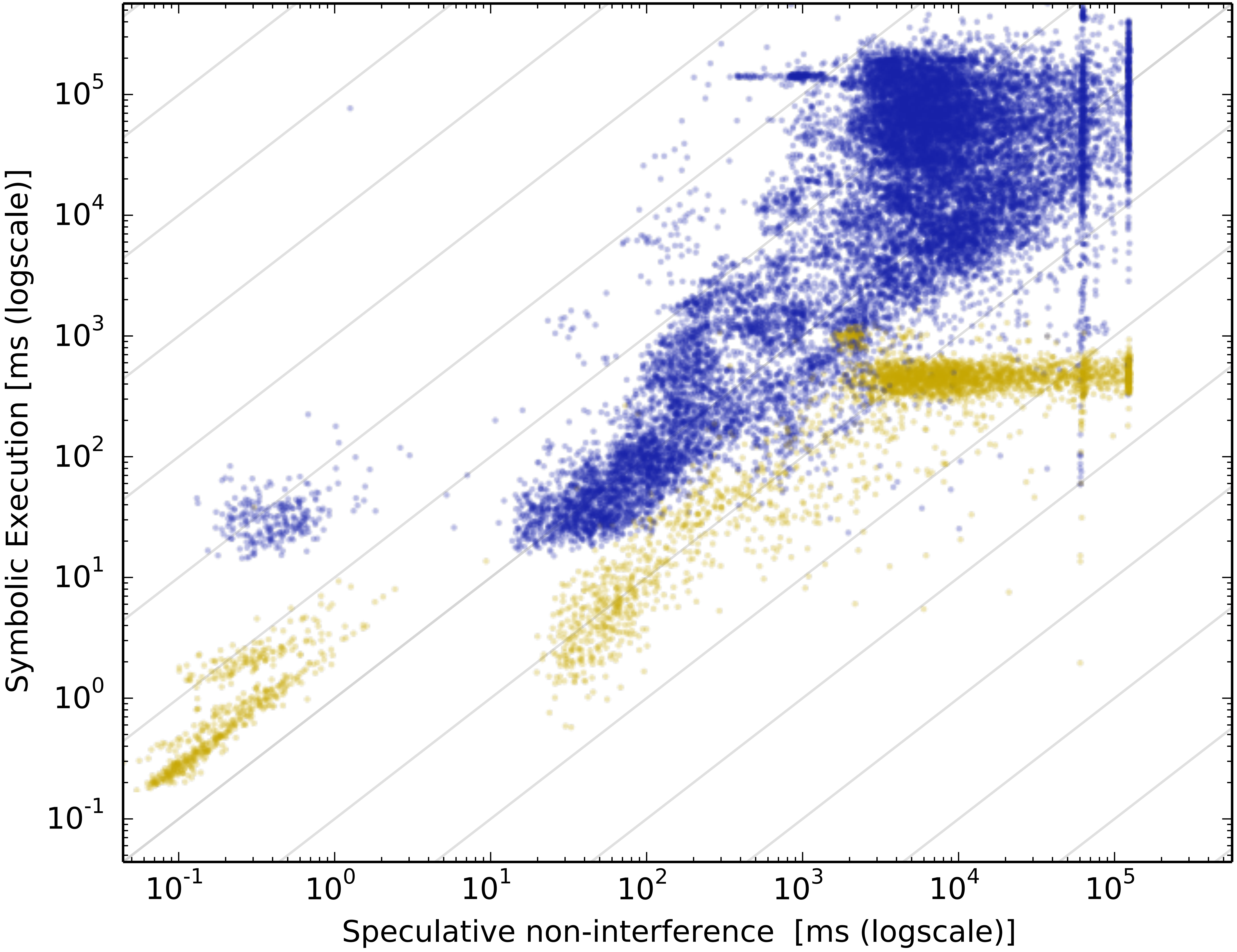
- Policy definition
- ISA coverage
- Path explosion

Trade-offs affect analysis soundness and completeness

How:

- Analyze scalability of checking SNI ***relative to*** symbolic execution
    - *24'000* symbolic paths of < *10'000* instructions (from ~ *4'000* functions)

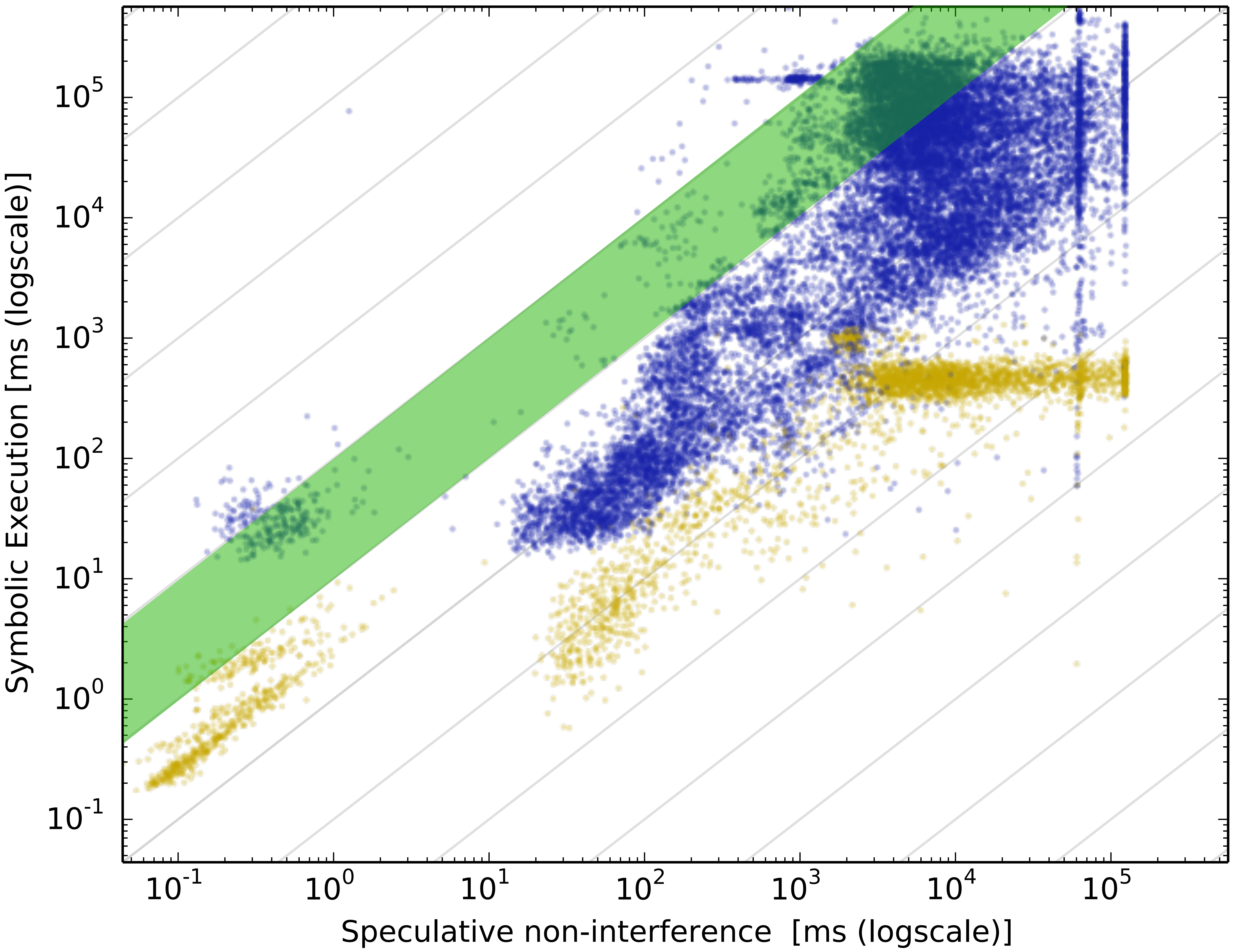
Results



Symbolic Execution [ms (logscale)]
Speculative non-interference [ms (logscale)]
10^5
10^4
10^3
10^2
10^1
10^0
10^-1

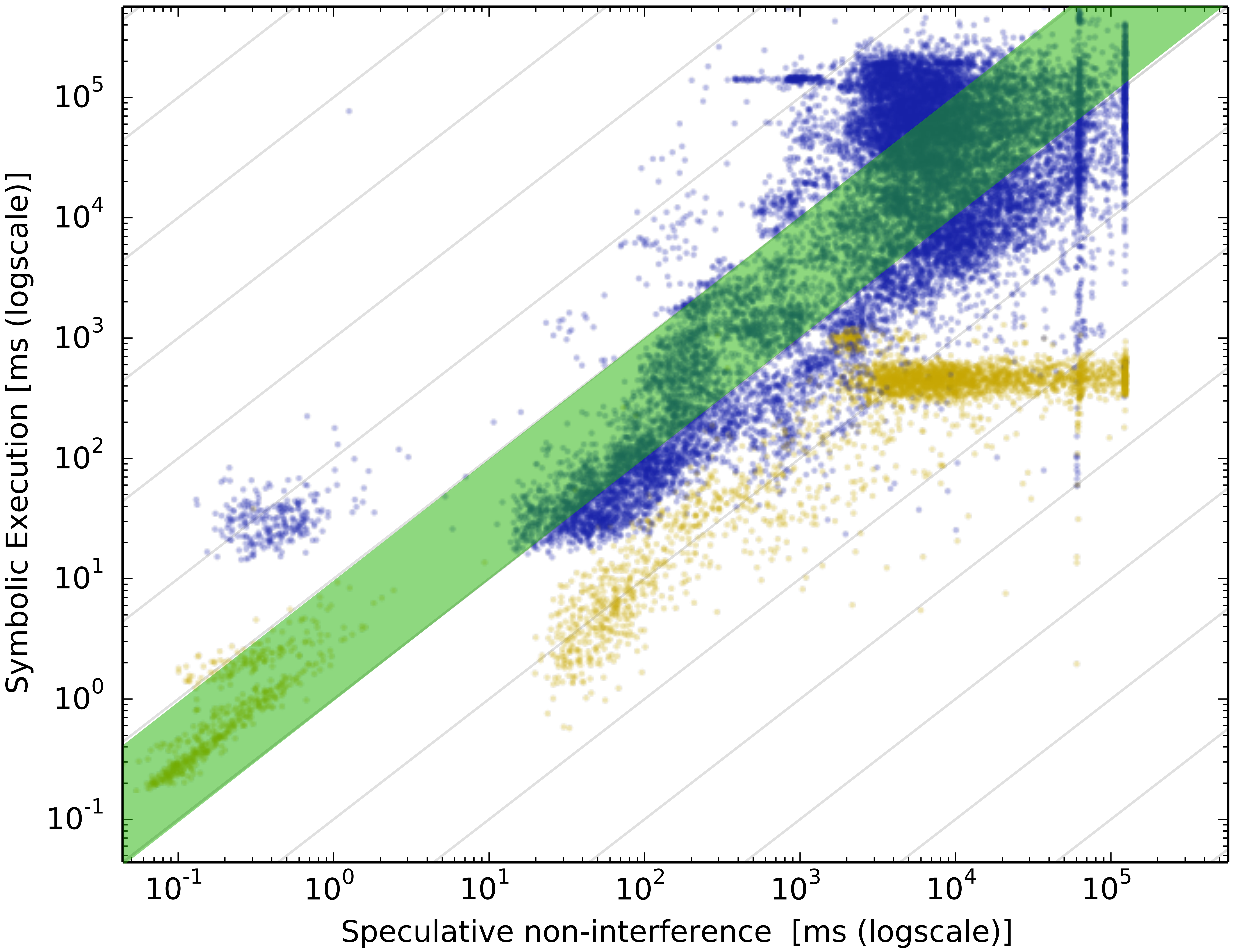
Results

- SNI 10x-100x faster
  - 20.2% traces



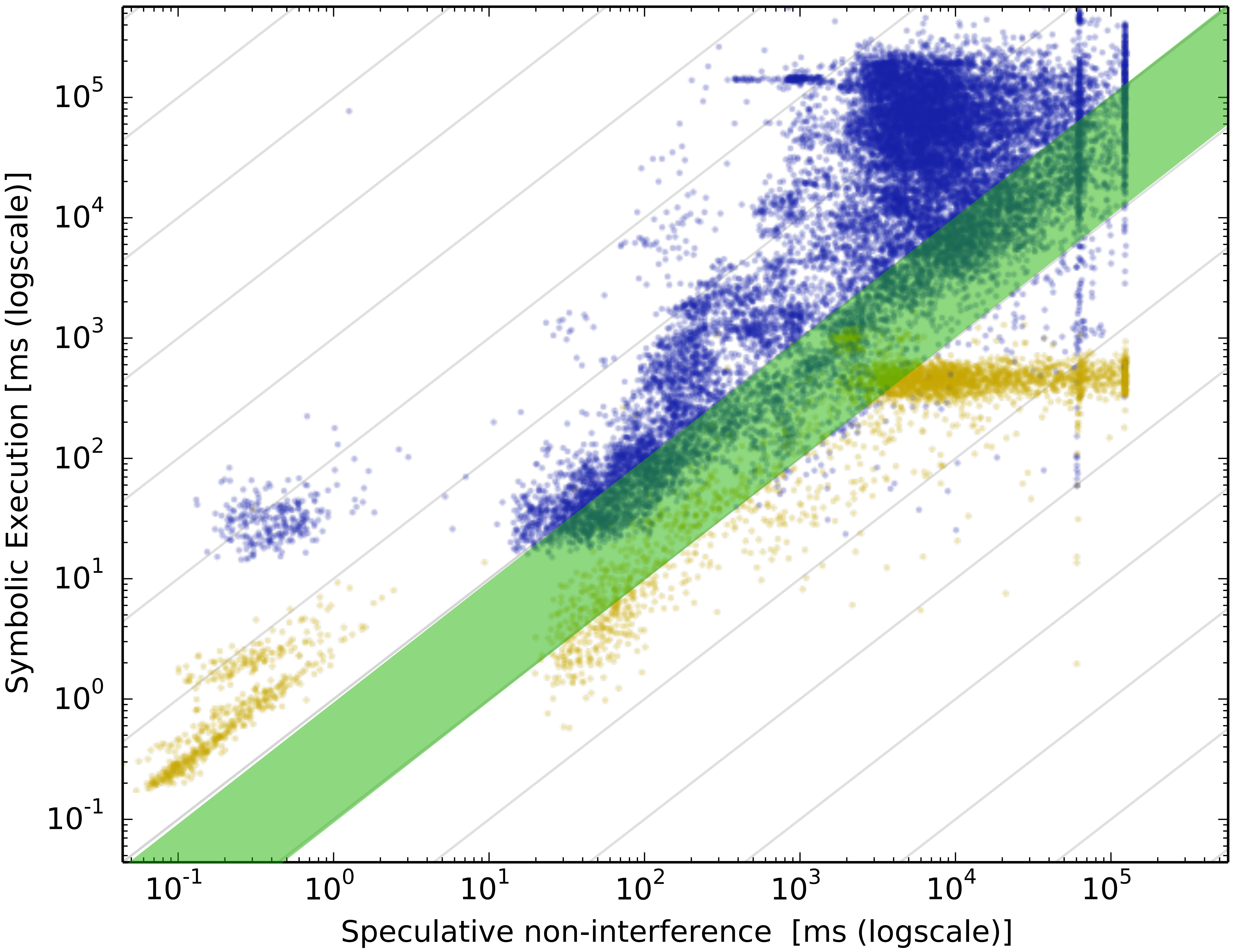
Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces



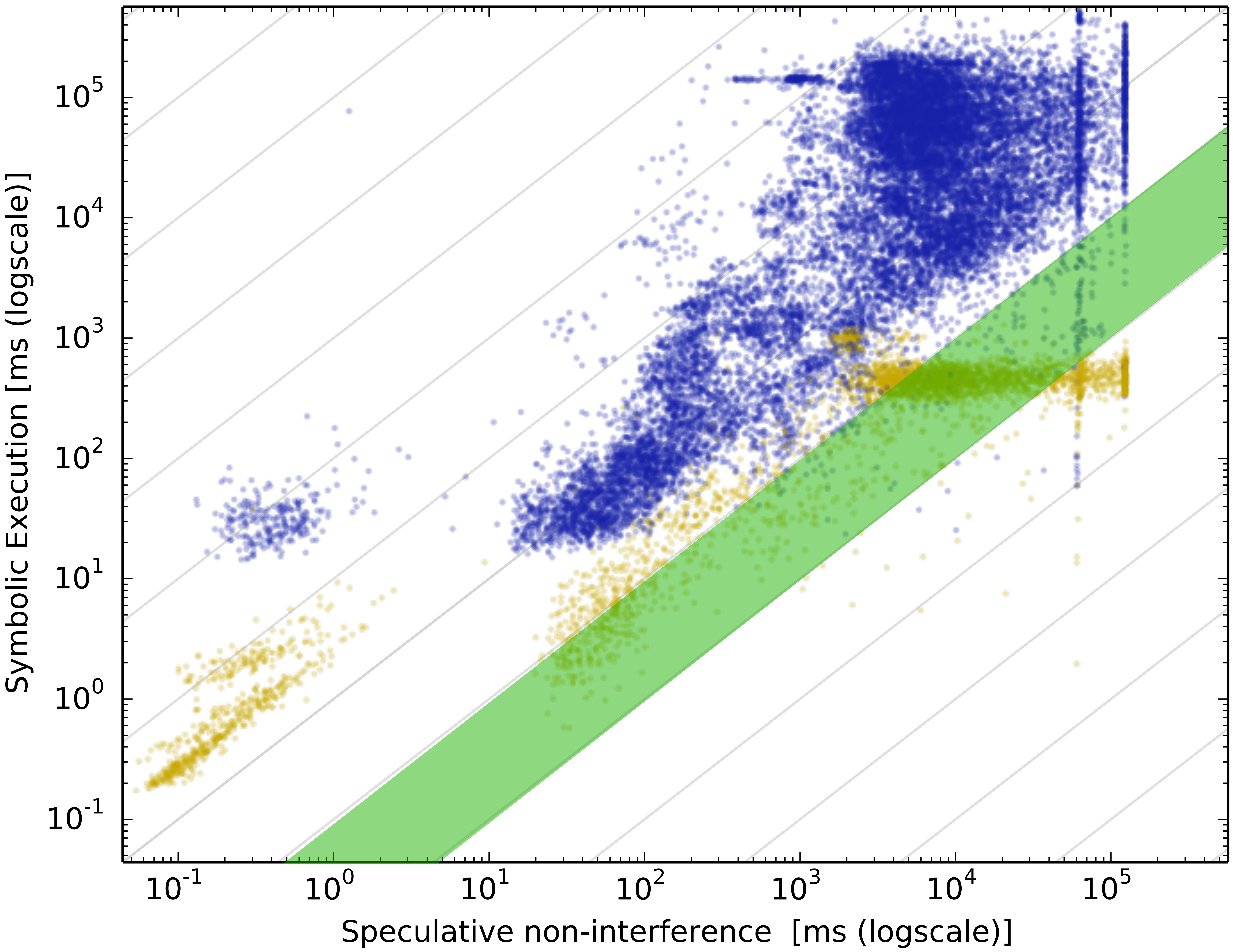
Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces
- SNI ≤10x slower
  - 26.9% traces



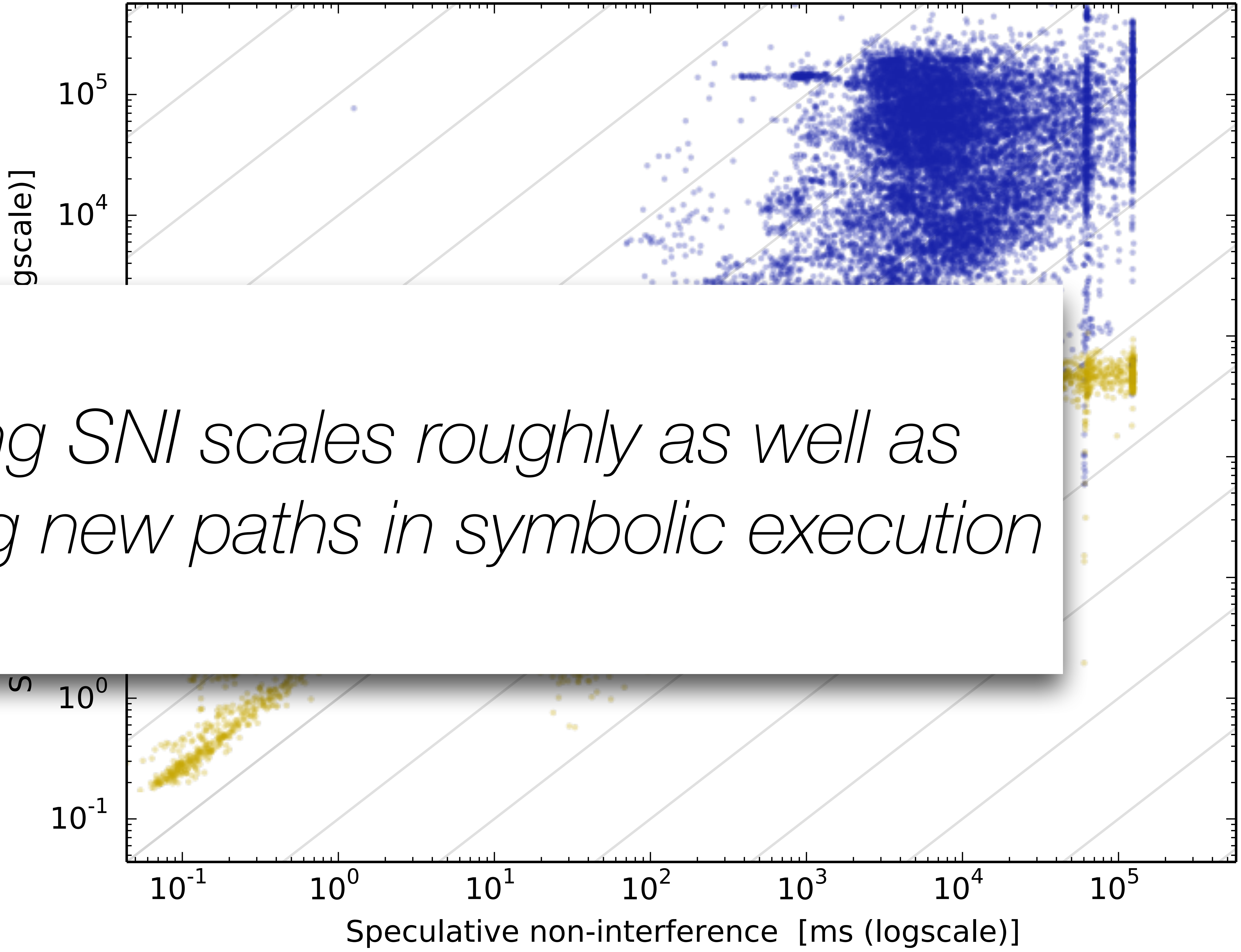
Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces
- SNI ≤10x slower
  - 26.9% traces
- SNI 10x-100x slower
  - 7.9% traces



Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI
  - 4
- SNI
  - 26
- SNI 10x-100x slower
  - 7.9% traces



Checking SNI scales roughly as well as discovering new paths in symbolic execution

Conclusion

Speculative non-interference

Formally!

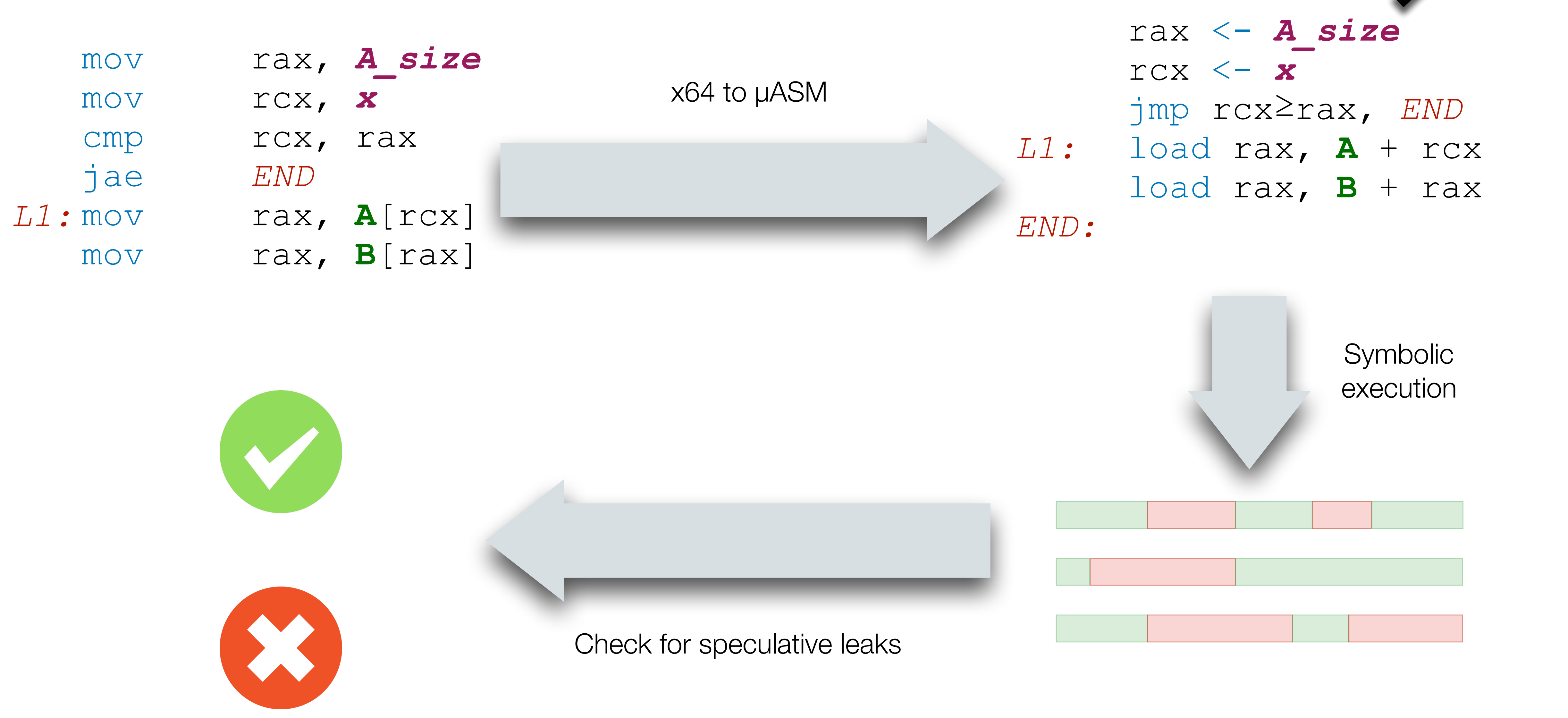
Program **P** is **speculatively non-interferent** for prediction oracle ***O*** if

For all program states s and s':

$$P_{\texttt{non-spec}}(s) = P_{\texttt{non-spec}}(s') \implies P_{\texttt{spec}}(s, O) = P_{\texttt{spec}}(s', O)$$

See paper for: reasoning about ***arbitrary prediction oracles***

Spectector

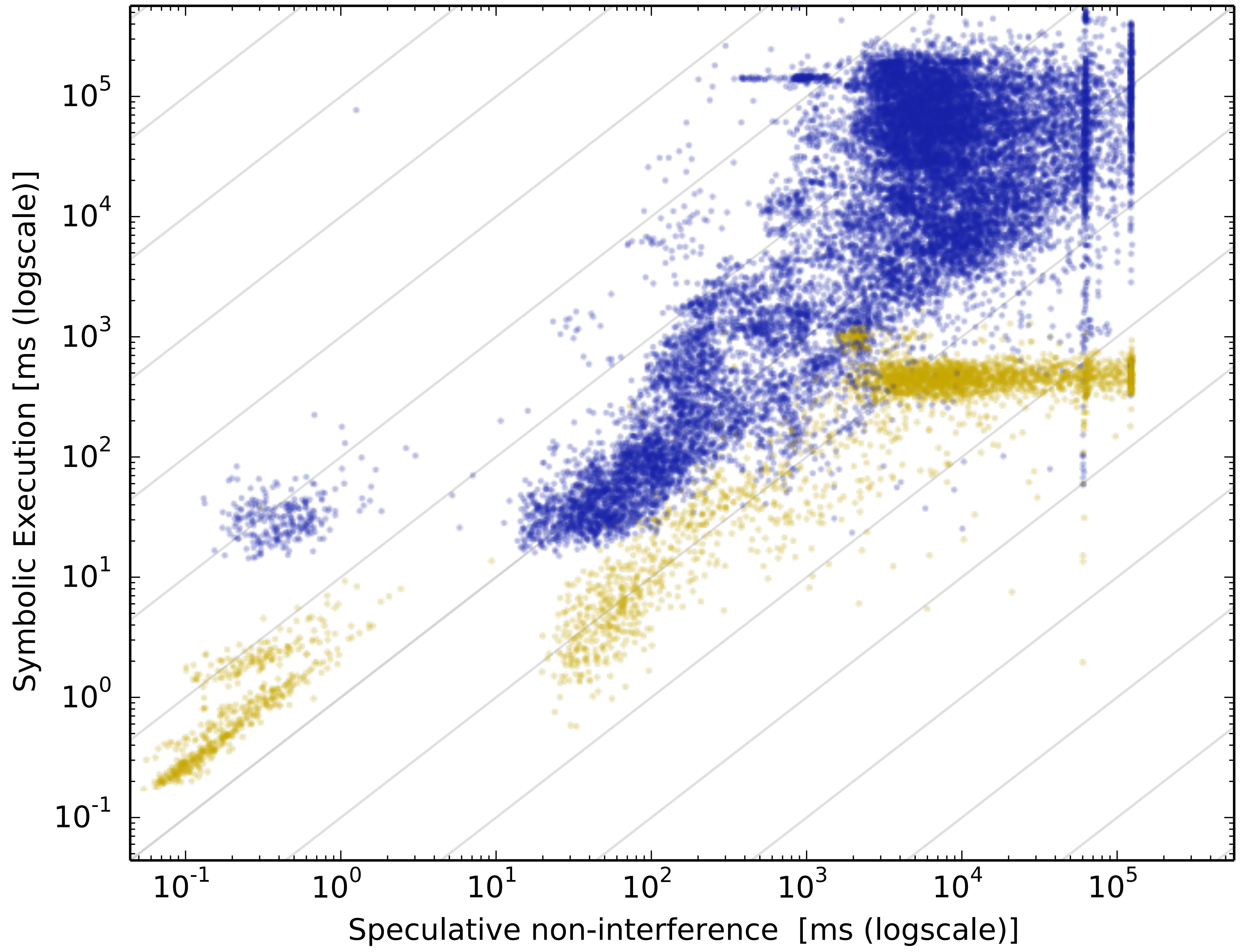


Results

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces
- SNI ≤10x slower
  - 26.9% traces
- SNI 10x-100x slower
  - 7.9% traces



Speculative non-interference

Formally!

Program **P** is **speculatively non-interferent** for prediction oracle ***O*** if

For al…
P_{nc}…

See paper for: r…

Spectector

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

x64 to µASM

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

Symbolic execution

Spectector

https://spectector.github.io

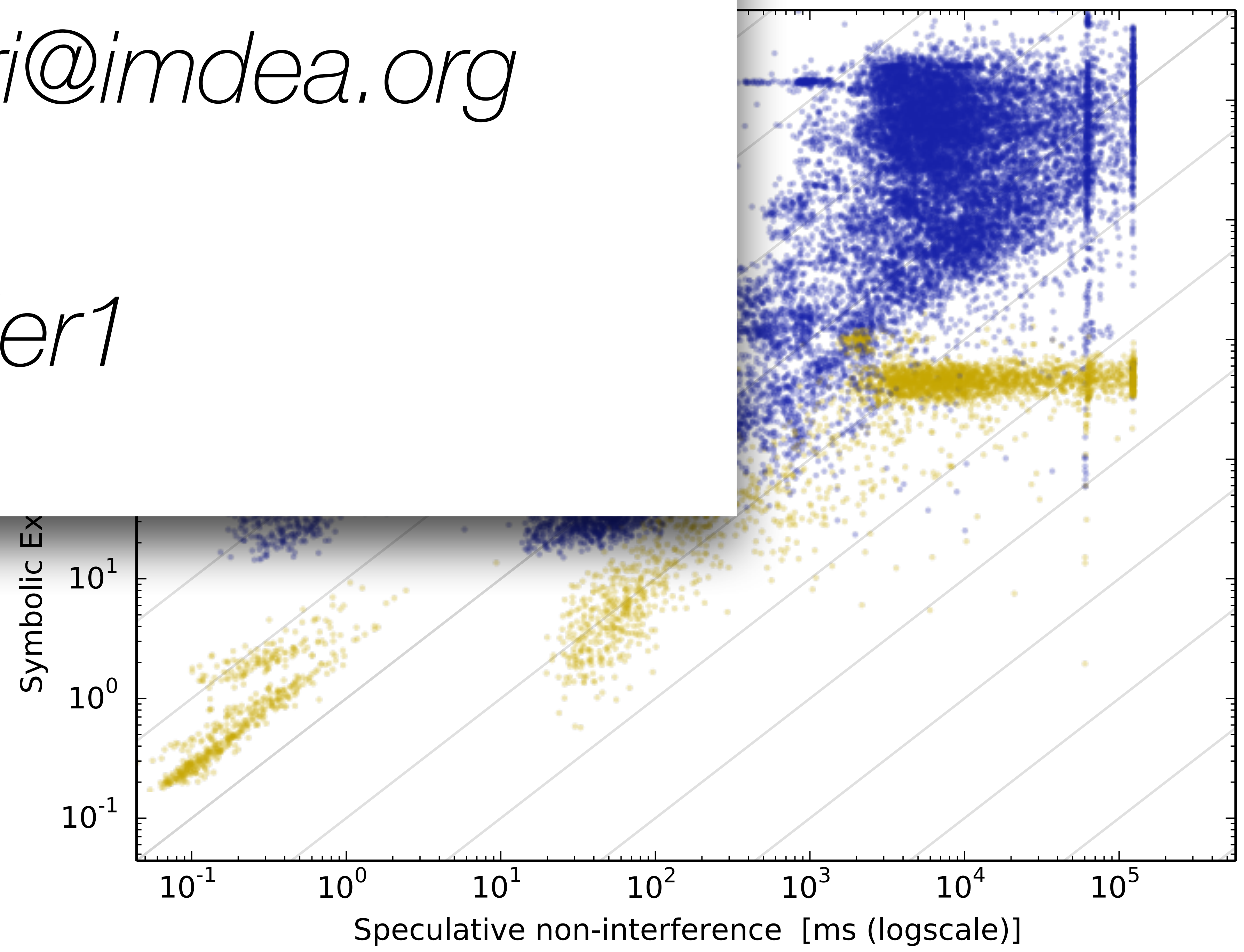
marco.guarnieri@imdea.org

@MarcoGuarnier1

Results

Ex.	Vcc Unp -O0	Vcc Unp -O2	Vcc Fen 19.15 -O0	Vcc Fen 19.15 -O2												
01	○	○	●	●												
02	○	○	●	●												
03	○	○	●	○												
04	○	○	○	○												
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

- SNI ≤10x slower
  - 26.9% traces
- SNI 10x-100x slower
  - 7.9% traces



Backup

Speculative non-interference

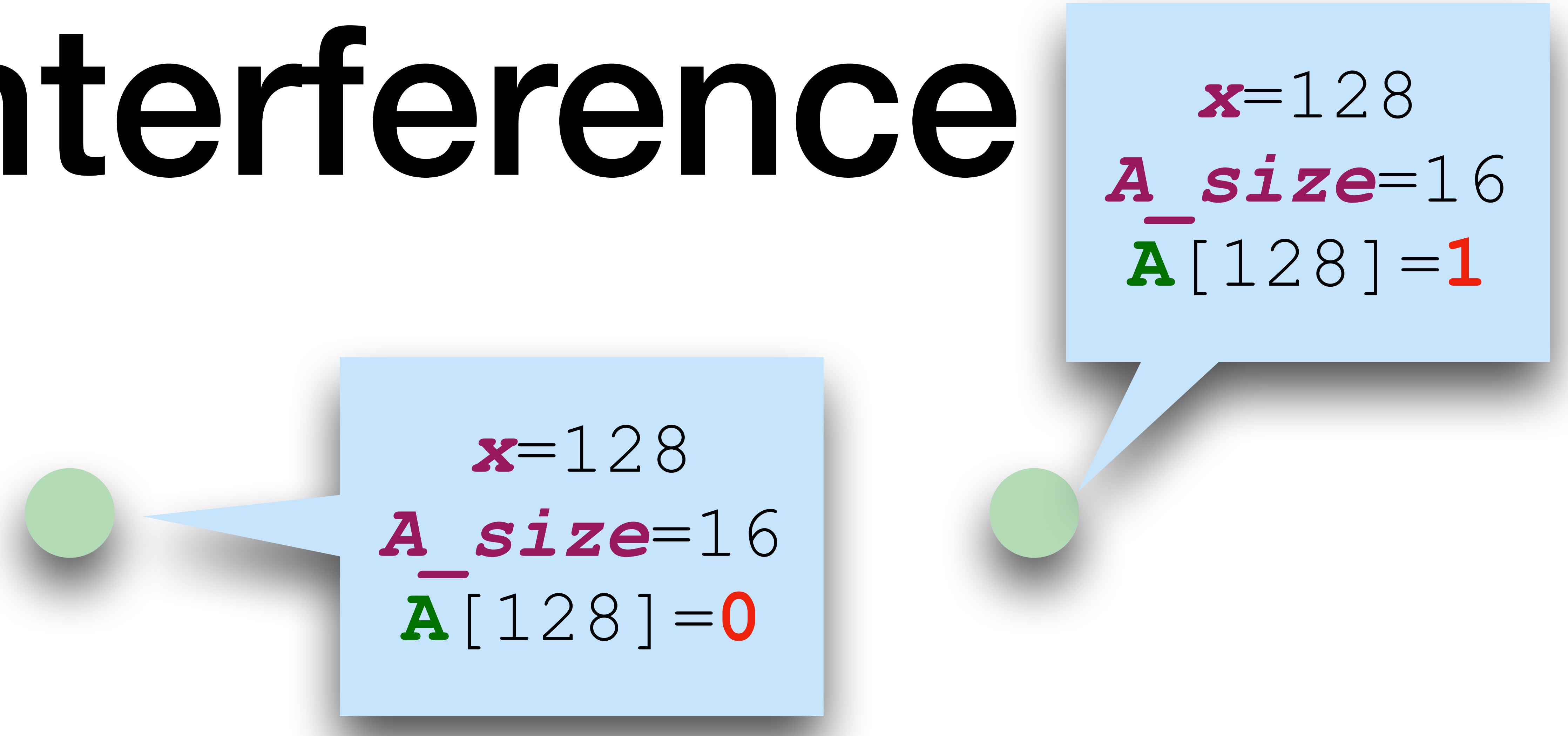
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```

Speculative non-interference

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

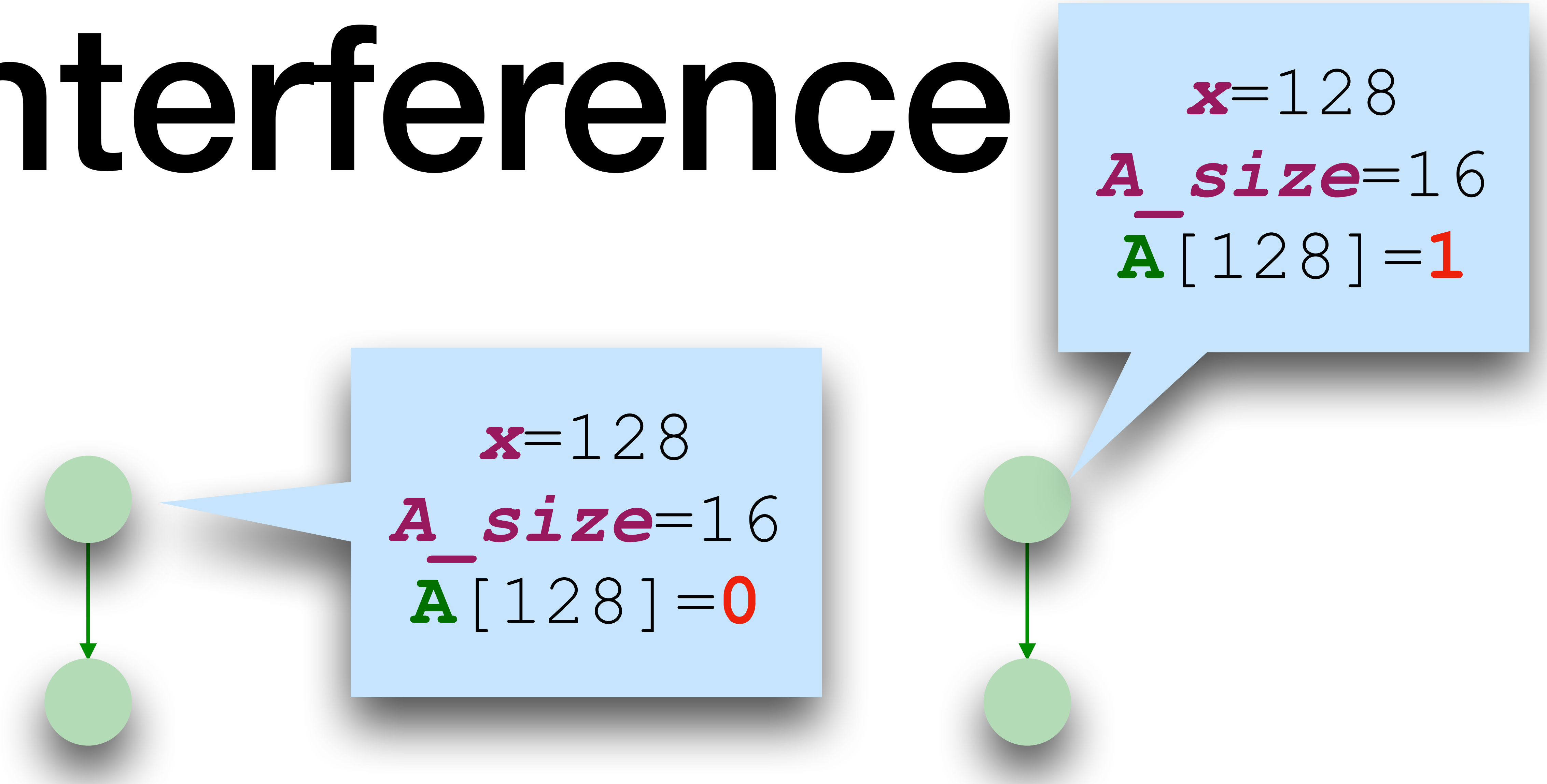
Speculative non-interference

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



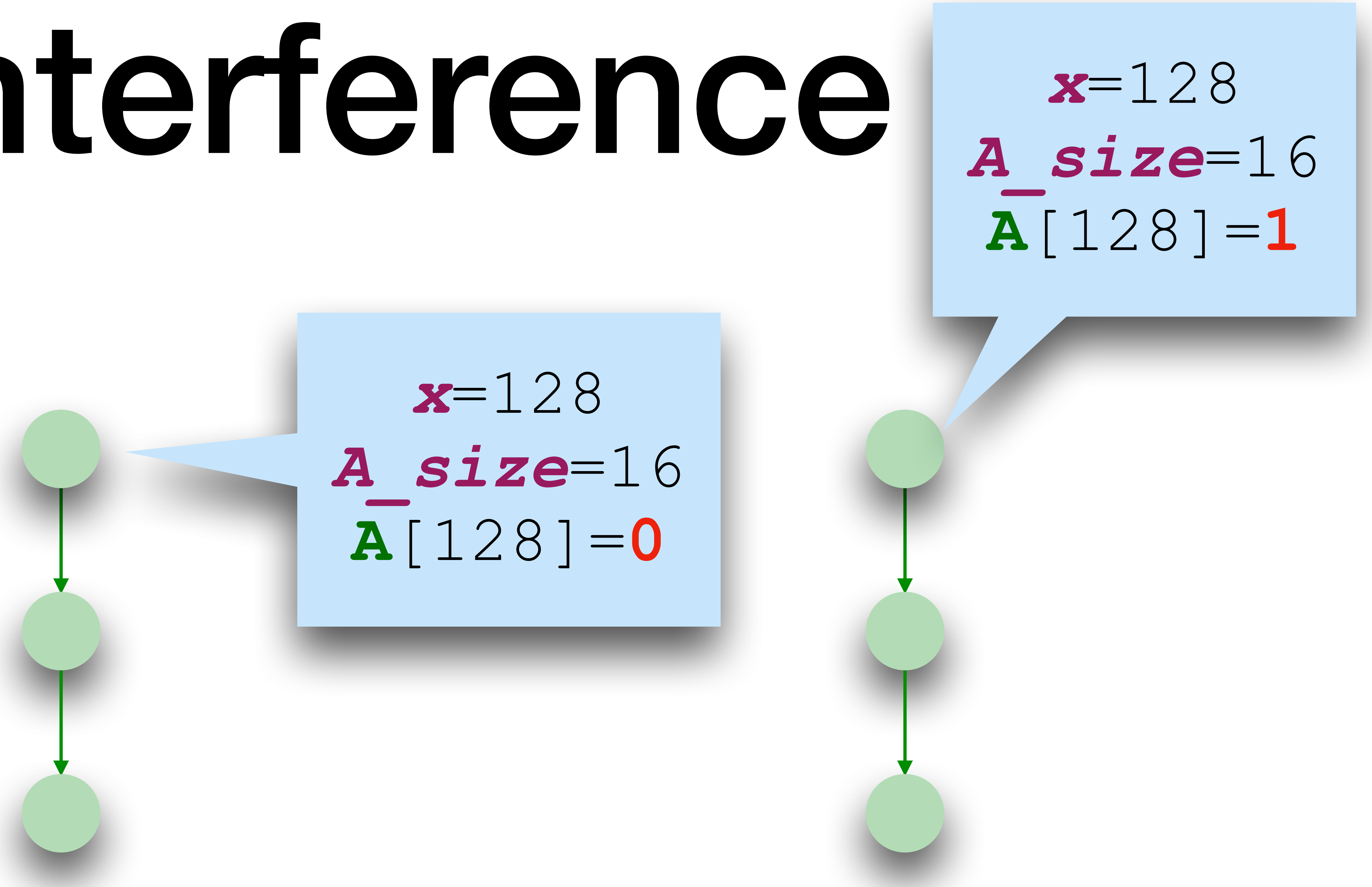
Speculative non-interference

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



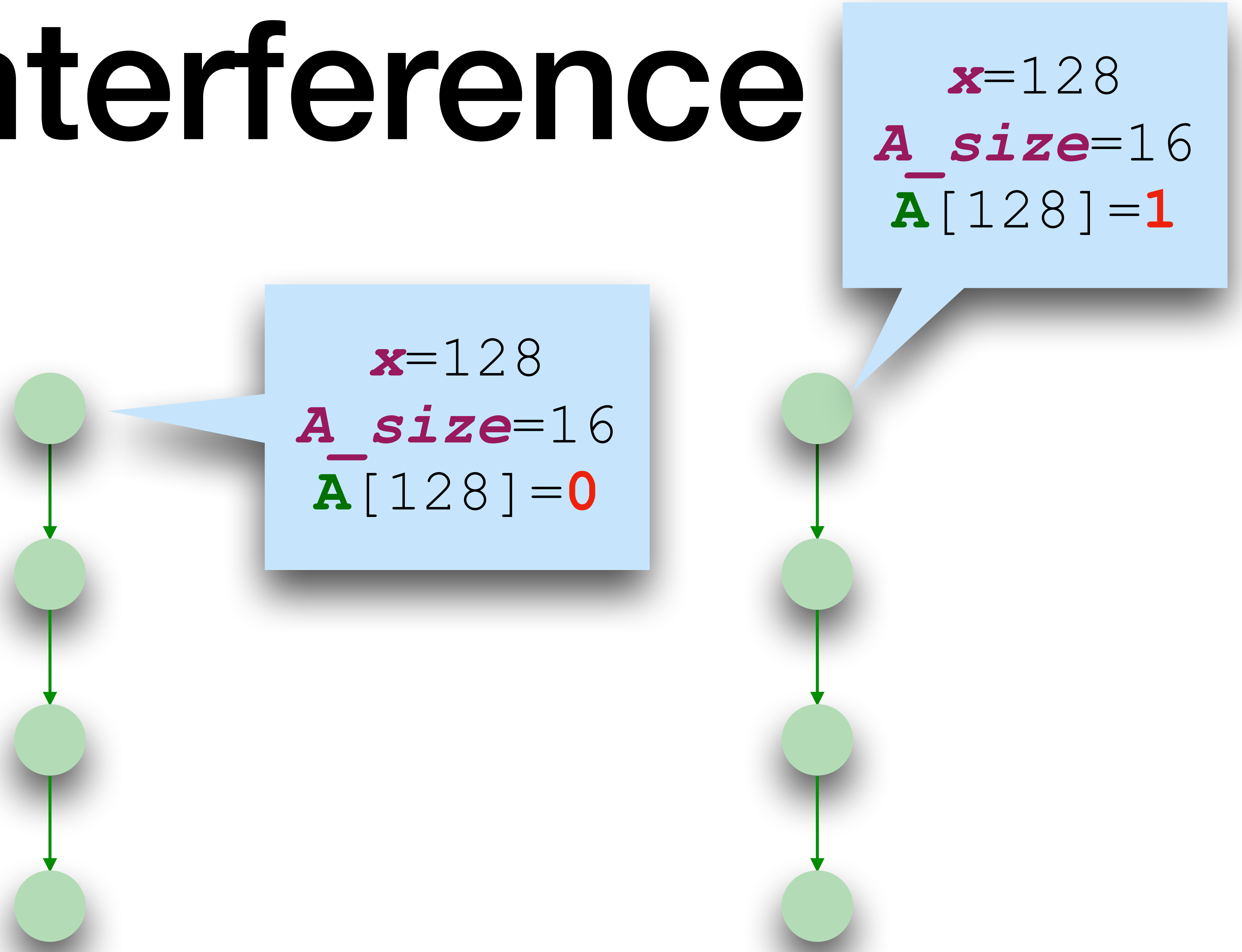
Speculative non-interference

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



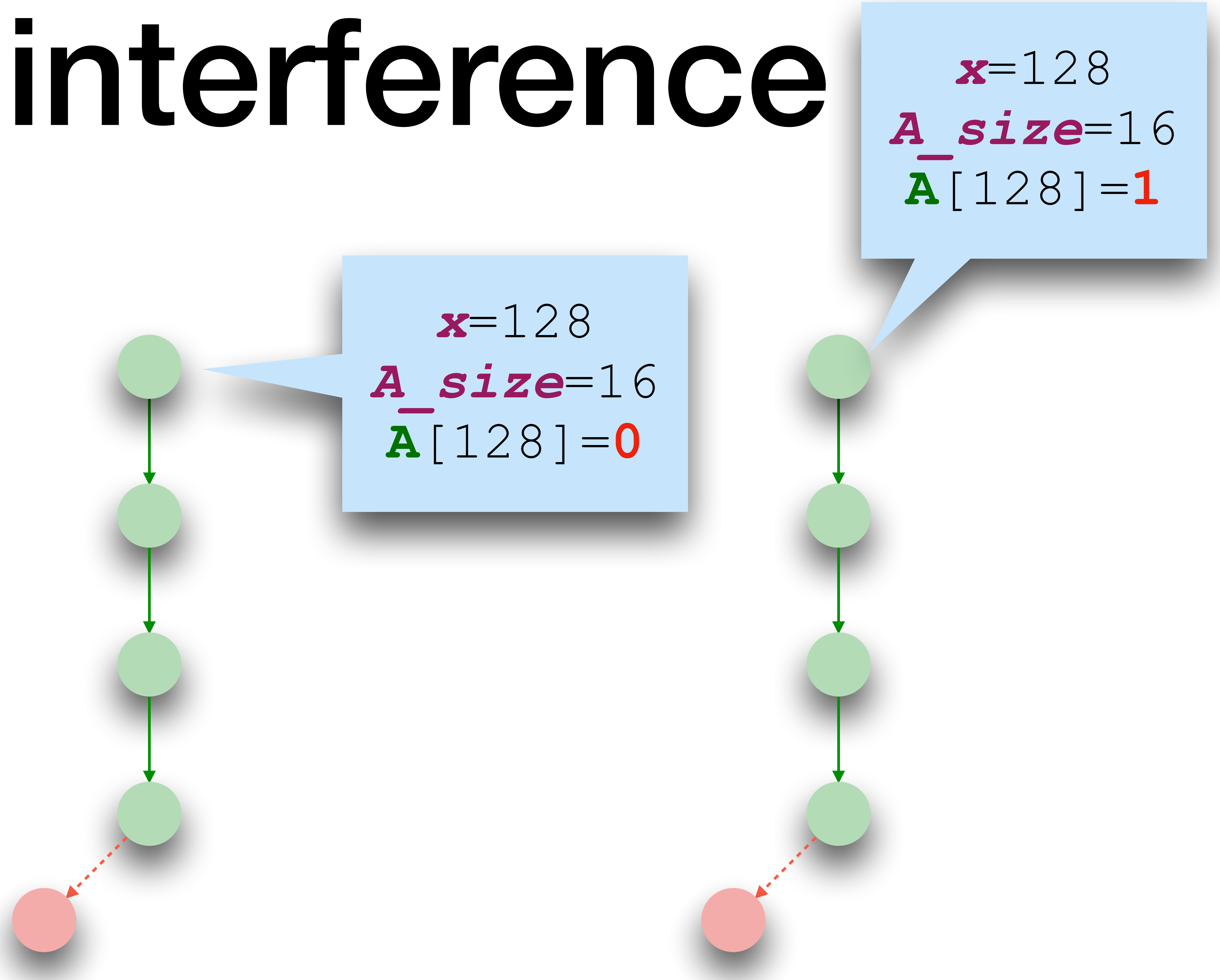
Speculative non-interference

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



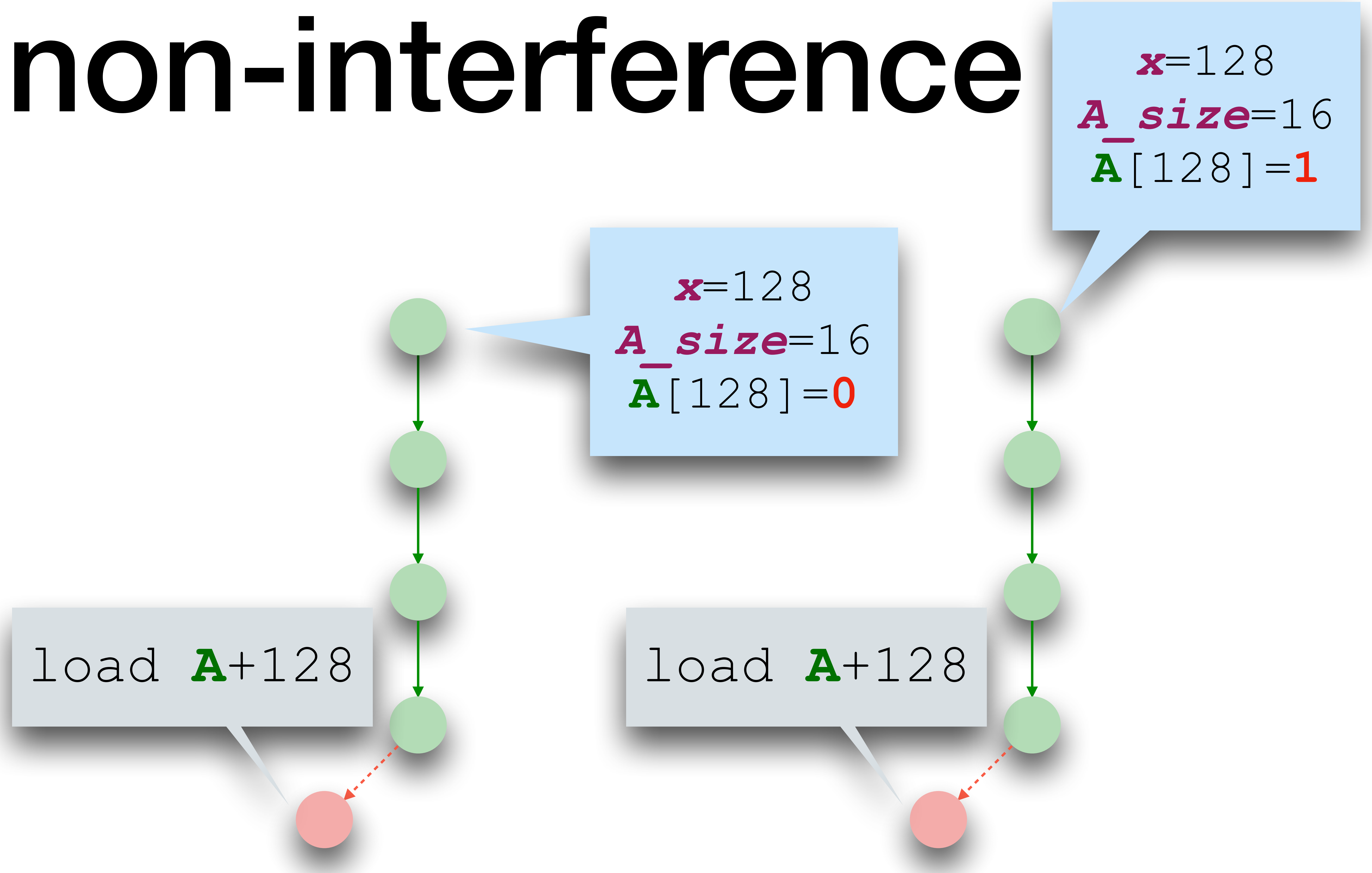
Speculative non-interference

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



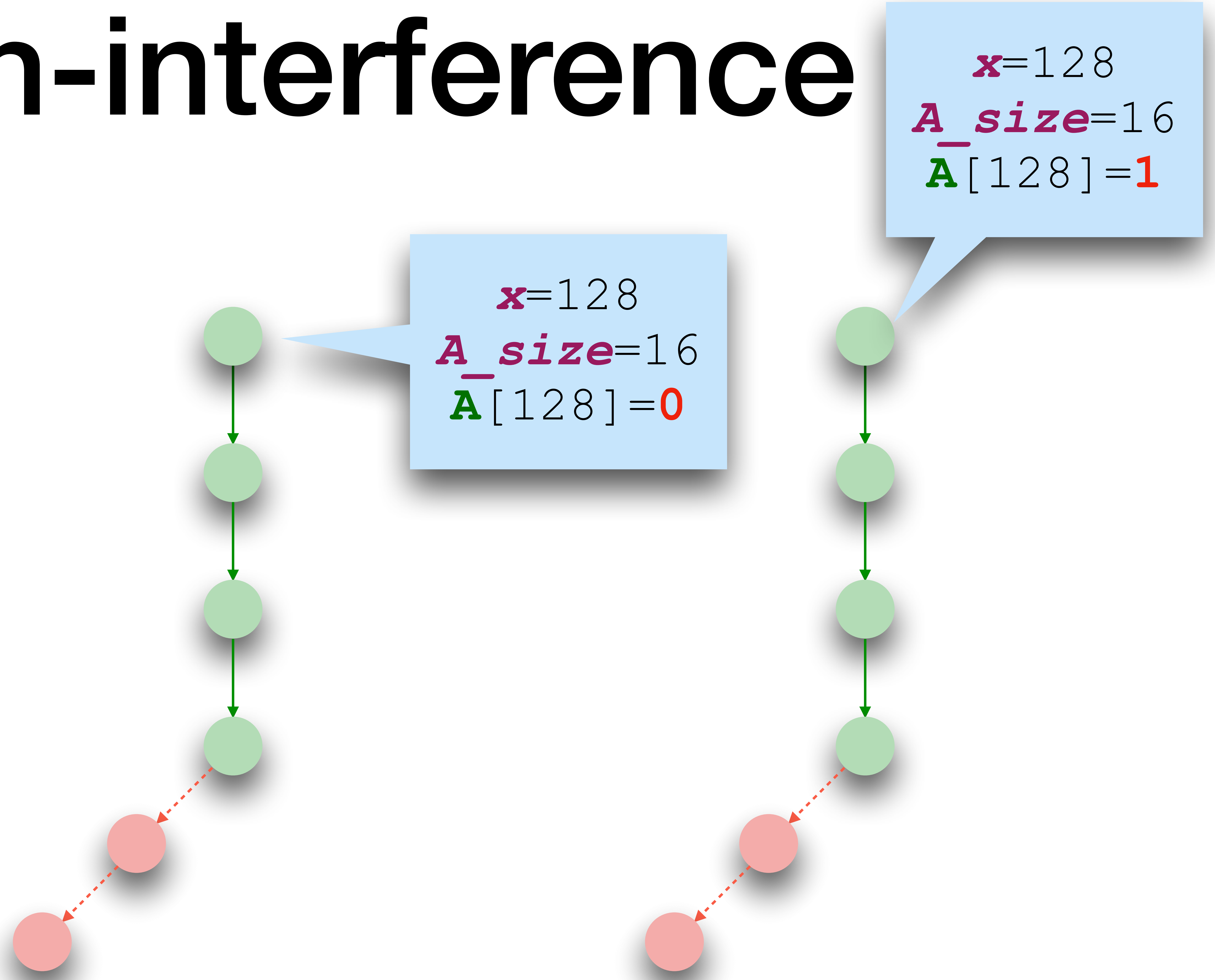
Speculative non-interference

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



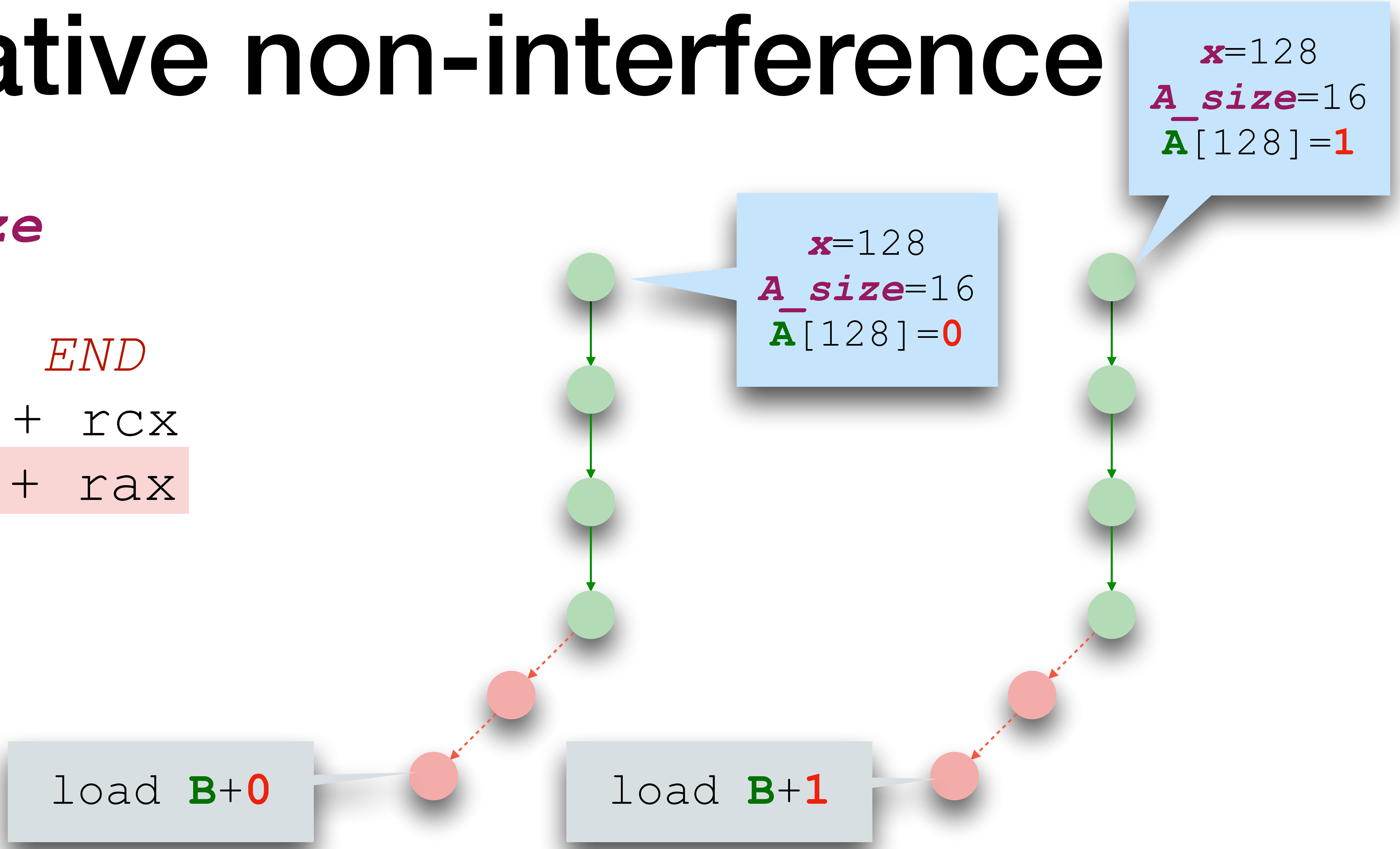
Speculative non-interference

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



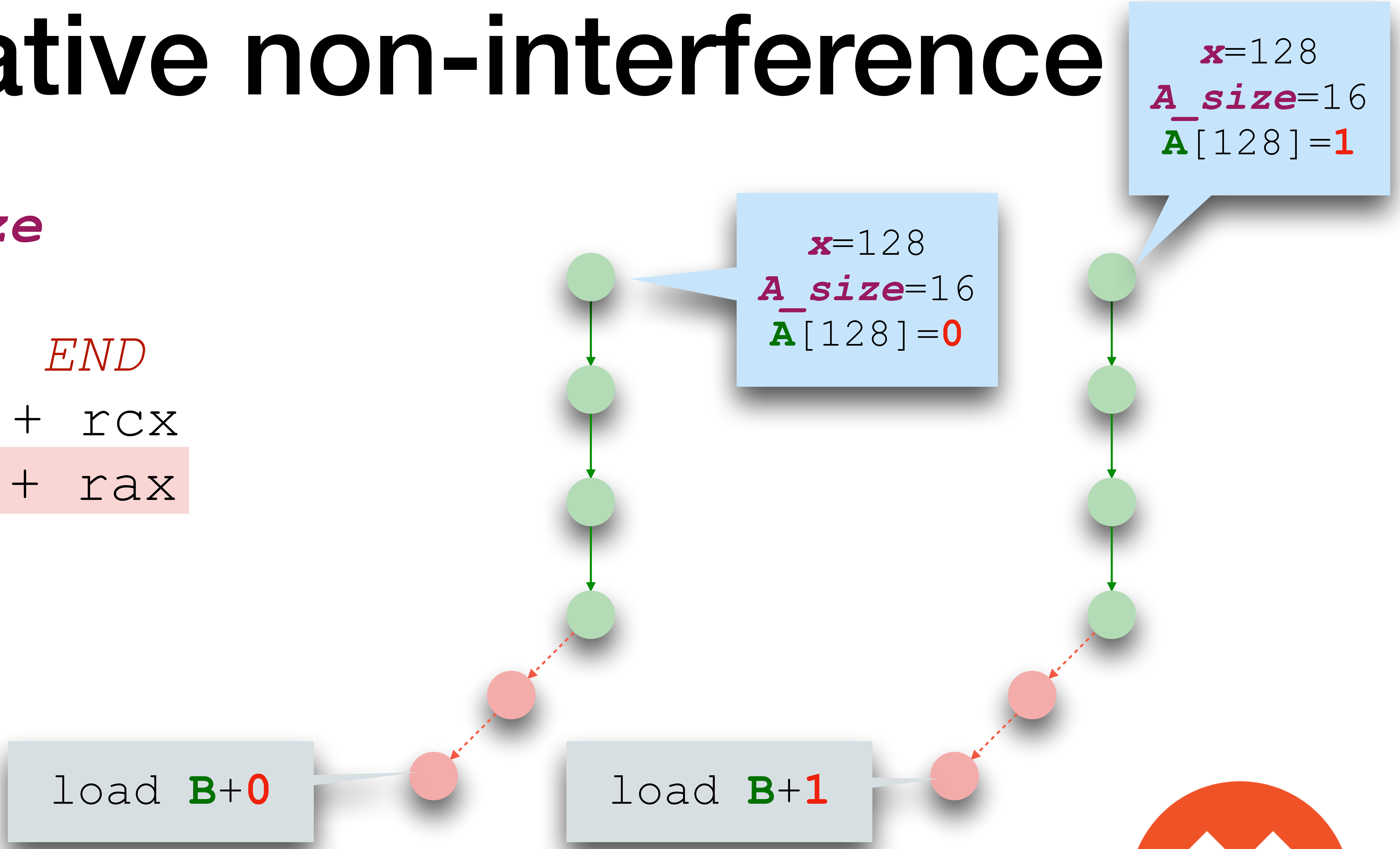
Speculative non-interference

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



Speculative non-interference

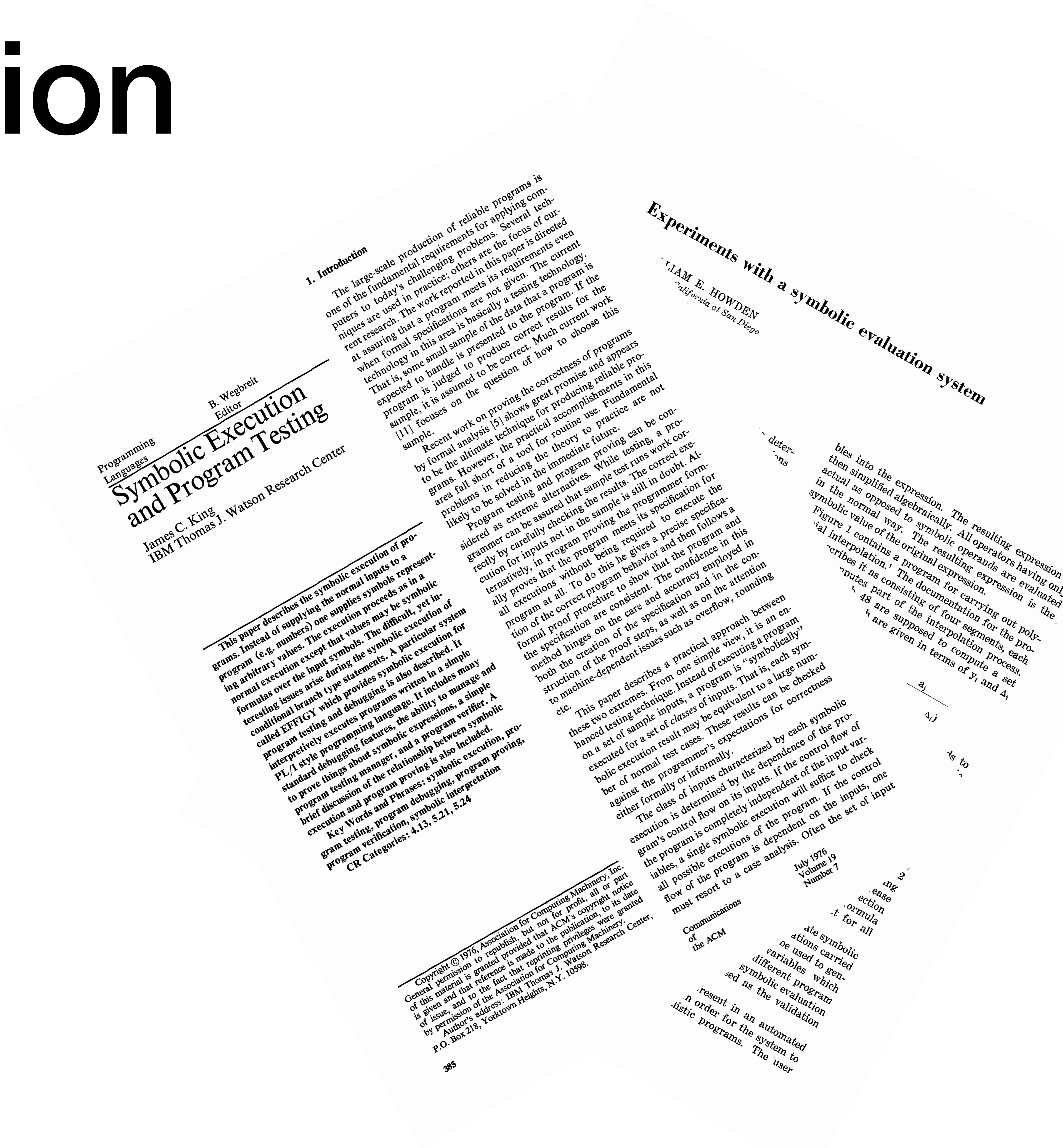
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Symbolic execution

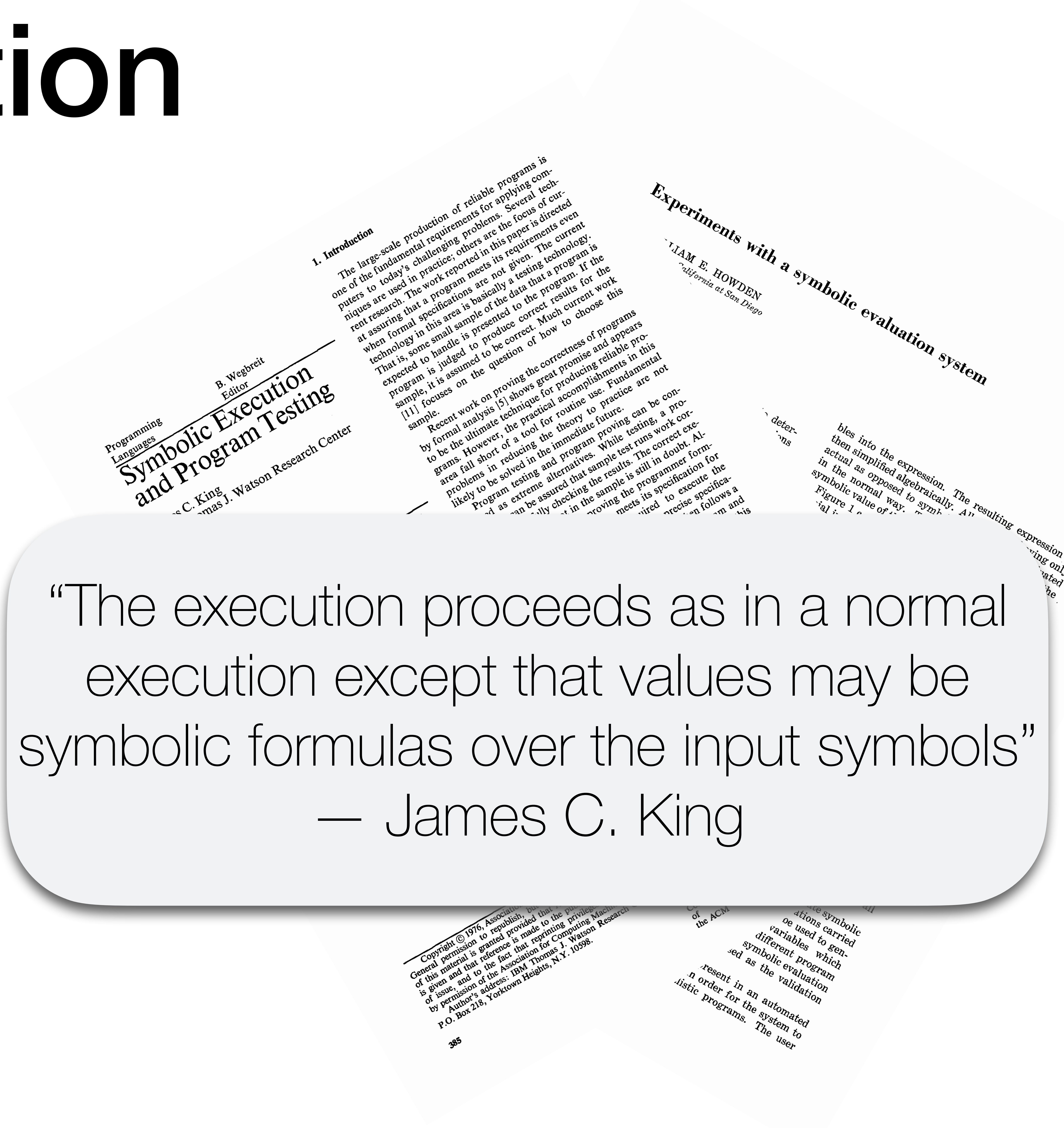
Symbolic execution

- Program analysis technique



Symbolic execution

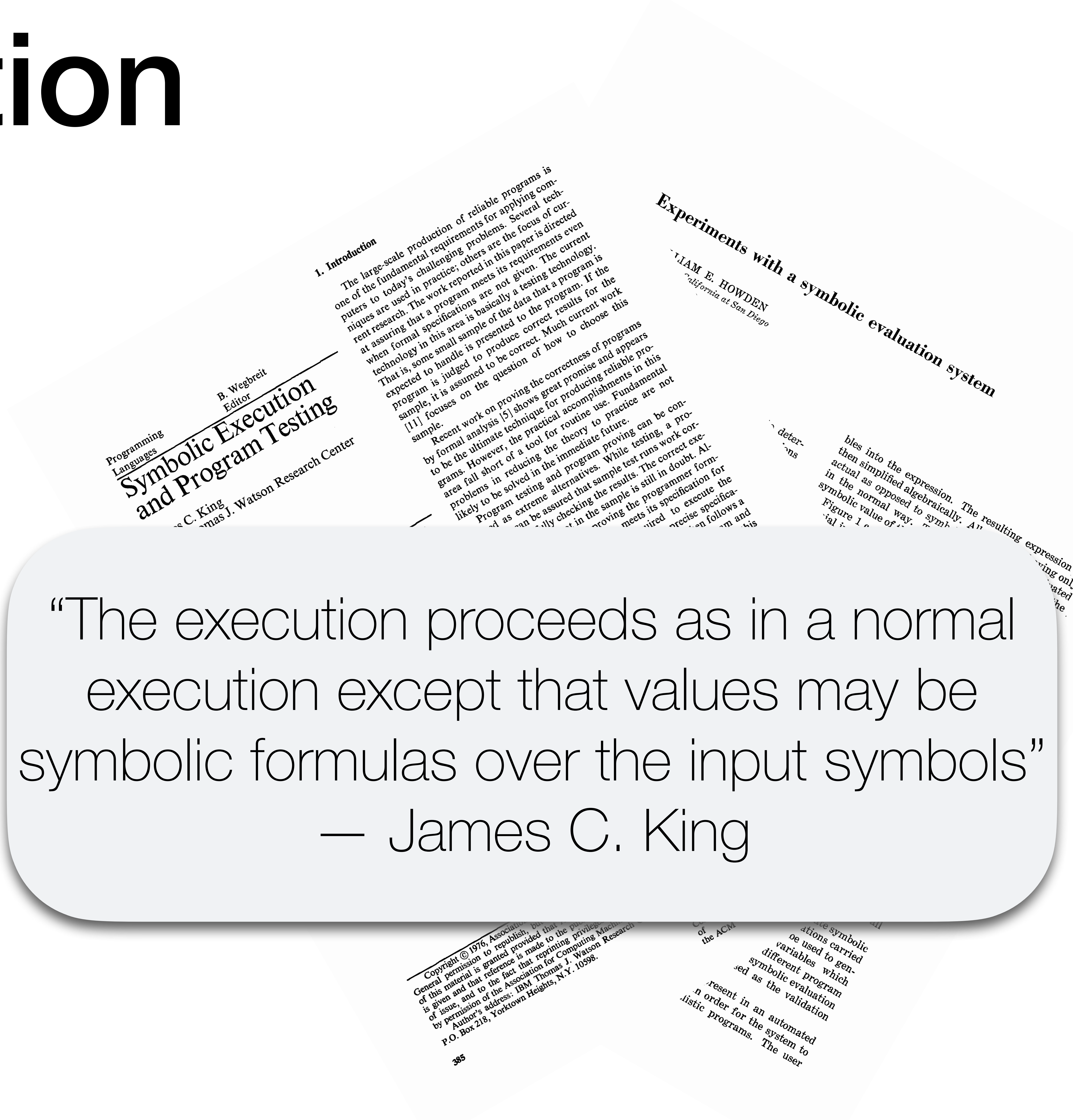
- Program analysis technique
- Execute programs over symbolic values



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

Symbolic execution

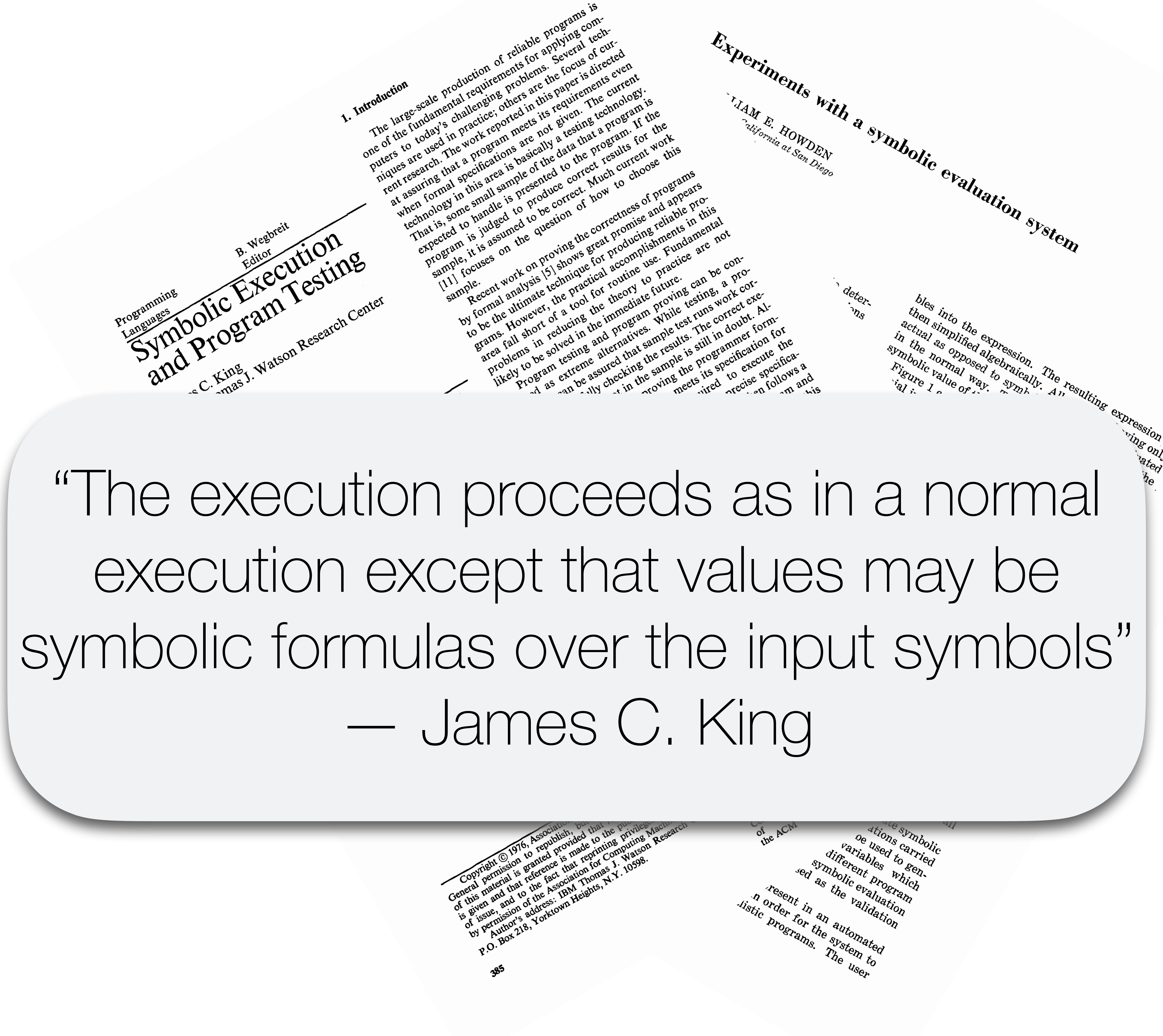
- Program analysis technique
- Execute programs over symbolic values
  - Explore all paths, each with its own path constraint



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

Symbolic execution

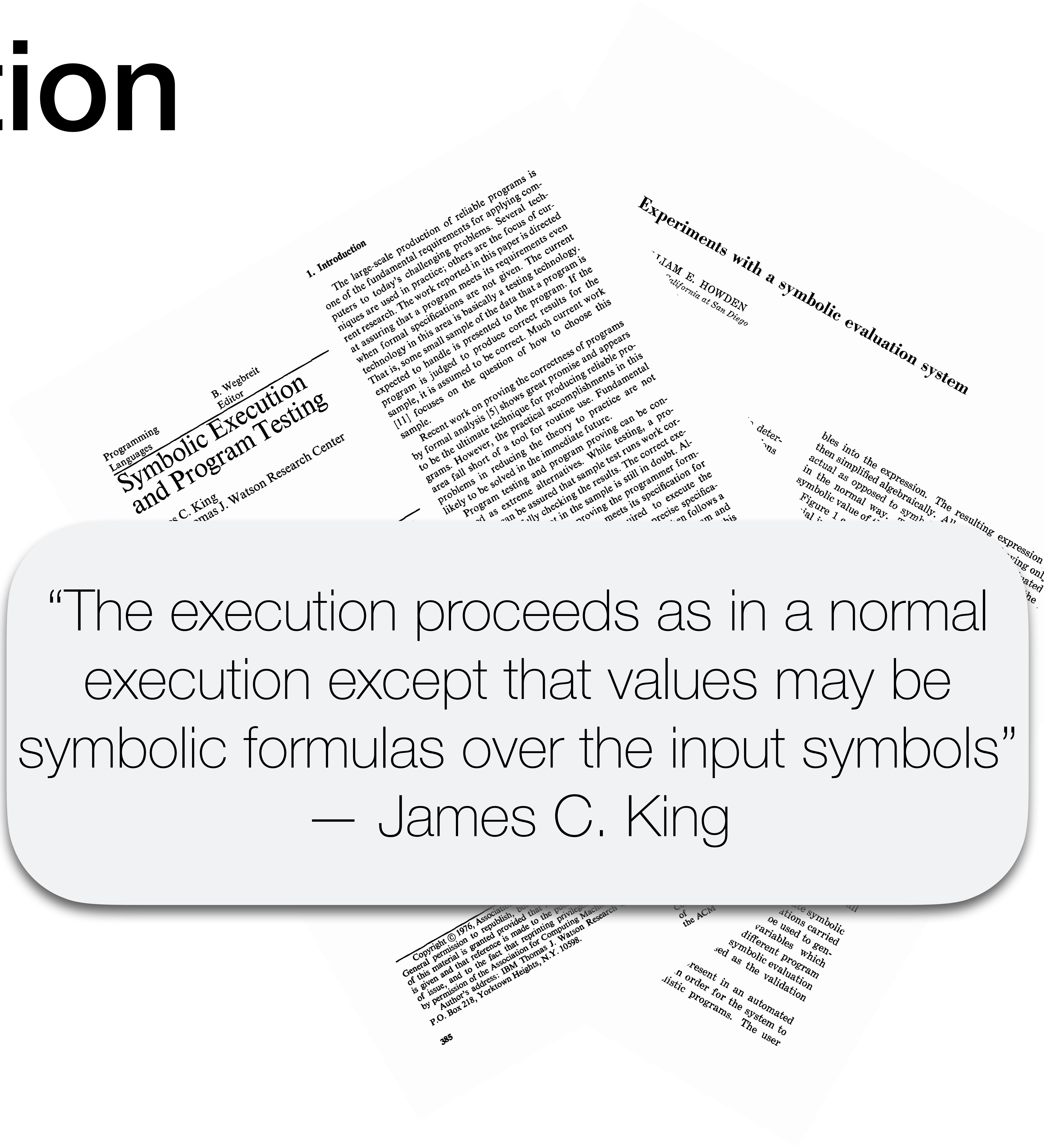
- Program analysis technique
- Execute programs over symbolic values
  - Explore all paths, each with its own path constraint
  - Each path represents all concrete executions satisfying the constraint



"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
  - Each path represents all concrete executions satisfying the constraint
  - Branch and jump instructions: fork paths and update path constraint



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

Symbolic execution

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

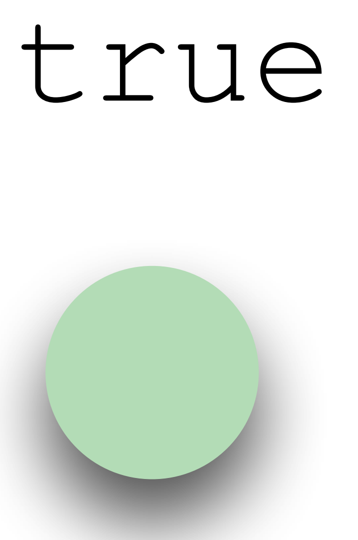
Symbolic execution

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

Always mispredict branch instructions

Symbolic execution

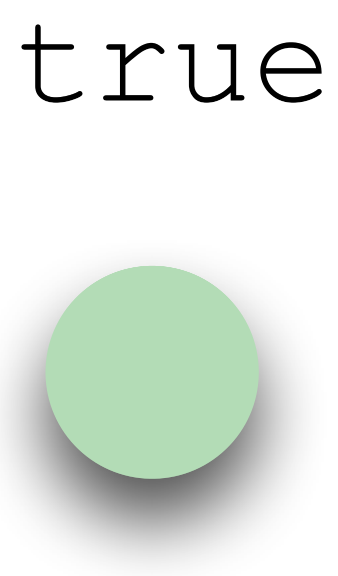
```
if (x < A_size)
    y = A[x]
    z = B[y]
 end
```



Always mispredict
branch instructions

Symbolic execution

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



Always mispredict
branch instructions

Symbolic execution

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

$x \geq$ *A_size*

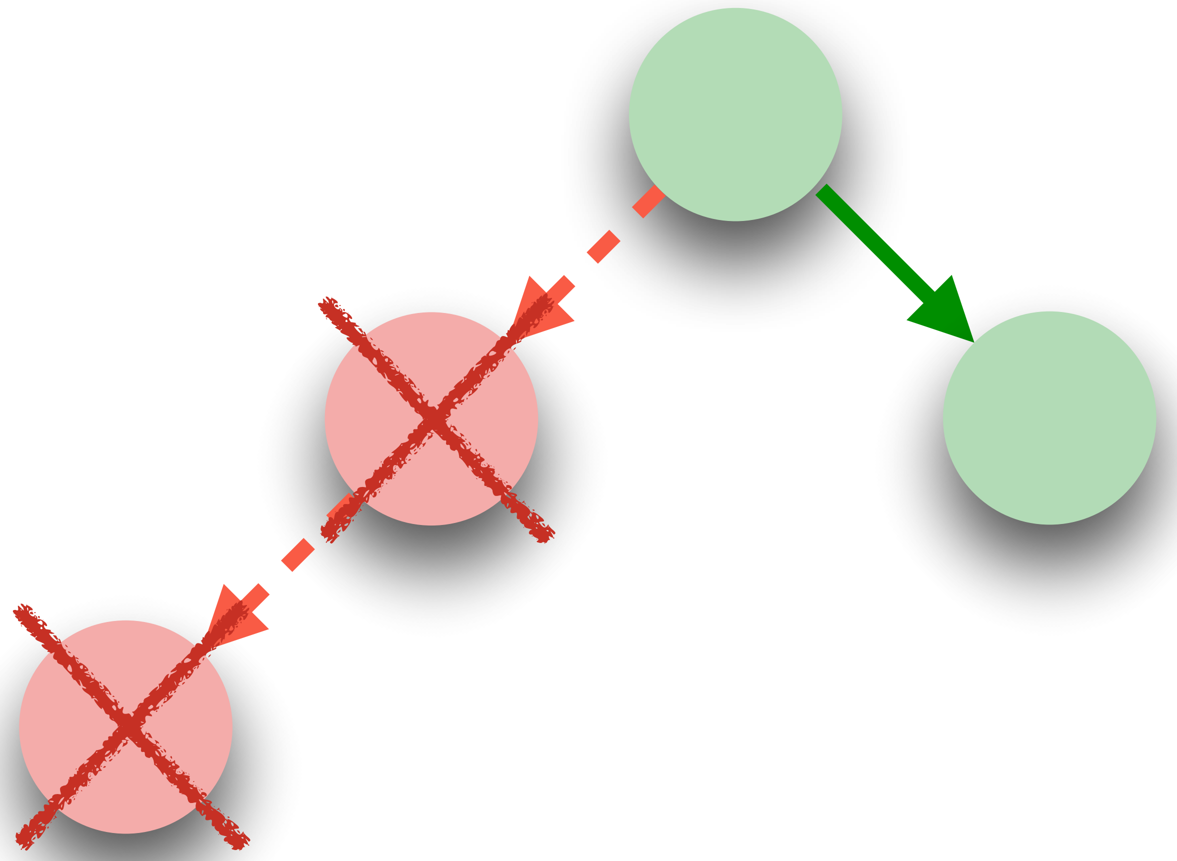
$x <$ *A_size*

Always mispredict branch instructions

Symbolic execution

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

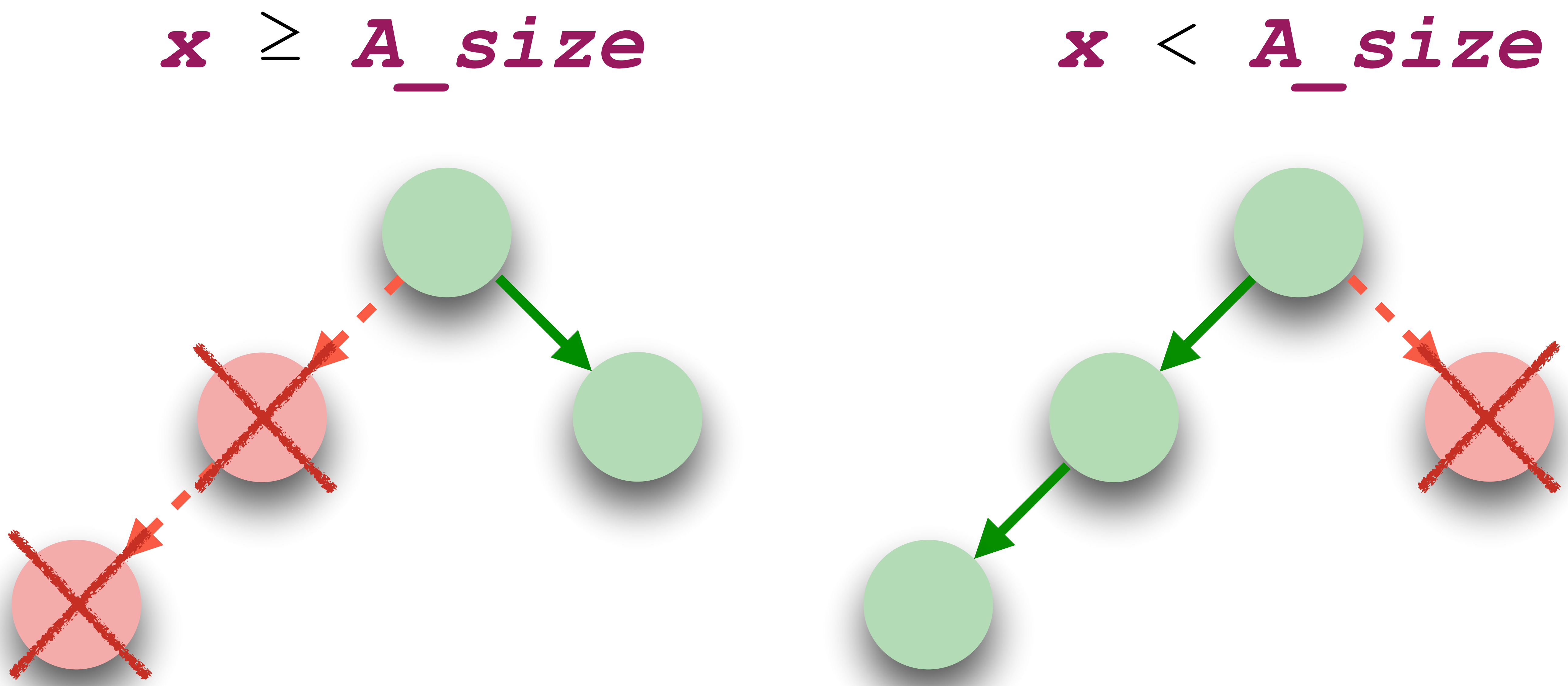
$x \geq A\_size$ $\quad$ $x < A\_size$



Always mispredict branch instructions

Symbolic execution

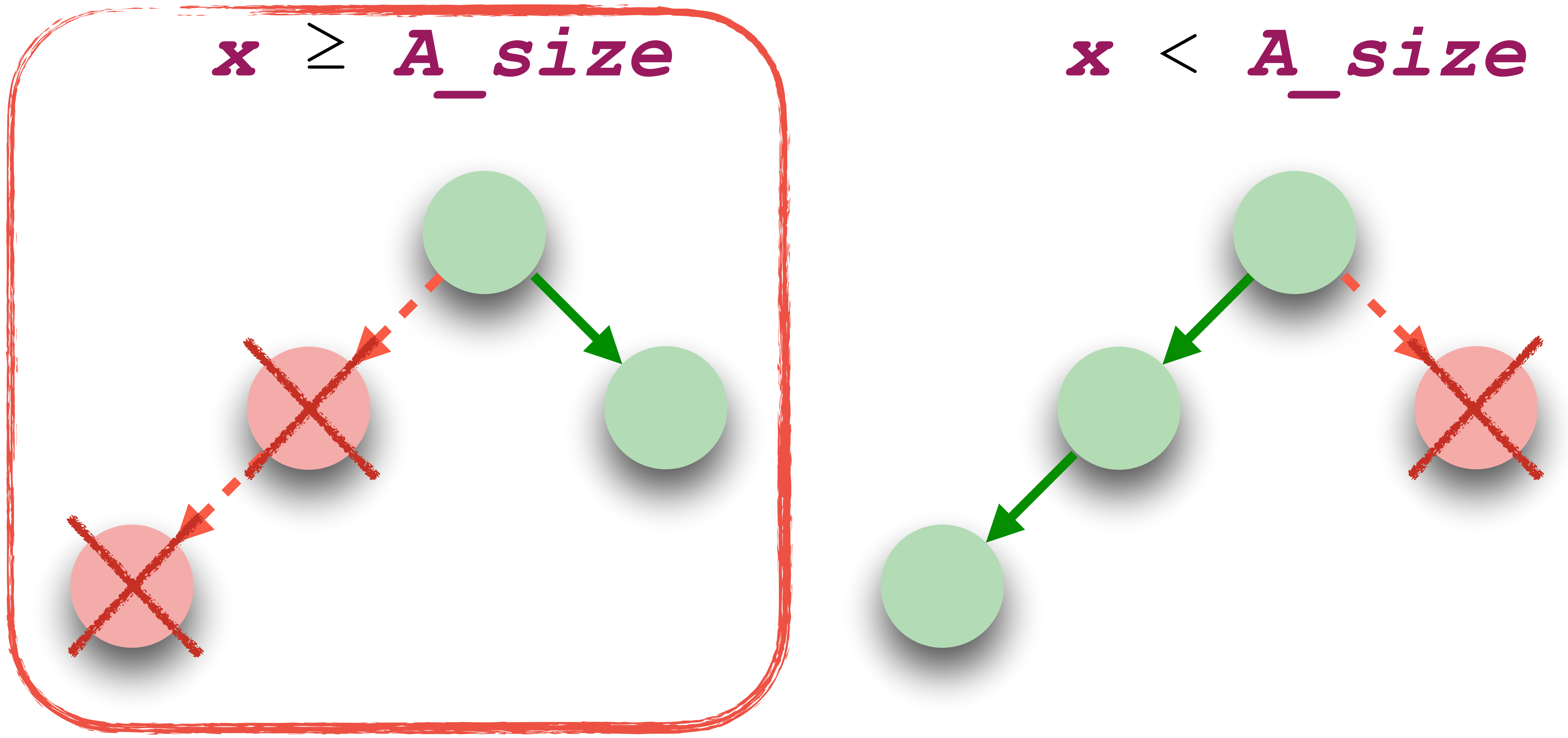
```
if (x < A_size)
    y = A[x]
    z = B[y]
end
```



Always mispredict
branch instructions

Symbolic execution

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

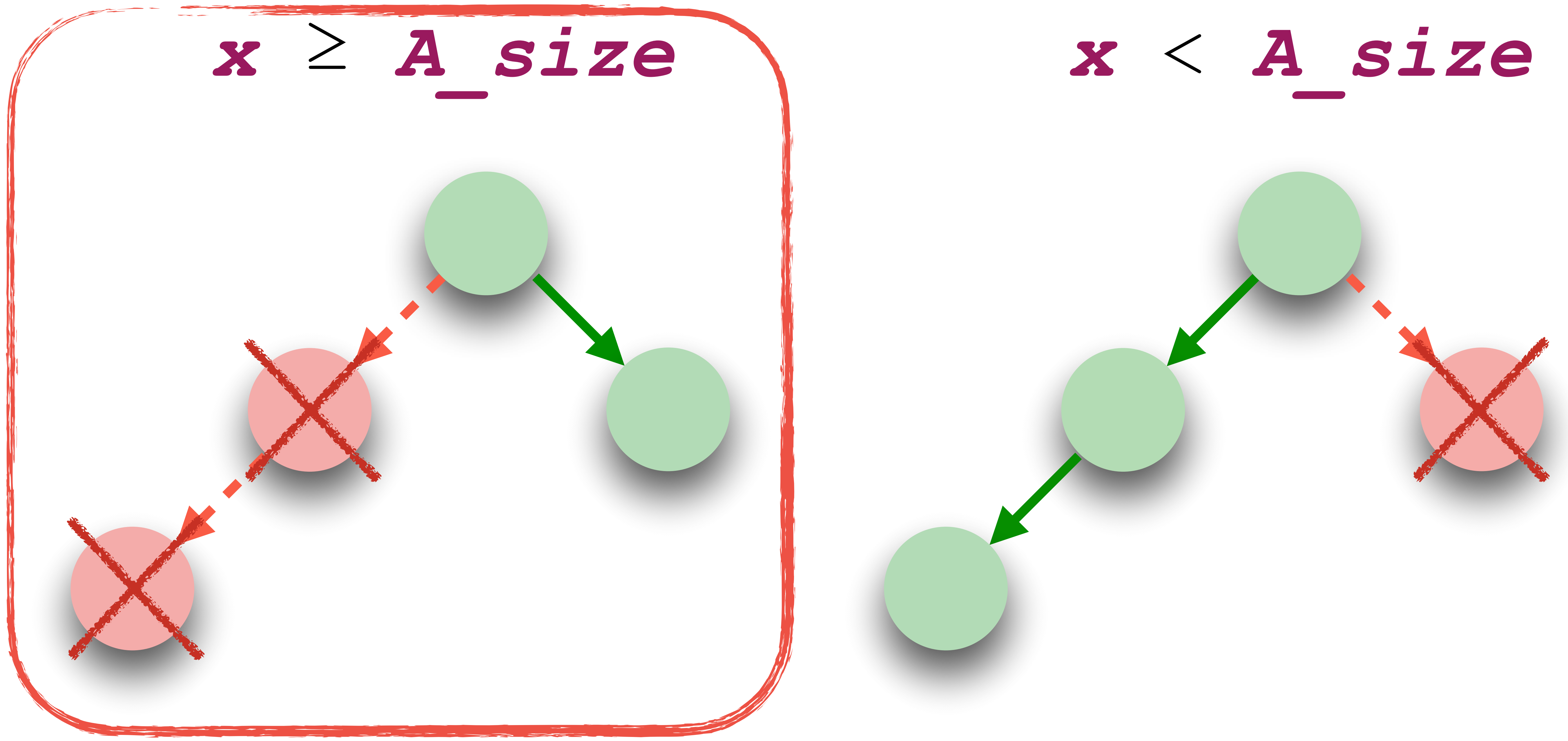


Always mispredict
branch instructions

Symbolic execution

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

Always mispredict branch instructions

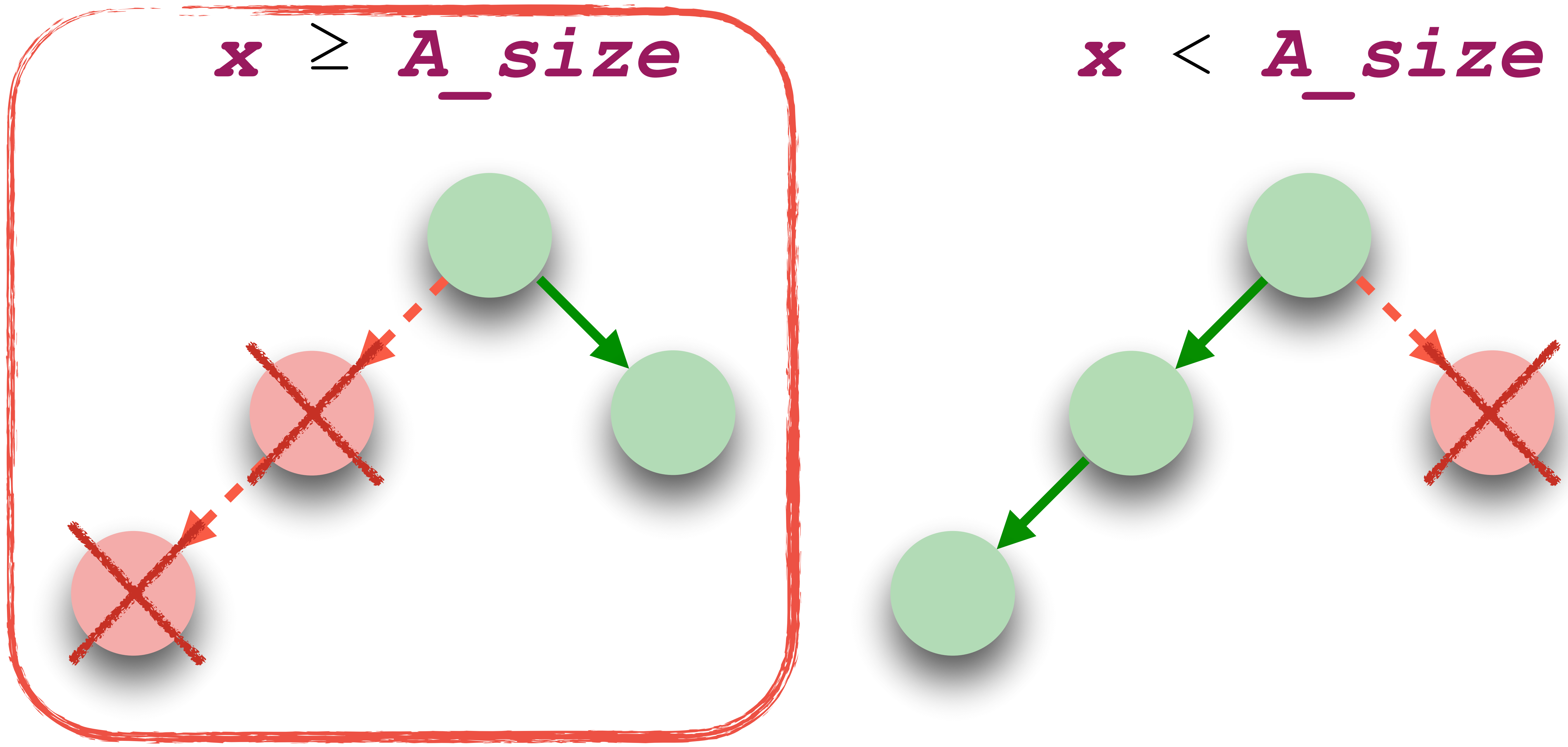


start pc *2* load **A**+x load **B**+**A**[x] rollback pc *4*

Symbolic execution

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

Always mispredict
branch instructions

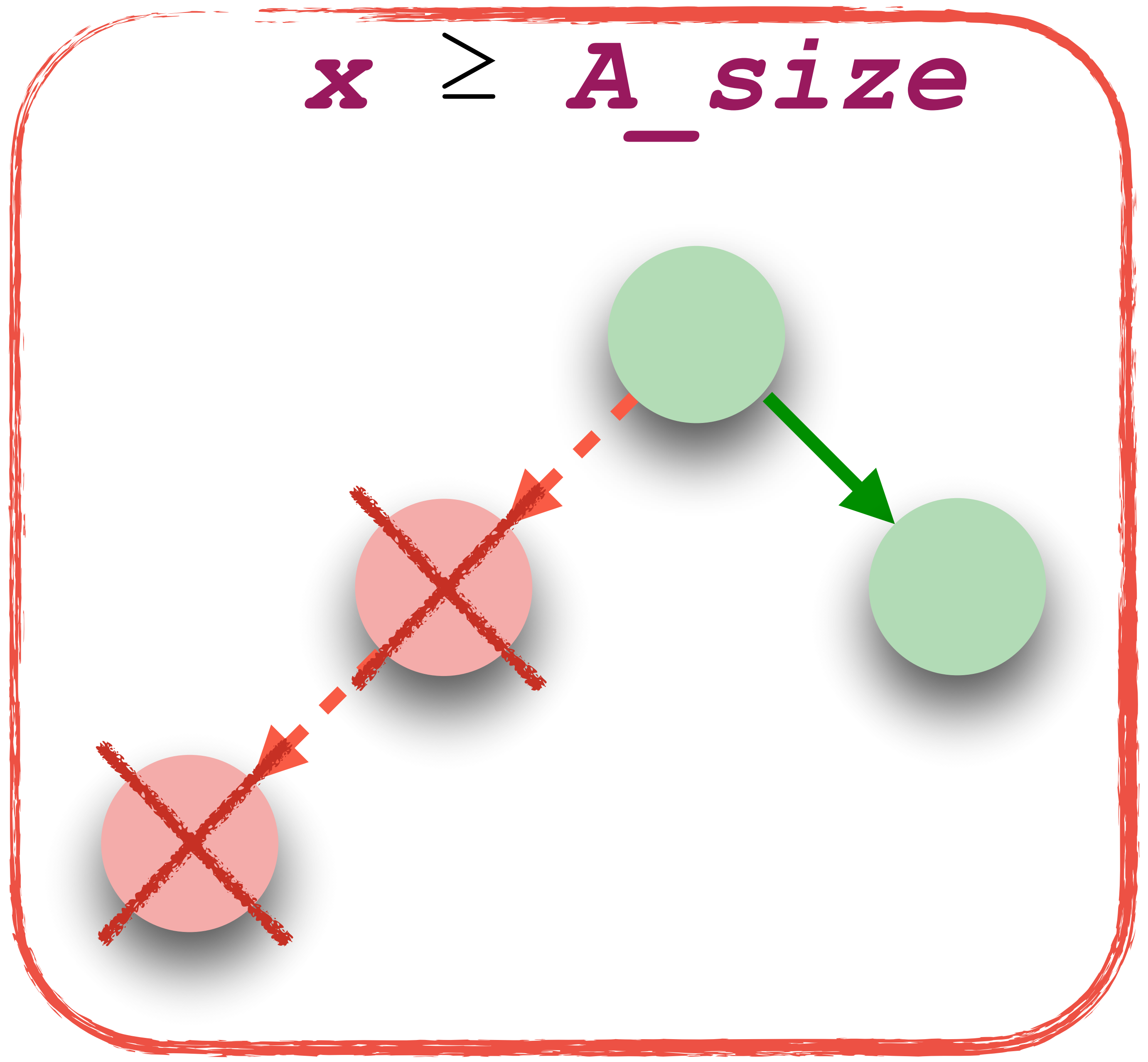


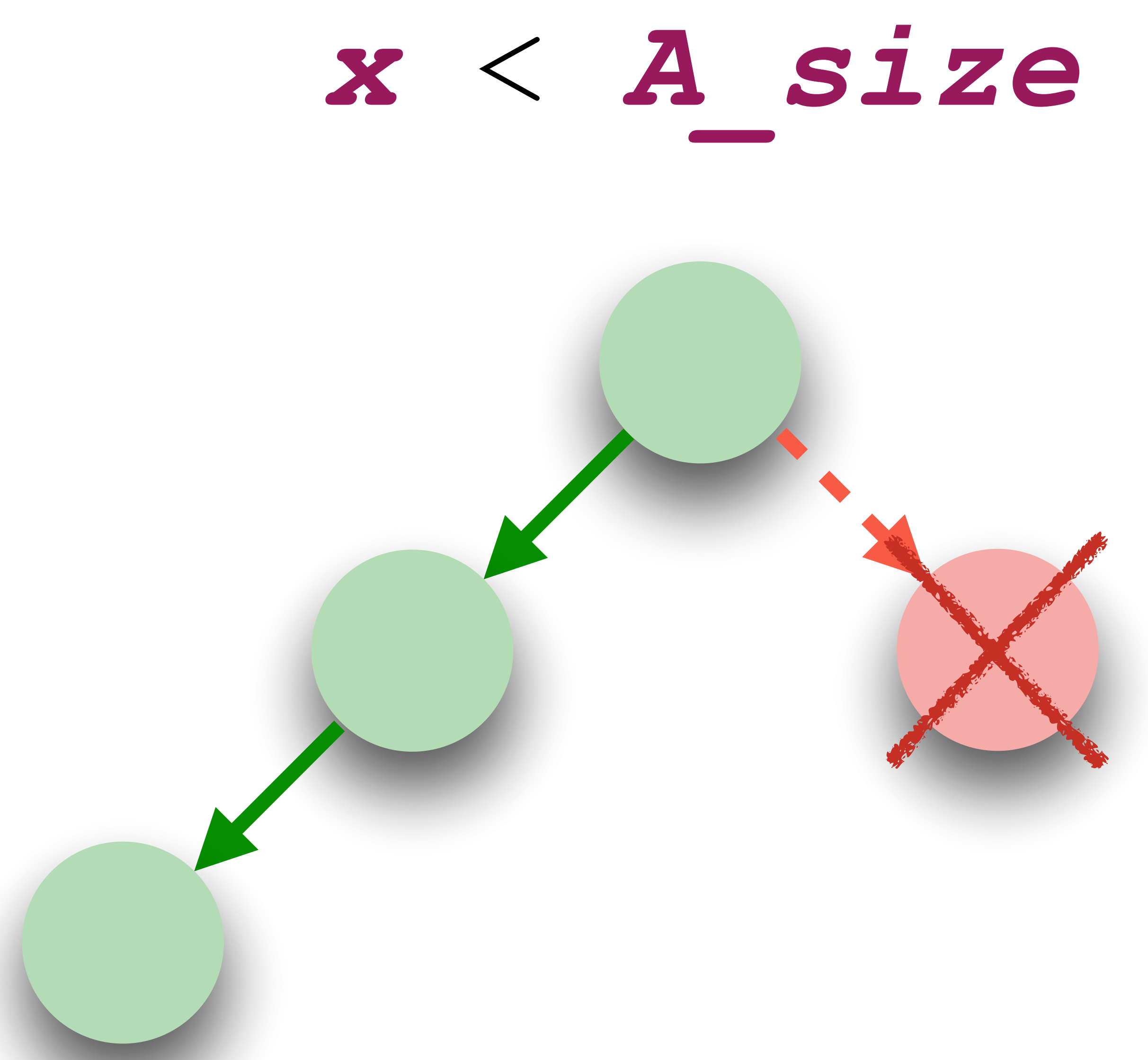
`start pc 2 load A+x load B+A[x] rollback pc 4`

Symbolic execution

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

Always mispredict branch instructions

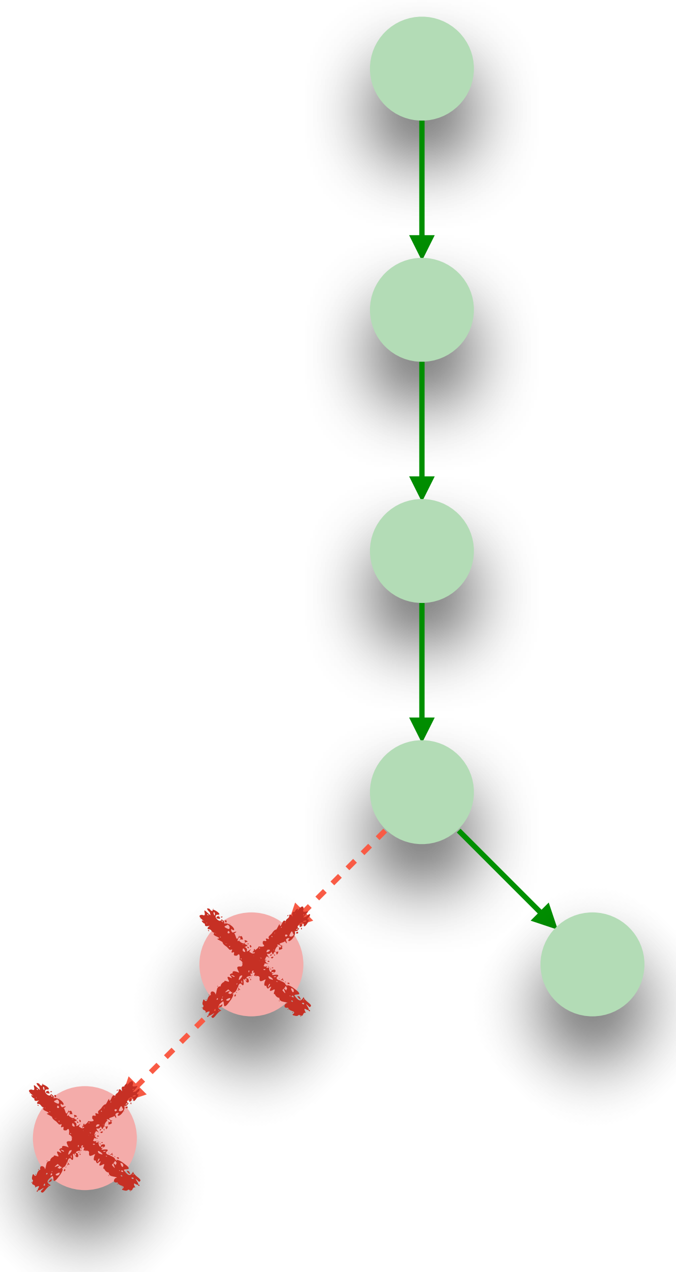




`start pc 2 load A+x load B+A[x] rollback pc 4`

Memory leaks

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



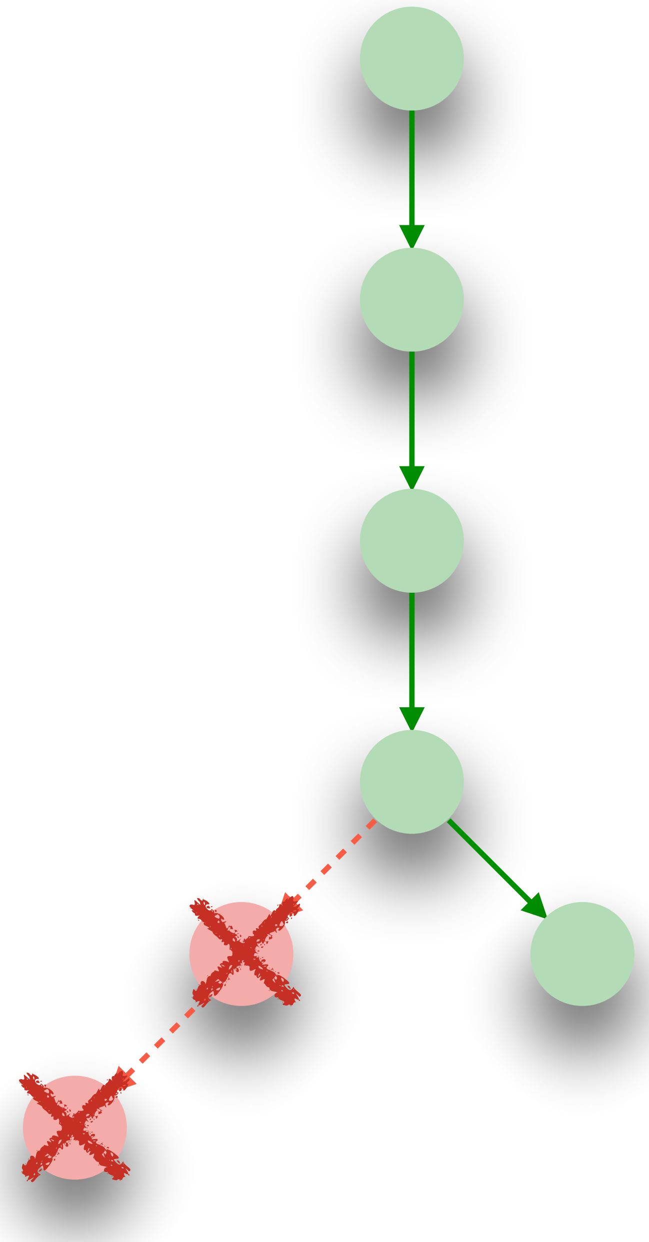
Policy

x, ***A_size***, **A**, **B** are public

`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

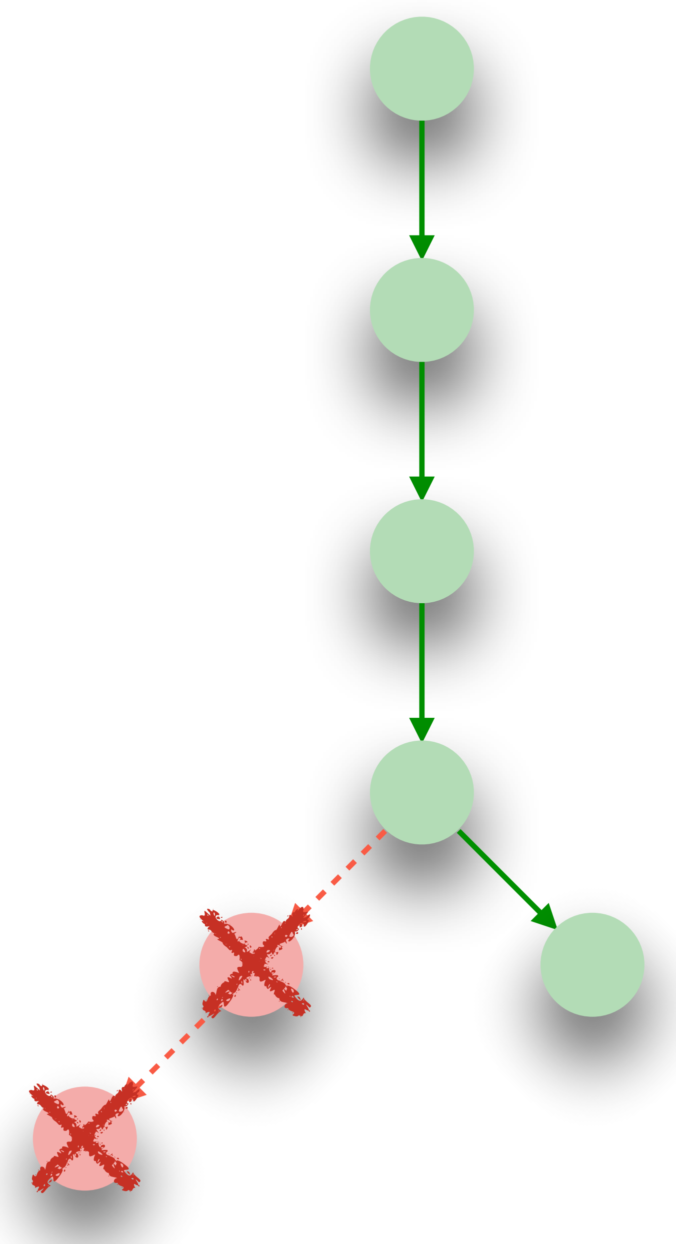
x, **A_size**, **A**, **B** are public

`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

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

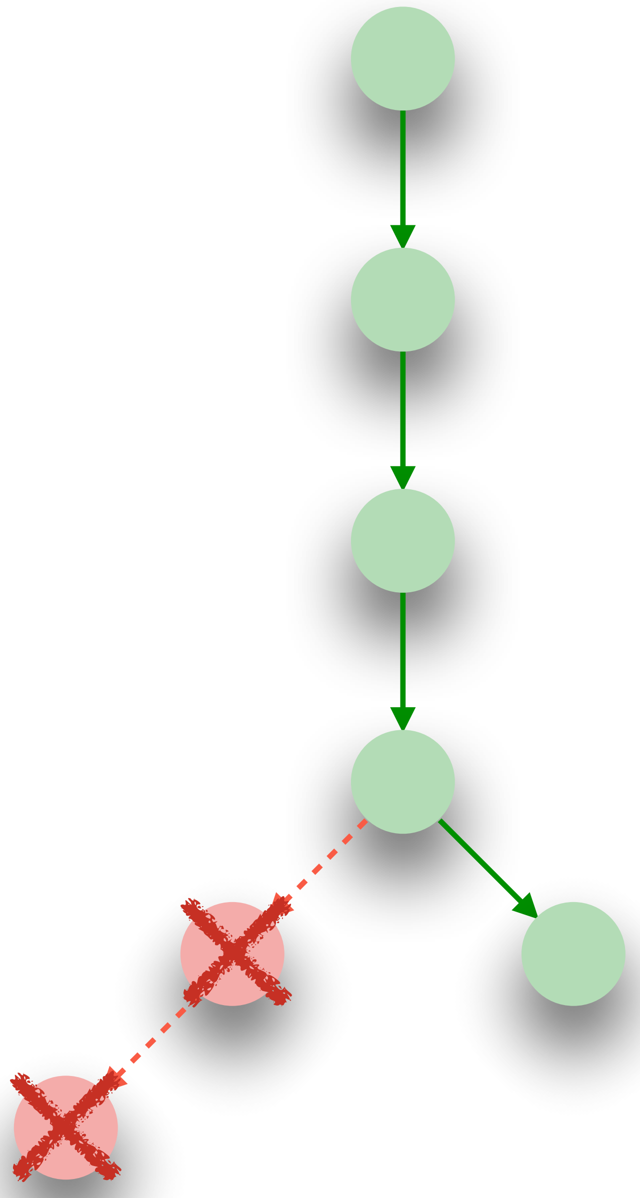
$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

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

x, **A_size**, **A**, **B**

are public

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

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

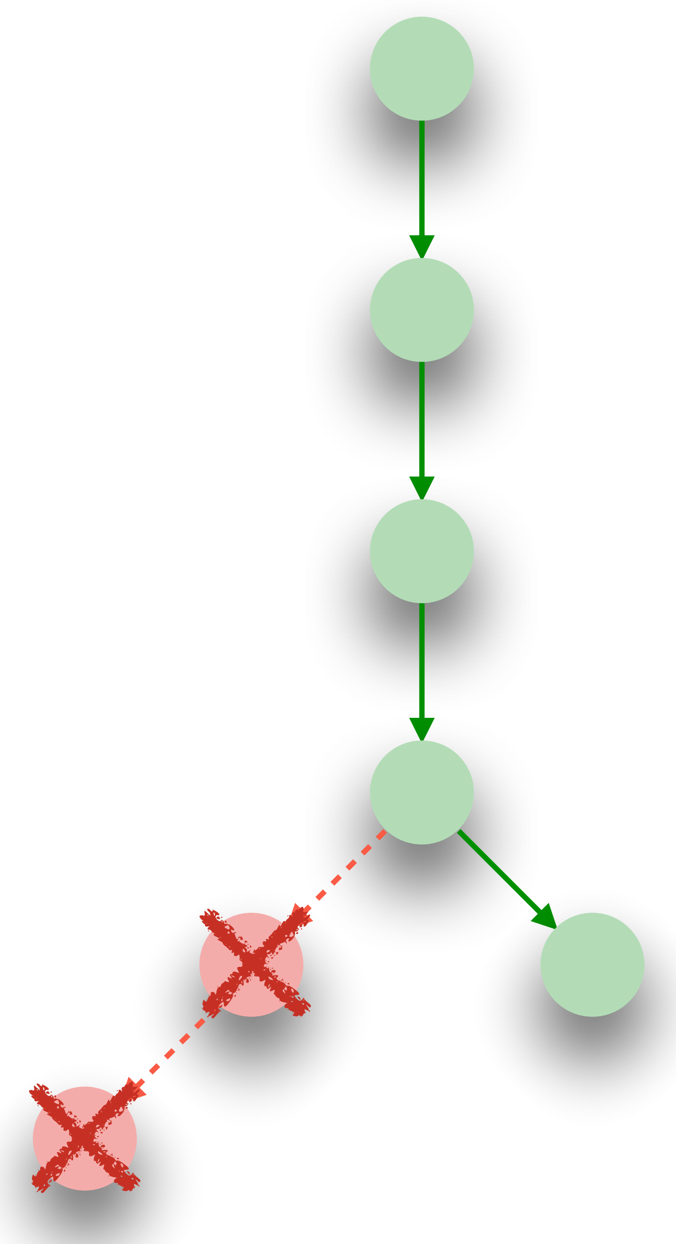
s_1

s_2

$x_1 = x_2 \wedge A\_size_1 = A\_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

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

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

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

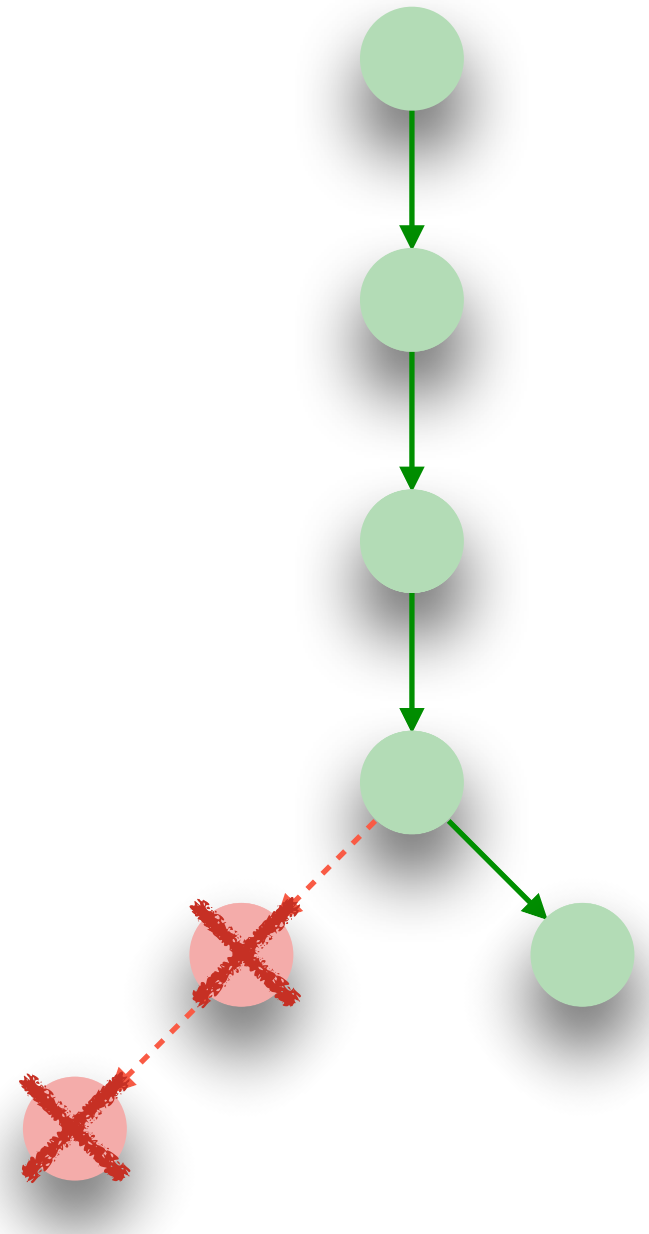
$s_1 \models \quad x_1 \geq A\_size_1$

$s_2 \models \quad x_2 \geq A\_size_2$

$x_1 = x_2 \wedge A\_size_1 = A\_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

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

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

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

$s_1 \models x_1 \geq A\_size_1$

$||$

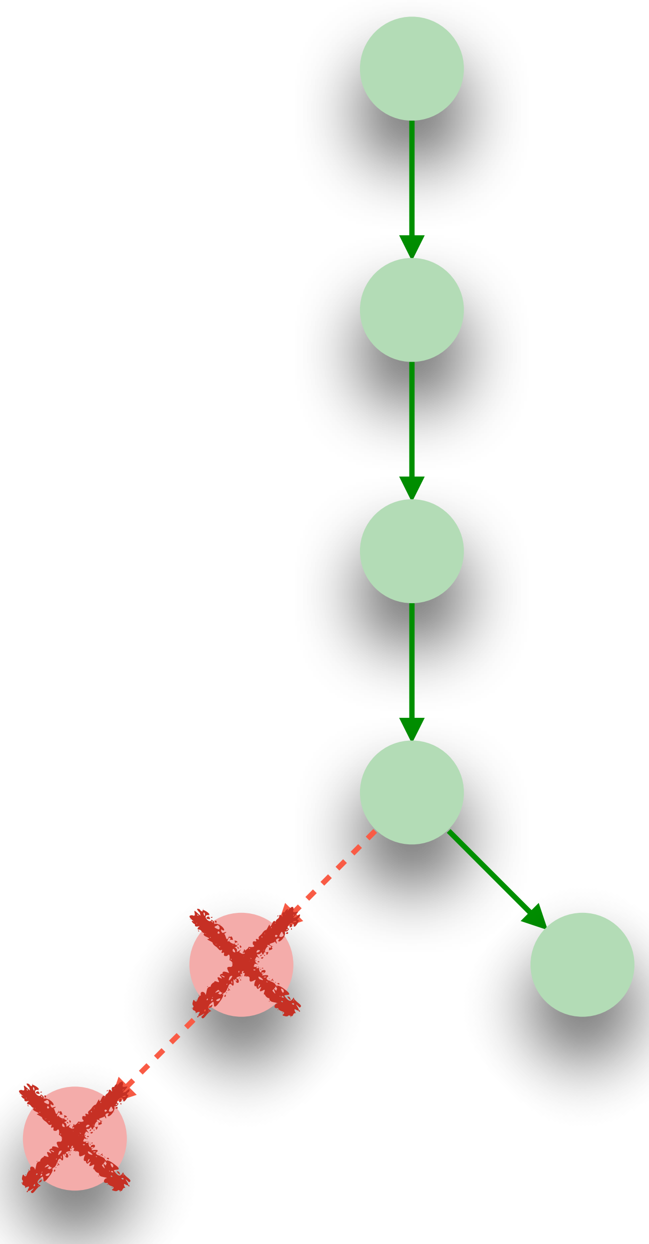
$s_2 \models x_2 \geq A\_size_2$

$x_1 = x_2 \wedge A\_size_1 = A\_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

Always true!

Memory leaks

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



Policy

x, **A_size**, **A**, **B** are public

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

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

$s_1 \models$ $x_1 \geq A\_size_1$

||

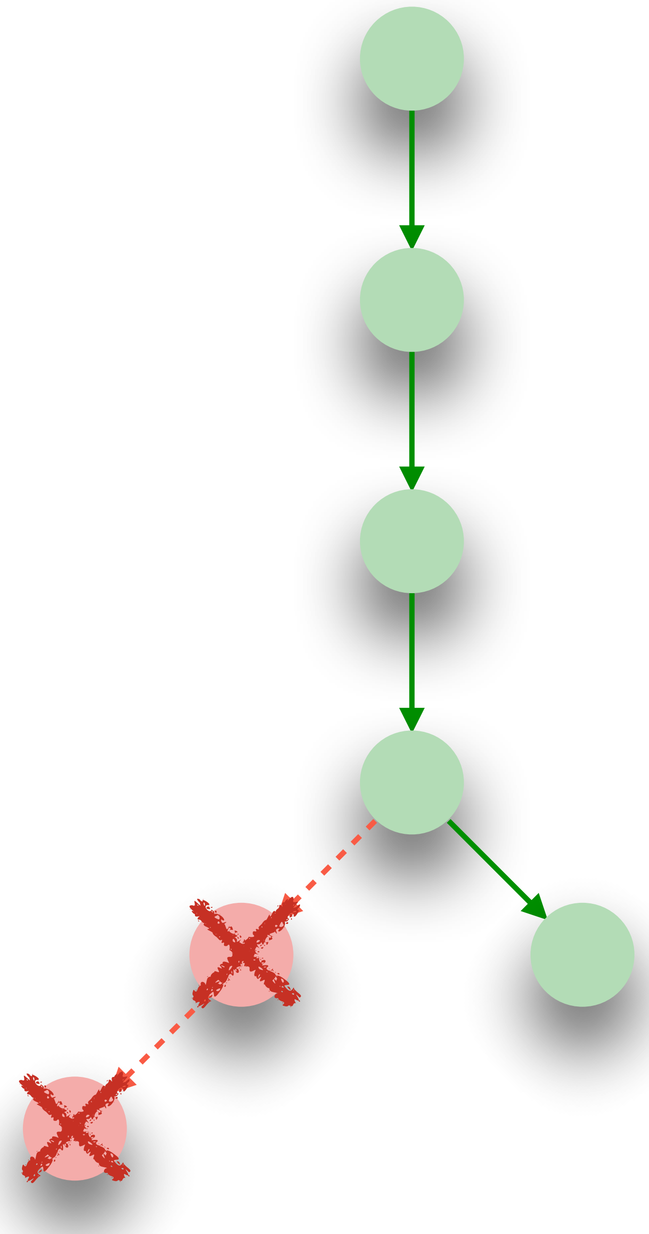
$s_2 \models$ $x_2 \geq A\_size_2$

$x_1 = x_2 \wedge A\_size_1 = A\_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

Always true!

Memory leaks

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



Policy

x, **A_size**, **A**, **B**

are public

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

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

$s_1 \models x_1 \geq A\_size_1$ $A_1 + x_1$

$\qquad\qquad\qquad\qquad = \qquad\qquad \neq$

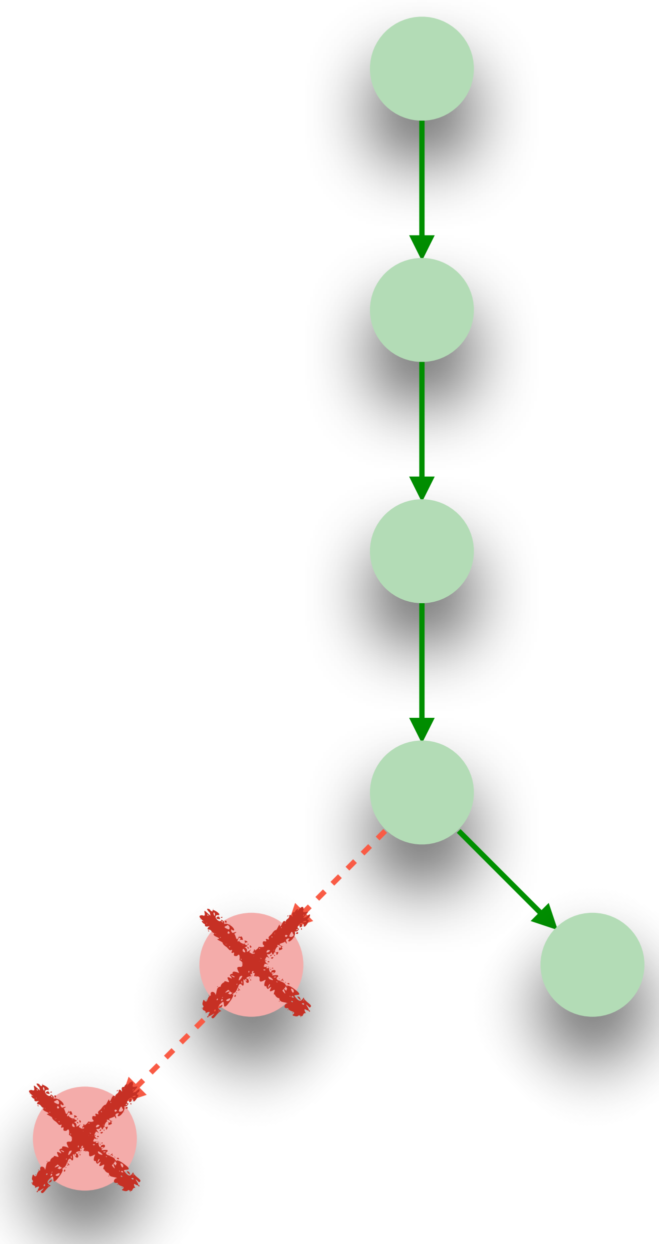
$s_2 \models x_2 \geq A\_size_2$ $A_2 + x_2$

$x_1 = x_2 \wedge A\_size_1 = A\_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

Always true!

Memory leaks

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



Policy
x, A_size, A, B
are public

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

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

$s_1 \models x_1 \geq A\_size_1$ $\quad A_1 + x_1$

$= \quad \neq \quad \vee$

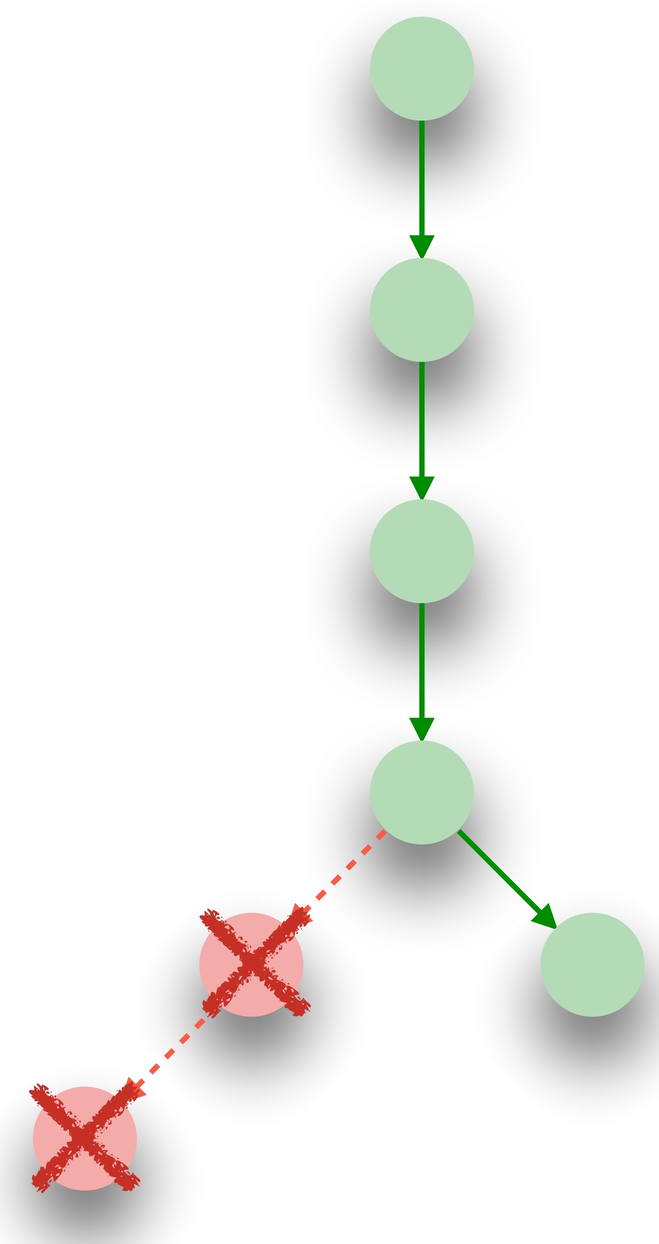
$s_2 \models x_2 \geq A\_size_2$ $\quad A_2 + x_2$

$x_1 = x_2 \wedge A\_size_1 = A\_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

Always true!

Memory leaks

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



Policy

x, **A_size**, **A**, **B**

are public

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

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

$s_1 \models$ $x_1 \geq A\_size_1$ $A_1+x_1 \not\Vdash A_2+x_2 \;\vee\; B_1+A_1[x_1] \not\Vdash B_2+A_2[x_2]$

$||$

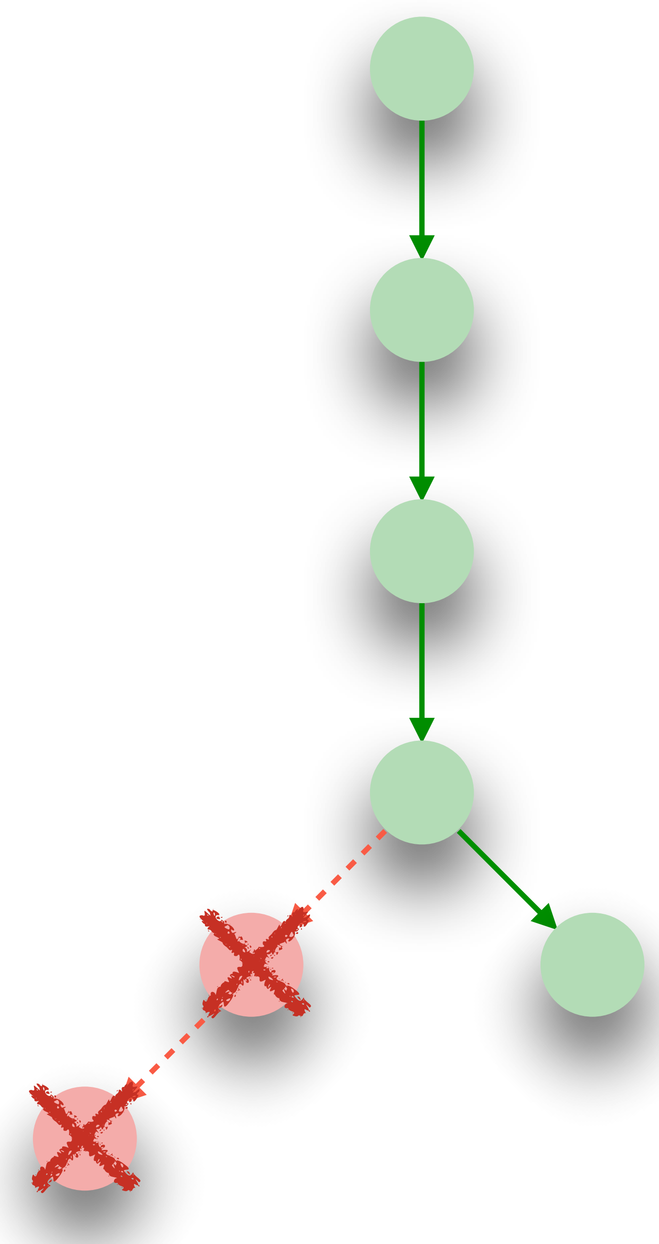
$s_2 \models$ $x_2 \geq A\_size_2$

$x_1 = x_2 \wedge A\_size_1 = A\_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

Always true!

Memory leaks

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



Policy

x, ***A_size***, **A**, **B**

are public

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

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

$s_1 \models x_1 \geq A\_size_1$

$\parallel$

$s_2 \models x_2 \geq A\_size_2$

$(A_1 + x_1 \nparallel A_2 + x_2) \vee (B_1 + A_1[x_1] \nparallel B_2 + A_2[x_2])$

$x_1 = x_2 \wedge A\_size_1 = A\_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

Always true!

Reasoning about arbitrary oracles

Reasoning about arbitrary oracles

Always-mispredict speculative semantics

Mispredict ***all*** branch instructions

Fixed speculative window

Rollback of every transaction

Reasoning about arbitrary oracles

Always-mispredict speculative semantics

Mispredict ***all*** branch instructions

Fixed speculative window

Rollback of every transaction

Always-mispredict is ***worst-case***

$$\mathbf{P}_{\mathbf{am}}(s) = \mathbf{P}_{\mathbf{am}}(s') \iff$$

$$\forall \mathcal{O}.\ \mathbf{P}_{\mathbf{spec}}(s, \mathcal{O}) = \mathbf{P}_{\mathbf{spec}}(s', \mathcal{O})$$

Reasoning about arbitrary oracles

Always-mispredict speculative semantics

Mispredict ***all*** branch instructions

Fixed speculative window

Rollback of every transaction

Always-mispredict is ***worst-case***

$$\mathbf{P}_{\mathbf{am}}(s) = \mathbf{P}_{\mathbf{am}}(s') \iff$$
$$\forall O.\ \mathbf{P}_{\mathbf{spec}}(s, O) = \mathbf{P}_{\mathbf{spec}}(s', O)$$

If program $\mathbf{P}$ satisfies

$$\forall s, s'. \mathbf{P}_{\mathbf{non\text{-}spec}}(s) = \mathbf{P}_{\mathbf{non\text{-}spec}}(s')$$
$$\Rightarrow \mathbf{P}_{\mathbf{am}}(s) = \mathbf{P}_{\mathbf{am}}(s')$$

then $\mathbf{P}$ satisfies ***SNI*** w.r.t. all O

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
cmovae  -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]
```

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

`rax` is -1 whenever `x` ≥ `A_size`
We can prove security

Example #10 - SLH

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

Example #10 - SLH

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

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

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
cmovae  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]
```

```
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]
```

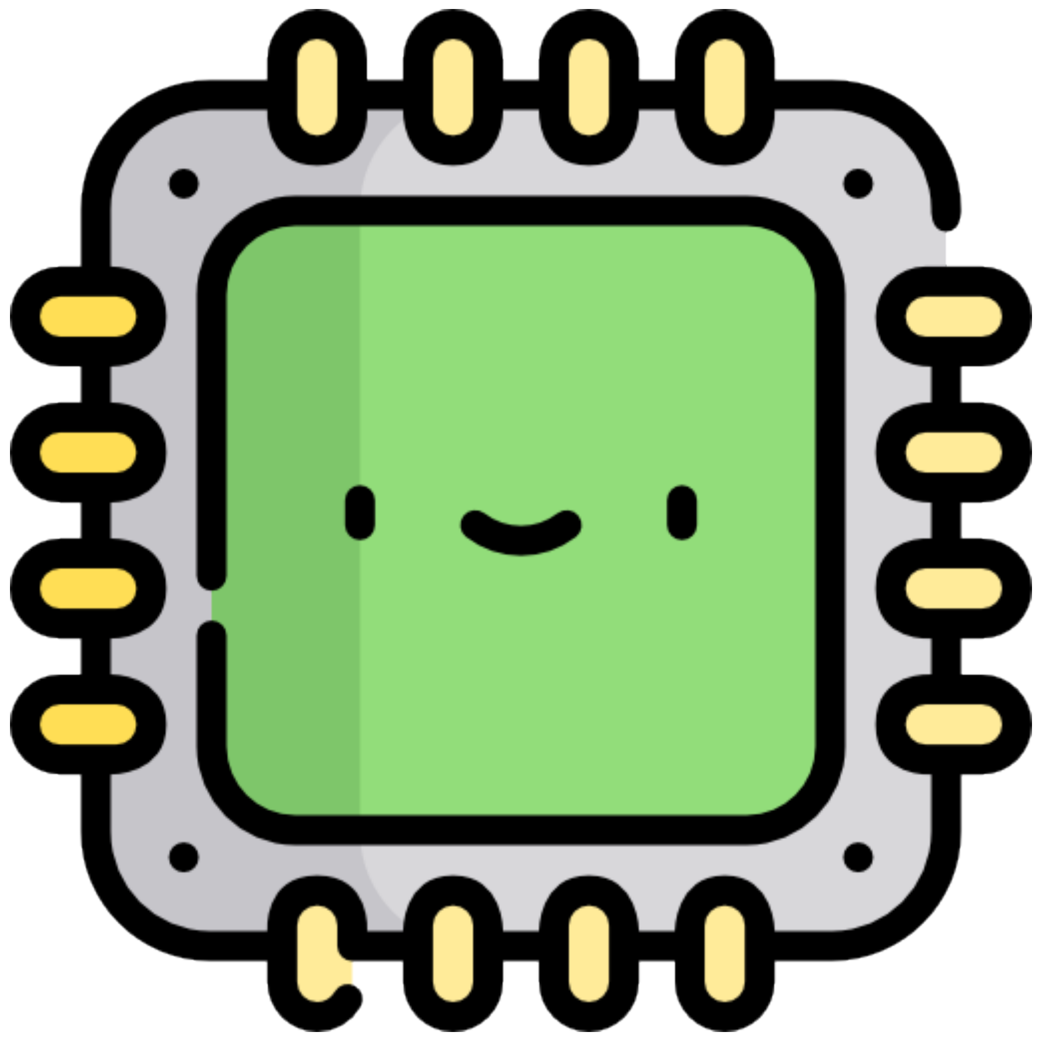
`lfence` is unnecessary

Spectre V1

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

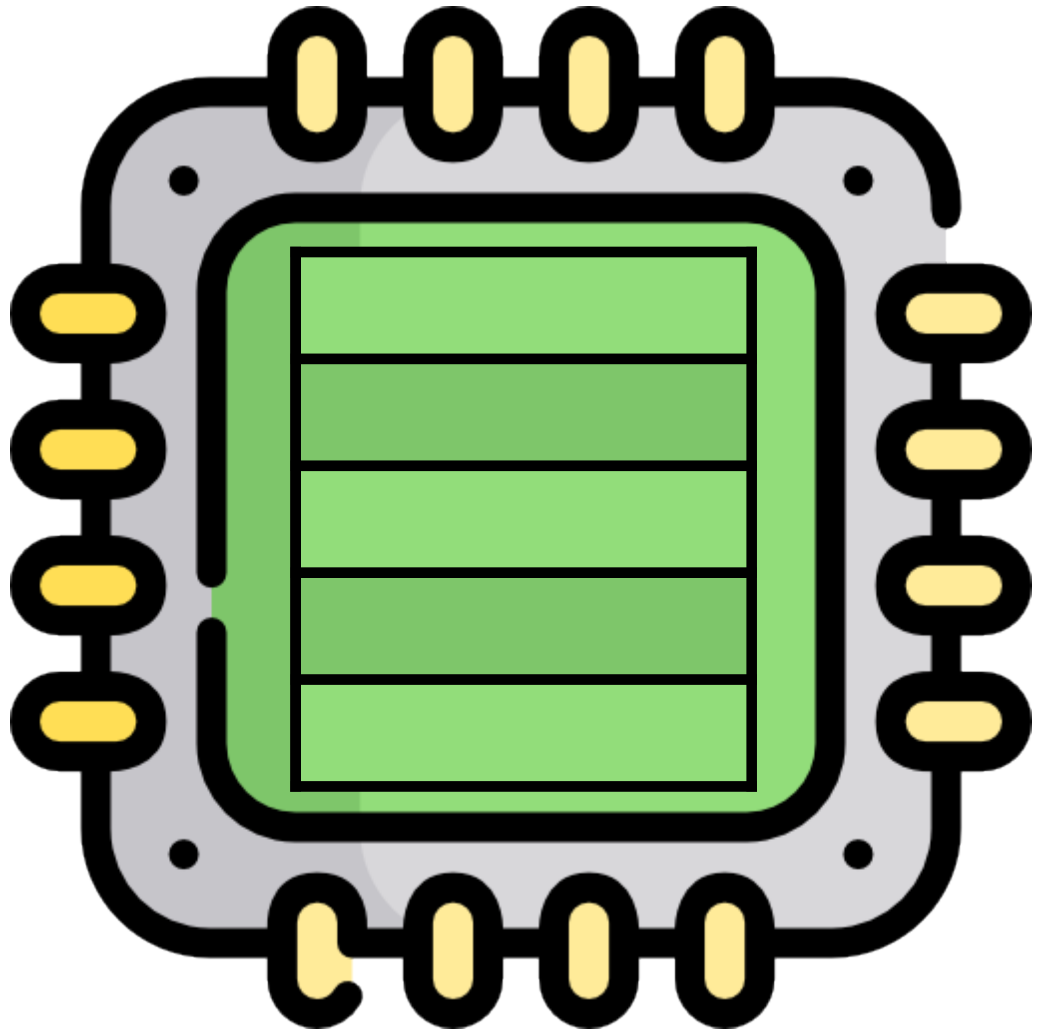
Spectre V1

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



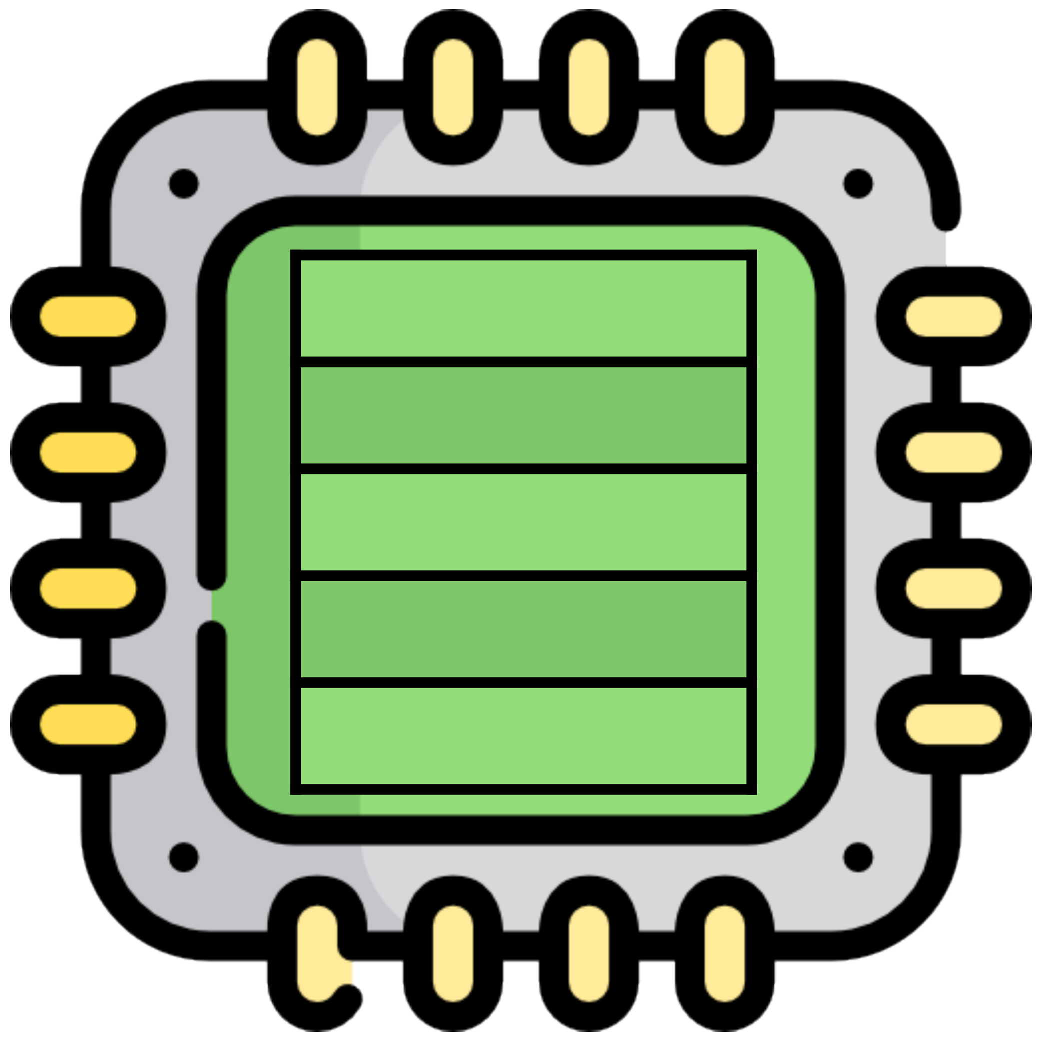
Spectre V1

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

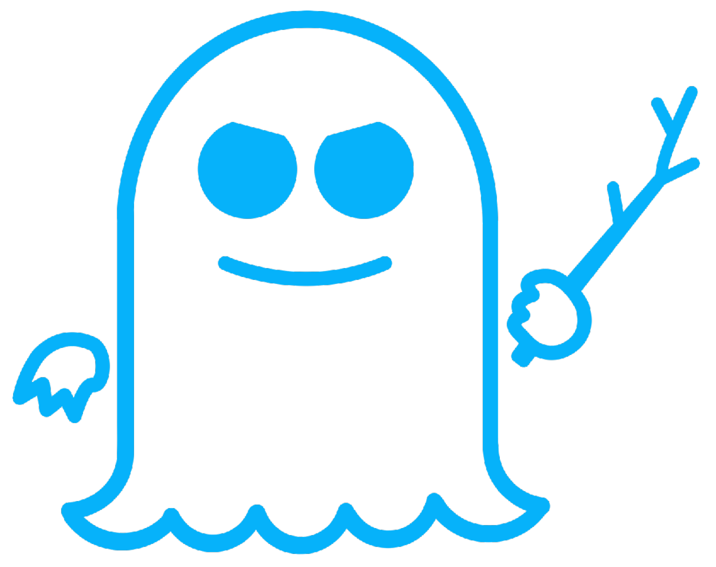


Spectre V1

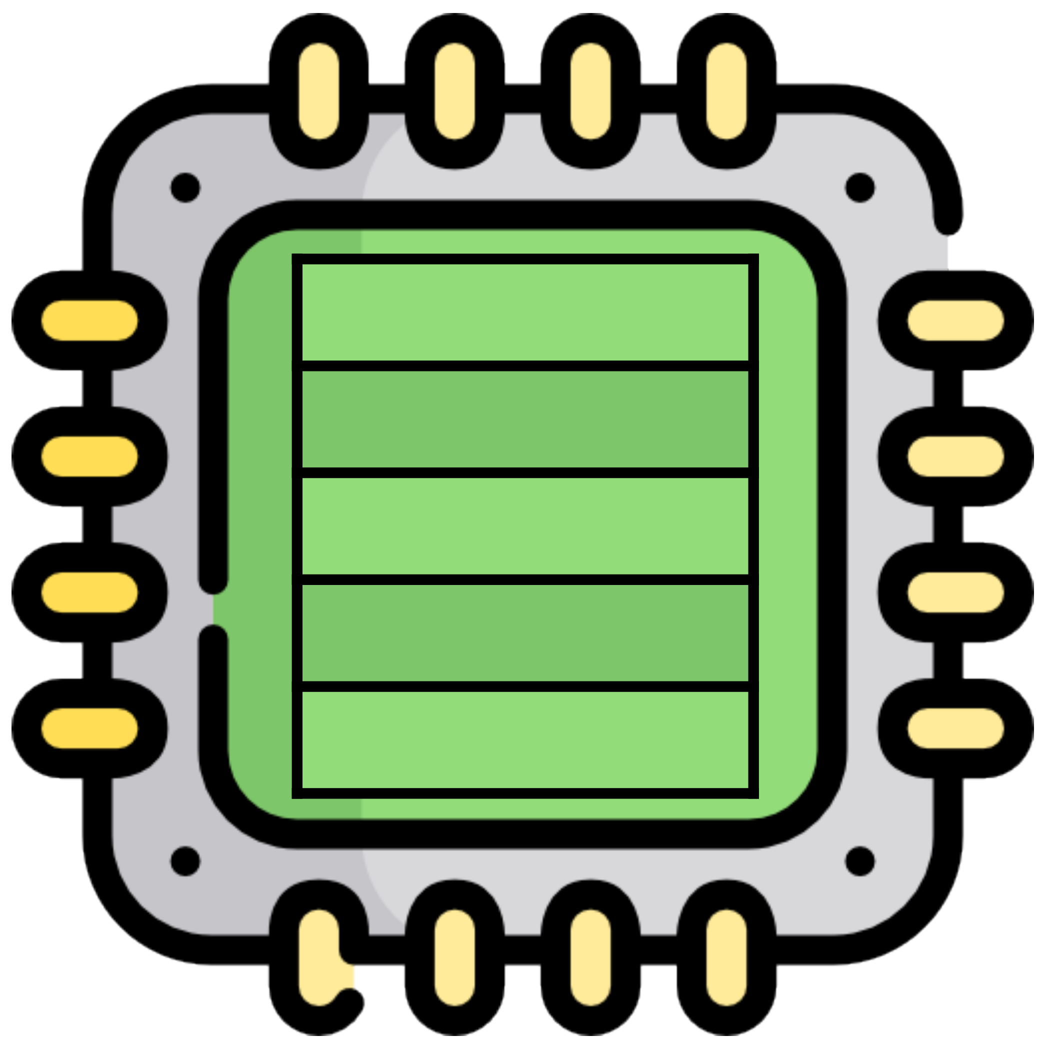
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



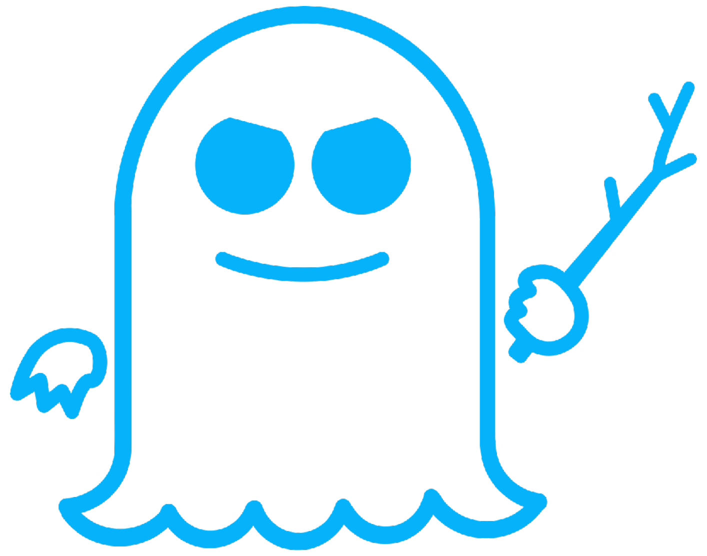
Spectre V1



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

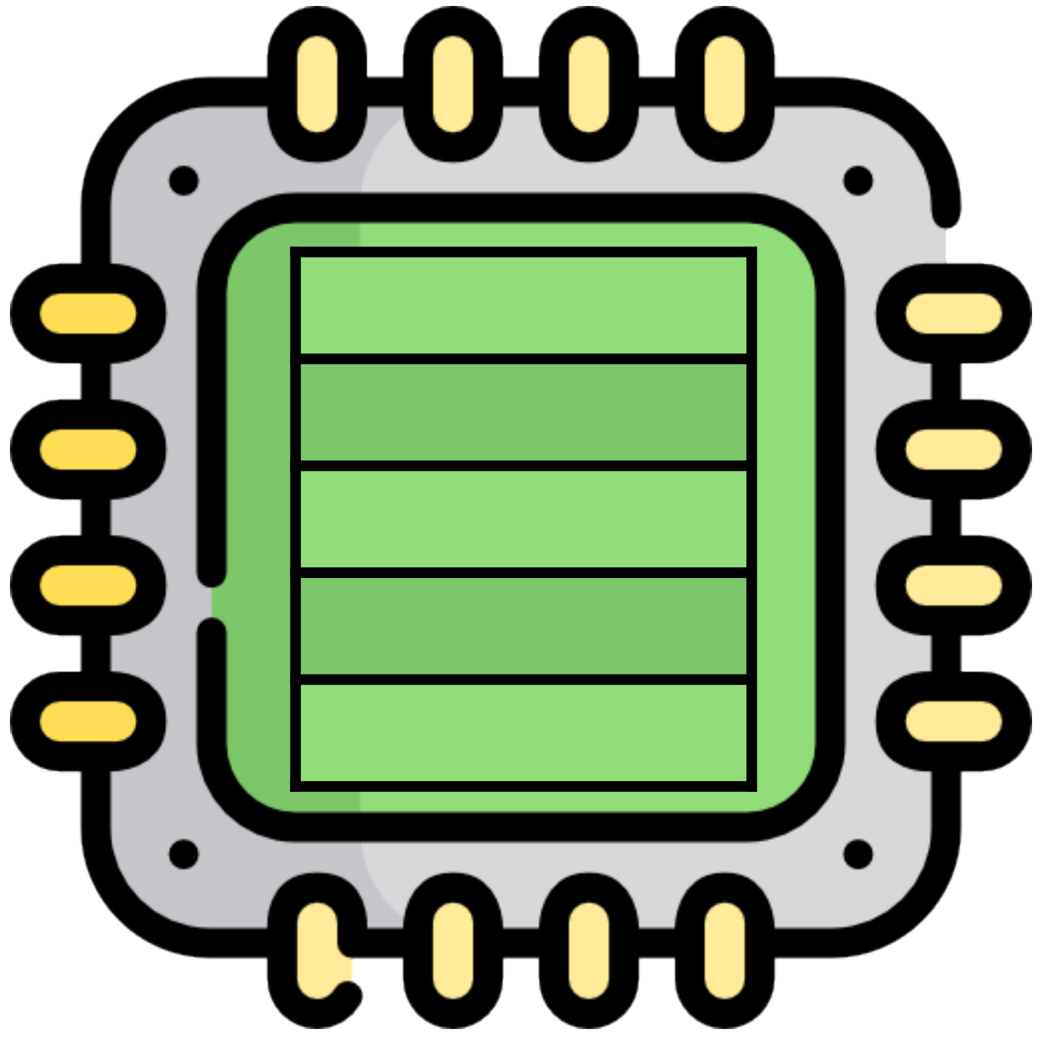


Spectre V1



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

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

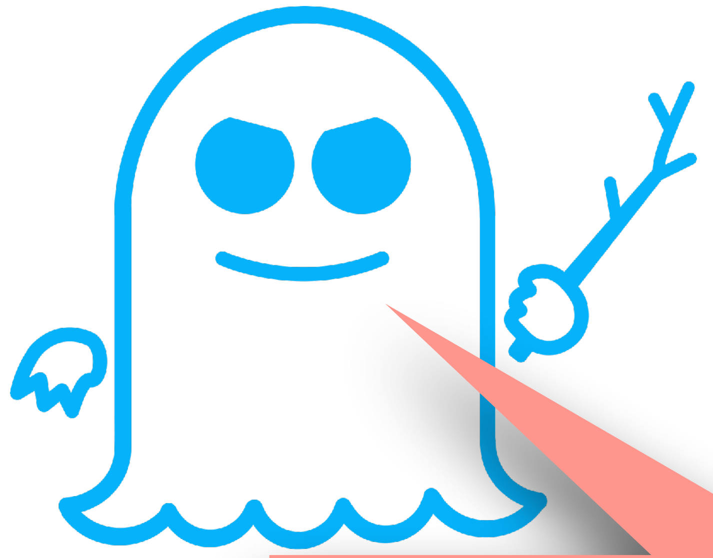


Spectre V1

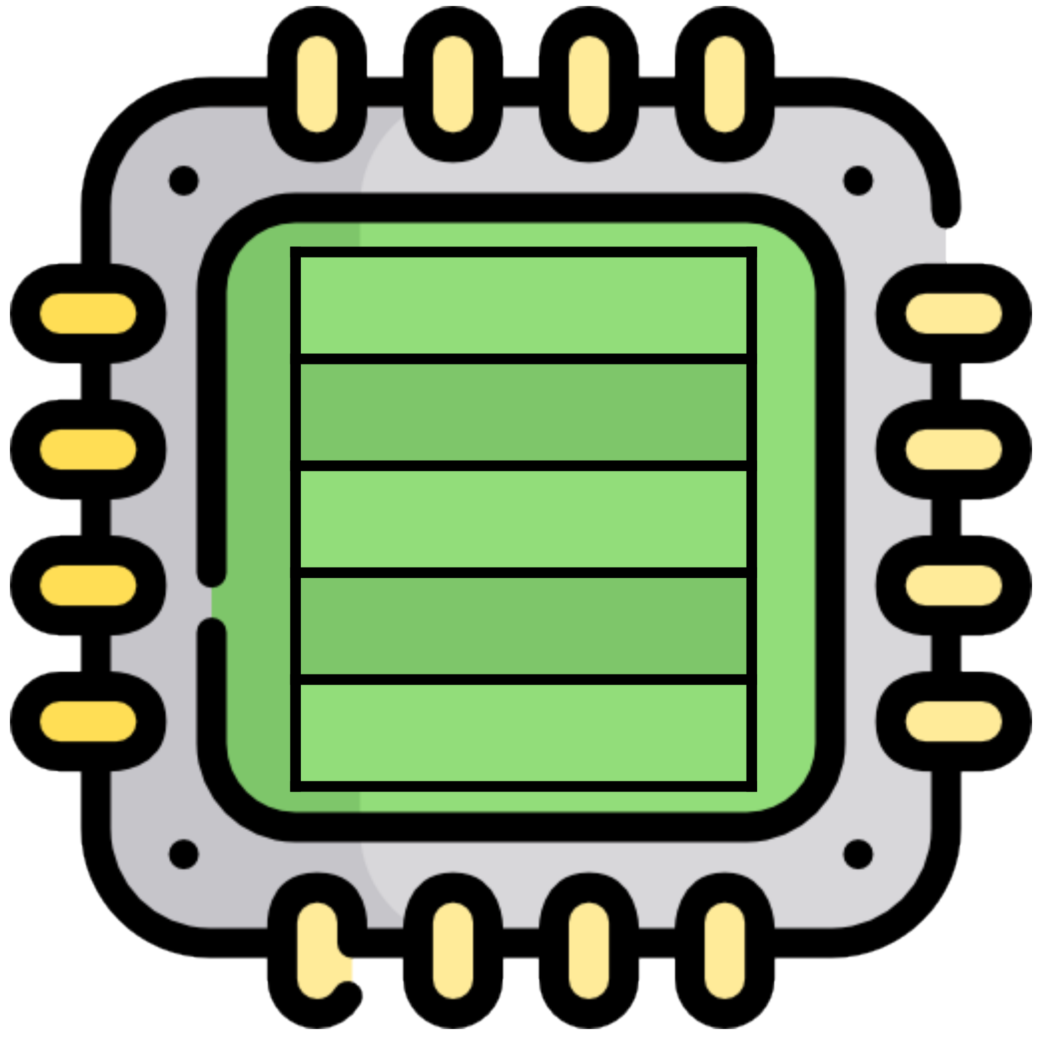
A_size=16

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

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



What is in A[128]?



Spectre V1

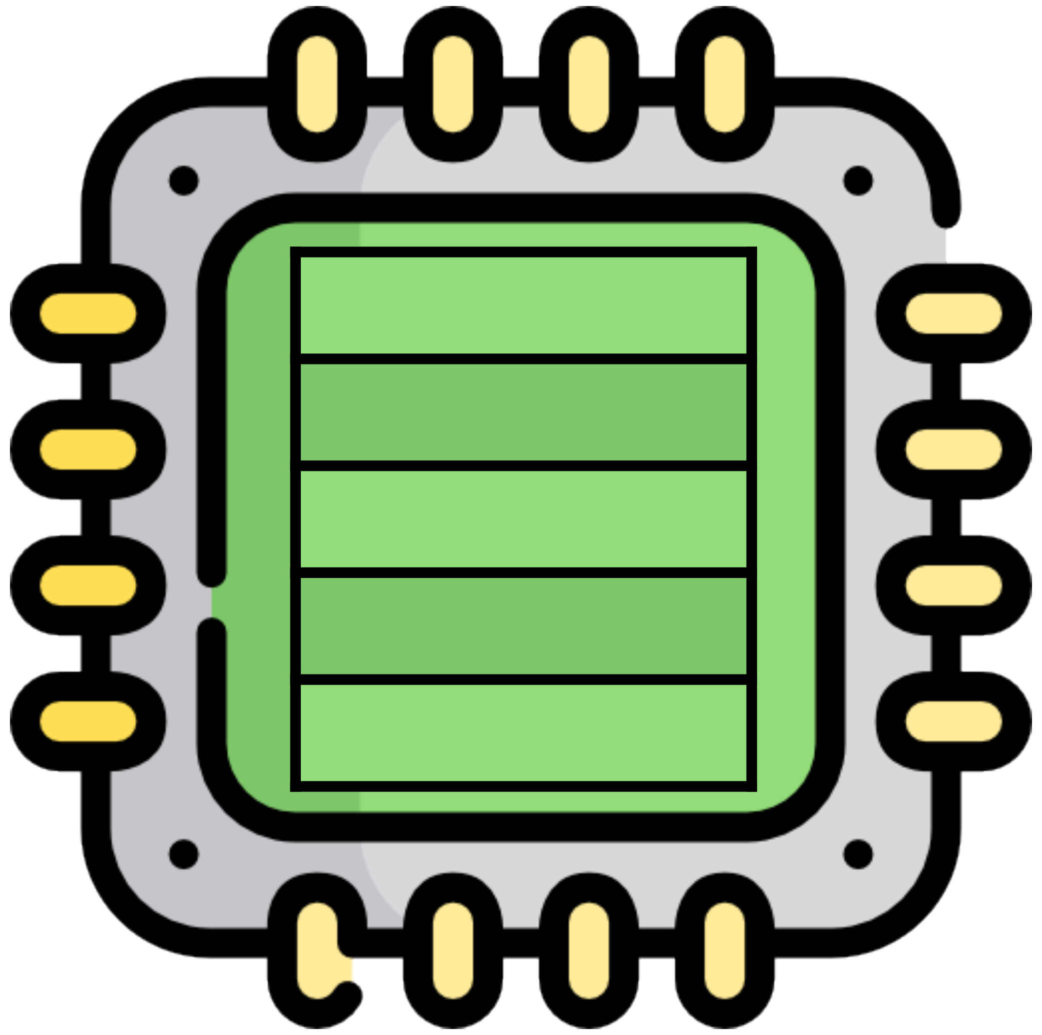
A_size=16

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

What is in A[128]?

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

1) Training

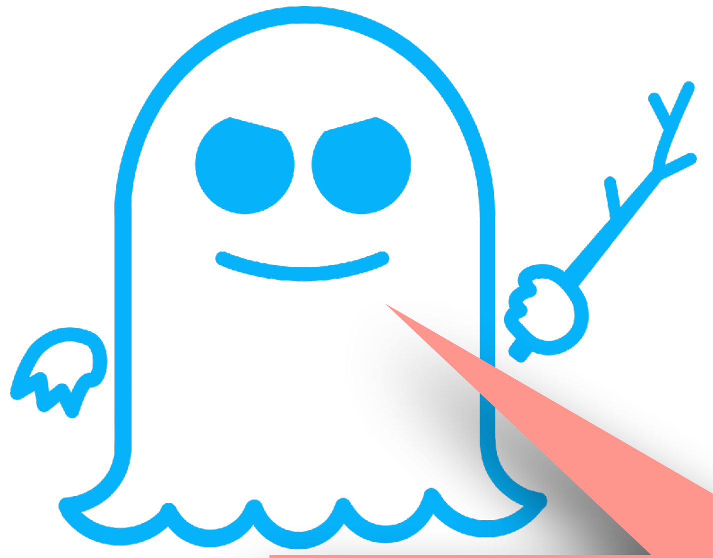


Spectre V1

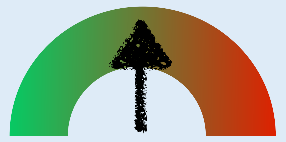
A_size=16

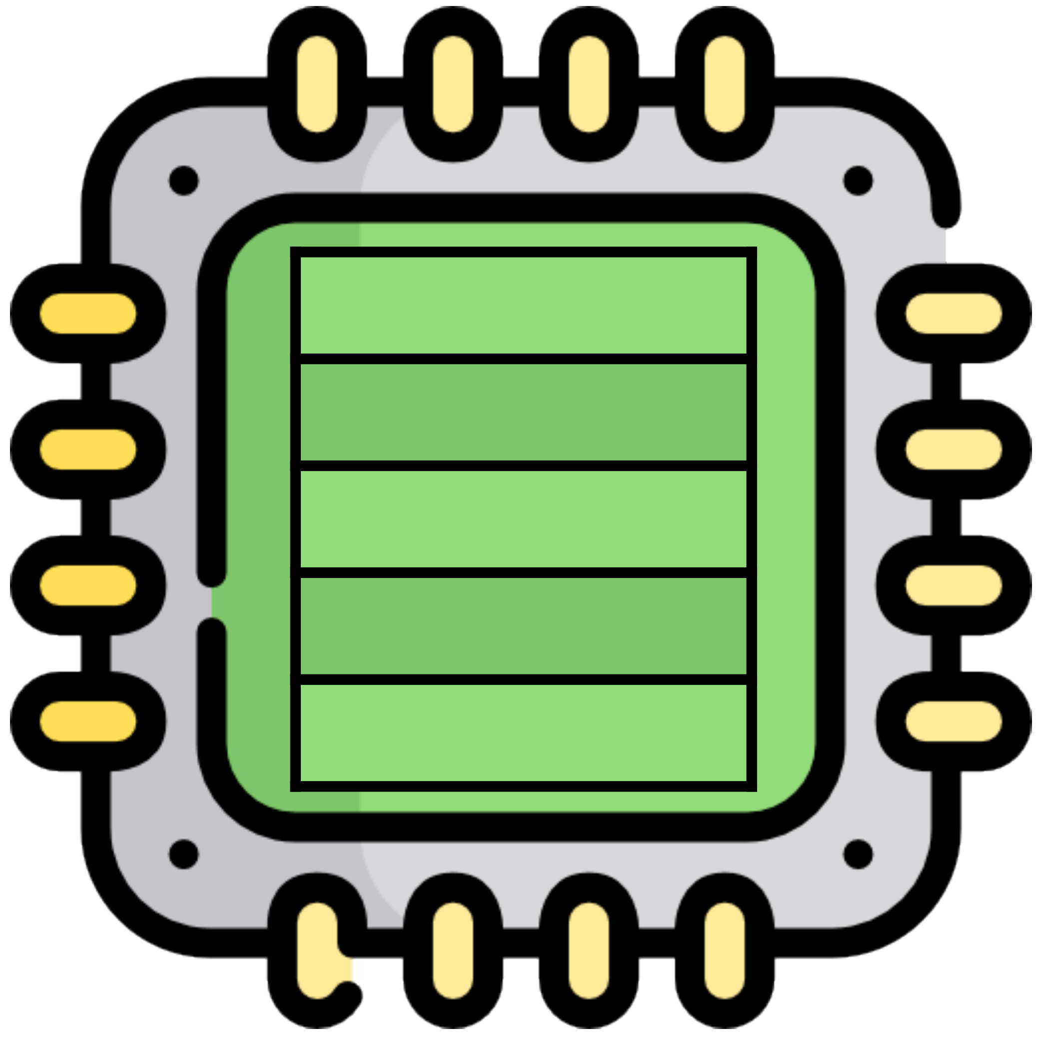
B | B[0] | B[1] | ...

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



What is in A[128]?

1) Training 

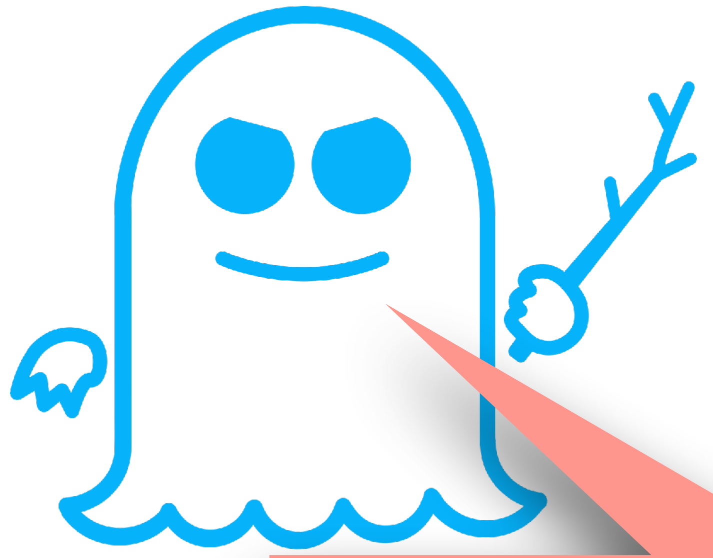


Spectre V1

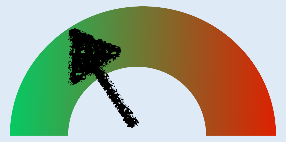
A_size=16

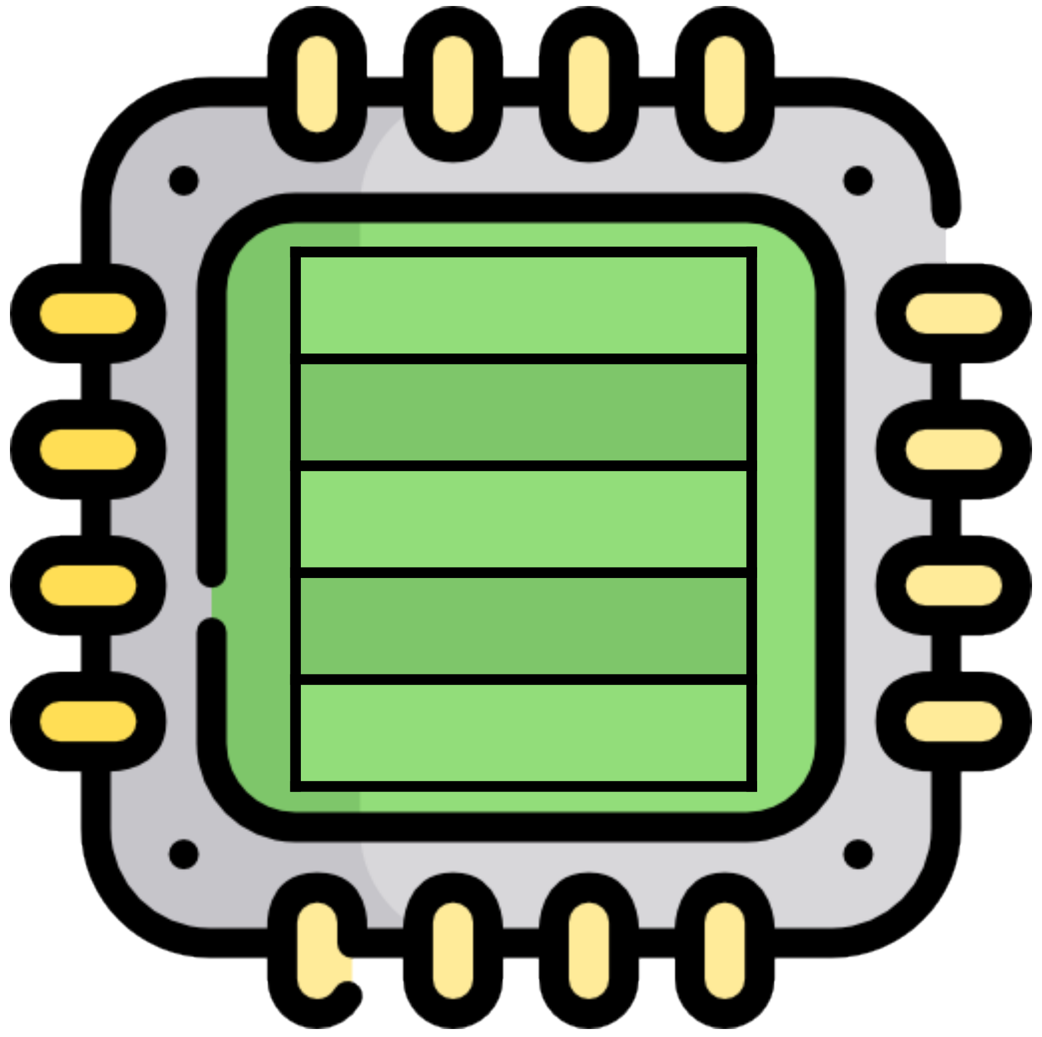
B | B[0] | B[1] | ...

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



What is in A[128]?

1) Training  f(0);

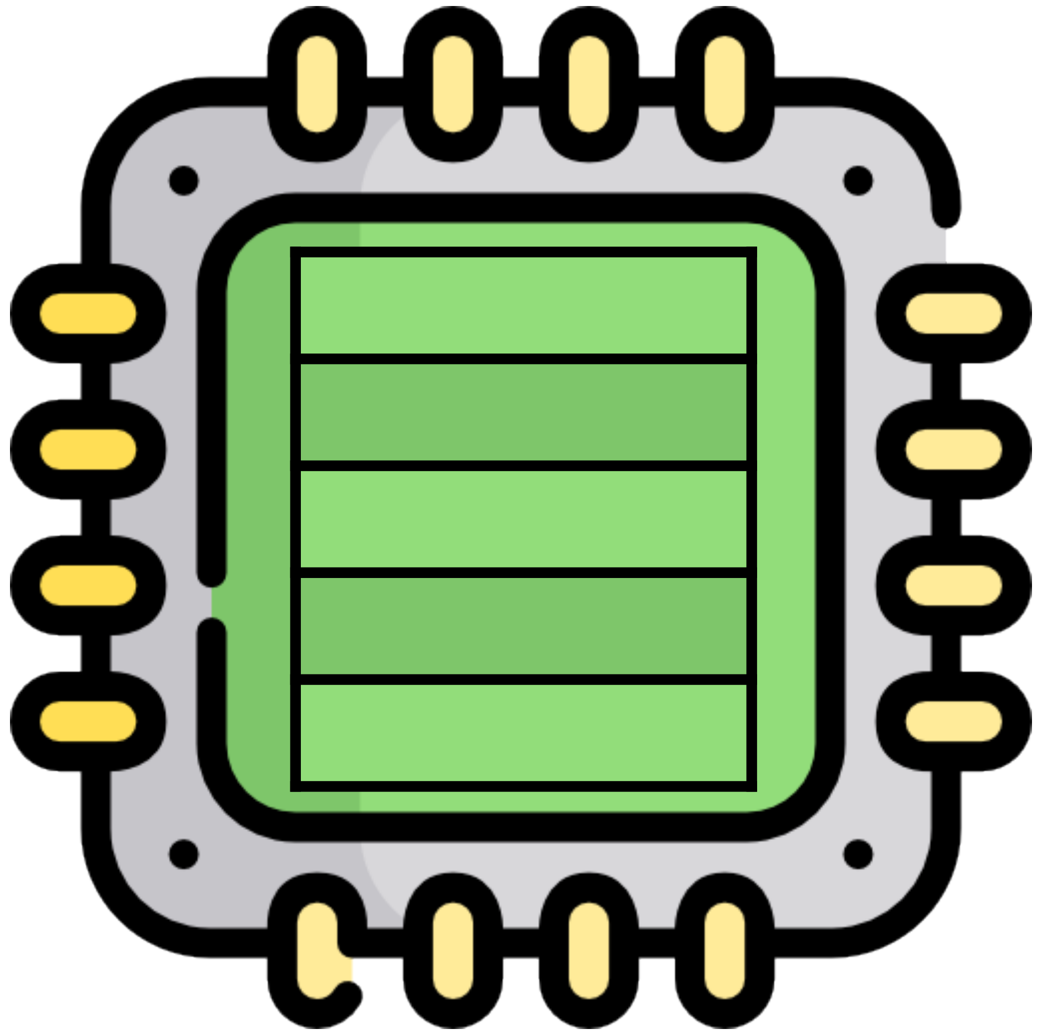


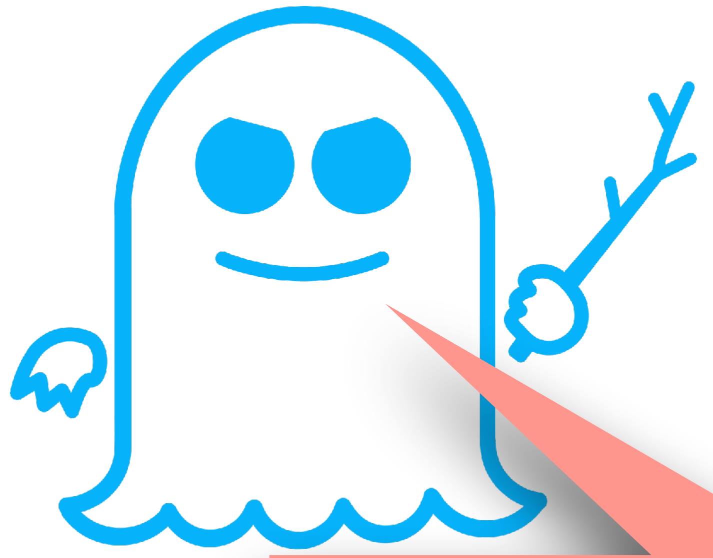
Spectre V1

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]?

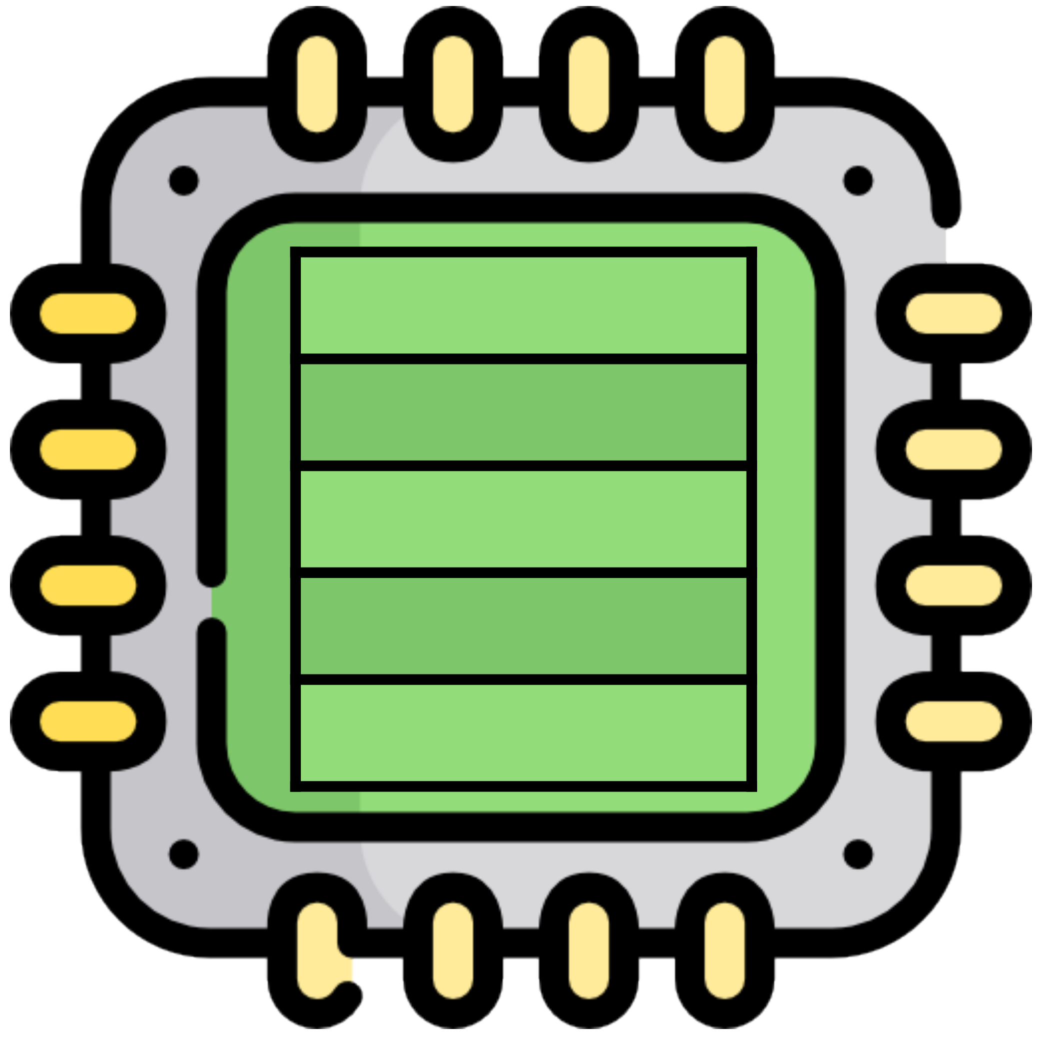
1) Training f(0);f(1);

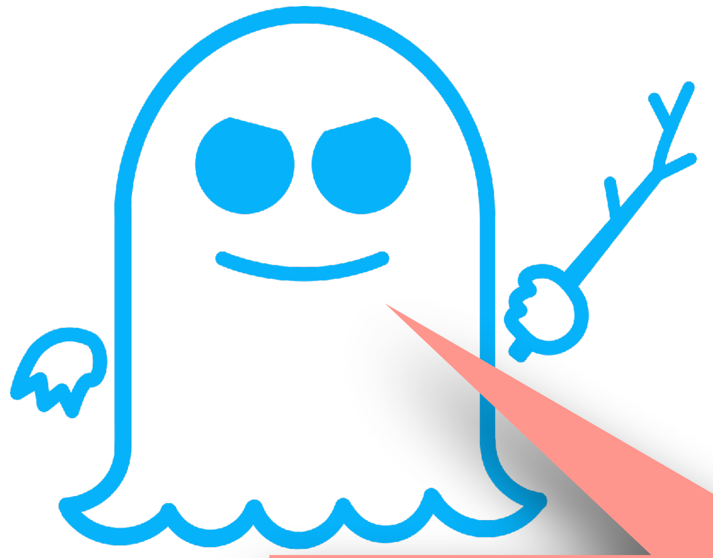
Spectre V1

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

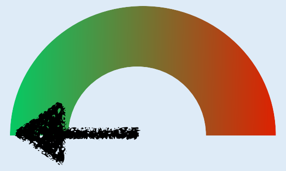
A_size=16

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





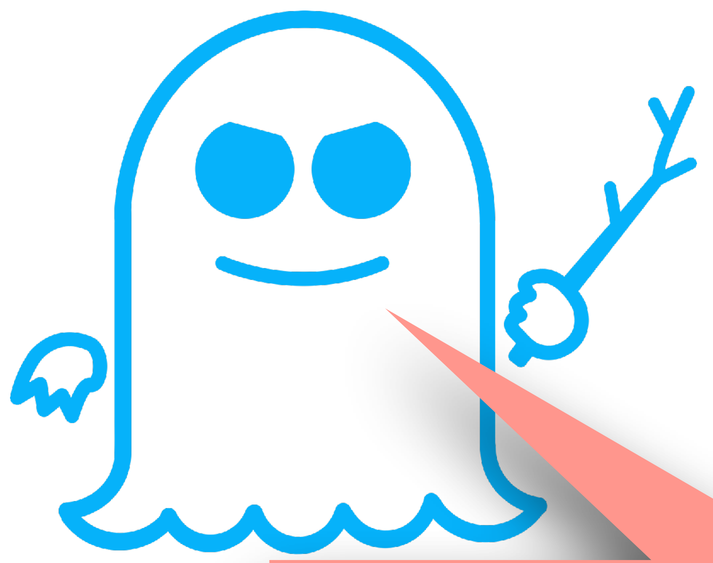
What is in A[128]?

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

Spectre V1

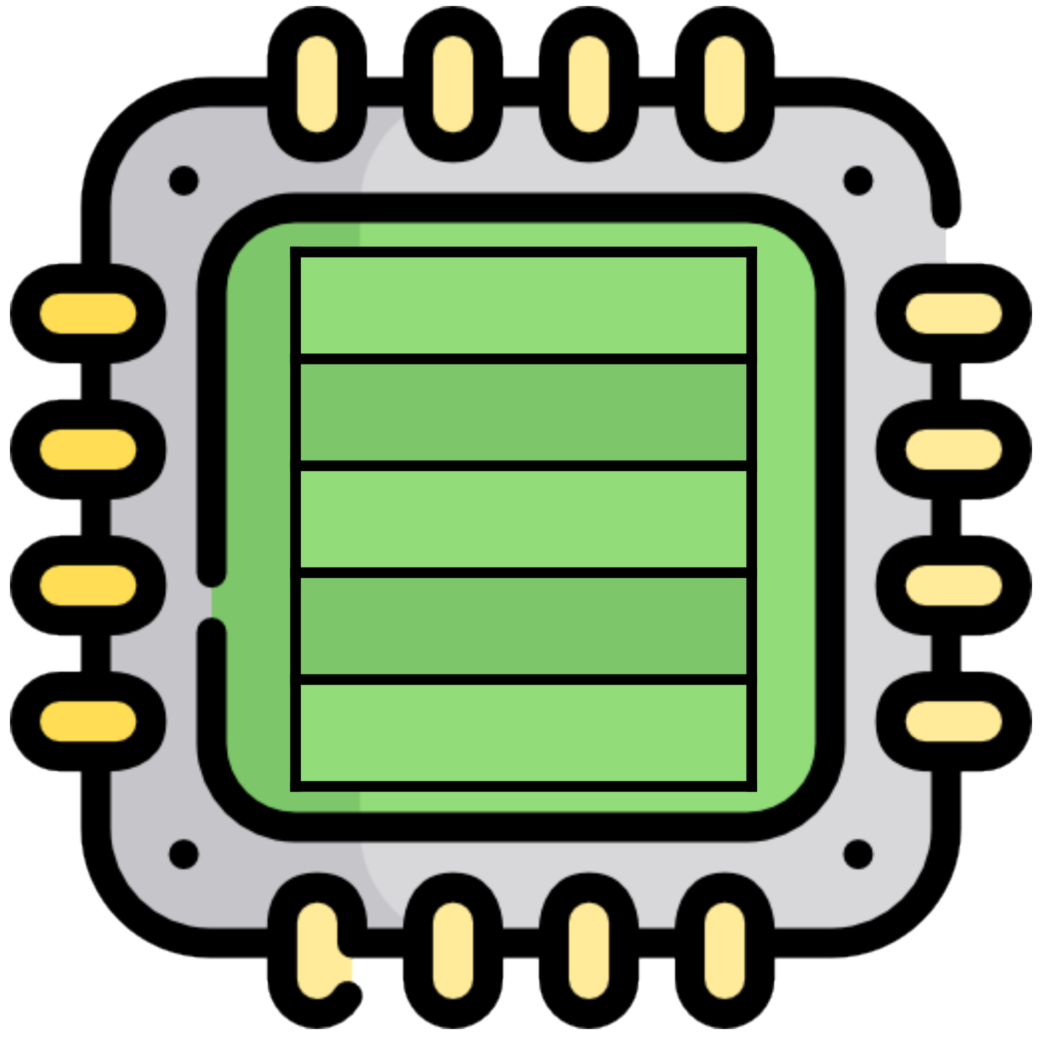
`A_size`=16

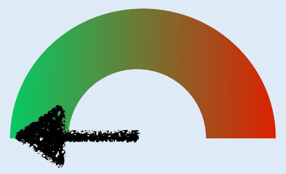
B `B[0]` `B[1]` ...



What is in `A[128]`?

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



1) Training  `f(0);f(1);f(2);` …

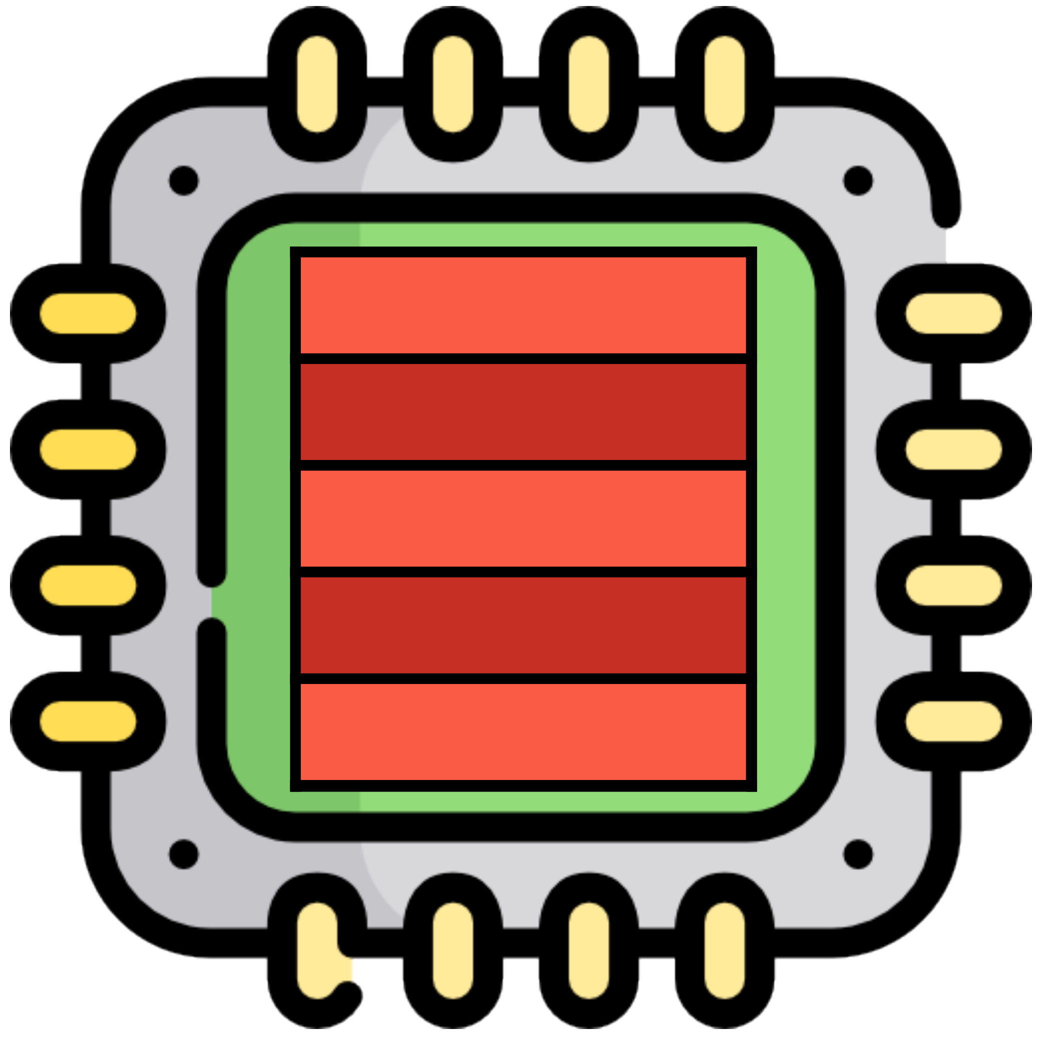
2) Prepare cache

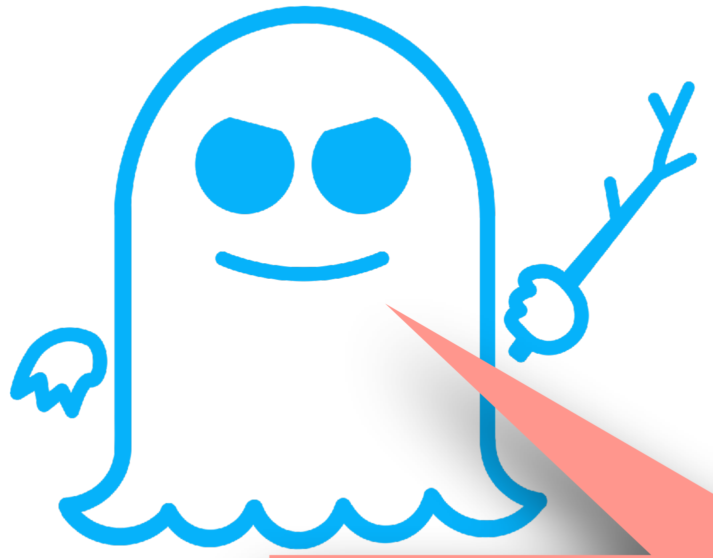
Spectre V1

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]?

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

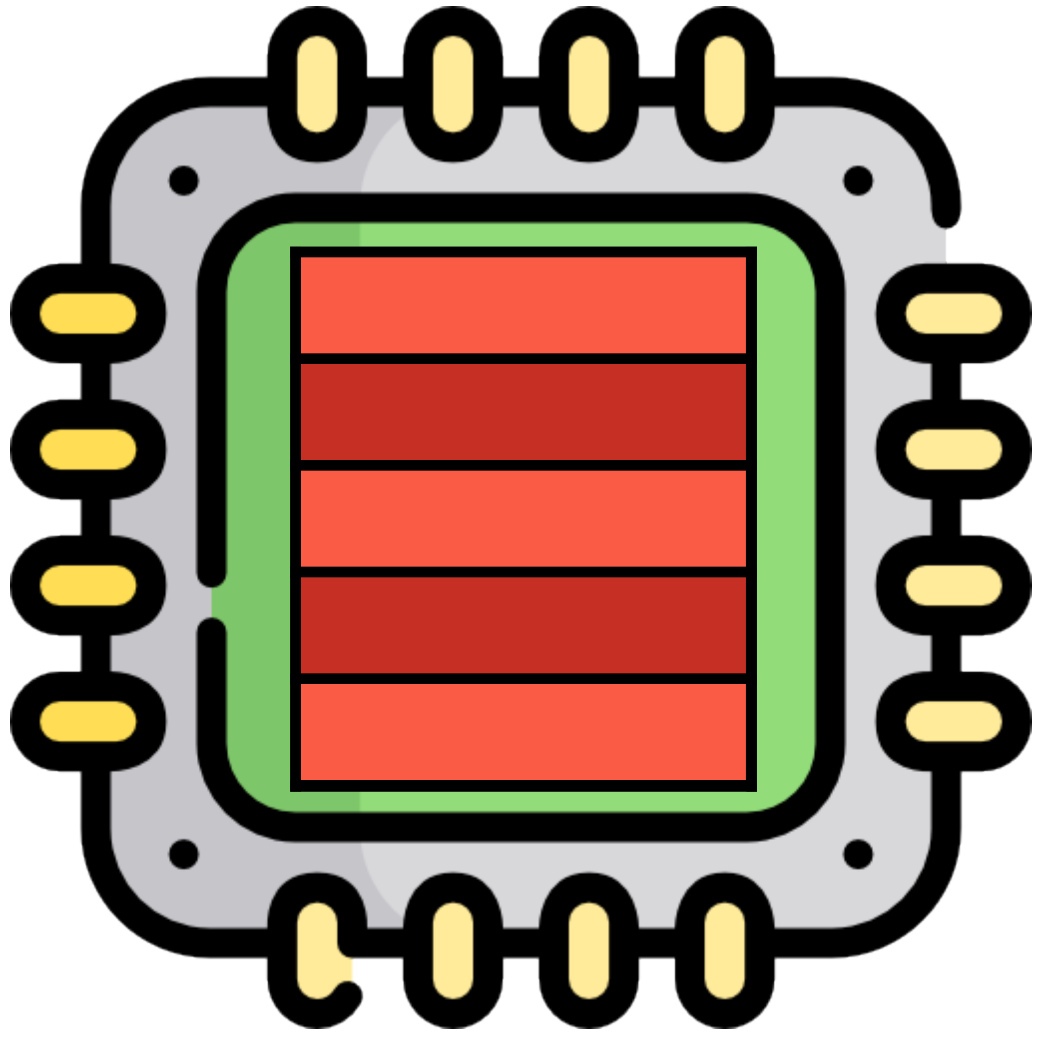
2) Prepare cache

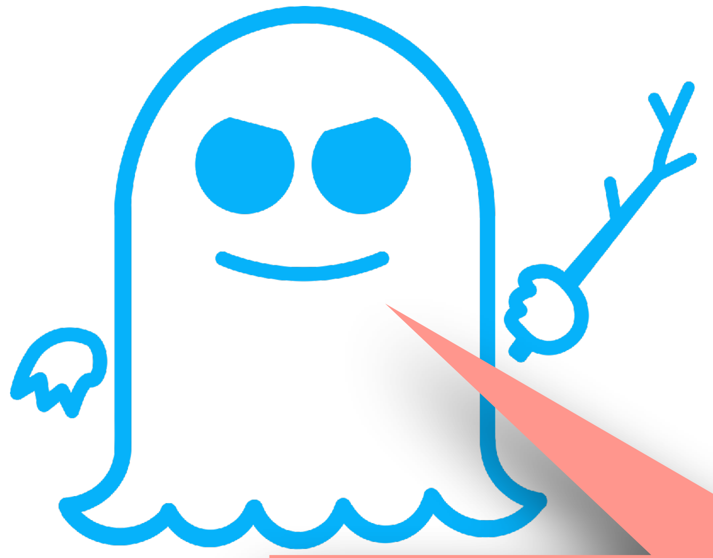
Spectre V1

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]?

1) **Training** f(0);f(1);f(2); …

2) **Prepare cache**

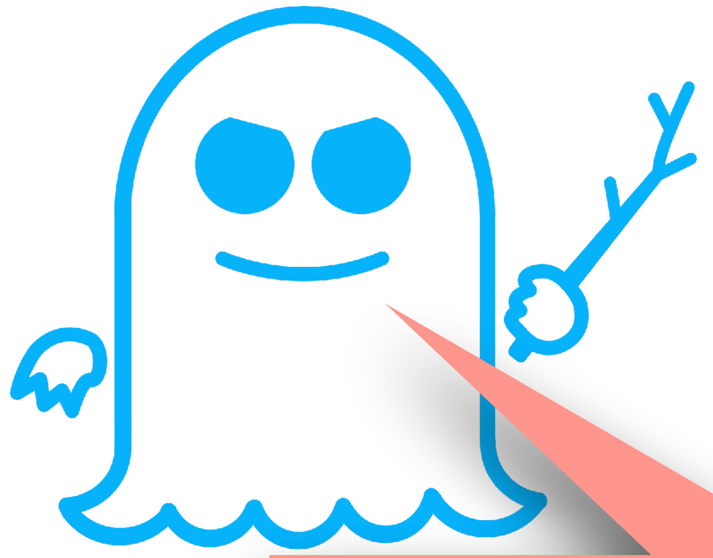
3) **Run with** x = 128

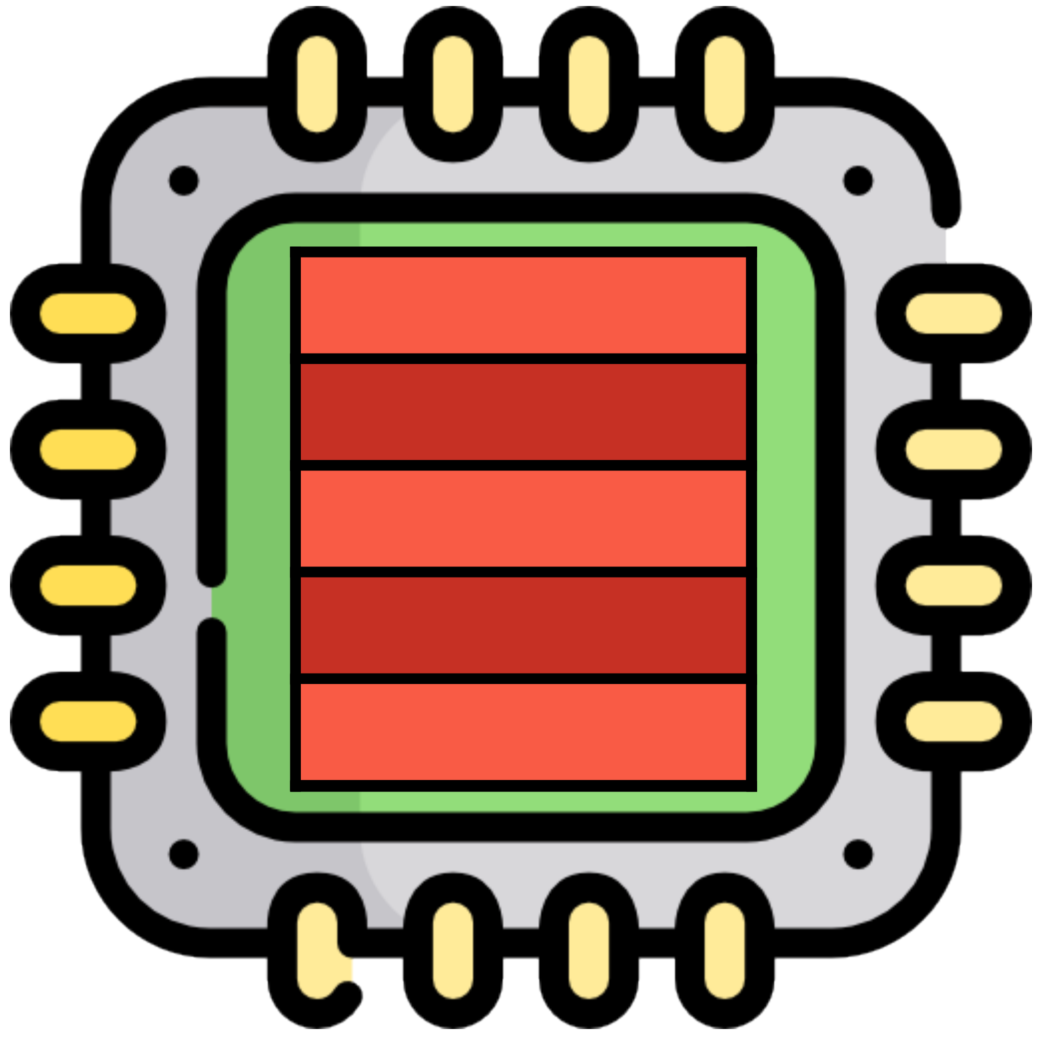
Spectre V1

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]?

1) Training f(0);f(1);f(2); …

2) Prepare cache

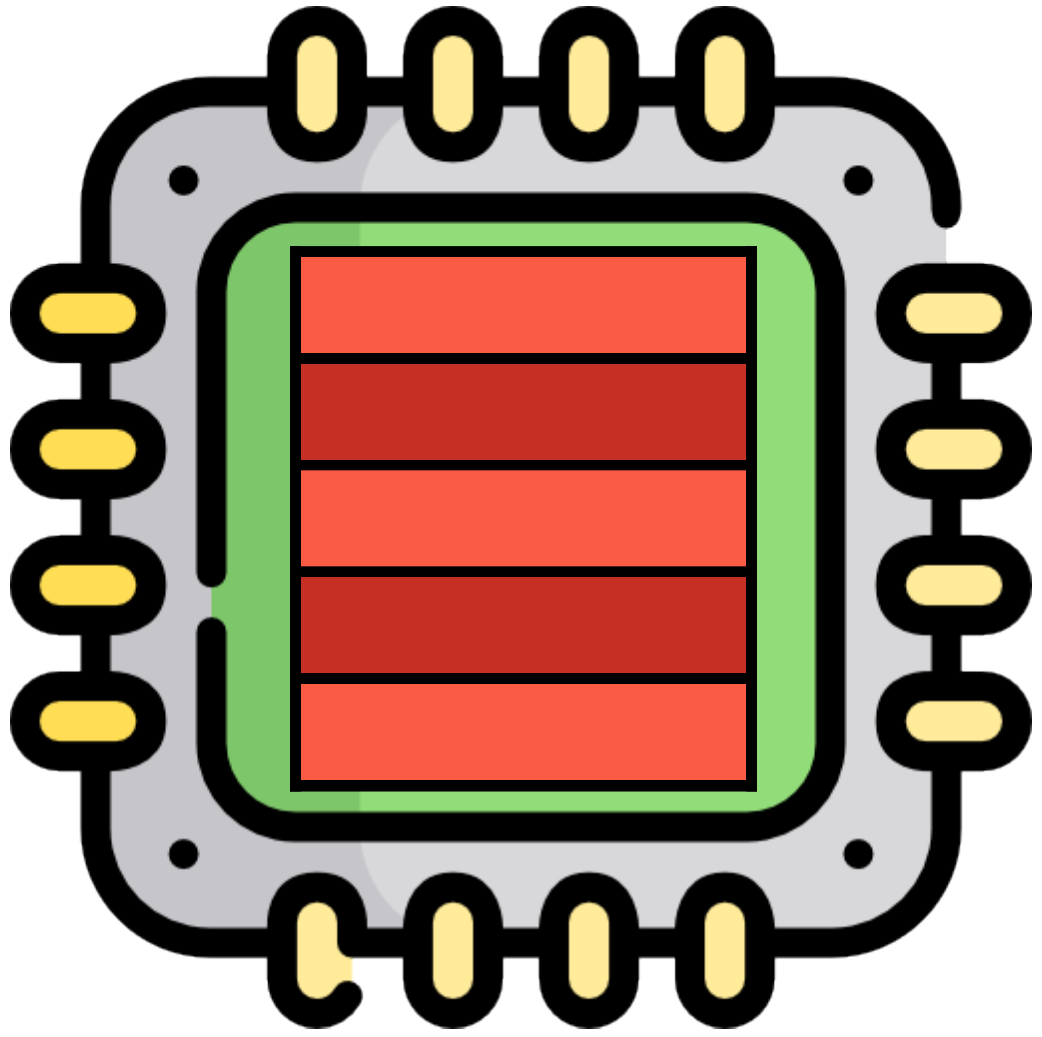
3) Run with *x* = 128

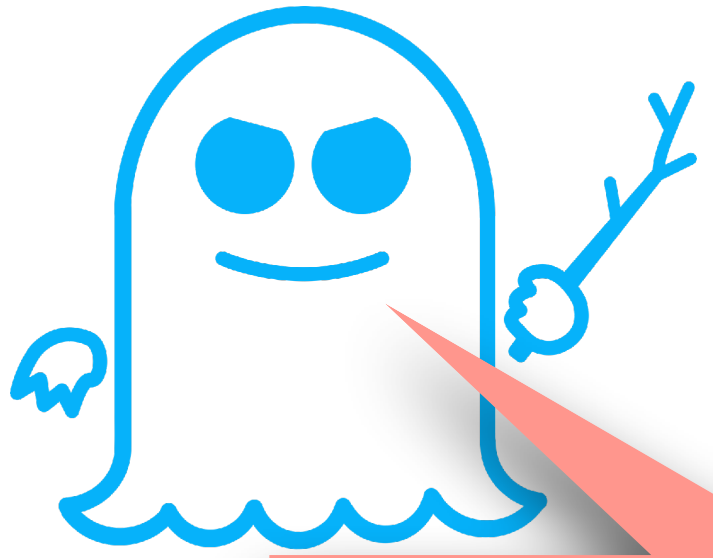
Spectre V1

A_size=16

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

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





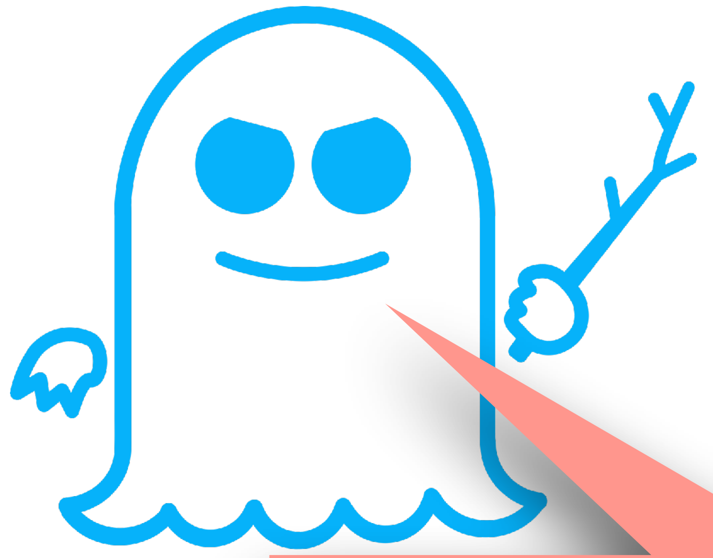
What is in A[128]?

1) Training `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with x `= 128`

Spectre V1

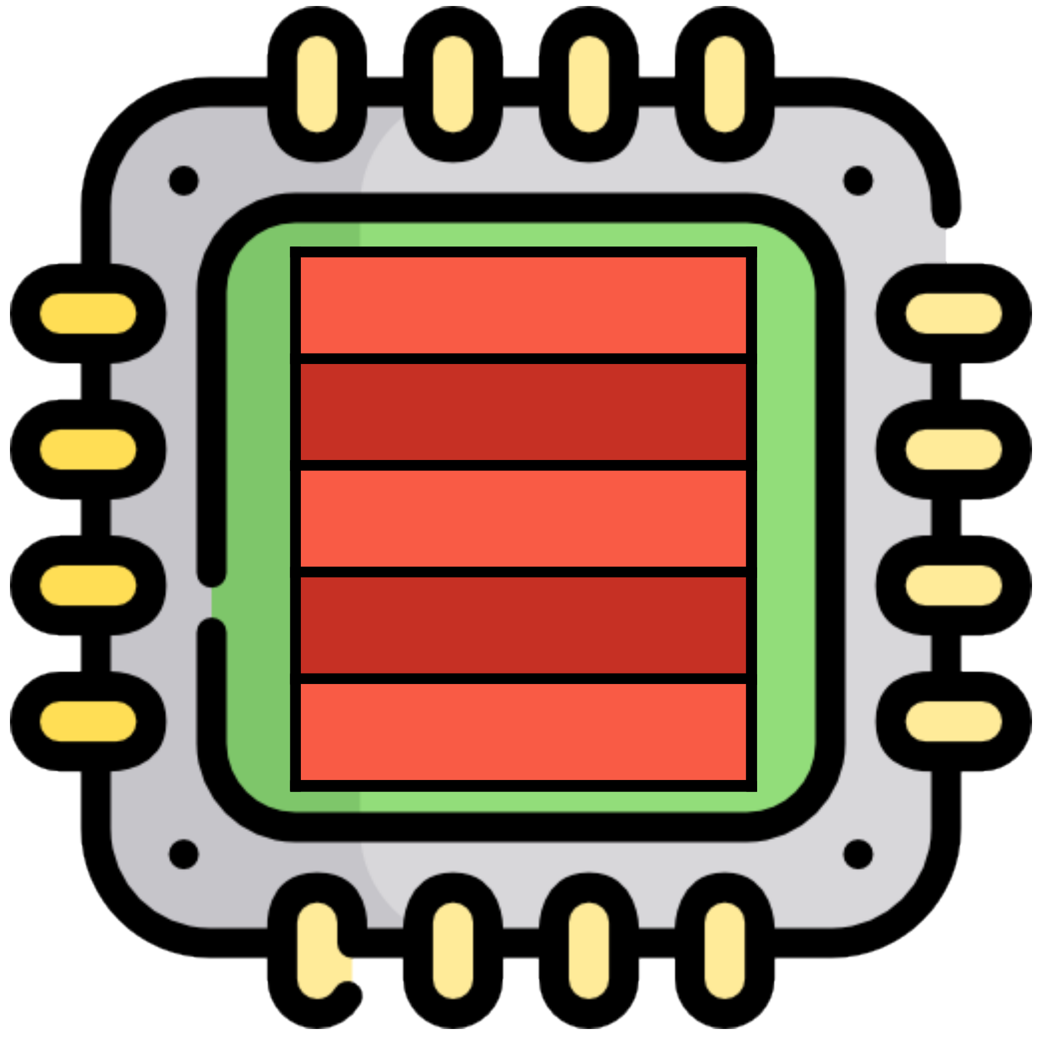


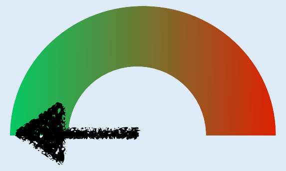
What is in `A[128]`?

```
A_size=16
```

B `B[0]` `B[1]` `B[A[128]]`

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

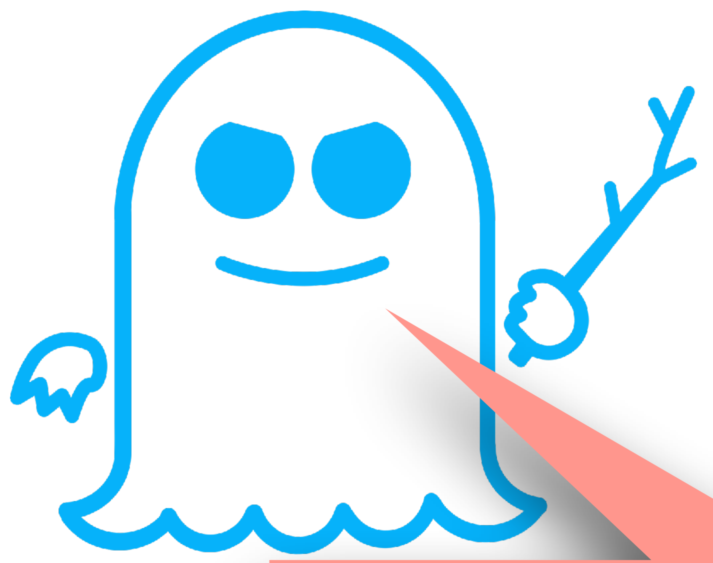


1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with `x = 128`

Spectre V1

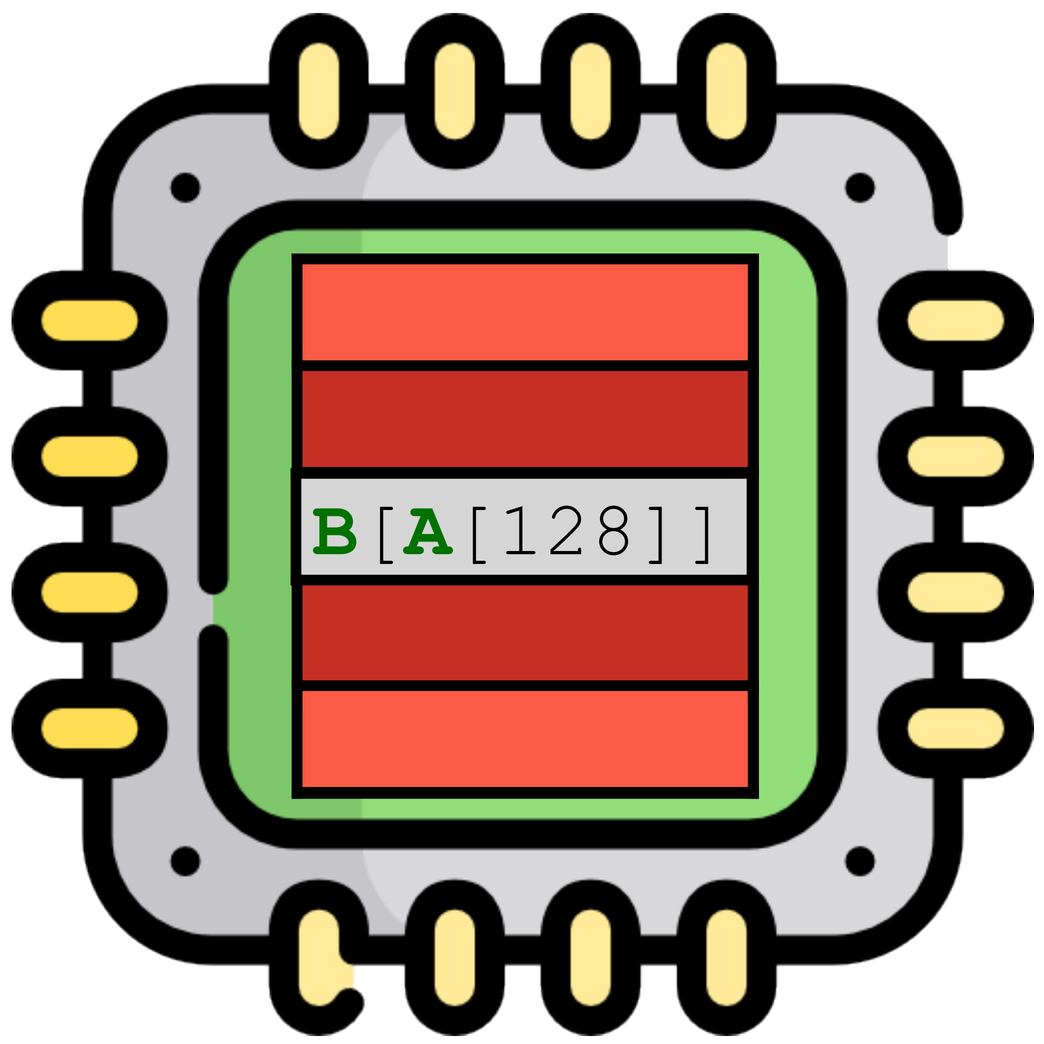


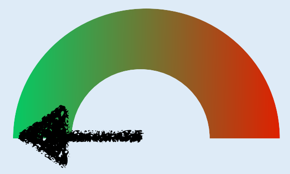
What is in A[128]?

A_size=16

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

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



1) Training  f(0);f(1);f(2); …

2) Prepare cache

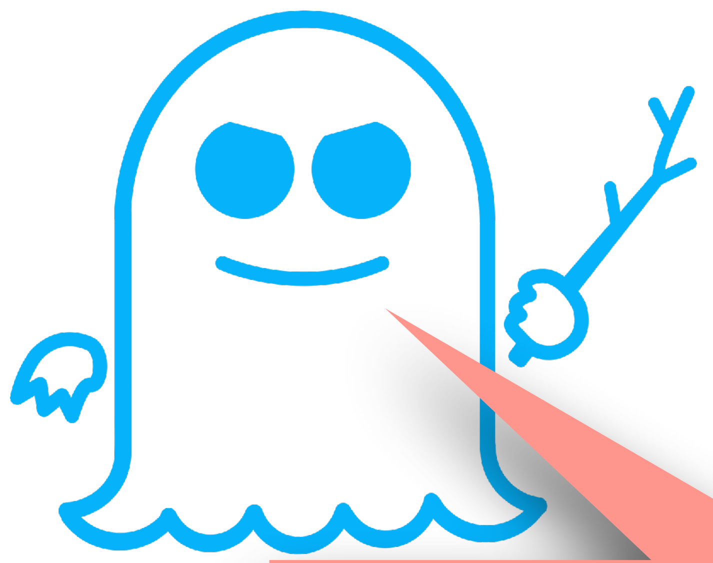
3) Run with x = 128

Spectre V1

A_size=16

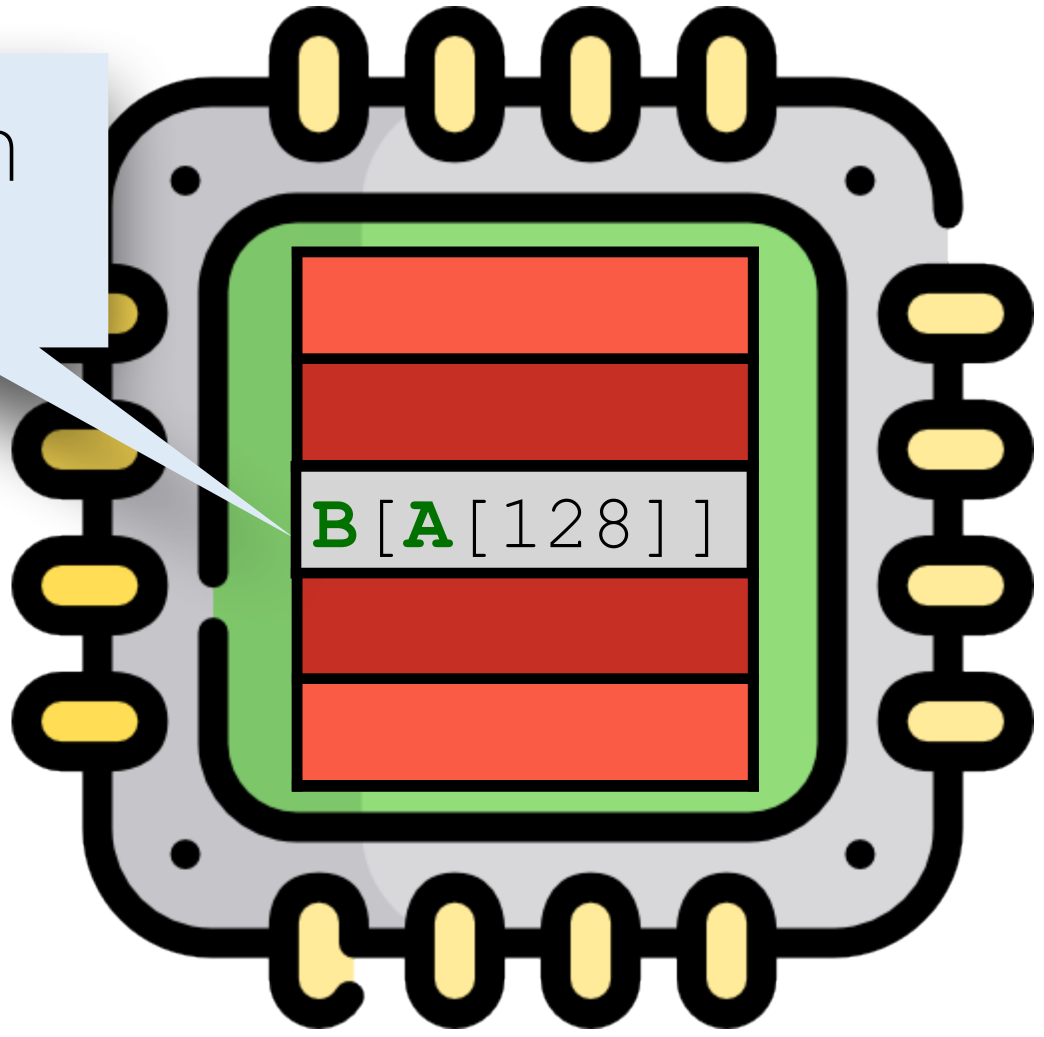
B | B[0] | B[1] | | B[A[128]] |

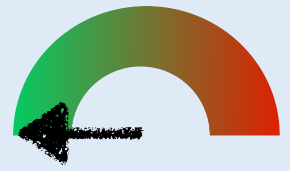
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



What is in A[128]?

Depends on A[128]



1) Training  f(0);f(1);f(2); …

2) Prepare cache

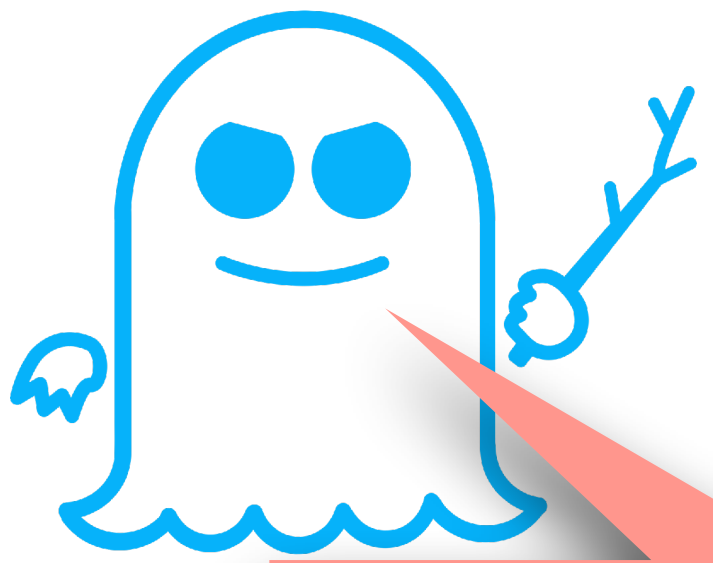
3) Run with x = 128

Spectre V1

A_size=16

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

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



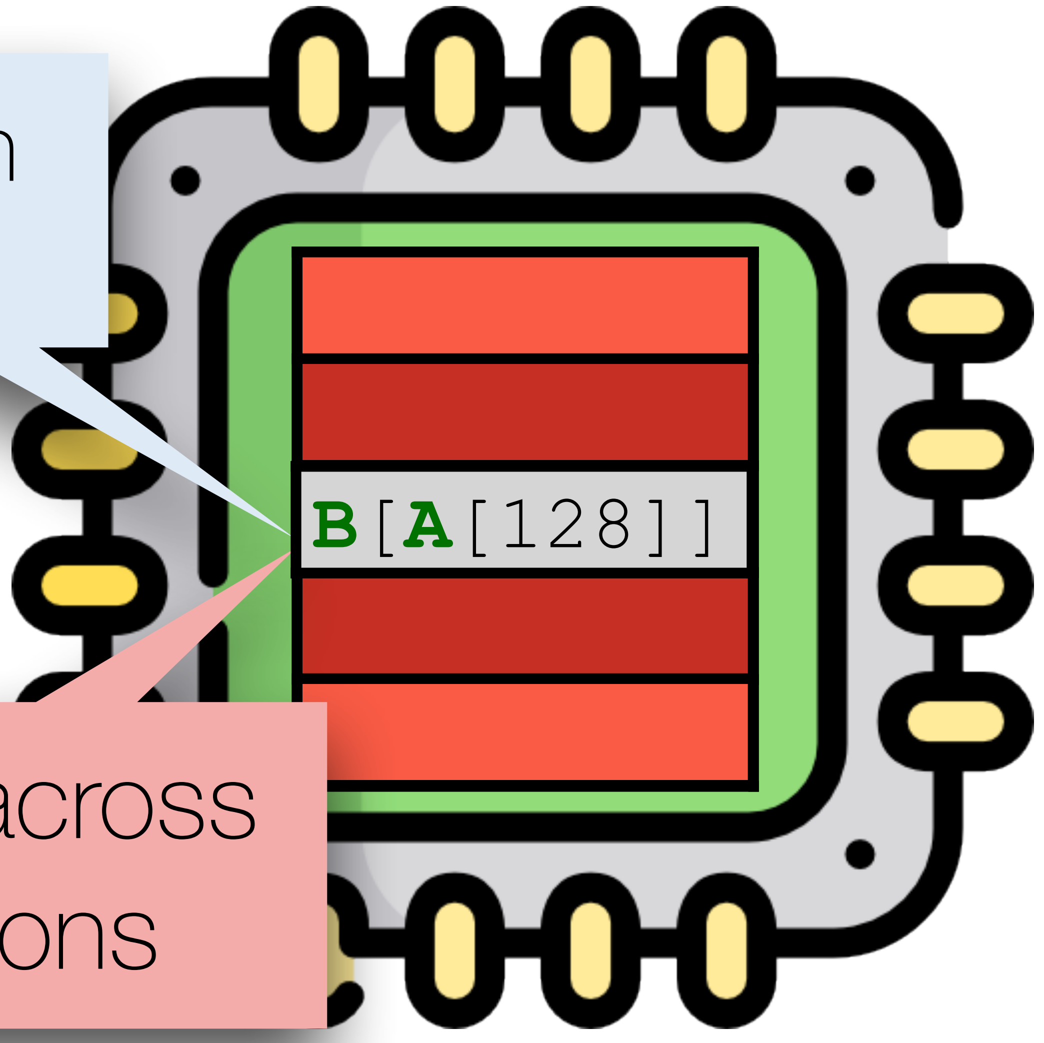
What is in A[128]?

1) Training f(0);f(1);f(2); …

2) Prepare cache

3) Run with x = 128

Depends on A[128]



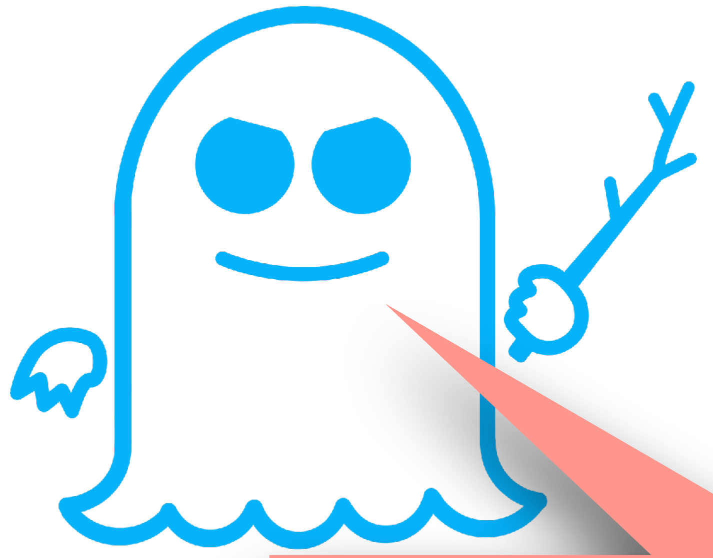
Persistent across speculations

Spectre V1

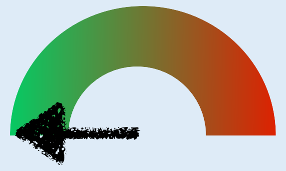
A_size=16

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

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



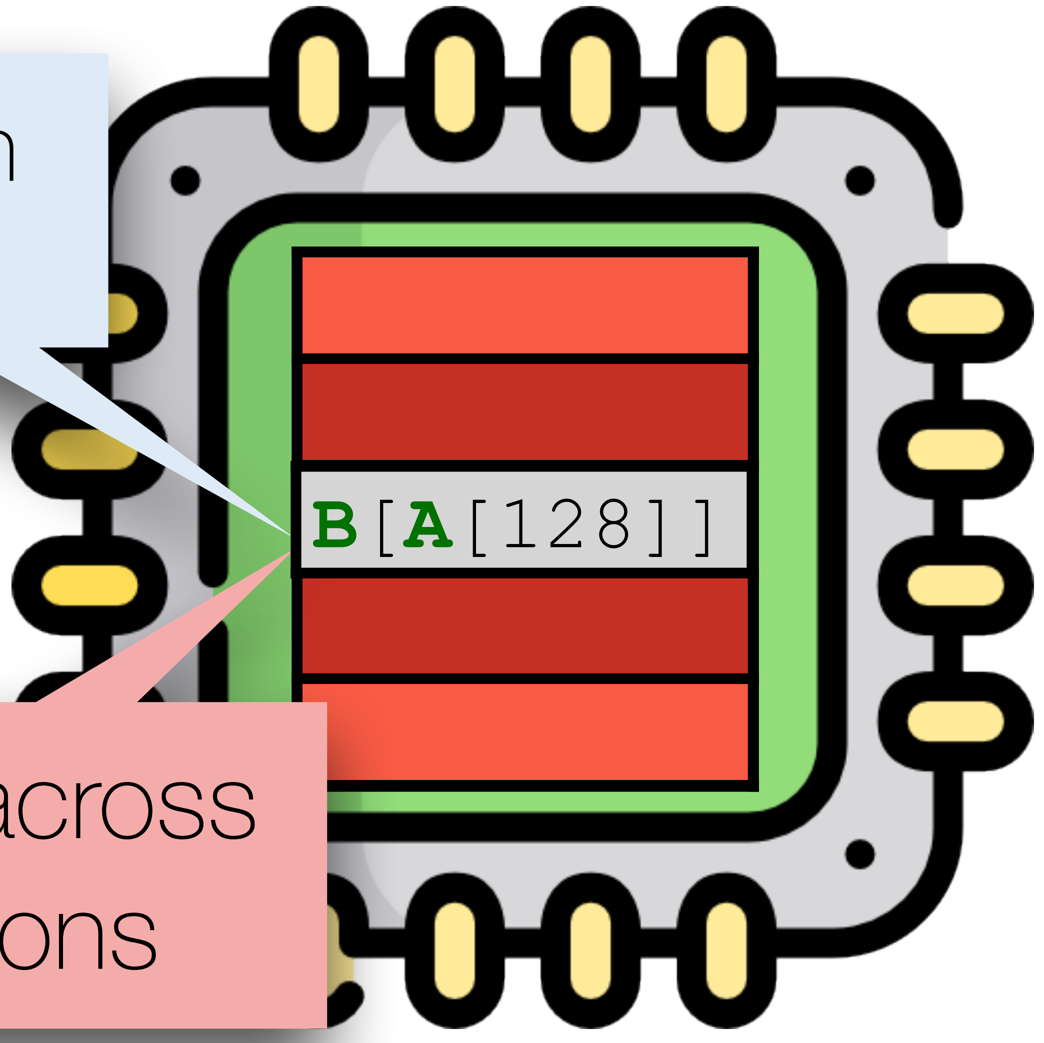
What is in A[128]?

1) Training  f(0);f(1);f(2); …

2) Prepare cache

3) Run with x = 128

4) Extract from cache



Depends on A[128]

Persistent across speculations

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

Compiler-level countermeasures

- Example: insert LFENCE to selectively stop 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

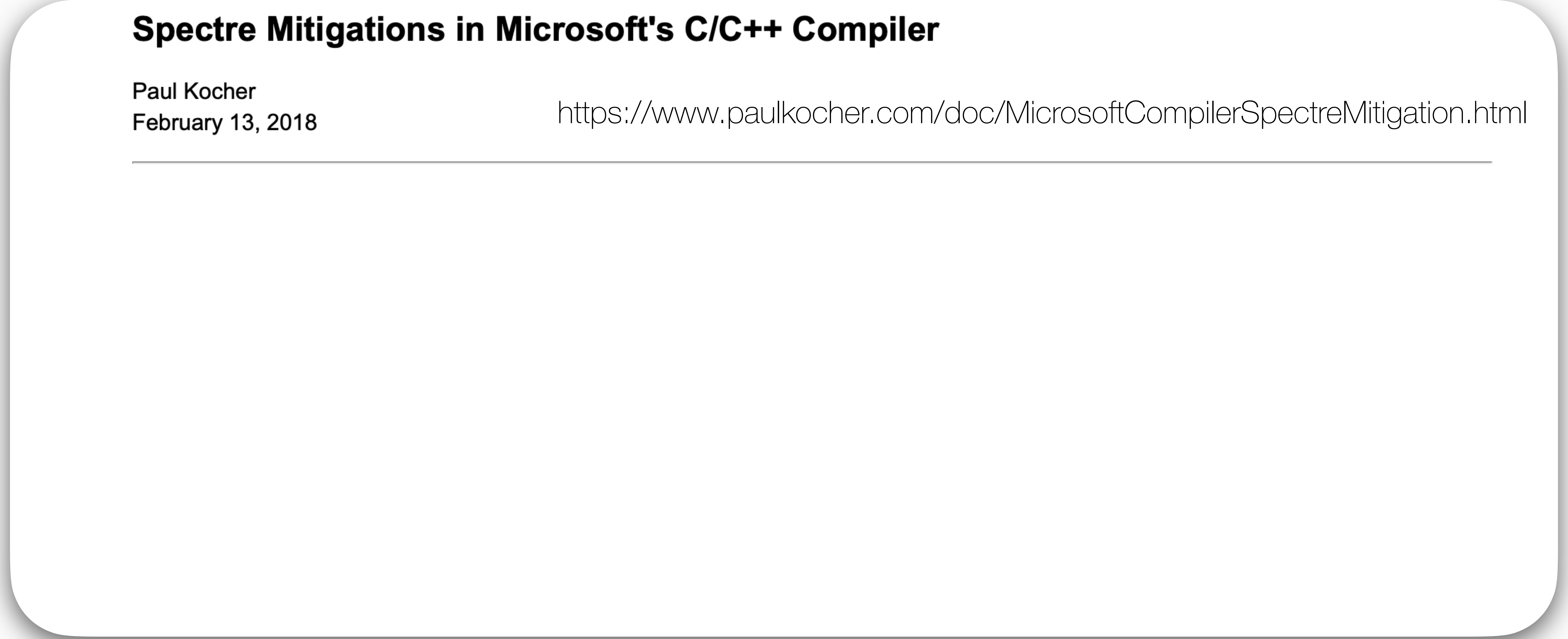
PROBLEM SOLVED ?

Compiler-level countermeasures

- Example: insert LFENCE to selectively stop speculative execution
- Implemented in major compilers (Microsoft Visual C++, Intel ICC, Clang)

Compiler-level countermeasures

Compiler-level countermeasures


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

Paul Kocher
February 13, 2018

https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html

Compiler-level countermeasures

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

Paul Kocher
February 13, 2018

https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html

---

“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

https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html

---

“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

https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html

---

"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!

Outline

1. Speculative execution attacks 101
2. Speculative non-interference
3. Detecting speculative leaks
4. Spectector + Case studies

Speculative execution attacks 101

Speculative execution + branch prediction

Speculative execution + branch prediction

Size of array **A**

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

Speculative execution + branch prediction

Size of array **A**

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

Speculative execution + branch prediction

Size of array **A**

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

Branch predictor

Speculative execution + branch prediction

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

Size of array **A**

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

Branch predictor

Speculative execution + branch prediction

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

Size of array **A**

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

Branch predictor

Speculative execution + branch prediction

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

Size of array **A**

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

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

- Architectural (ISA) state
- Microarchitectural state

Branch predictor

Speculative non-interference

Speculative non-interference

Speculative non-interference

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

Speculative non-interference

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

Informally:

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

How to capture leakage?

Non-speculative semantics

Speculative semantics

+

Attacker model

How to capture leakage?

Non-speculative semantics

Speculative semantics

+

Attacker model

Model program's behavior

How to capture leakage?

Capture attacker's observational power

Non-speculative semantics

+

Attacker model

Speculative semantics

Model program's behavior

µAssembly + 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:
```

µAssembly + non-speculative semantics

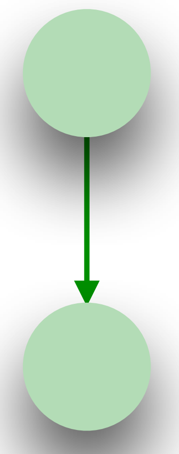
```
if (x < A_size)
    y = B[A[x]]
```

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

μAssembly + non-speculative semantics

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

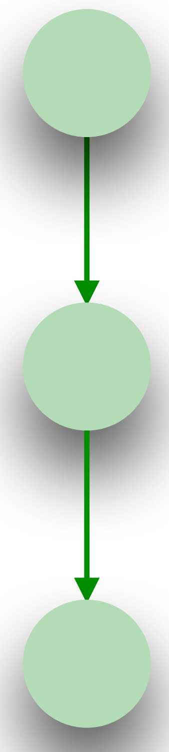
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```



µAssembly + non-speculative semantics

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

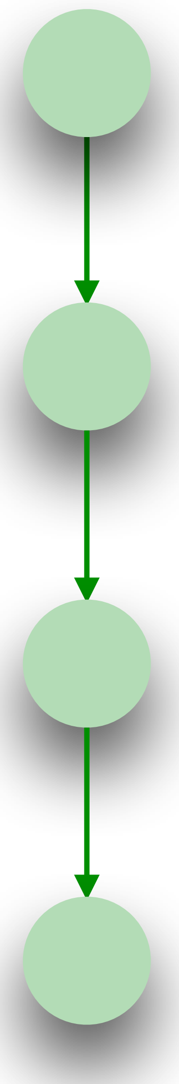
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```



μAssembly + non-speculative semantics

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

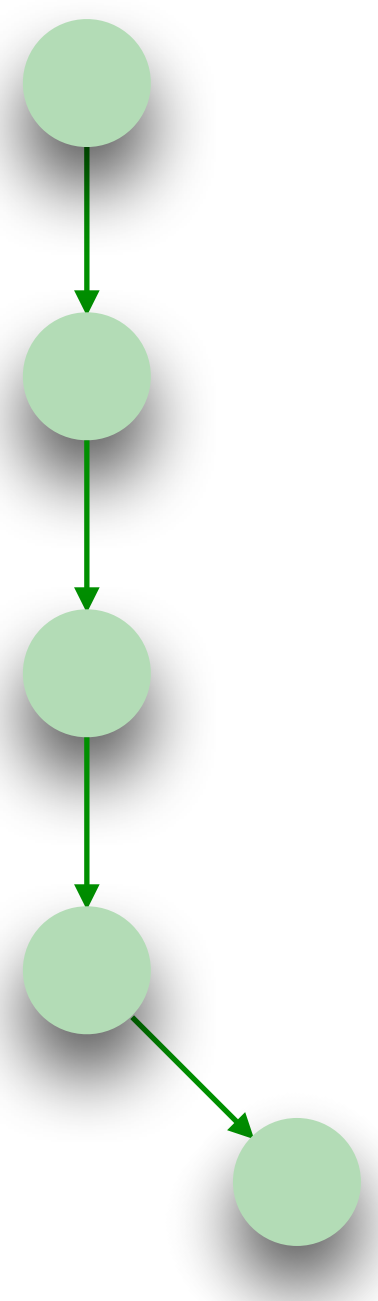
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```



µAssembly + 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:
```



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
END:
```

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

Committed upon correct speculation

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle ***O*** : branch prediction + length of speculative window

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 branch instructions

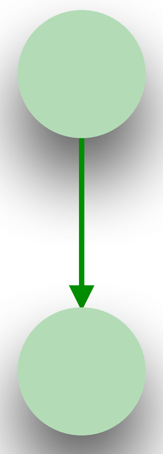
Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

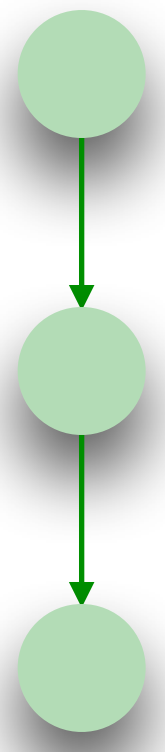
Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

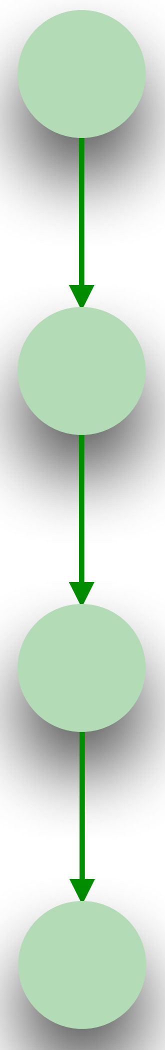
Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

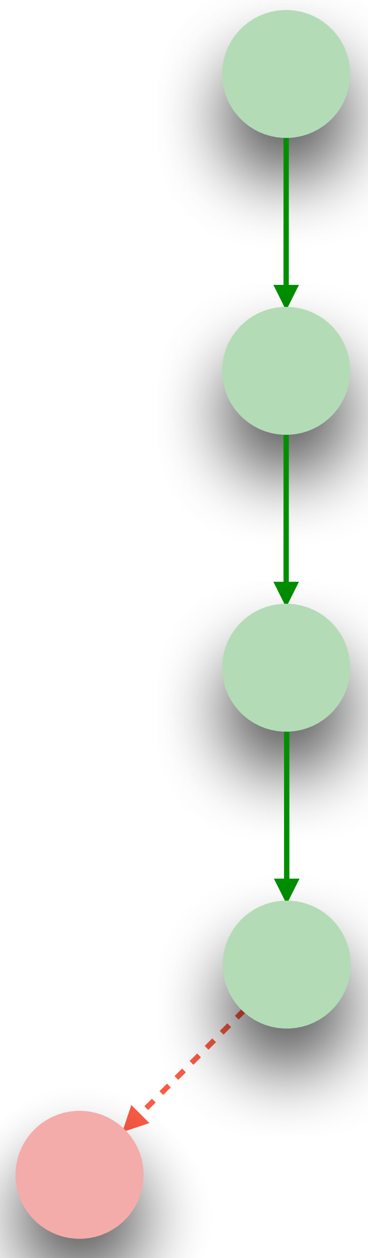
Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

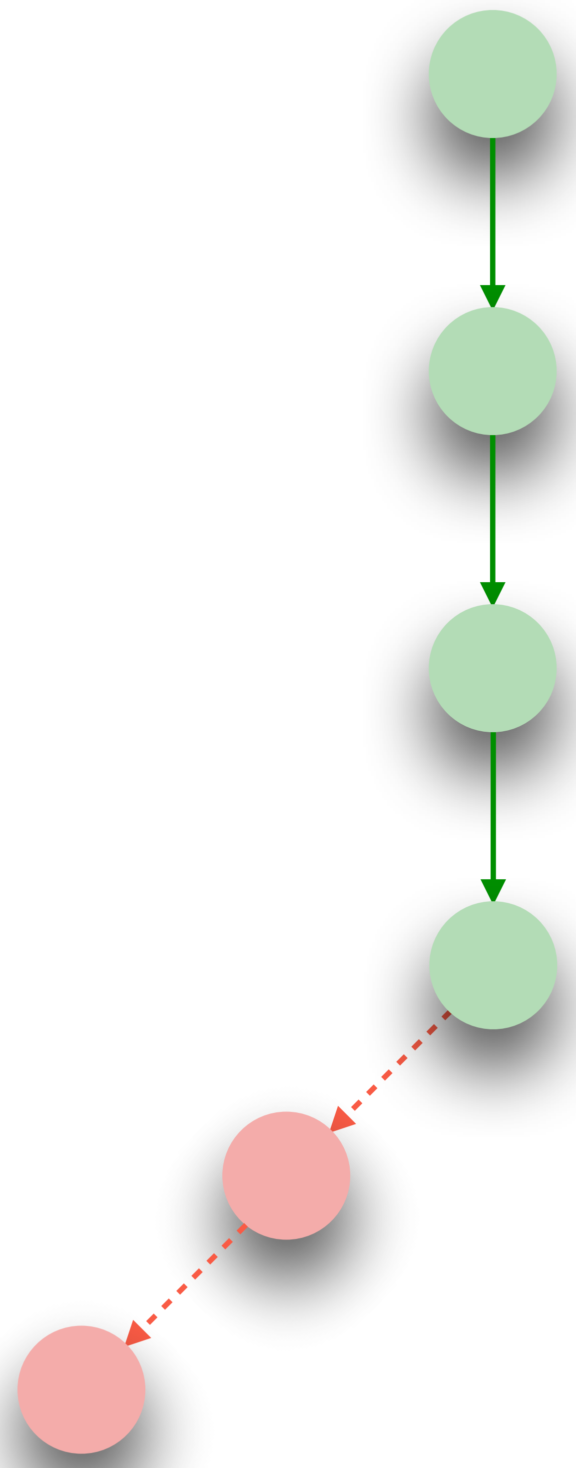
Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

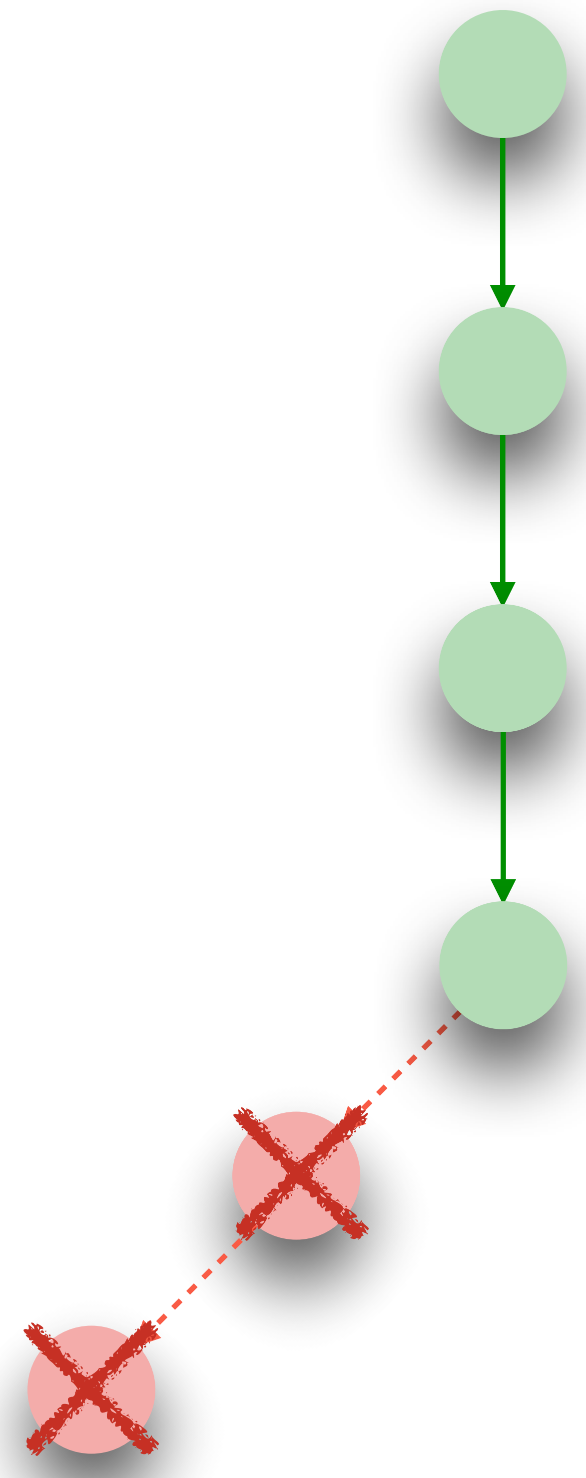
Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

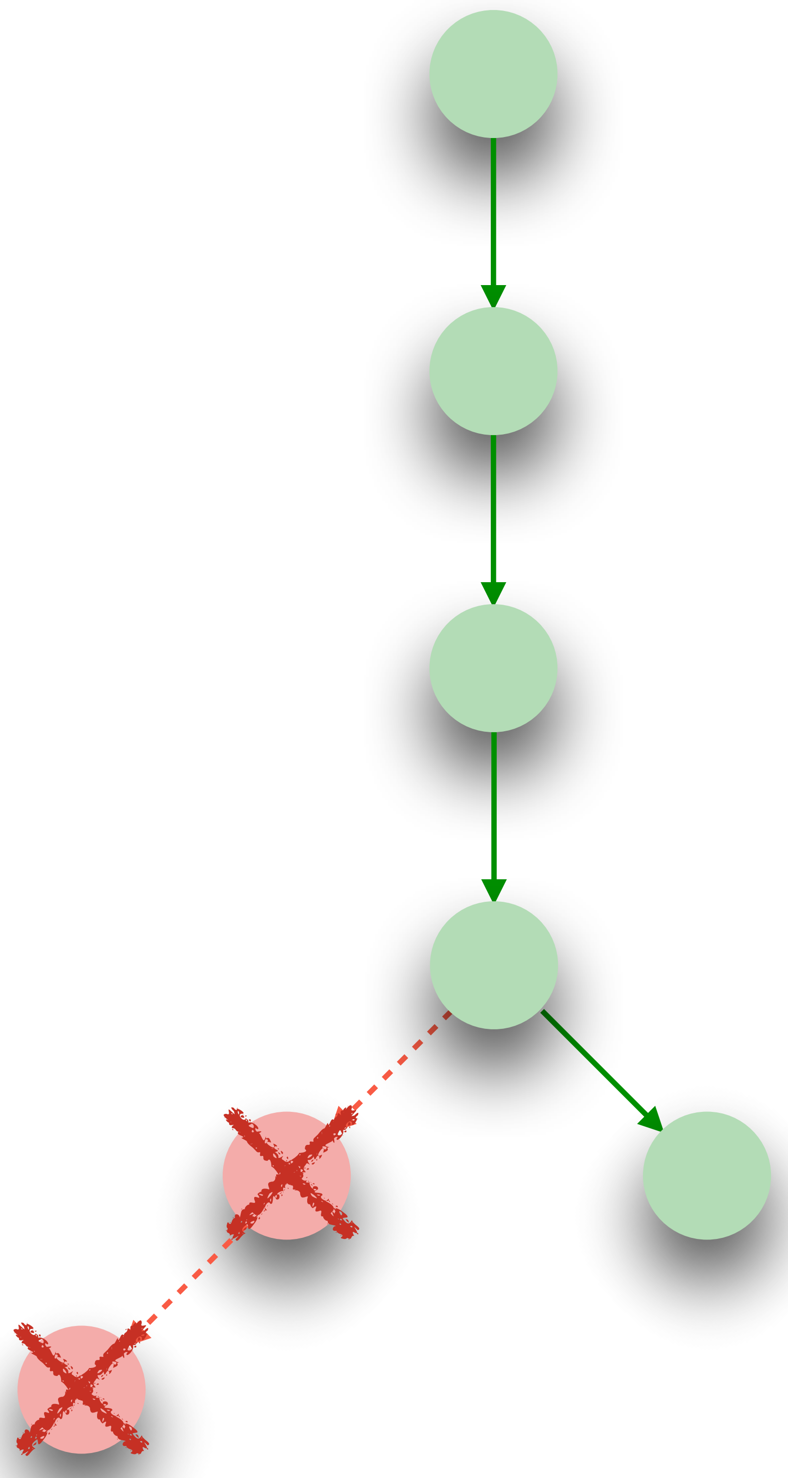
Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle *O* : branch prediction + length of speculative window

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 branch instructions

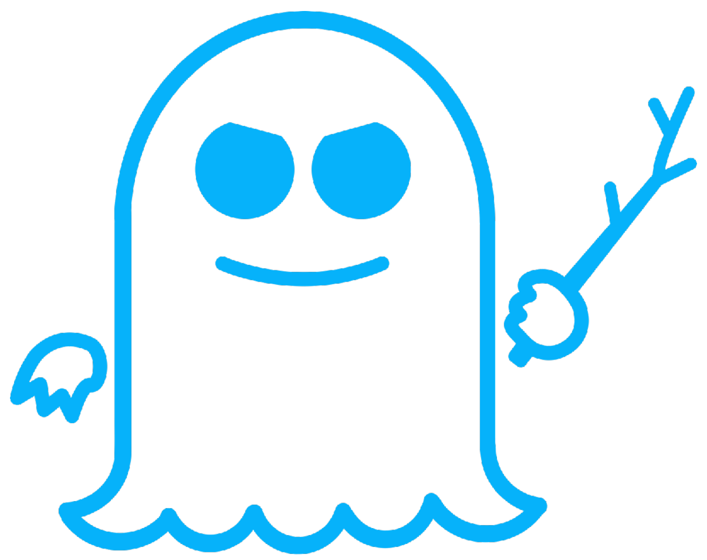
Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle *O* : branch prediction + length of speculative window

Leakage into μarchitecture

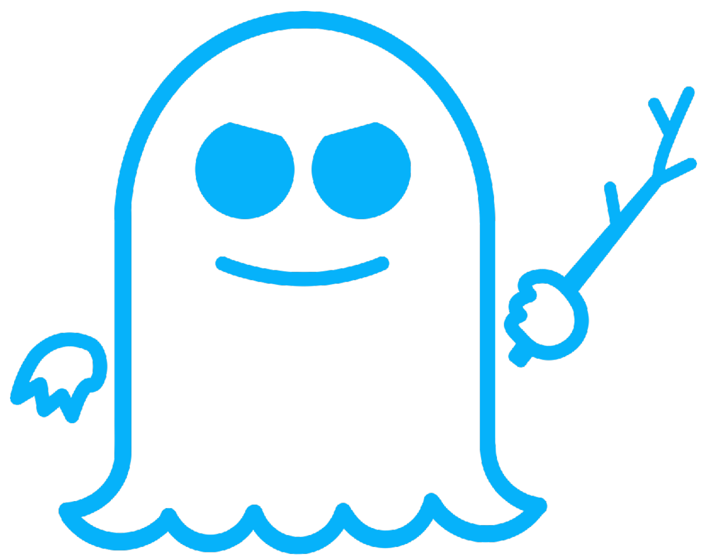
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Leakage into μarchitecture

```
    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

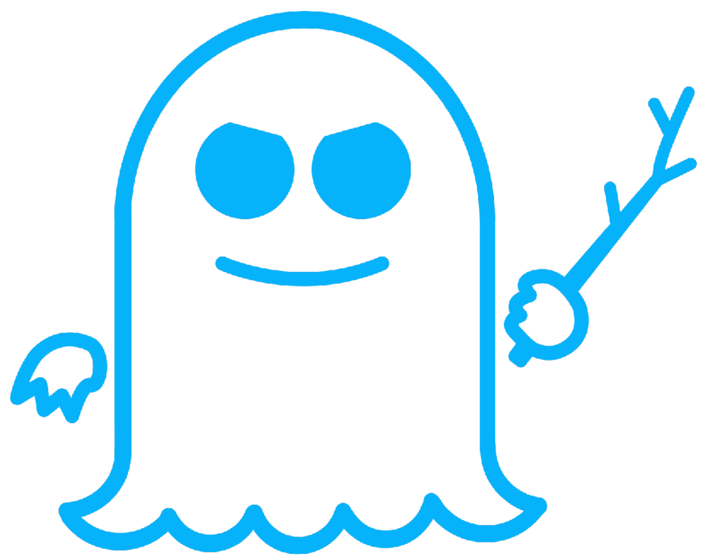


Leakage into μarchitecture

```
    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" rqmts



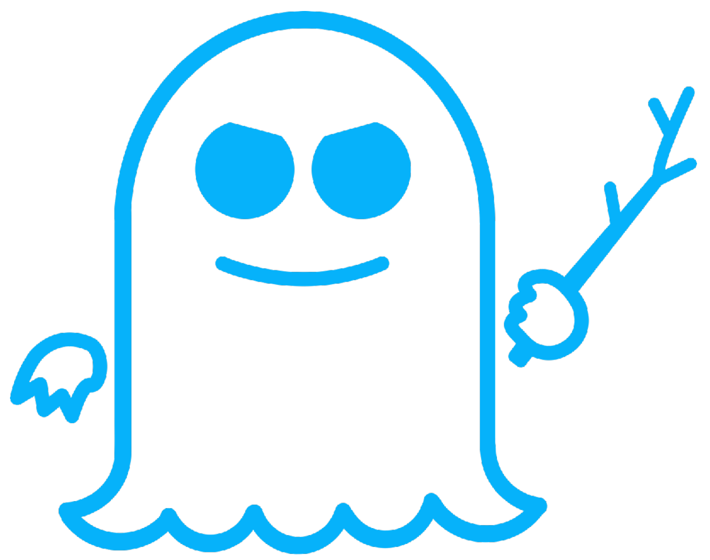
Leakage into μarchitecture

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

Attacker can observe:

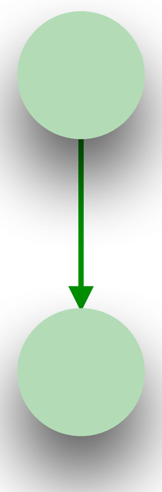
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by “constant-time” rqmts



Leakage into μarchitecture

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

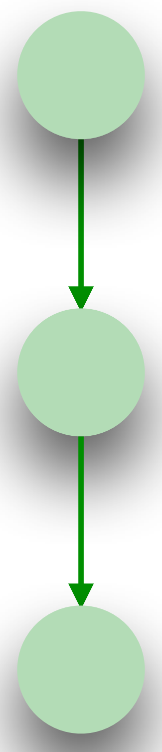


Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by “constant-time” rqmts

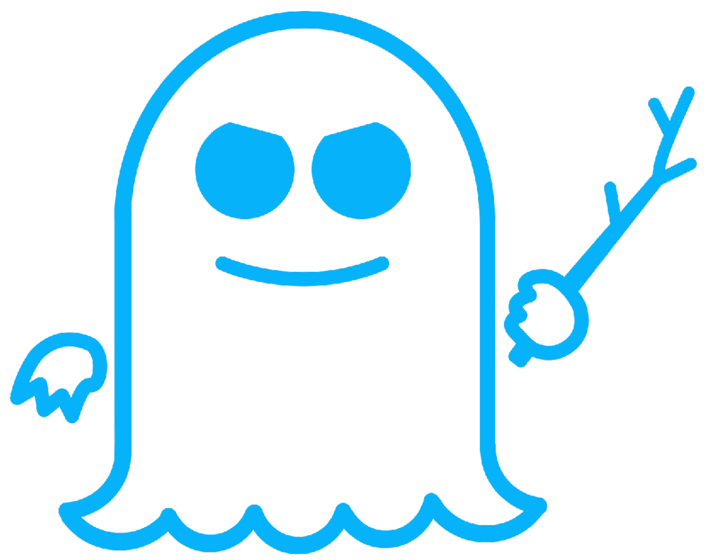
Leakage into μarchitecture

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



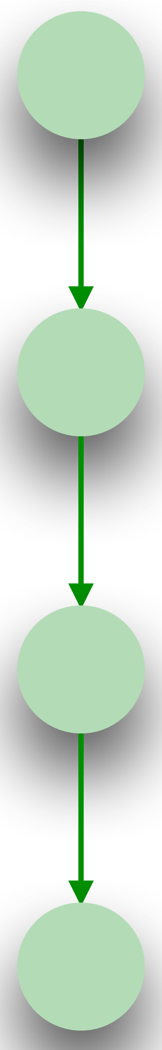
Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by "constant-time" rqmts



Leakage into μarchitecture

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

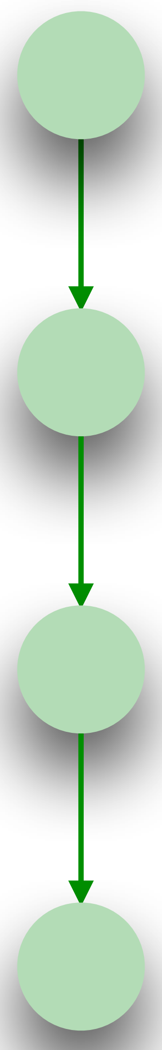


Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by "constant-time" rqmts

Leakage into µarchitecture

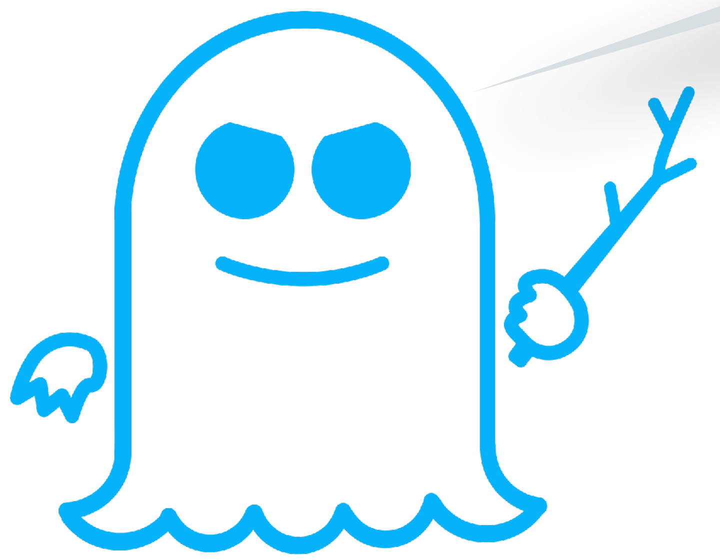
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



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

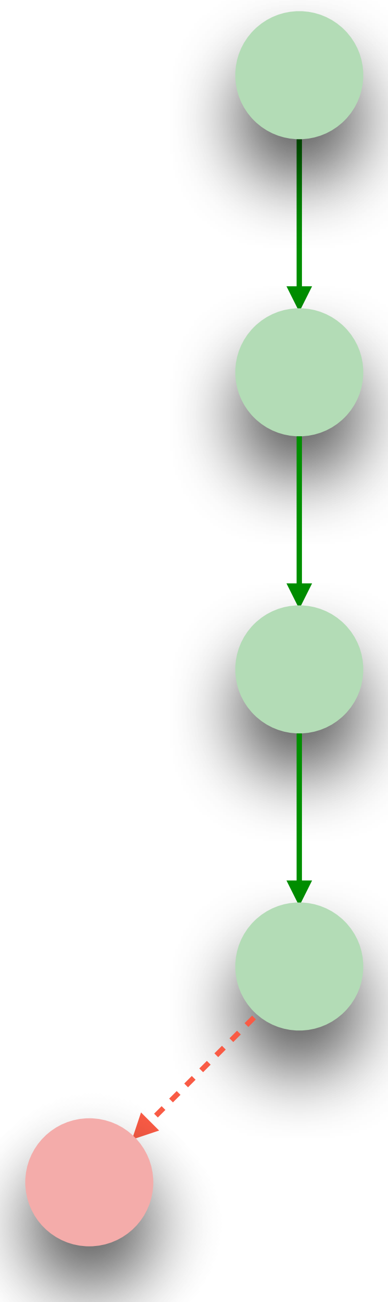
Inspired by “constant-time” rqmts

start
pc L1



Leakage into μarchitecture

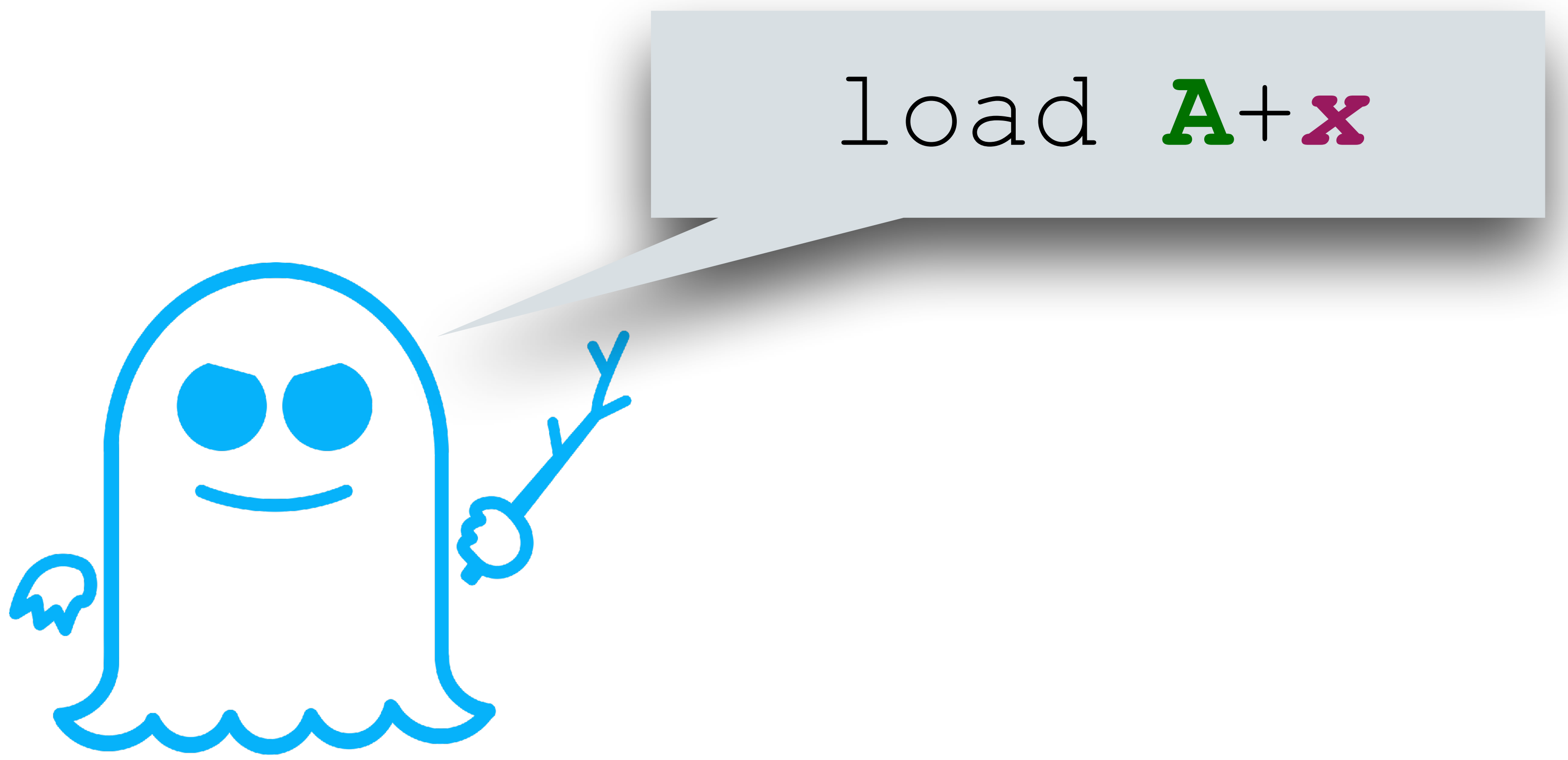
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Attacker can observe:

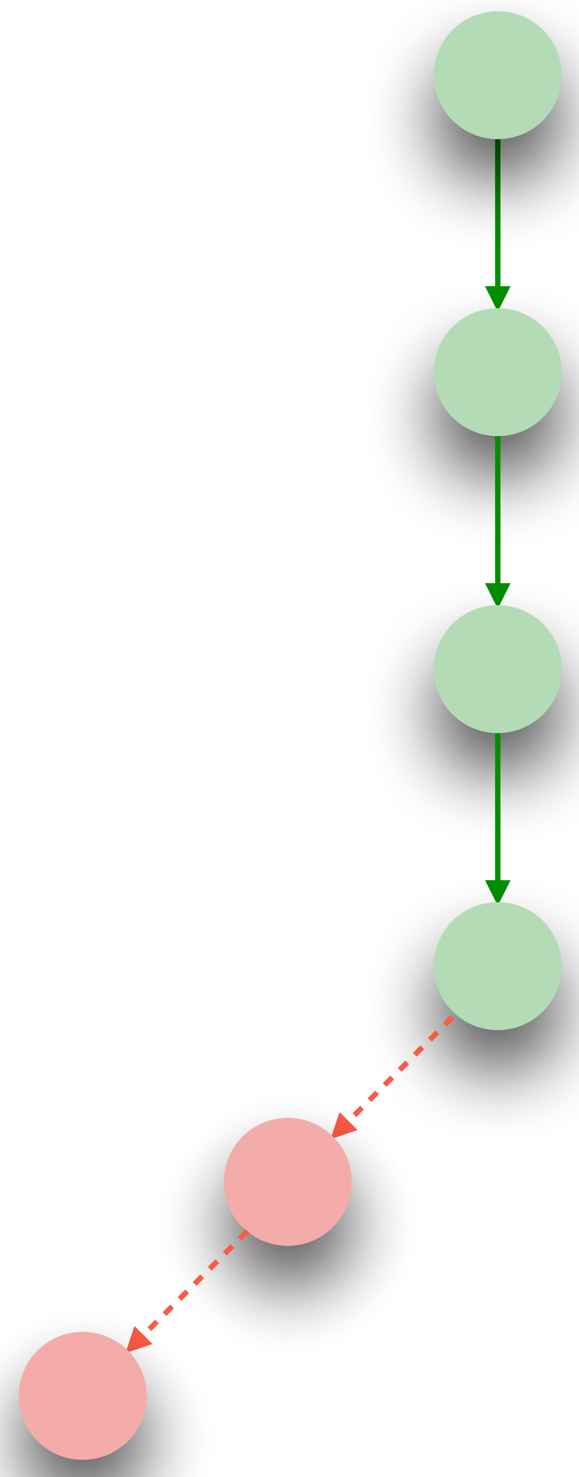
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by "constant-time" rqmts



Leakage into μarchitecture

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



```
load B+A[x]
```

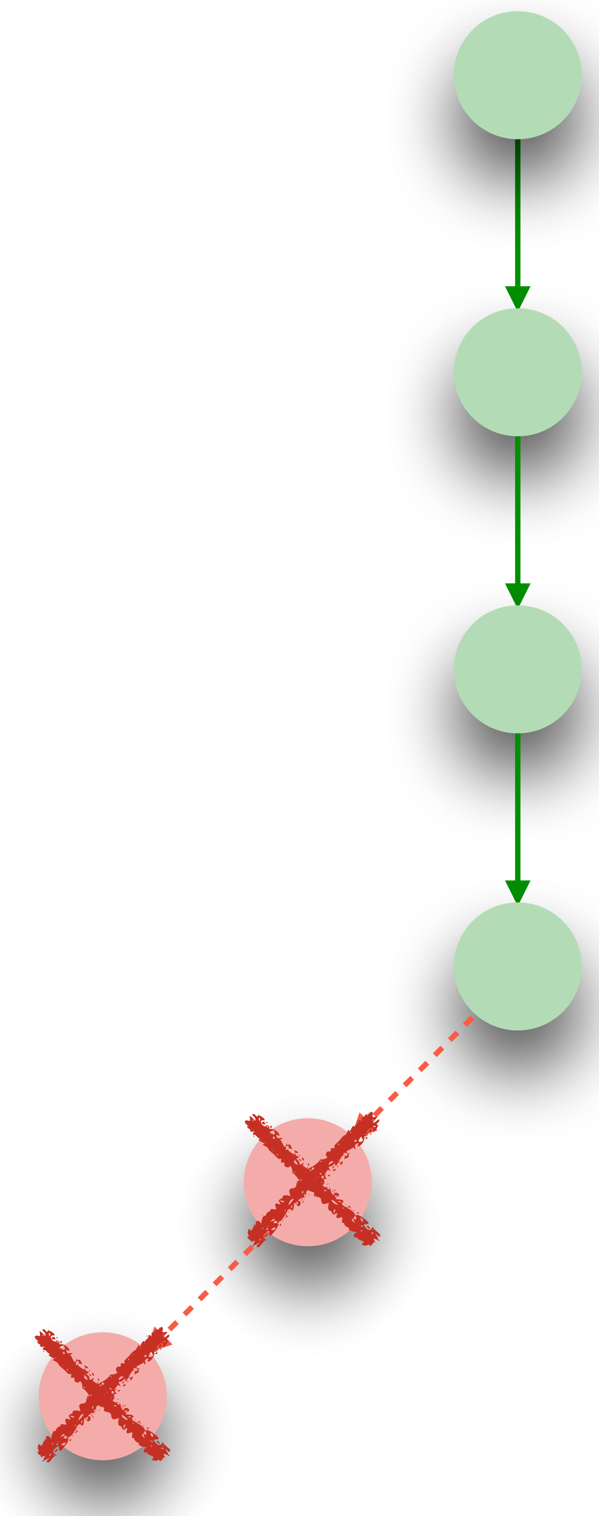
Attacker can observe:

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

Inspired by “constant-time” rqmts

Leakage into μarchitecture

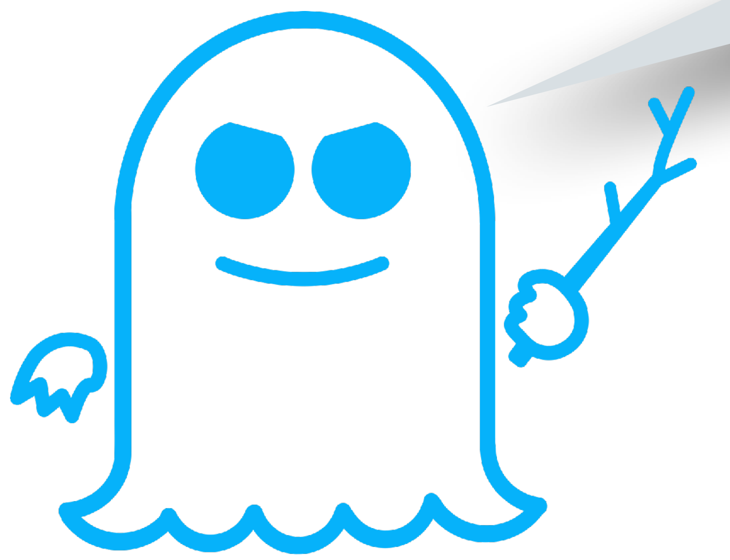
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Attacker can observe:

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

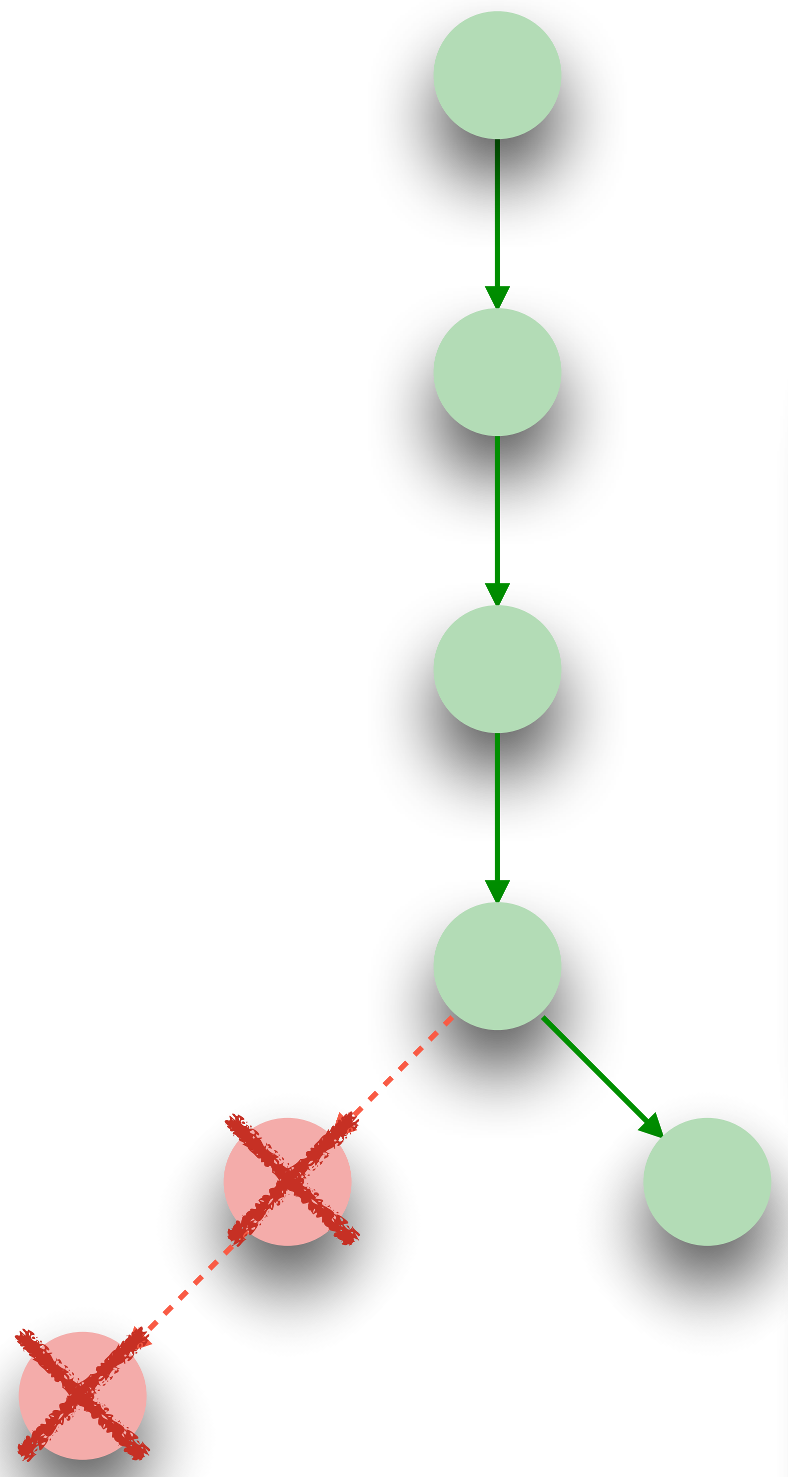
Inspired by "constant-time" rqmts



```
rollback
pc END
```

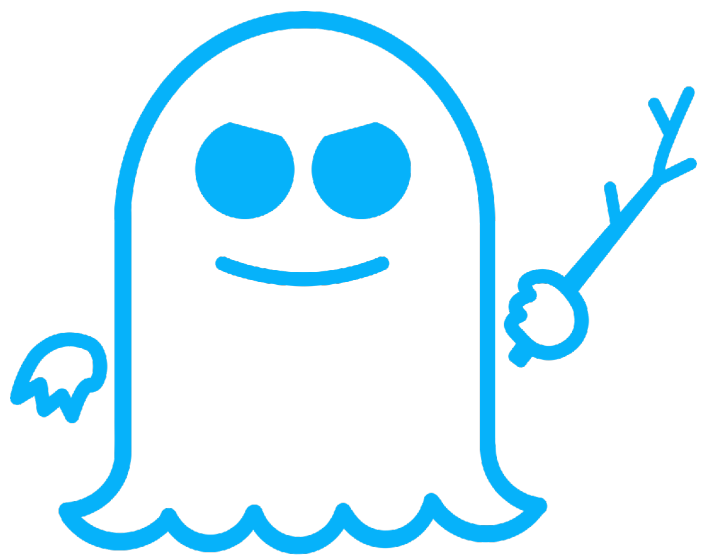
Leakage into µarchitecture

```
    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" rqmts



Speculative non-interference

Formally!

Speculative non-interference

Formally!

Program **P** is **speculatively non-interferent** for prediction oracle ***O*** if

Speculative non-interference

Formally!

Program **P** is **speculatively non-interferent** for prediction oracle ***O*** if

For all program states ***s*** and ***s'***:

Speculative non-interference

Formally!

Program $\mathtt{P}$ is **speculatively non-interferent** for prediction oracle $\boldsymbol{O}$ if

For all program states $\boldsymbol{s}$ and $\boldsymbol{s}'$:

$$\mathtt{P}_{\mathtt{non\text{-}spec}}(\boldsymbol{s}) = \mathtt{P}_{\mathtt{non\text{-}spec}}(\boldsymbol{s}')$$

Speculative non-interference

Formally!

Program $\mathtt{P}$ is **speculatively non-interferent** for prediction oracle $\boldsymbol{O}$ if

For all program states s and s':

$$\mathtt{P}_{\mathtt{non\text{-}spec}}(s) = \mathtt{P}_{\mathtt{non\text{-}spec}}(s') \implies \mathtt{P}_{\mathtt{spec}}(s,O) = \mathtt{P}_{\mathtt{spec}}(s',O)$$

Reasoning about arbitrary oracles

Reasoning about arbitrary oracles

Always-mispredict speculative semantics

Mispredict ***all*** branch instructions

Fixed speculative window

Rollback of every transaction

Reasoning about arbitrary oracles

Always-mispredict speculative semantics

Mispredict ***all*** branch instructions

Fixed speculative window

Rollback of every transaction

Always-mispredict is ***worst-case***

$$\mathbf{P}_{\mathbf{am}}(s) = \mathbf{P}_{\mathbf{am}}(s') \Leftrightarrow \forall O.\ \mathbf{P}_{\mathbf{spec}}(s,O) = \mathbf{P}_{\mathbf{spec}}(s',O)$$

Reasoning about arbitrary oracles

Always-mispredict speculative semantics

Mispredict ***all*** branch instructions

Fixed speculative window

Rollback of every transaction

Always-mispredict is ***worst-case***

$$\mathbf{P}_{\mathbf{am}}(s) = \mathbf{P}_{\mathbf{am}}(s') \iff$$

$$\forall O.\ \mathbf{P}_{\mathbf{spec}}(s, O) = \mathbf{P}_{\mathbf{spec}}(s', O)$$

If program **P** satisfies

$$\forall s, s'. \mathbf{P}_{\mathbf{non\text{-}spec}}(s) = \mathbf{P}_{\mathbf{non\text{-}spec}}(s')$$

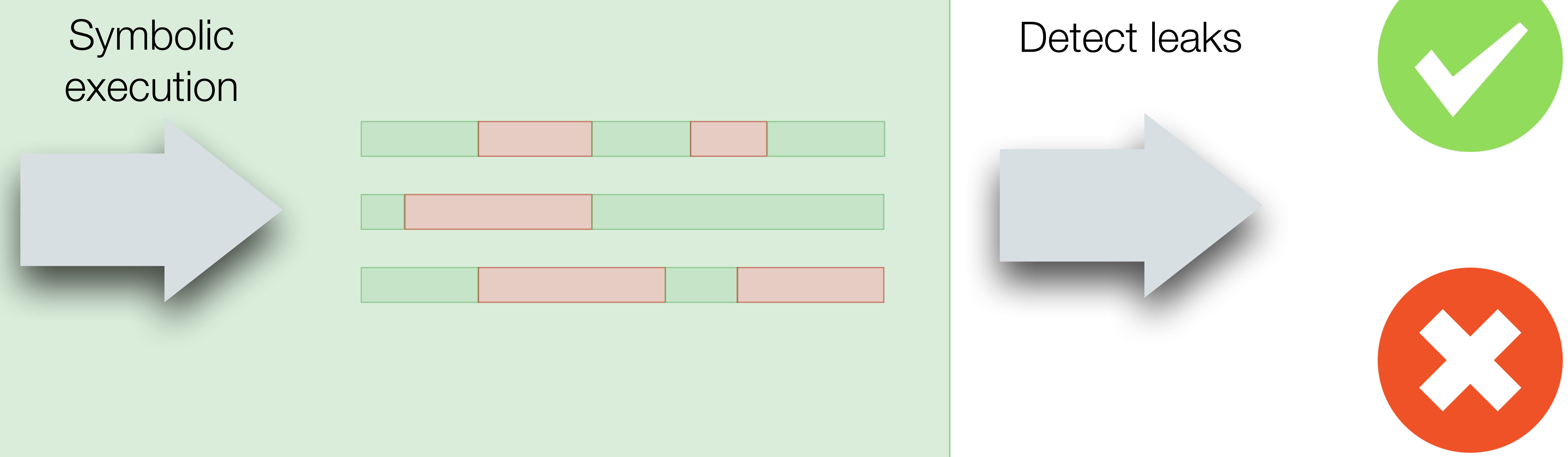
$$\Rightarrow \mathbf{P}_{\mathbf{am}}(s) = \mathbf{P}_{\mathbf{am}}(s')$$

then **P** satisfies ***SNI*** w.r.t. all ***O***

Detecting speculative leaks

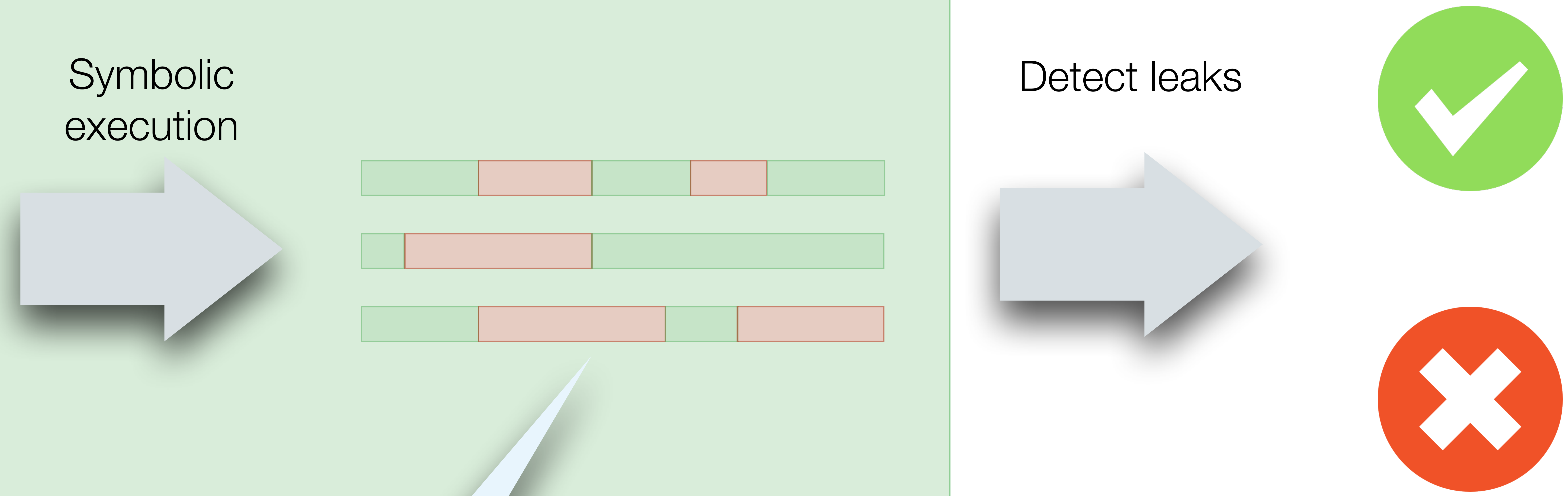
Detecting speculative leaks

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



Detecting speculative leaks

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



Symbolic trace: path condition + observations along the symbolic path

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:
```

Always mispredict
branch instructions

Symbolic execution

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

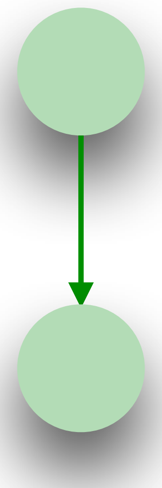
true

Always mispredict
branch instructions

Symbolic execution

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

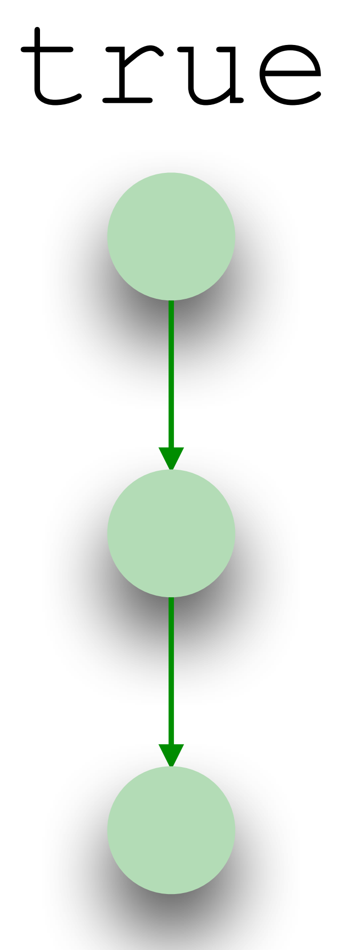
true



Always mispredict branch instructions

Symbolic execution

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

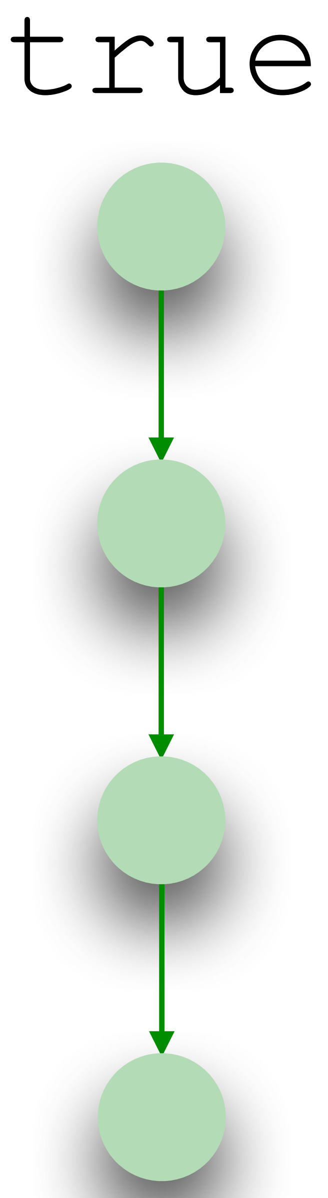


Always mispredict
branch instructions

Symbolic execution

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

Always mispredict
branch instructions



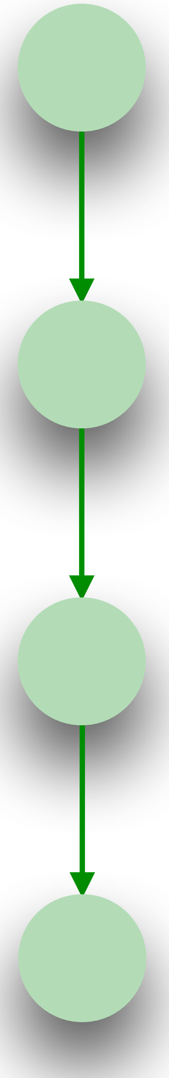
Symbolic execution

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

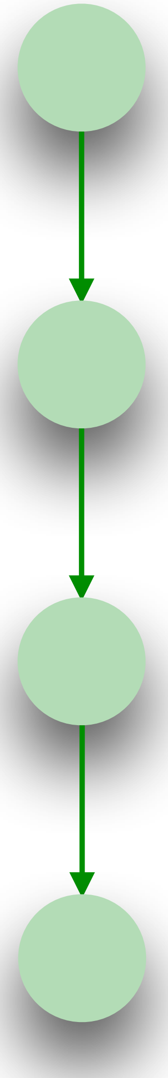
Always mispredict

branch instructions

$x \geq$ A_size



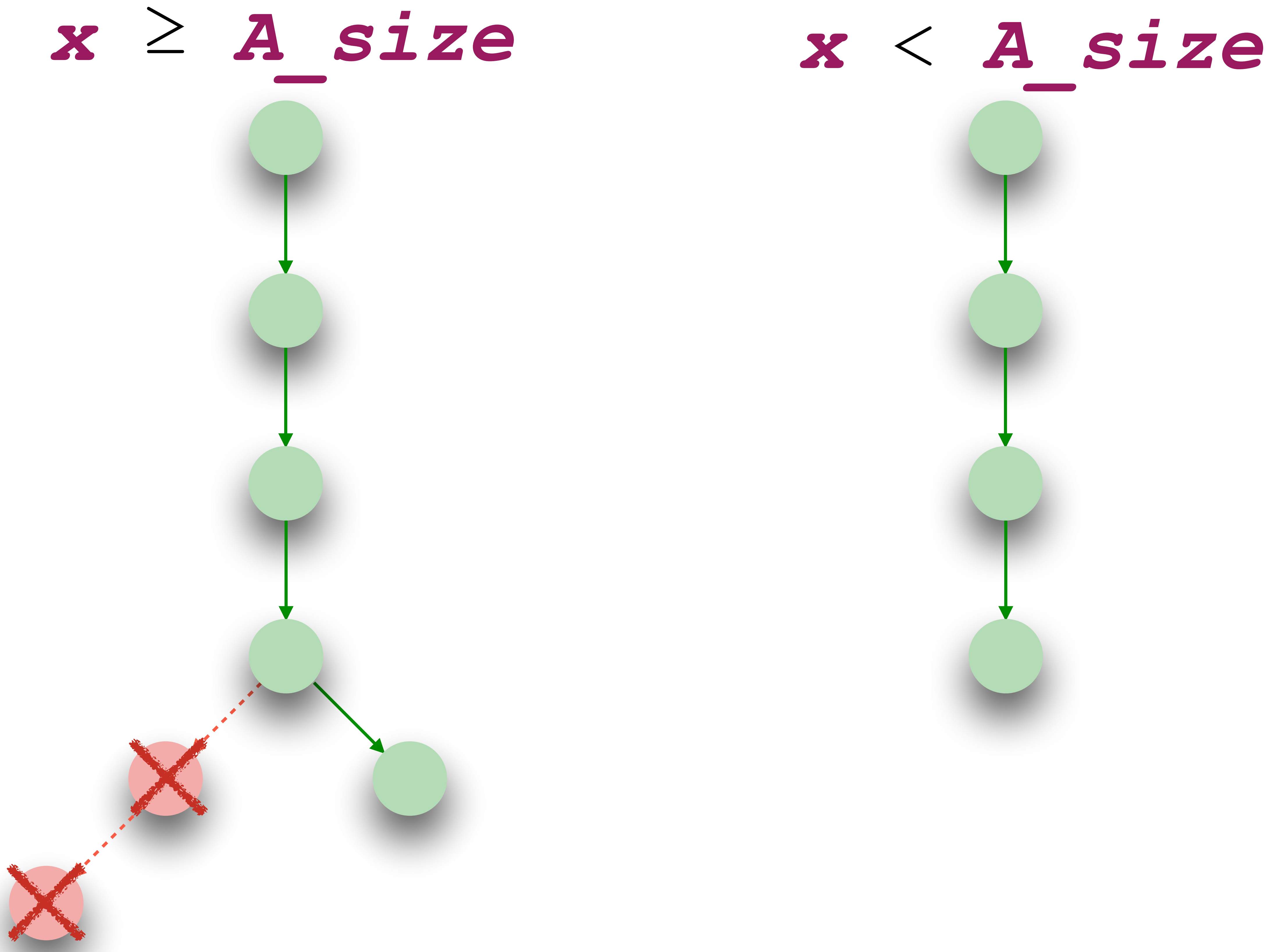
$x <$ A_size



Symbolic execution

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

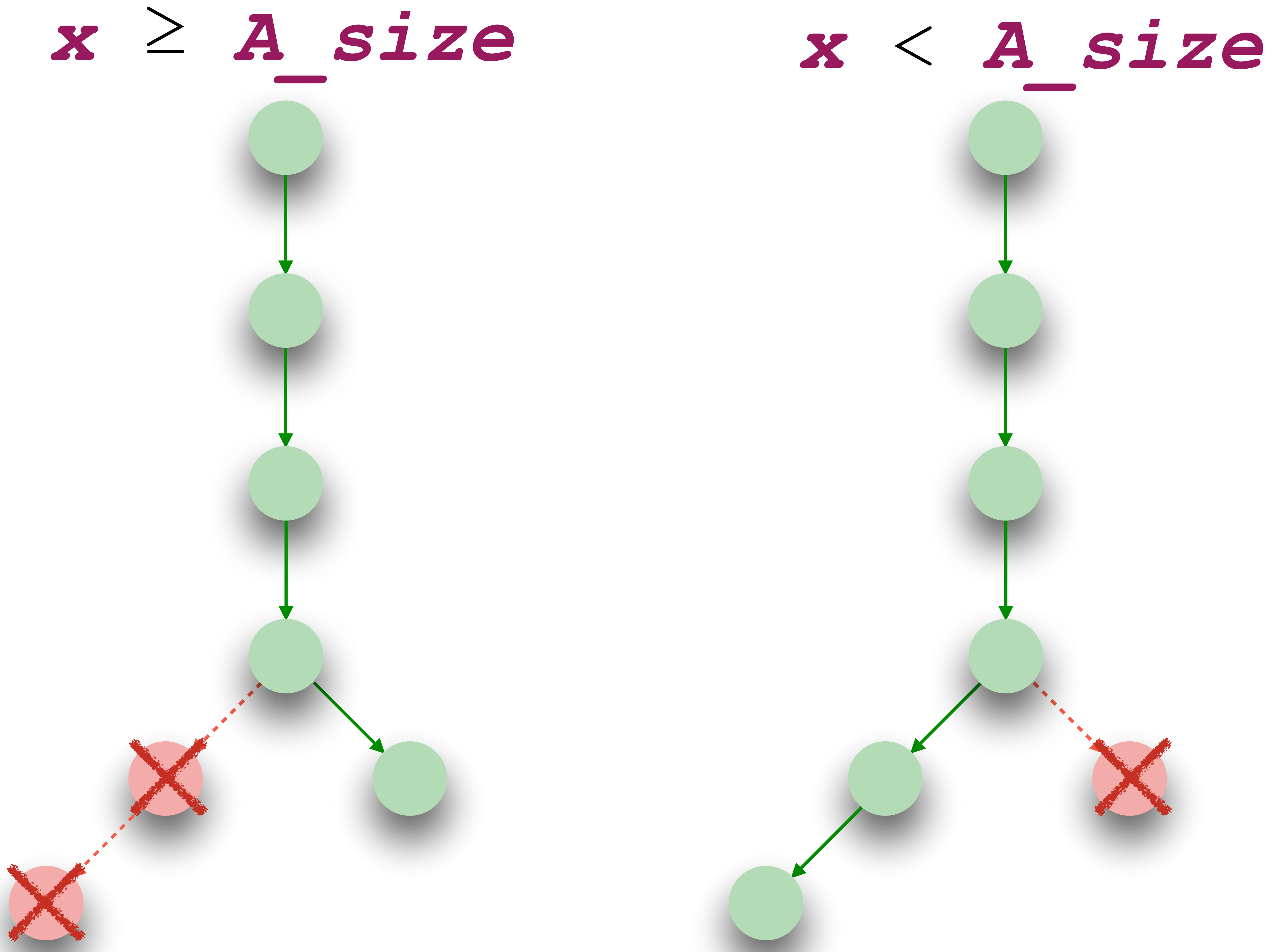
Always mispredict branch instructions



Symbolic execution

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

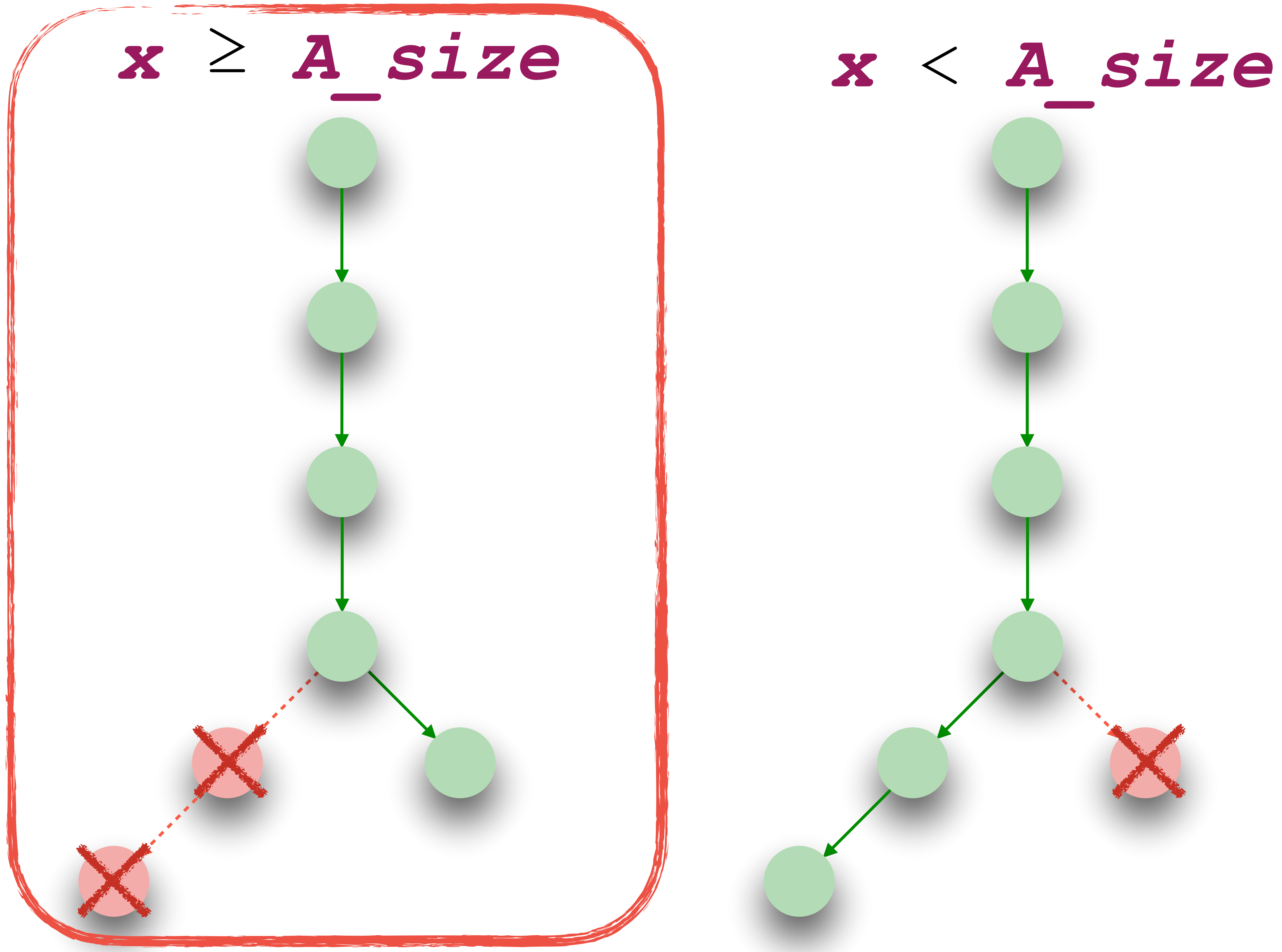
Always mispredict branch instructions



Symbolic execution

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

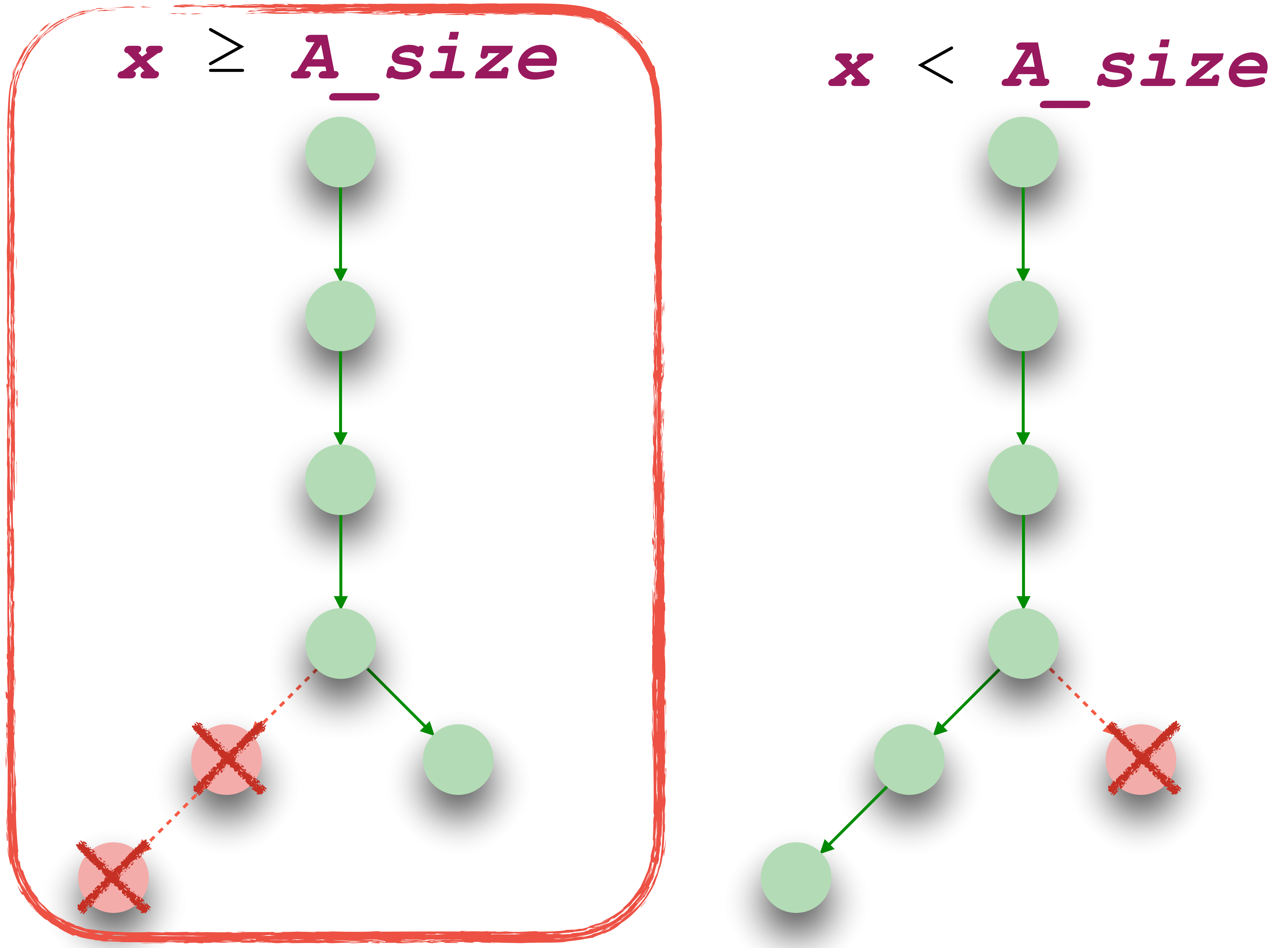
Always mispredict branch instructions



Symbolic execution

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

Always mispredict
branch instructions

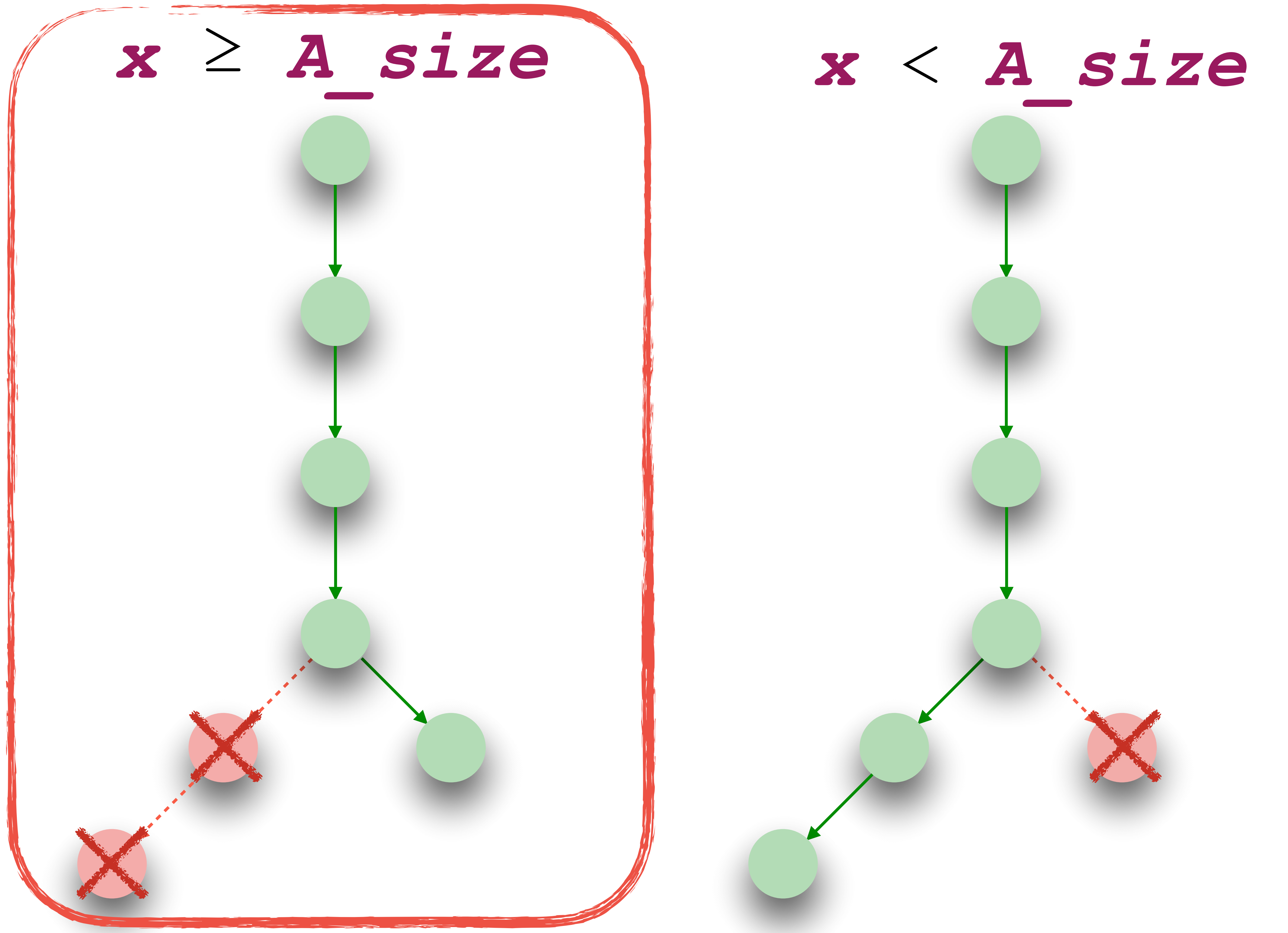


start pc *L1* load **A**+***x*** load **B**+**A**[***x***] rollback pc *END*

Symbolic execution

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

Always mispredict branch instructions

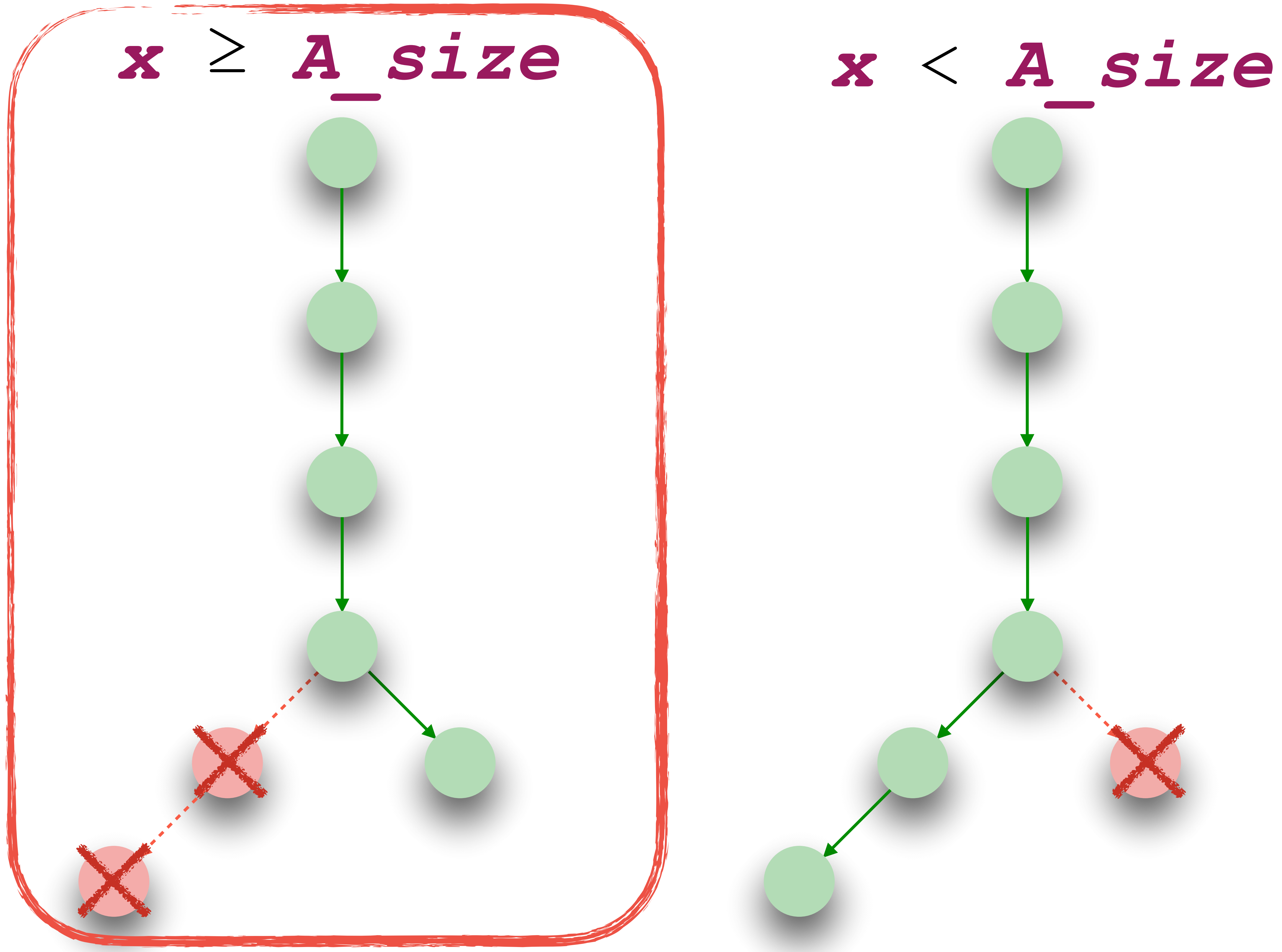


start pc *L1* load **A**+***x*** load **B**+**A**[***x***] rollback pc *END*

Symbolic execution

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

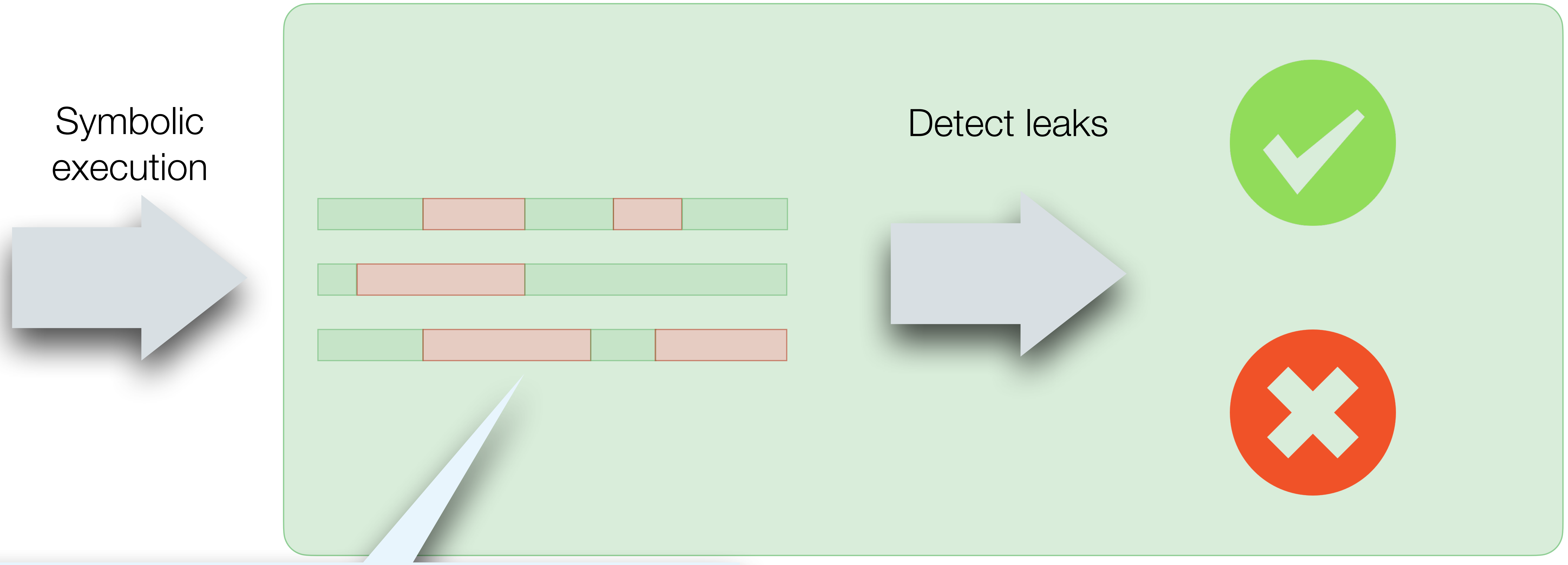
Always mispredict branch instructions



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 trace: path condition + observations along the symbolic path

Detecting speculative leaks

```
    rax
    rcx
    jmp
L1: loa
    loa
END:
```

For each symbolic trace $\tau \in traces(prg)$

  if $MemLeak(\tau)$ **then**

    return $INSECURE$

  if $CtrlLeak(\tau)$ **then**

    return $INSECURE$

return $SECURE$

Detecting speculative leaks

```
     rax
     rcx
     jmp
L1:  loa
     loa
END:
```

For each symbolic trace $\tau \in traces(prg)$

  if $\boxed{MemLeak(\tau)}$ **then**

    return $INSECURE$

  if $CtrlLeak(\tau)$ **then**

    return $INSECURE$

return $SECURE$

Memory leaks

Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations

Memory leaks

Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations

τ

Memory leaks

Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations

τ

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

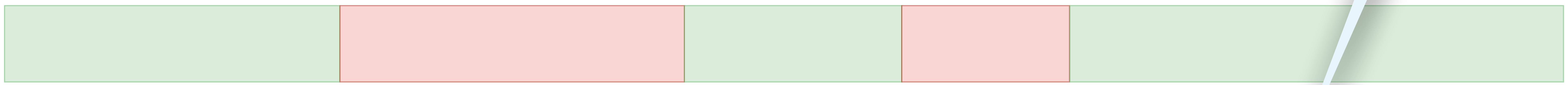
Memory leaks

Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations

Check with self-composition

τ



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

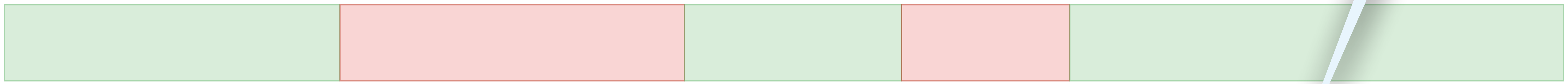
Memory leaks

Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations

Check with self-composition

τ



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

s_1

s_2

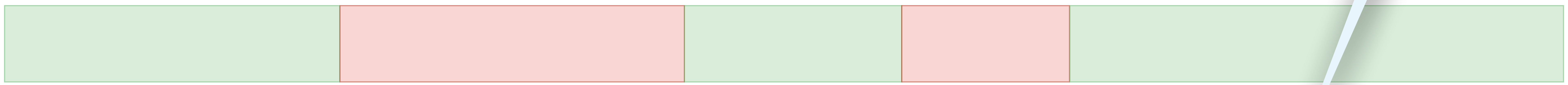
Memory leaks

Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations

Check with self-composition

τ



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

Equivalent wrt ***`policy`***

s_1

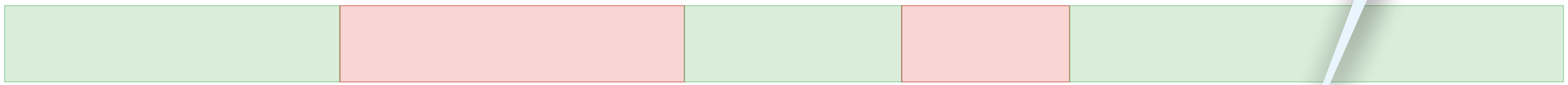
s_2

Memory leaks

Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations

Check with self-composition

τ 

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

Equivalent wrt **`policy`**

$$s_1 \models \varphi$$

$$s_2 \models \varphi$$

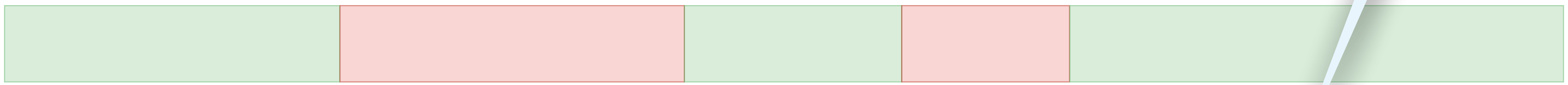
Memory leaks

Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations

Check with self-composition

τ

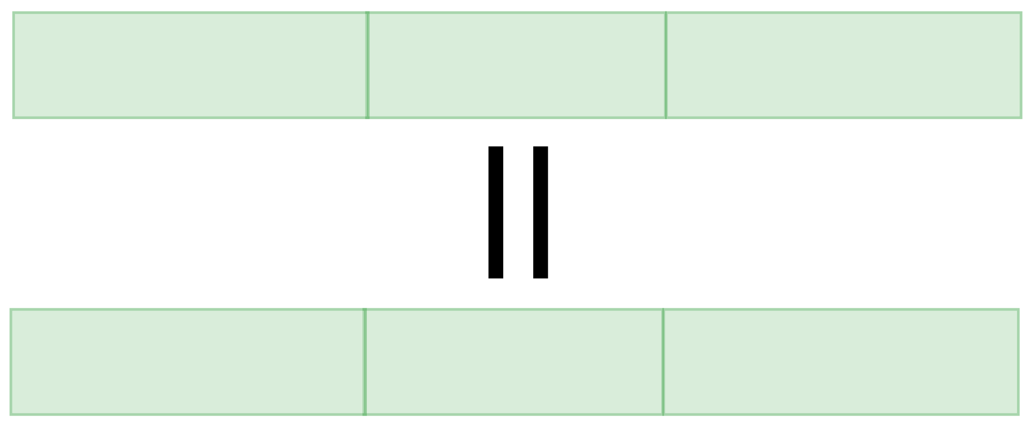


$$pathCnd(\tau) \land \boxed{obsEqv(\tau|_{non-spec})} \land \neg obsEqv(\tau|_{spec})$$

Equivalent wrt **`policy`**

$s_1 \models \varphi$

$s_2 \models \varphi$



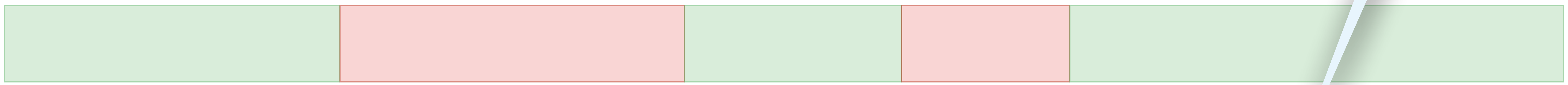
Memory leaks

Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations

Check with self-composition

τ

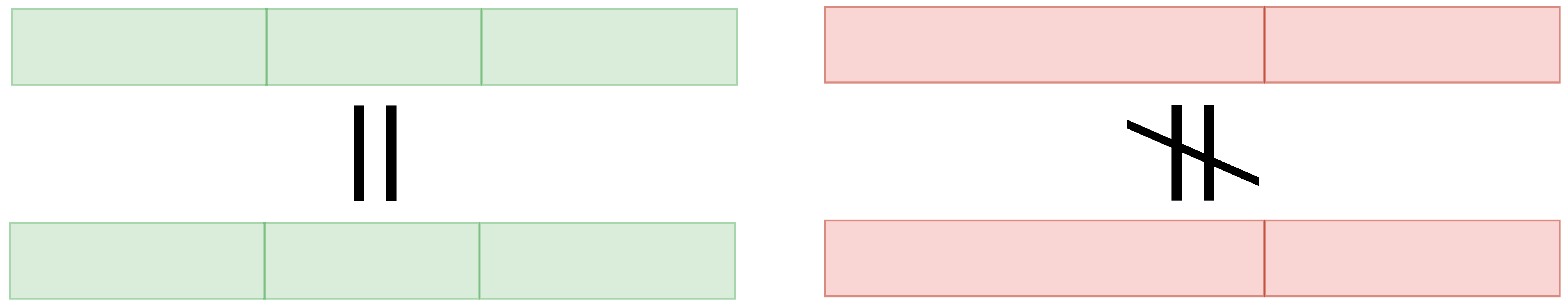


$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \boxed{\neg obsEqv(\tau|_{spec})}$$

Equivalent wrt `policy`

$s_1 \models \varphi$

$s_2 \models \varphi$



Spectector + Case studies

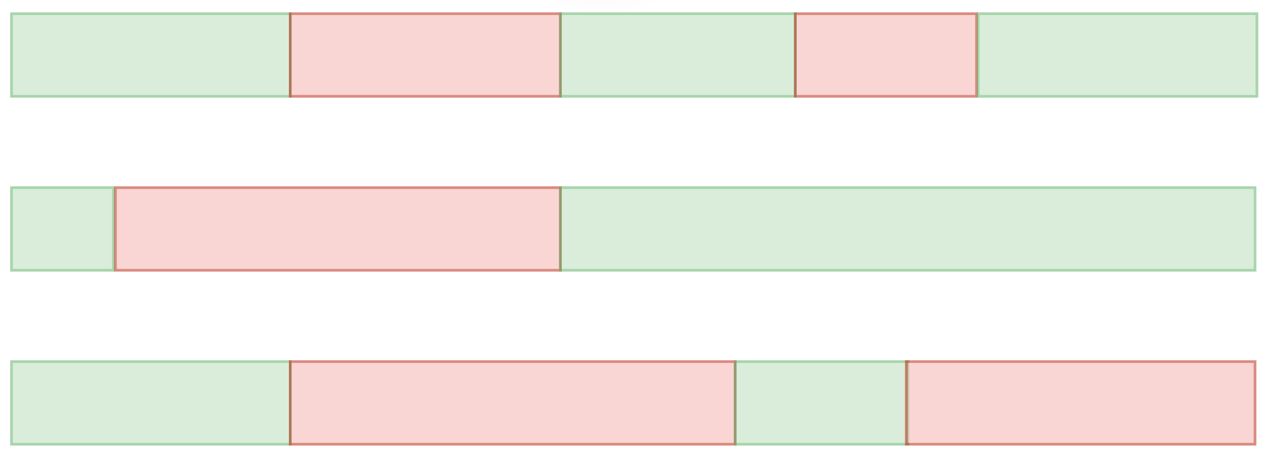
Spectector

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

x64 to µASM

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

Symbolic execution



Check for speculative leaks

Spectector

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

x64 to µASM

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

More details

- Built in Ciao Prolog
- **Z3** for symbolic execution and leak detection

Symbolic execution

Check for speculative leaks

Case study: compiler mitigations

Target:

- 15 variants of Spectre V1 by Paul Kocher*
- Compiled with Microsoft Visual C++, Intel ICC, and Clang with different mitigations and optimization levels
- 240 assembly programs of up to 200 instructions each

How:

- Use Spectector to prove security or detect leaks

* Paul Kocher - Spectre Mitigations in Microsoft C/C++ Compiler — https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html

Results

Ex.	VCC						ICC				CLANG					
	UNP		FEN 19.15		FEN 19.20		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	VCC						ICC				CLANG					
	UNP		FEN 19.15		FEN 19.20		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

No countermeasures

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Automated insertion of fences

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Speculative load hardening

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		SLH	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	VCC						ICC				CLANG					
	UNP		FEN 19.15		FEN 19.20		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

Ex.	VCC						ICC				CLANG					
	UNP		FEN 19.15		FEN 19.20		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2												-O2	-O0	-O2
01	○	○												●	●	●
02	○	○												●	●	●
03	○	○												●	●	●
04	○	○												●	●	●
05	○	○												●	●	●
06	○	○												●	●	●
07	○	○												●	●	●
08	○	●												●	●	●
09	○	○												●	●	●
10	○	○												●	●	○
11	○	○												●	●	●
12	○	○												●	●	●
13	○	○												●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

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
  - Unnecessary fences
- Programs with SLH are secure except #10 and #15

Case study: scalability

Target: Xen hypervisors

Main challenges for scalability:

- Policy definition
- ISA coverage
- Path explosion

How:

- Analyze scalability of checking SNI ***relative to*** symbolic execution
  - *24'000* symbolic paths of < *10'000* instructions (from ~ *4'000* functions)

Case study: scalability

Target: Xen hypervisors

Main challenges for scalability:

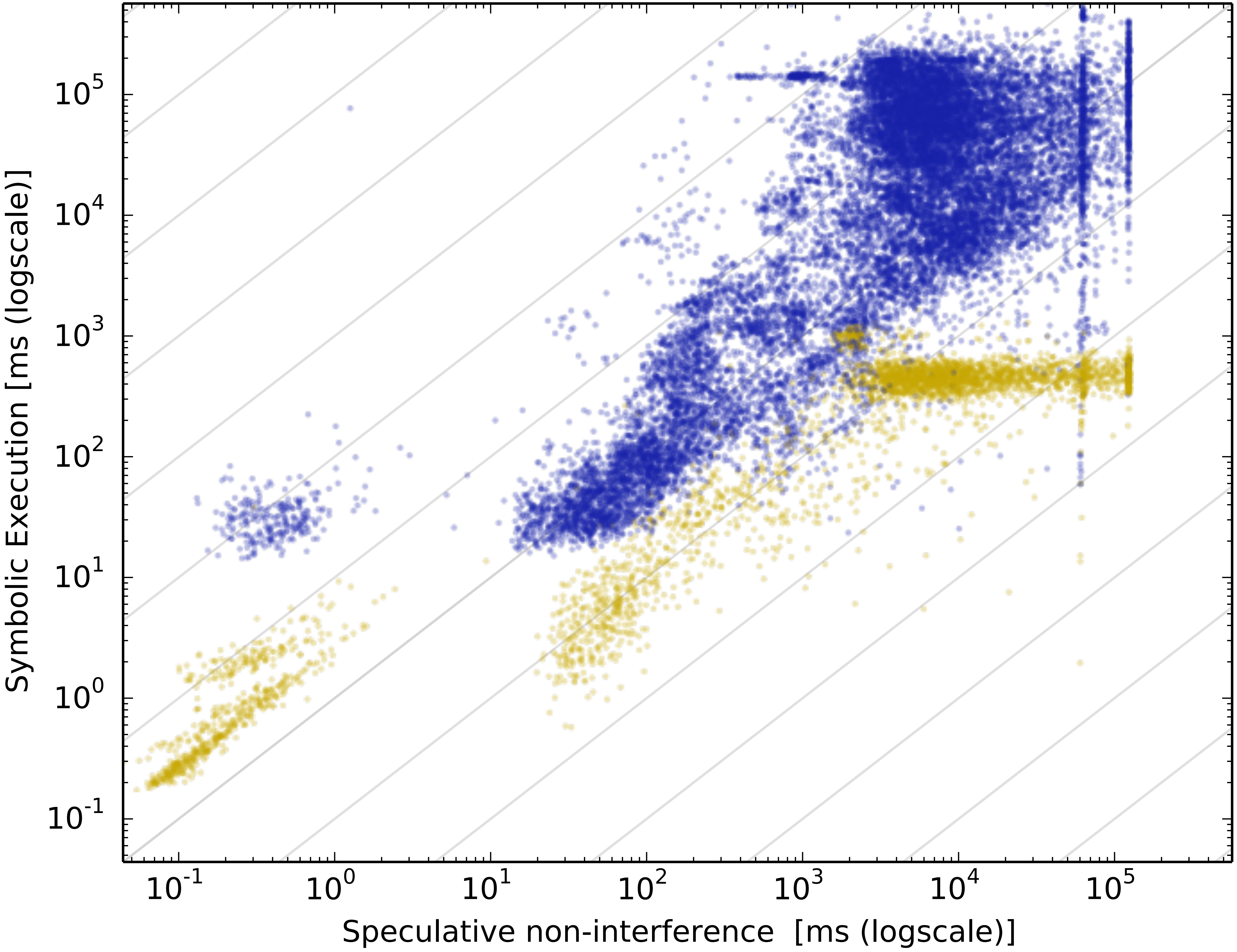
- Policy definition
- ISA coverage
- Path explosion

Trade-offs affect analysis soundness and completeness

How:

- Analyze scalability of checking SNI ***relative to*** symbolic execution
  - *24’000* symbolic paths of < *10’000* instructions (from ~ *4’000* functions)

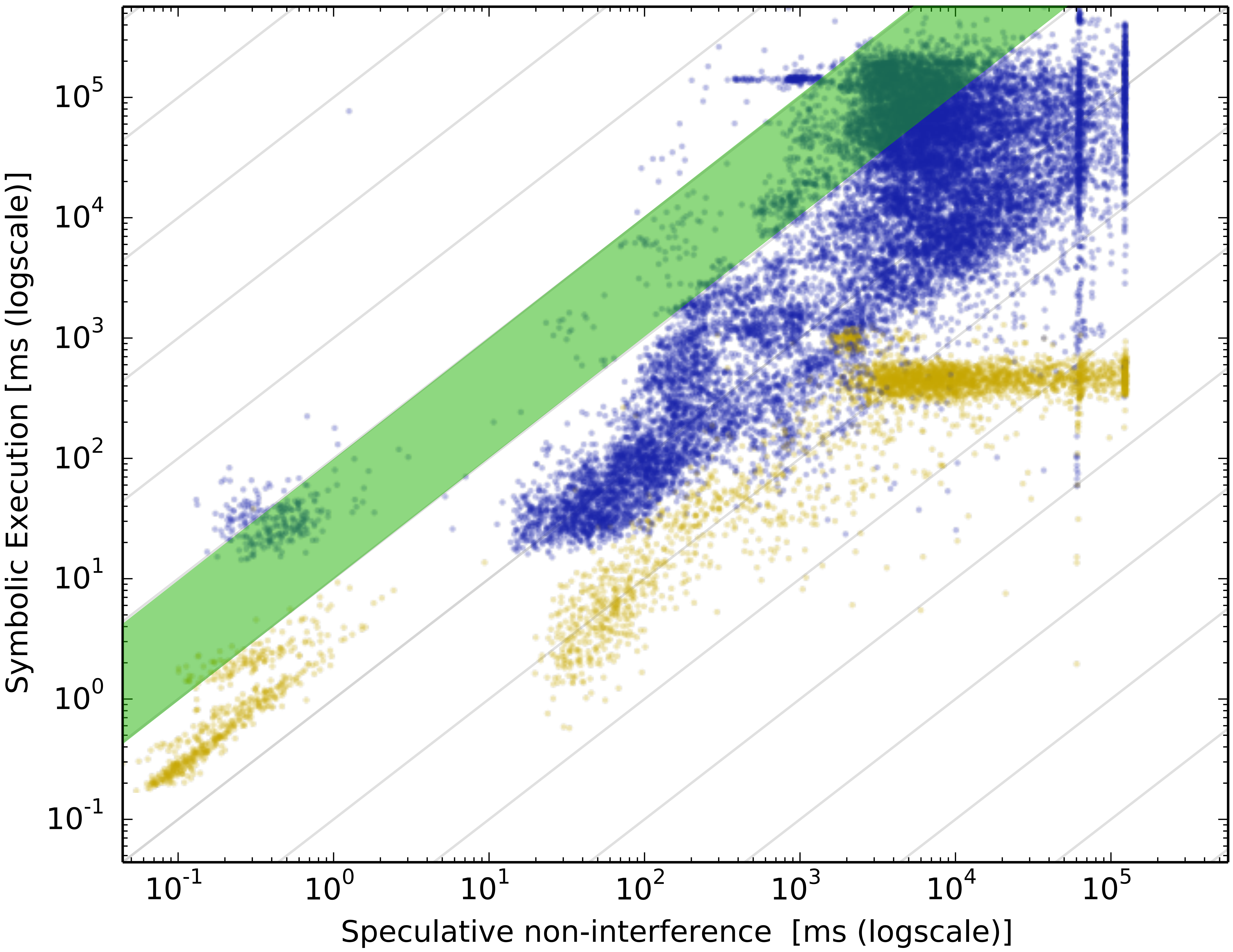
Results



Symbolic Execution [ms (logscale)]
Speculative non-interference [ms (logscale)]
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}

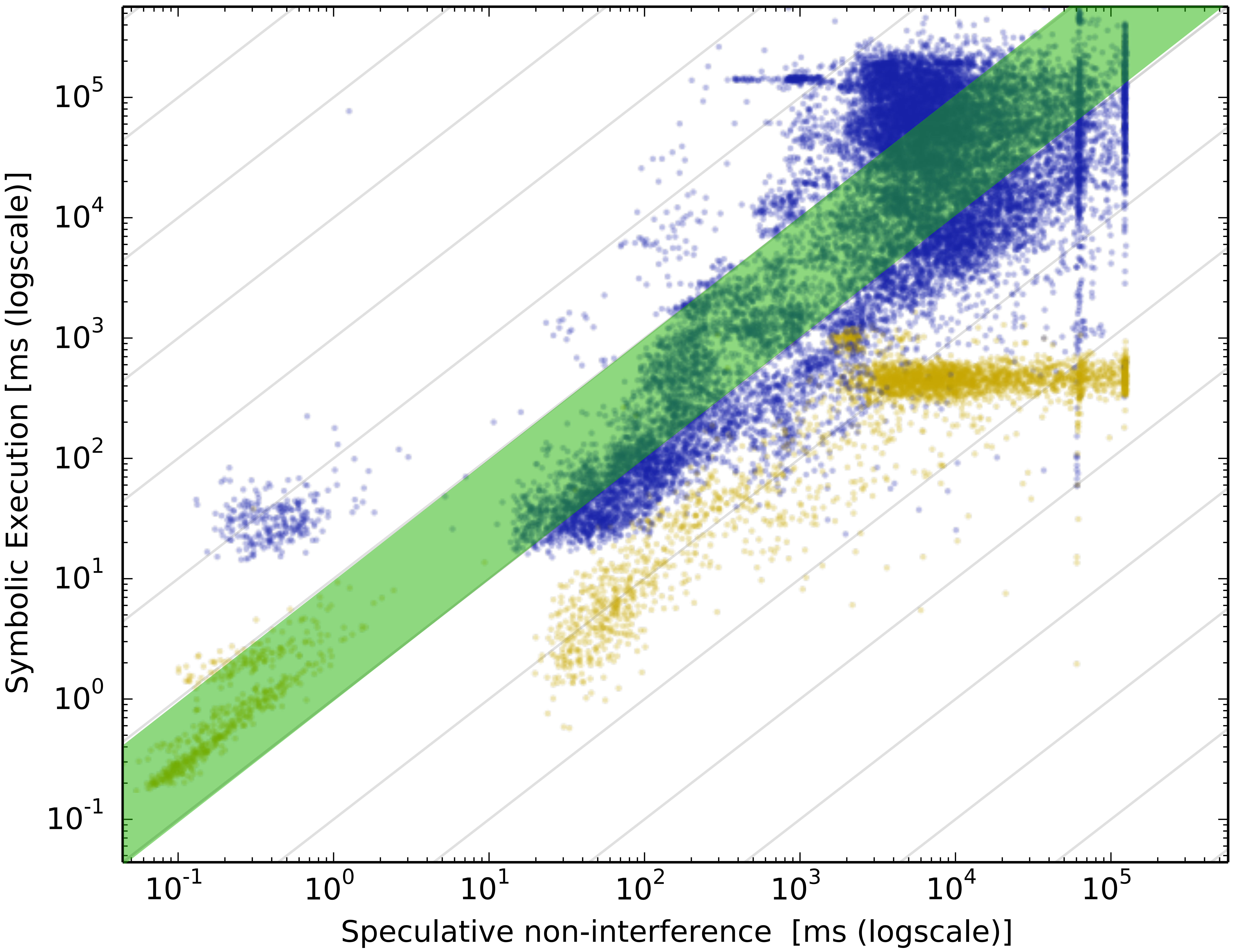
Results

- SNI 10x-100x faster
  - 20.2% traces



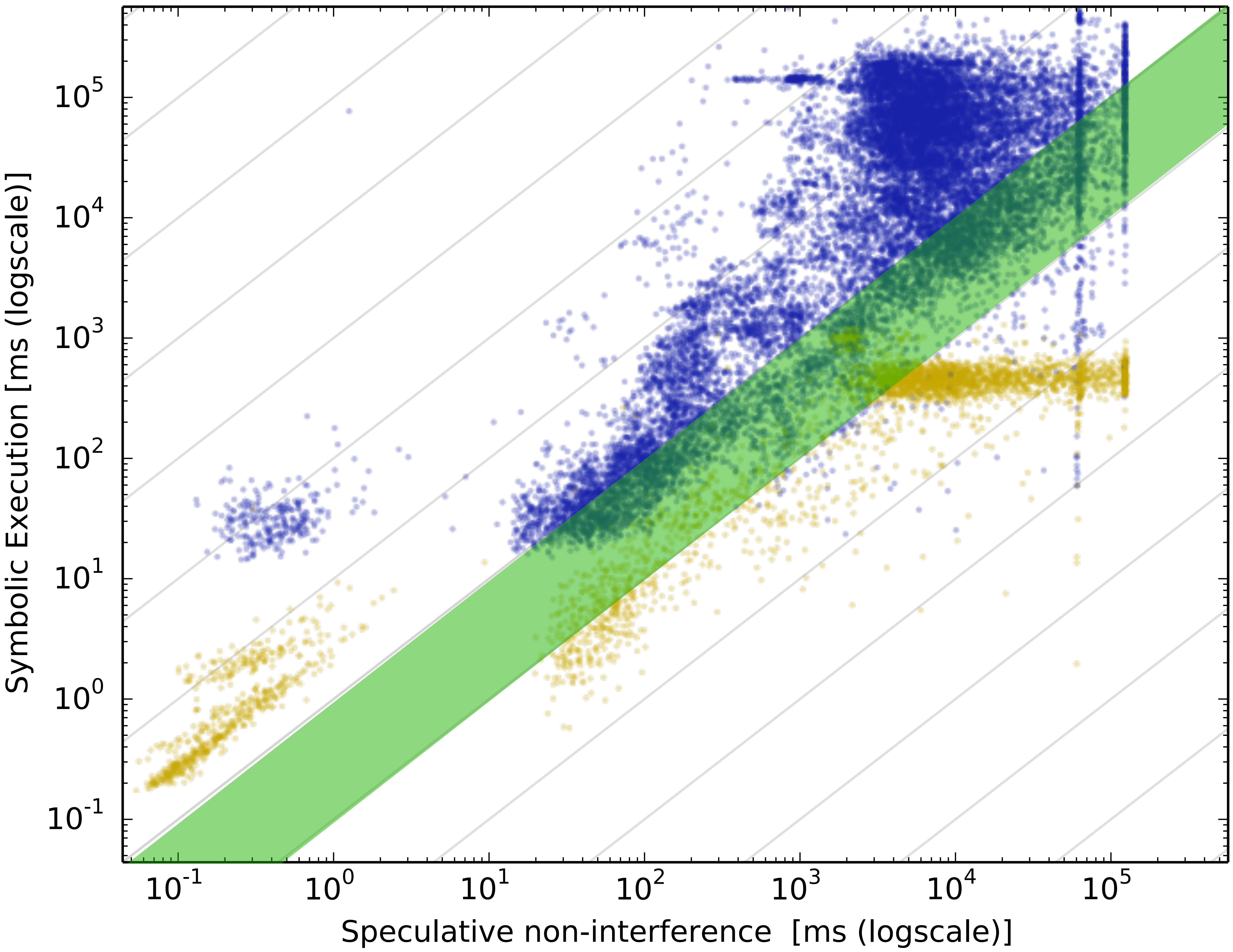
Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces



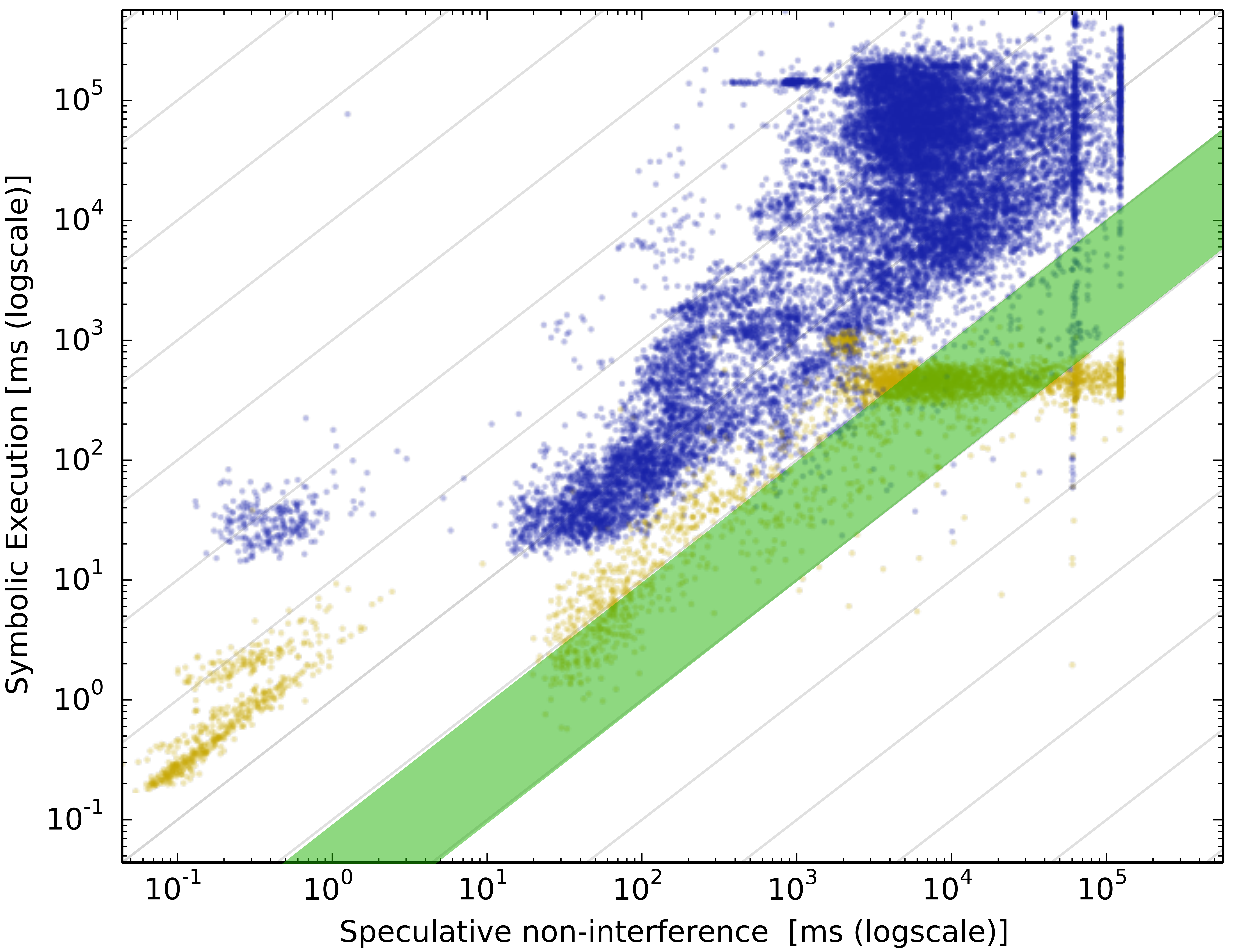
Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces
- SNI ≤10x slower
  - 26.9% traces



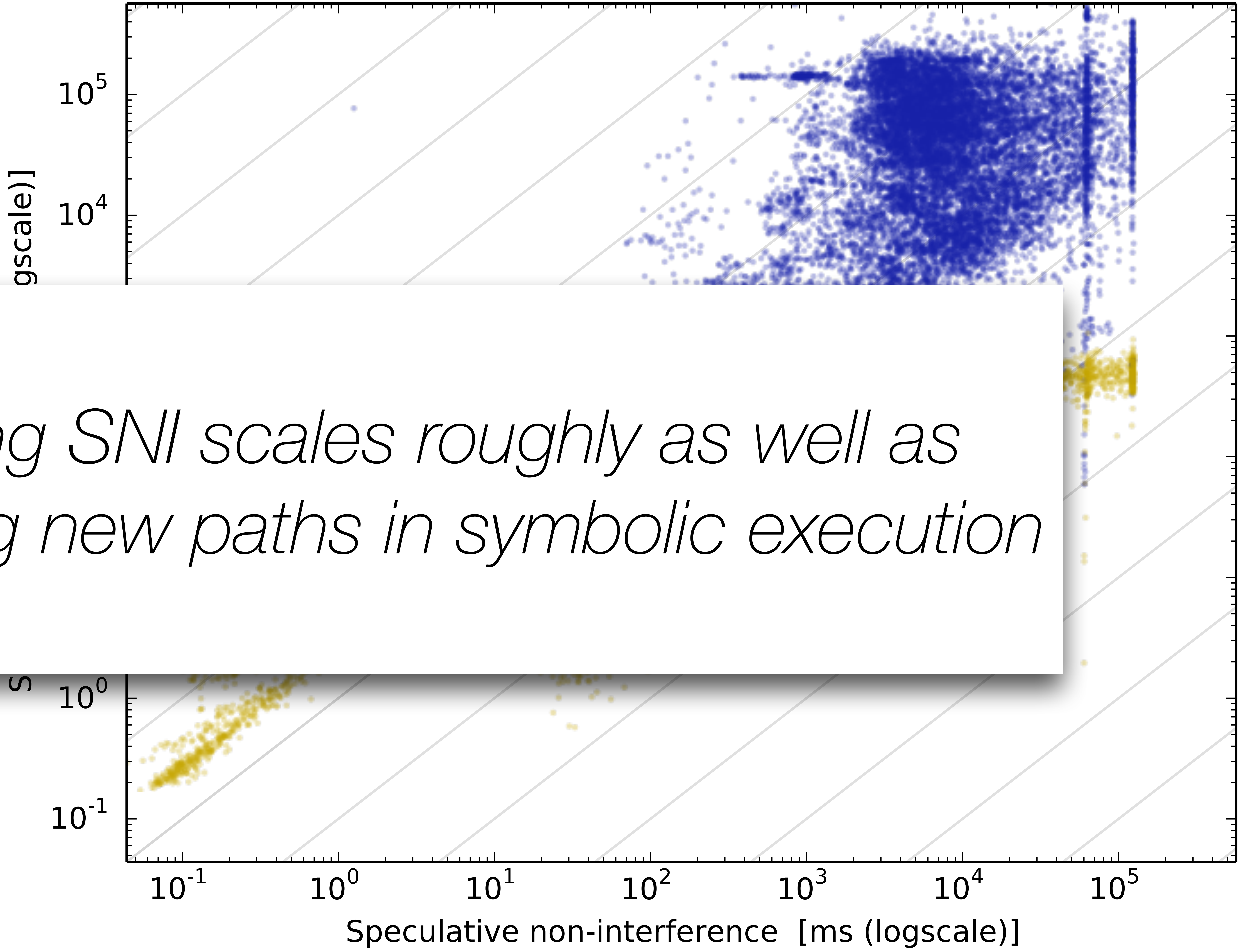
Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces
- SNI ≤10x slower
  - 26.9% traces
- SNI 10x-100x slower
  - 7.9% traces



Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI
  - 4
- SNI
  - 26
- SNI 10x-100x slower
  - 7.9% traces



Checking SNI scales roughly as well as discovering new paths in symbolic execution

Conclusion

Speculative non-interference

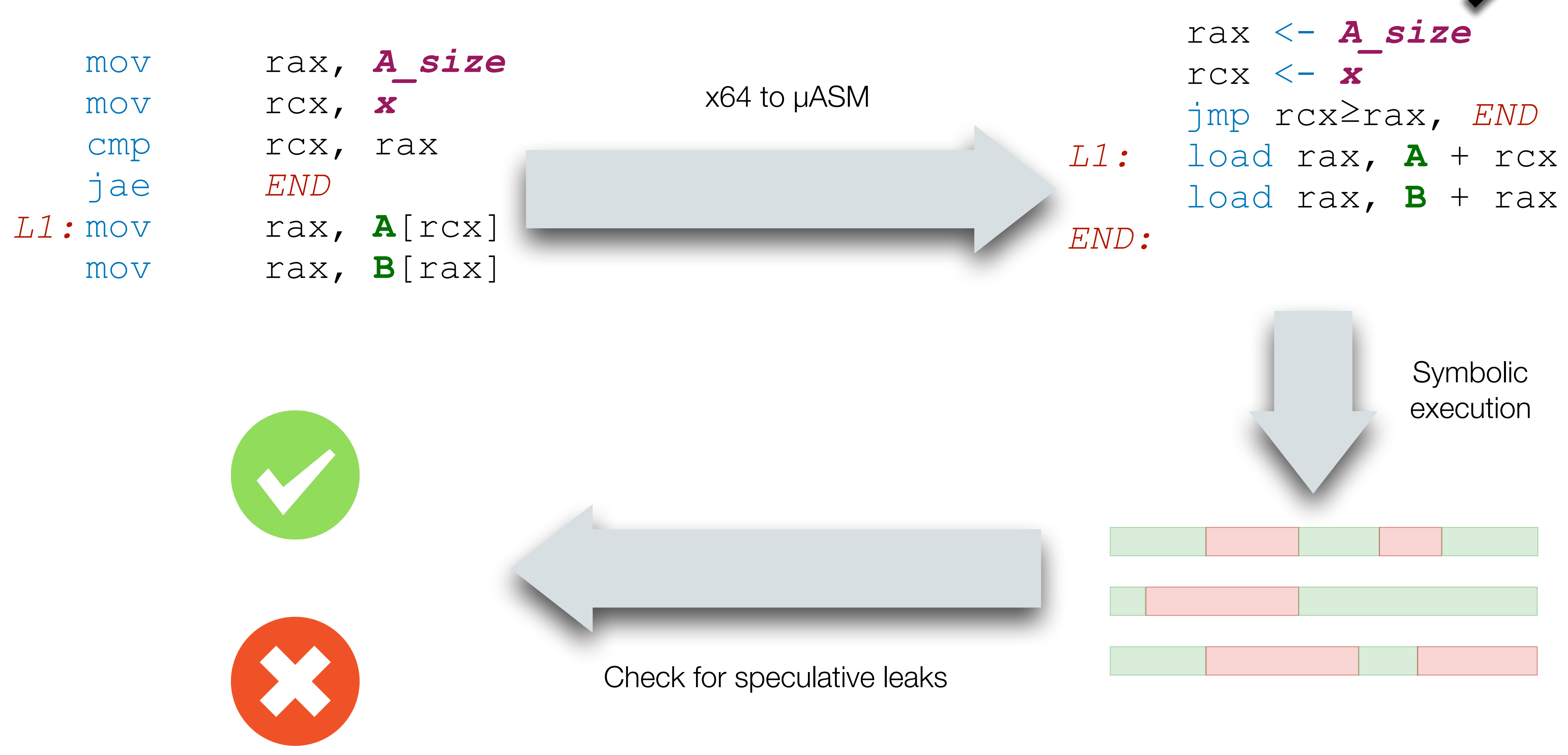
Formally!

Program **P** is **speculatively non-interferent** for prediction oracle ***O*** if

For all program states s and s':

$$P_{\texttt{non-spec}}(s) = P_{\texttt{non-spec}}(s') \Rightarrow P_{\texttt{spec}}(s,O) = P_{\texttt{spec}}(s',O)$$

Spectector

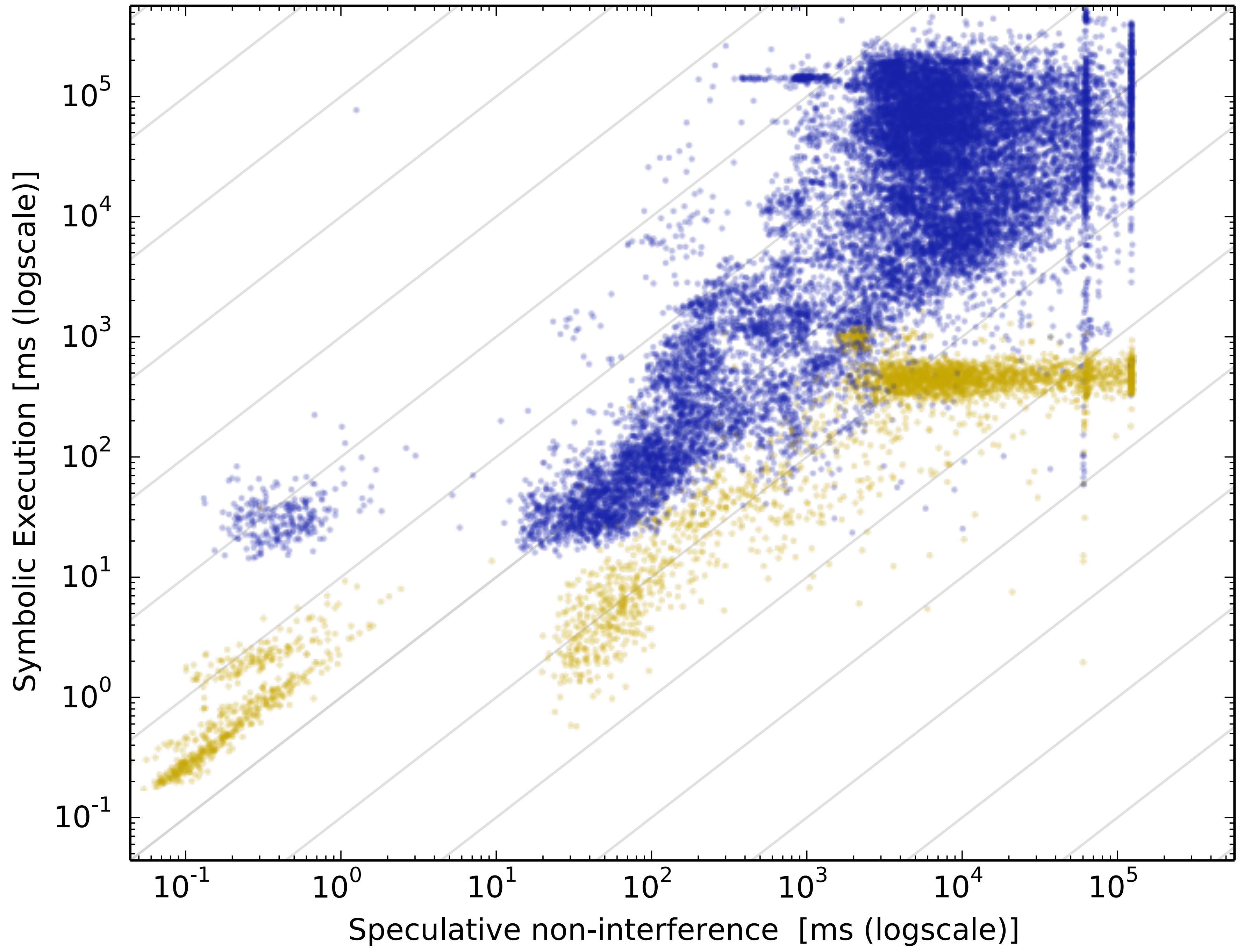


Results

Ex.	VCC UNP -O0	VCC UNP -O2	VCC FEN 19.15 -O0	VCC FEN 19.15 -O2	VCC FEN 19.20 -O0	VCC FEN 19.20 -O2	ICC UNP -O0	ICC UNP -O2	ICC FEN -O0	ICC FEN -O2	CLANG UNP -O0	CLANG UNP -O2	CLANG FEN -O0	CLANG FEN -O2	CLANG SLH -O0	CLANG SLH -O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces
- SNI ≤10x slower
  - 26.9% traces
- SNI 10x-100x slower
  - 7.9% traces



Speculative non-interference

Formally!

Program **P** is **speculatively non-interferent** for prediction oracle ***O*** if

For all

P_{non}

Spectector

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

x64 to µASM

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

Symbolic execution

Spectector

https://spectector.github.io

marco.guarnieri@imdea.org

@MarcoGuarnier1

Results

Ex.	Vcc															
	Unp		Fen 19.15													
	-O0	-O2	-O0	-O2												
01	○	○	●	●												
02	○	○	●	●												
03	○	○	●	○												
04	○	○	○	○												
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

- SNI ≤10x slower
  - 26.9% traces
- SNI 10x-100x slower
  - 7.9% traces