Spectector: Principled detection of speculative information flows

Jan Reineke @ UNIVERSITÄT DES SAARLANDES

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

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

SPECTRE

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

SPECTRE

Exploits **speculative execution** to leak sensitive information

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

SPECTRE

Exploits **speculative execution** to leak sensitive information

Almost all modern processors 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

Countermeasures

Countermeasures

Long Term: Co-Design of Software and Hardware countermeasures

Countermeasures

Long Term: Co-Design of Software and Hardware countermeasures

Short and Mid Term: Software countermeasures

In particular: Compiler-level countermeasures

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

Countermeasures

Long Term: Co-Design of Software and Hardware countermeasures

Short and Mid Term: Software countermeasures

In particular: Compiler-level countermeasures

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

PROBLEM SOLVED ?

Compiler-level countermeasures

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

Paul Kocher
February 13, 2018

Compiler-level countermeasures

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

Paul Kocher
February 13, 2018

"The countermeasure […] is conceptually straightforward
but **challenging in practice**"

Compiler-level countermeasures

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

Paul Kocher
February 13, 2018

"The countermeasure […] is conceptually straightforward but **challenging in practice**"

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

Compiler-level countermeasures

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

Paul Kocher
February 13, 2018

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

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

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

Compiler-level countermeasures

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

Paul Kocher
February 13, 2018

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

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

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

Bottom line: No guarantees!

Goals

Goals

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

Goals

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

2. Static analysis to **detect vulnerability** or to **prove security**

Outline

1. Speculative execution attacks

2. Speculative non-interference

3. Spectector: Detecting speculative leaks

4. Challenges

1. Speculative execution attacks

Background: Speculative execution

Background: Speculative execution

- Predict instructions' outcomes and speculatively continue execution

Background: Speculative execution

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

Background: Speculative execution

- Predict instructions' outcomes and speculatively continue execution

- Rollback changes if speculation was wrong

Only architectural (ISA, "logical") state,
not microarchitectural state

Background: Branch prediction

Size of array **A**

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

Background: Branch prediction

Size of array **A**

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

Background: Branch prediction

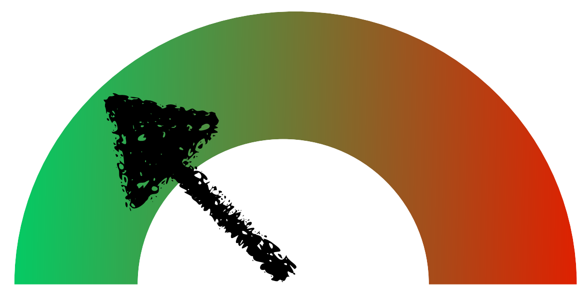
Size of array **A**

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

Background: Branch prediction

Size of array **A**

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

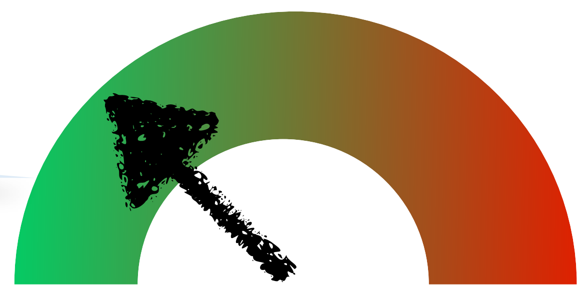


Background: Branch prediction

Size of array **A**

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

Predictions based on
branch history &
program structure

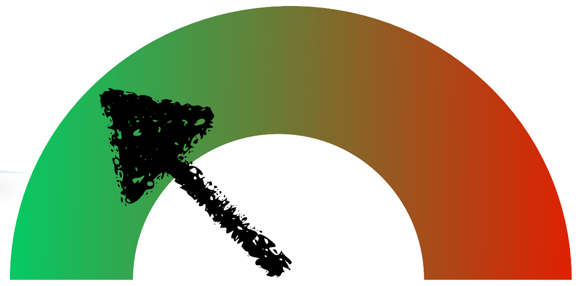


Background: Branch prediction

Size of array **A**

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

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

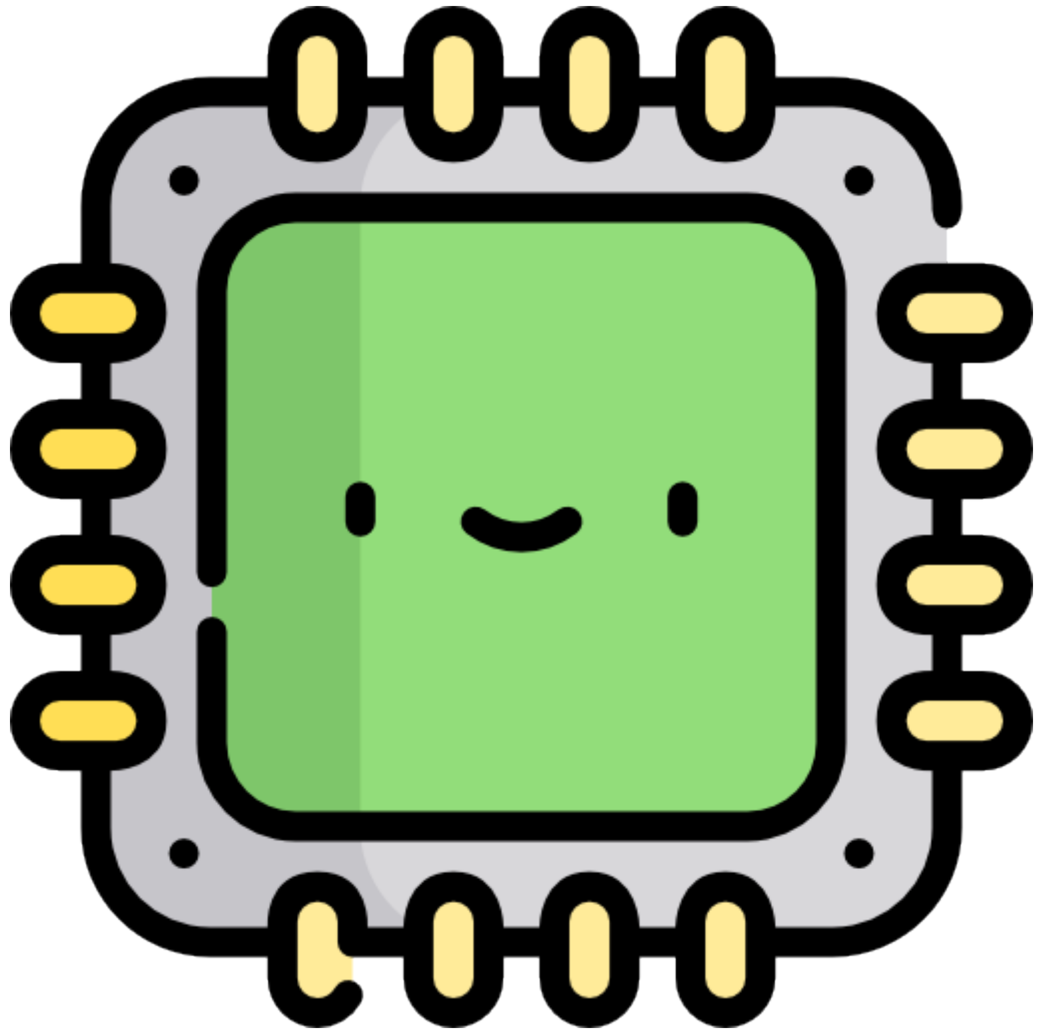


Spectre V1

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

Spectre V1

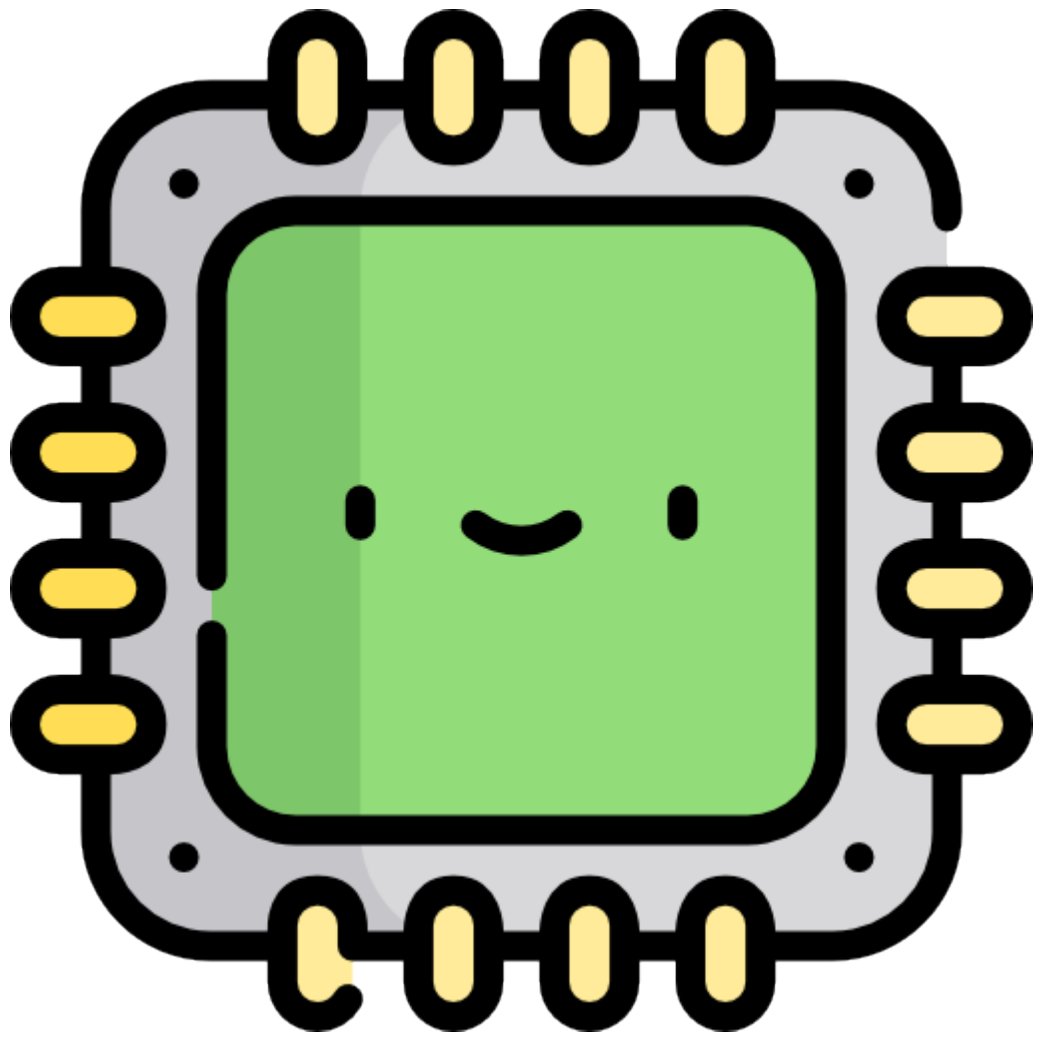
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



Cache state

Spectre V1

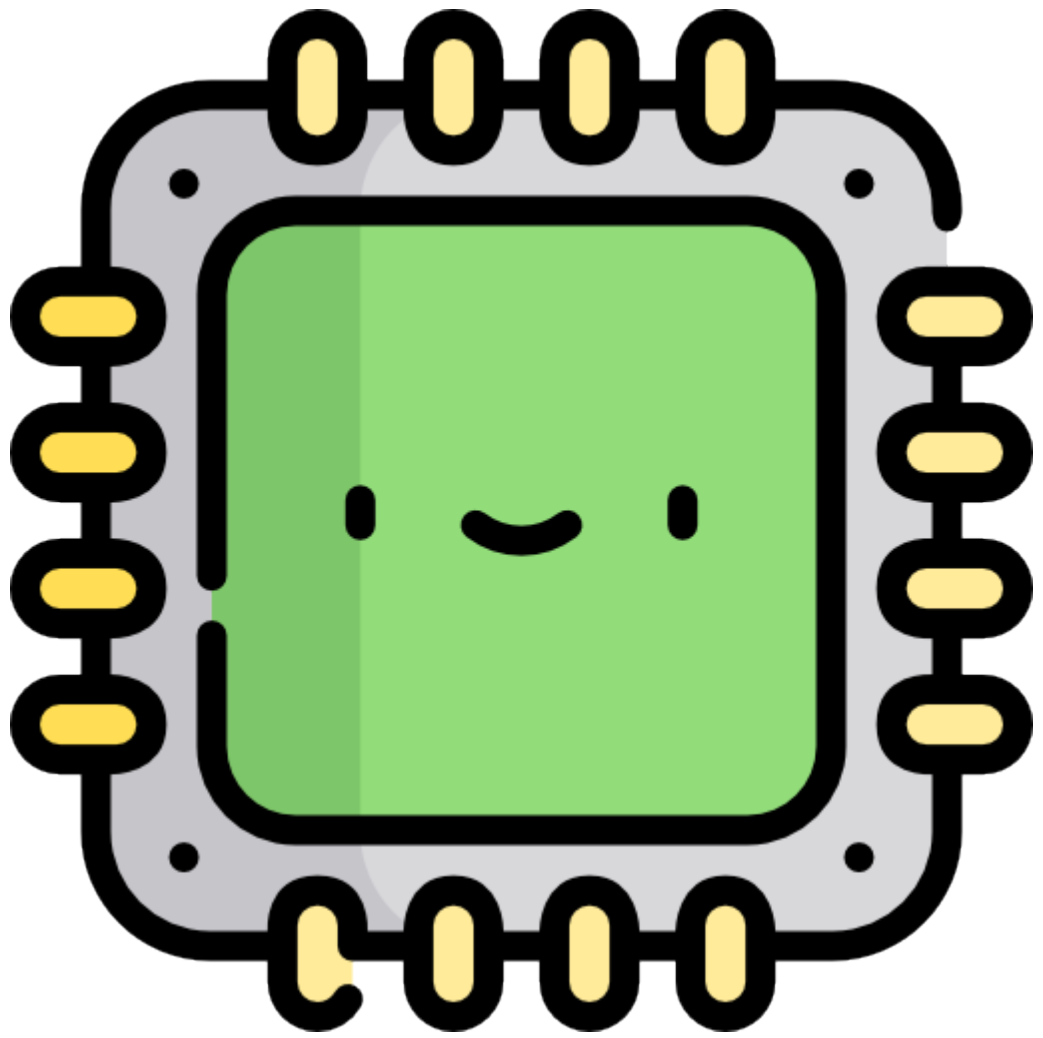
```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```



Cache state

Spectre V1


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



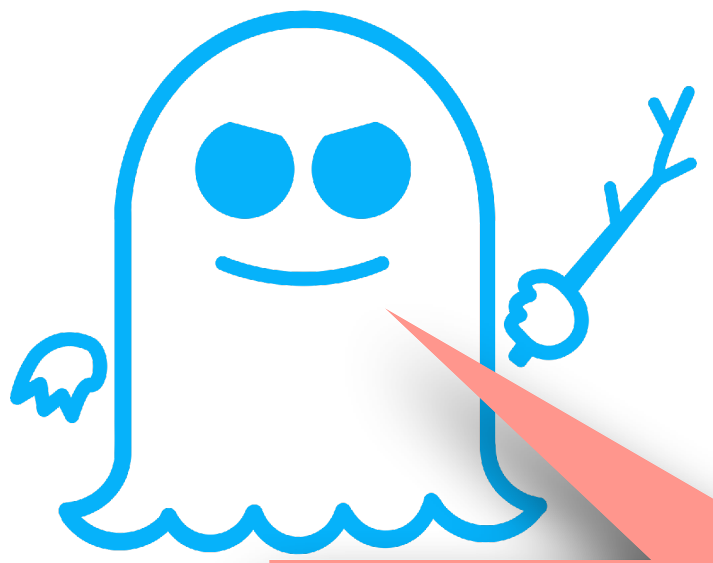
Cache state

Spectre V1

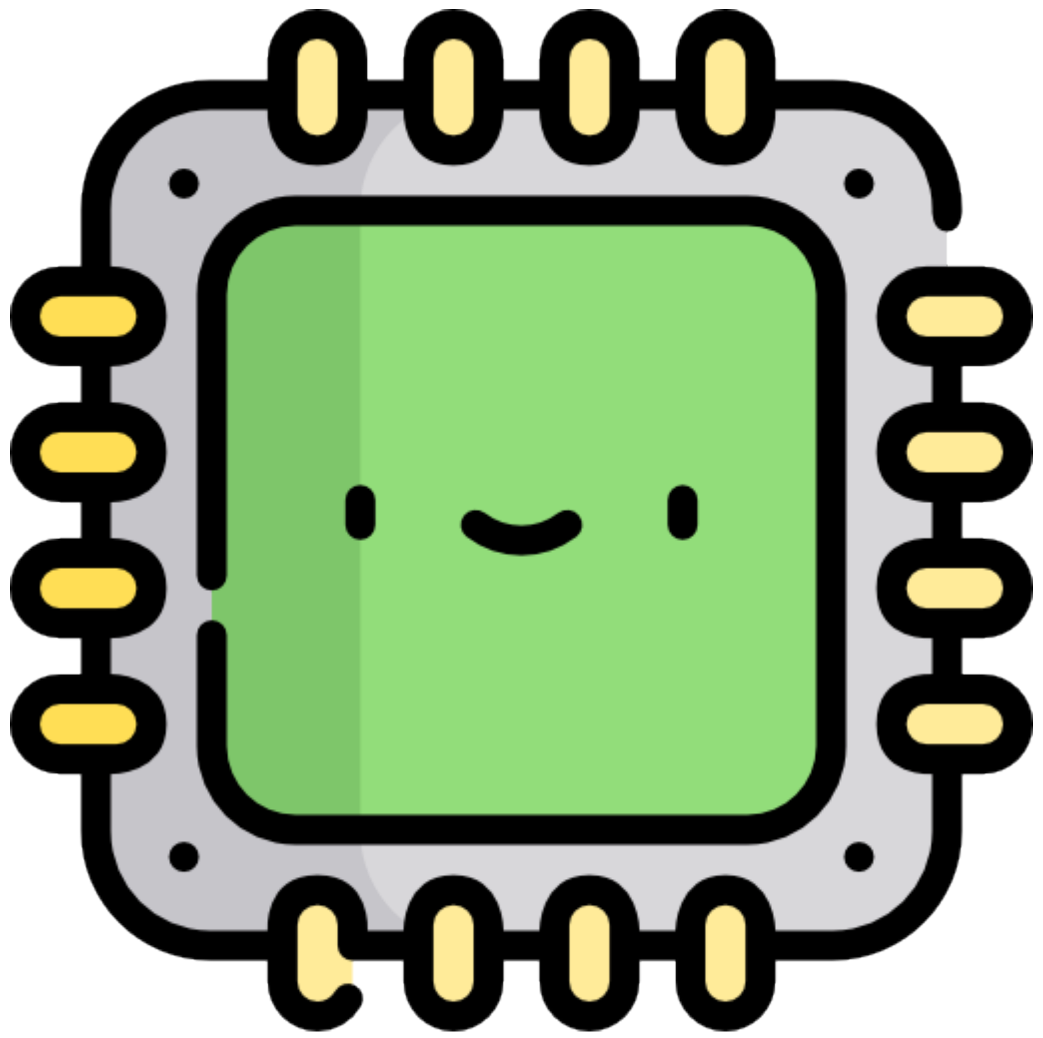
A_size=16

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

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



What is in A[128]?



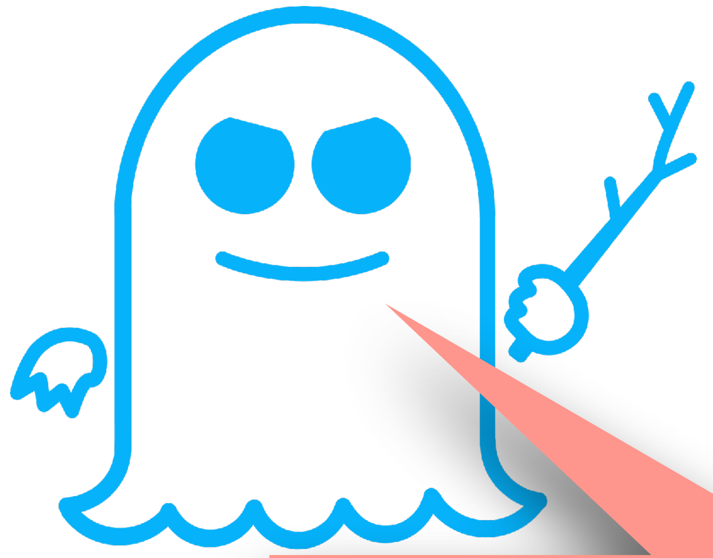
Cache state

Spectre V1

A_size=16

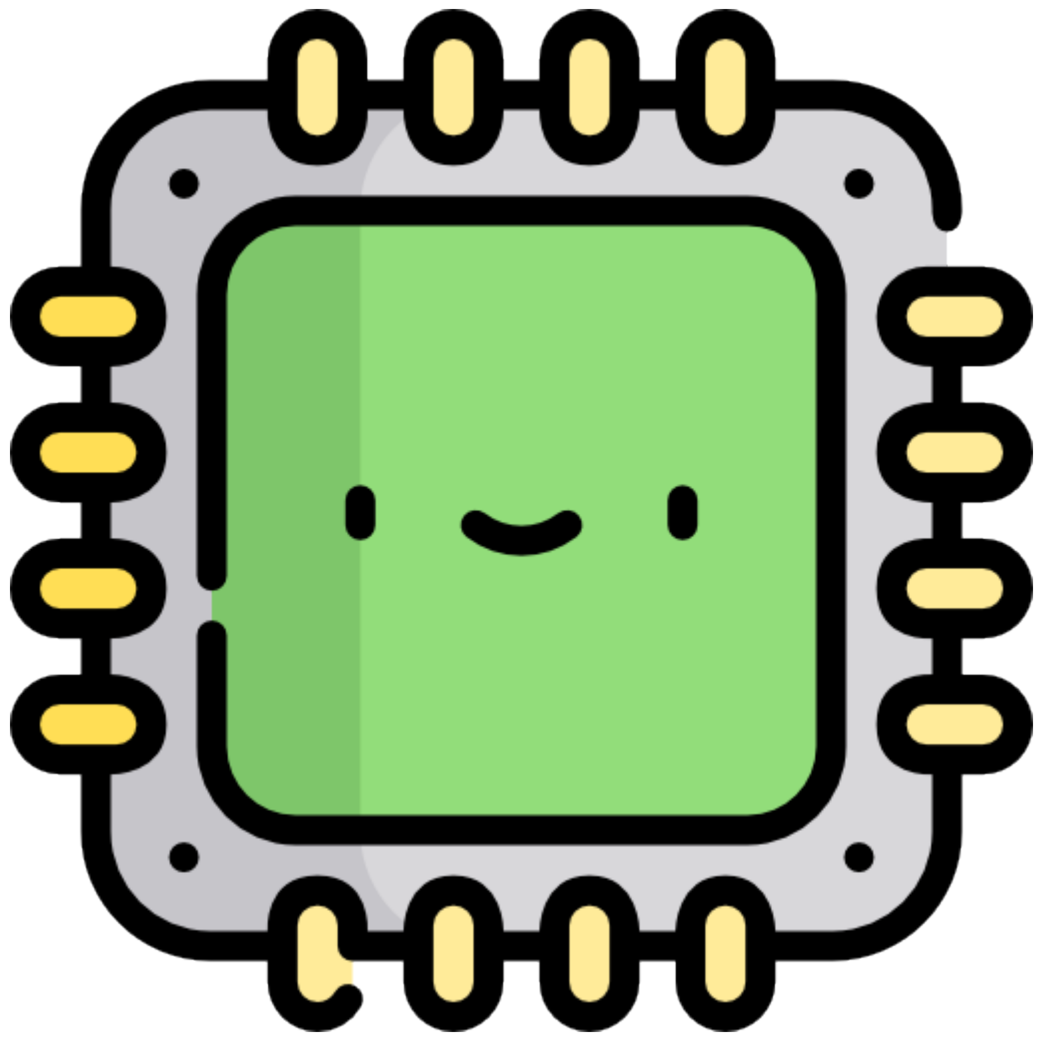
B B[0] B[1] ...

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



What is in A[128]?

1a) Training



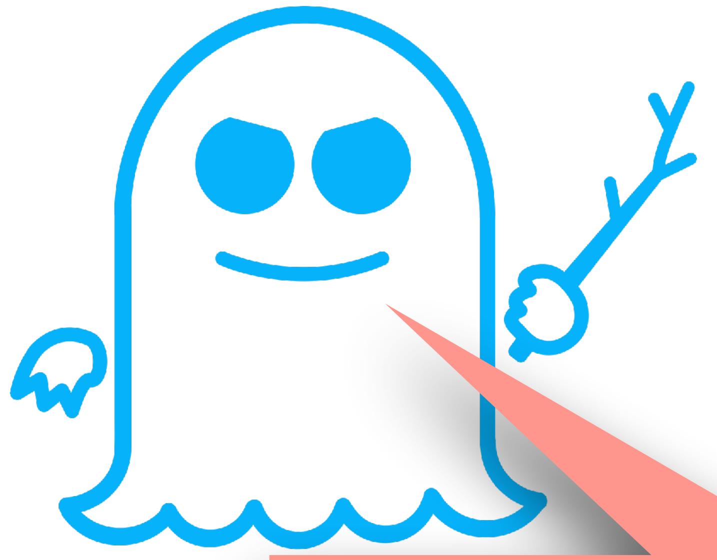
Cache state

Spectre V1

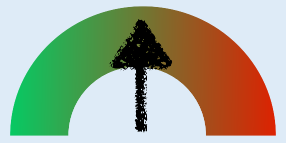
A_size=16

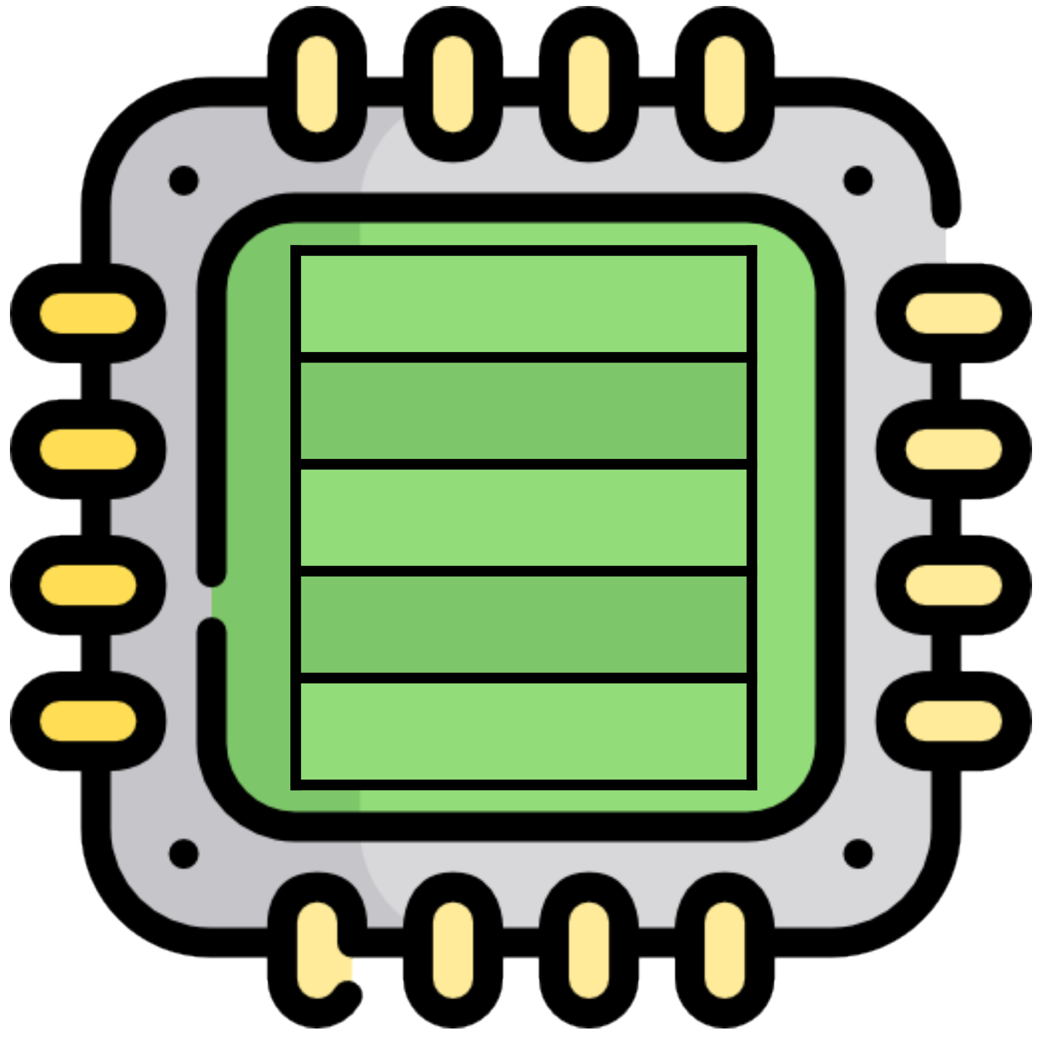
B B[0] B[1] ...

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



What is in A[128]?

1a) Training 



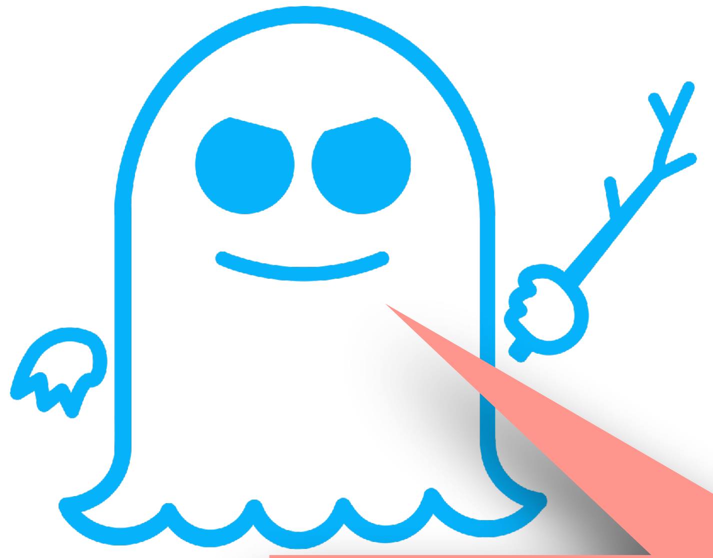
Cache state

Spectre V1

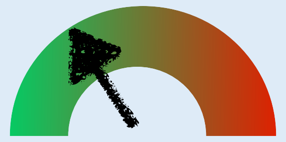
A_size=16

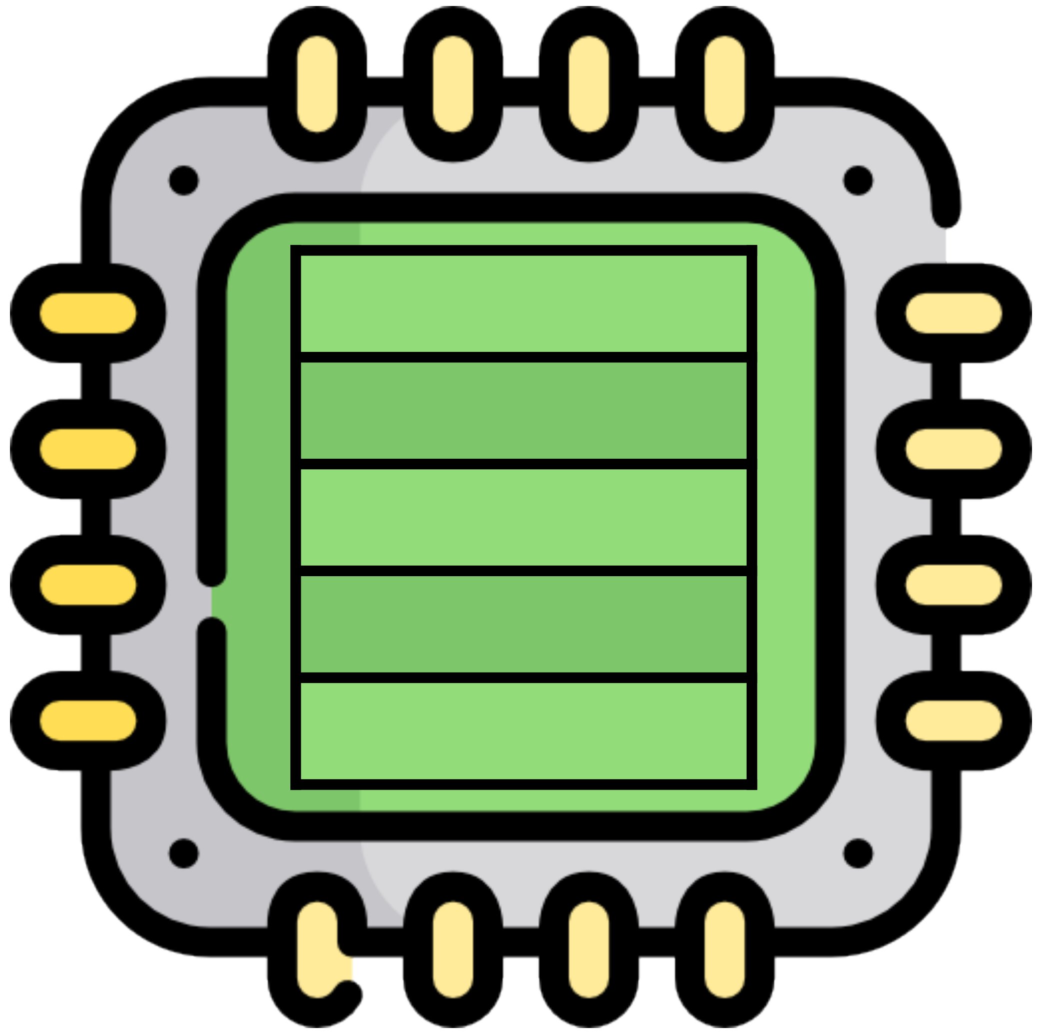
B: B[0] | B[1] | ...

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



What is in A[128]?

1a) Training  f(0);



Cache state

Spectre V1

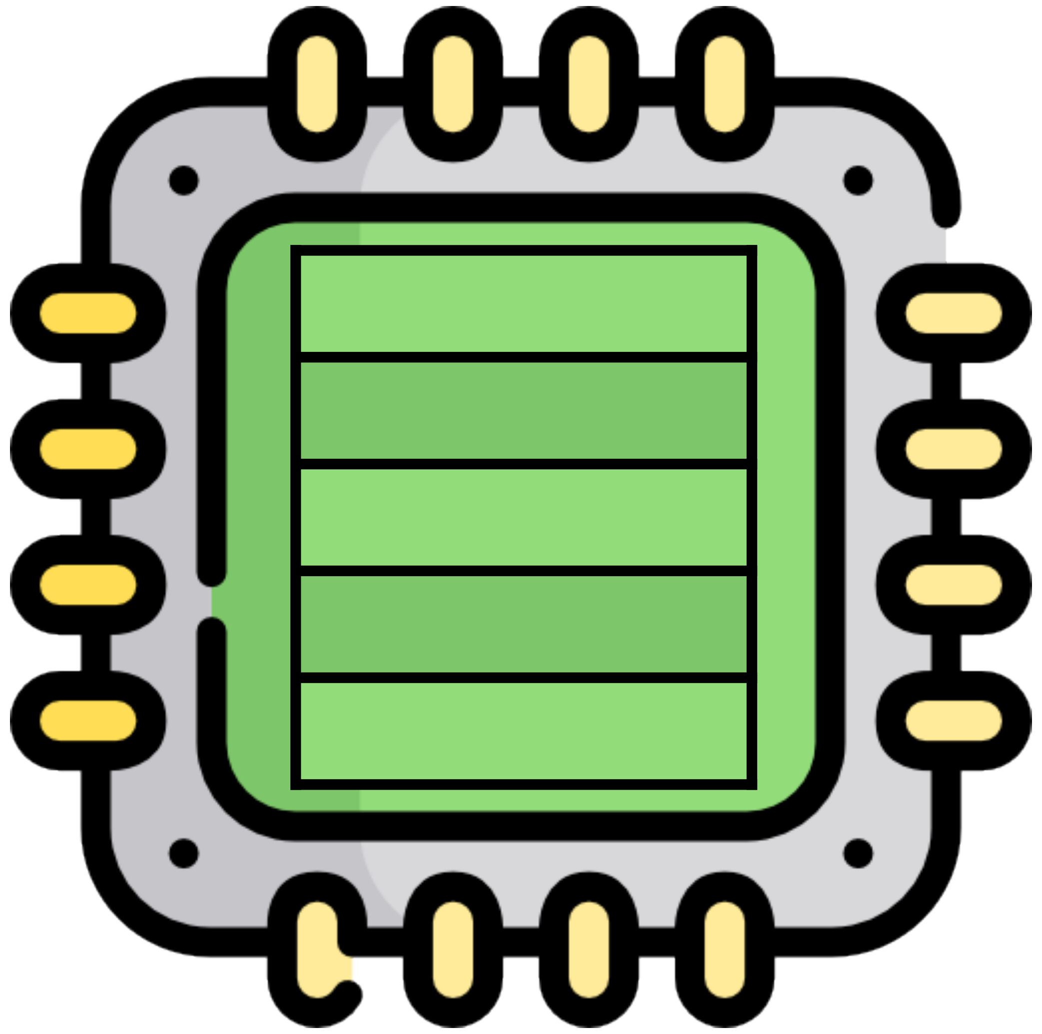
A_size=16

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

What is in A[128]?

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

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



Cache state

Spectre V1

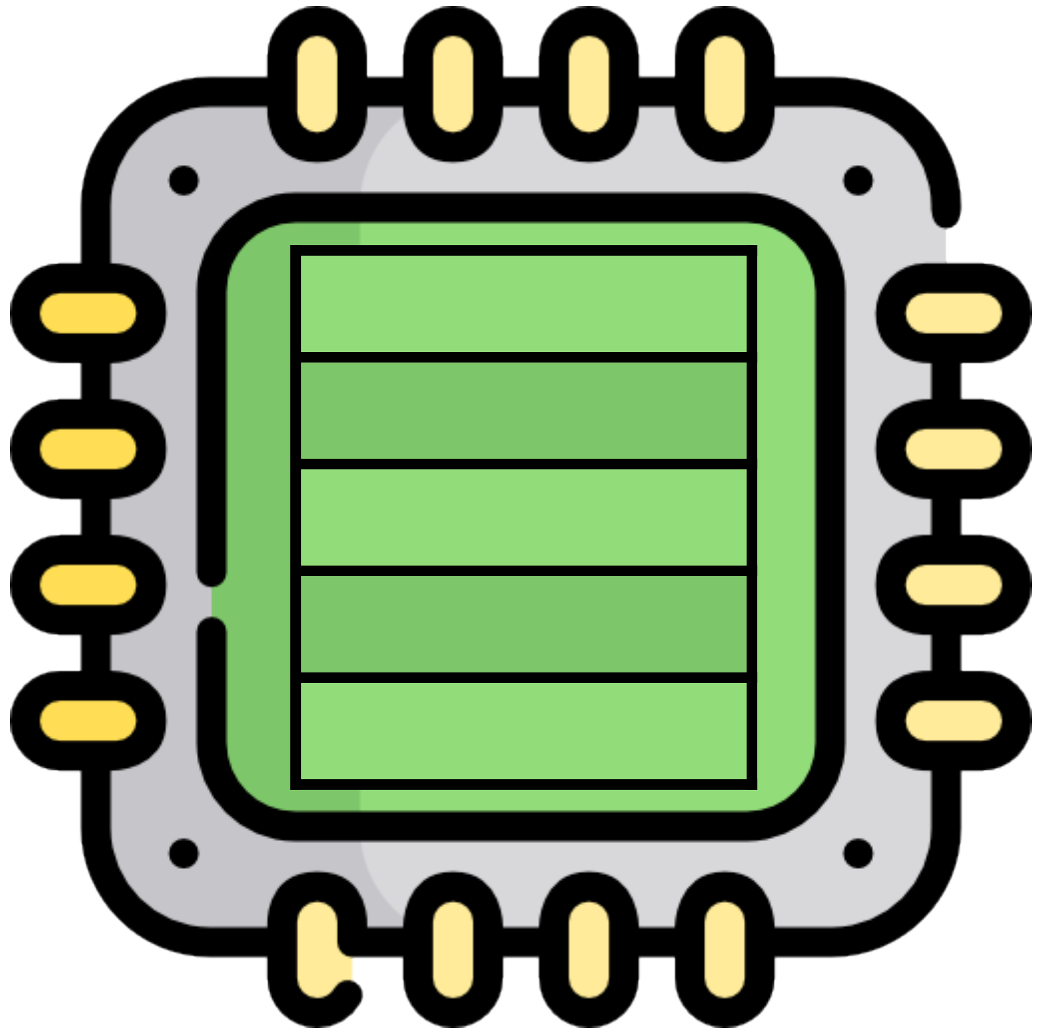
A_size=16

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

What is in A[128]?

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

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



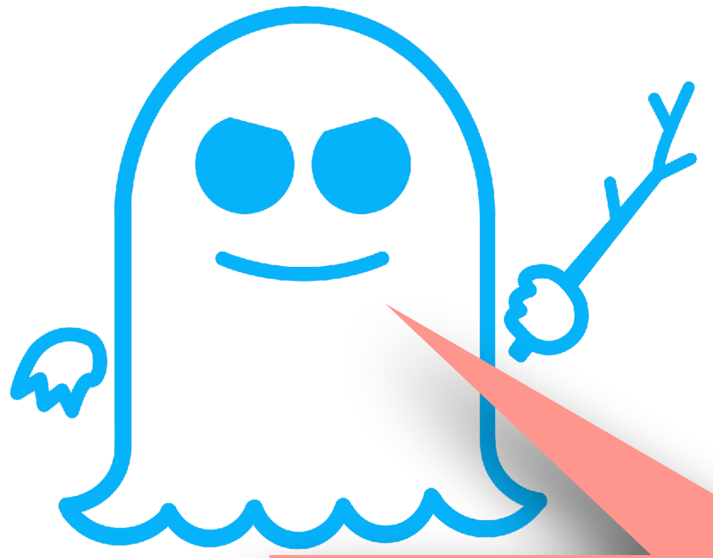
Cache state

Spectre V1

A_size=16

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

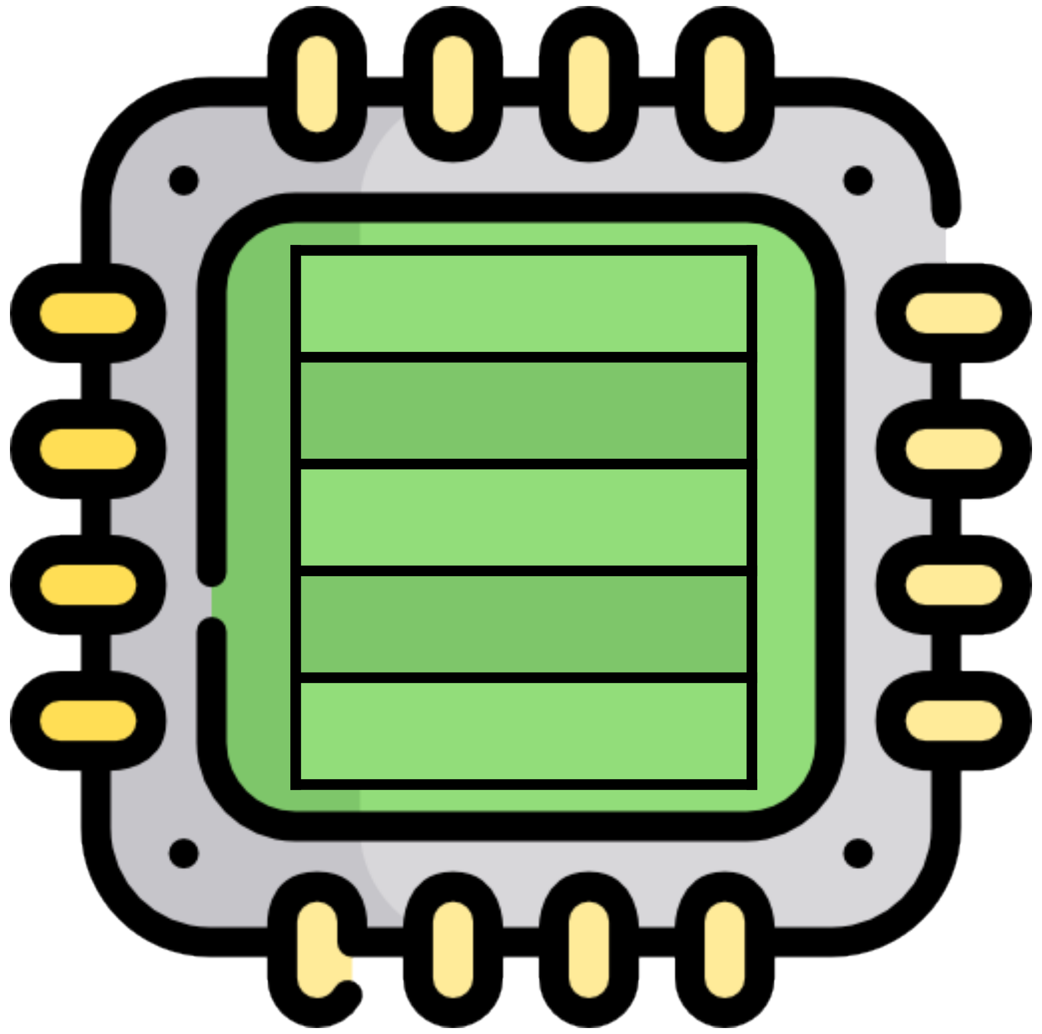
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



What is in A[128]?

1a) Training f(0);f(1);f(2); …

1b) Prepare cache



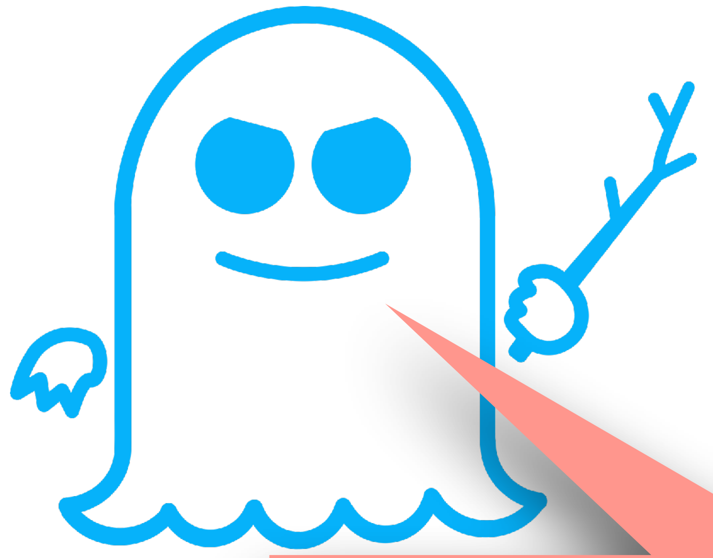
Cache state

Spectre V1

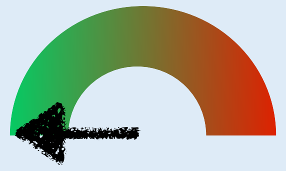
A_size=16

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

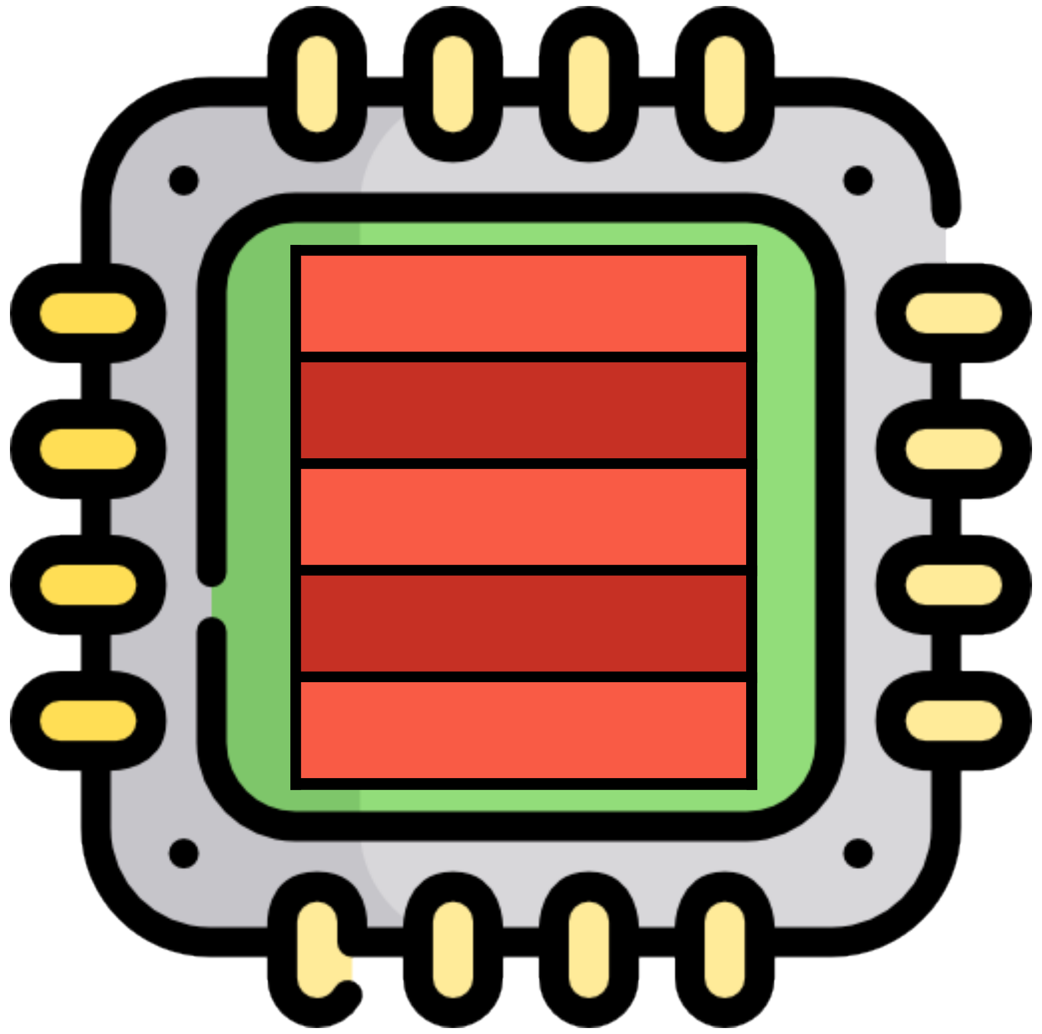
```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```



What is in A[128]?

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

1b) Prepare cache



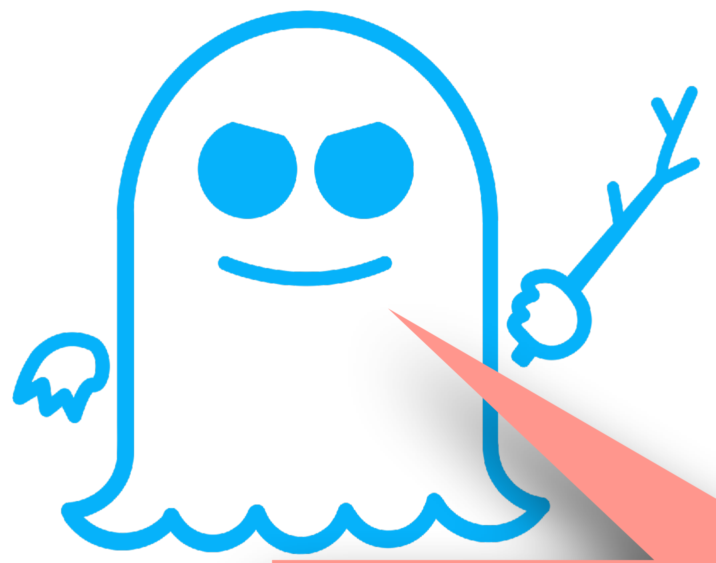
Cache state

Spectre V1

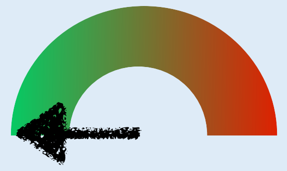
A_size=16

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

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

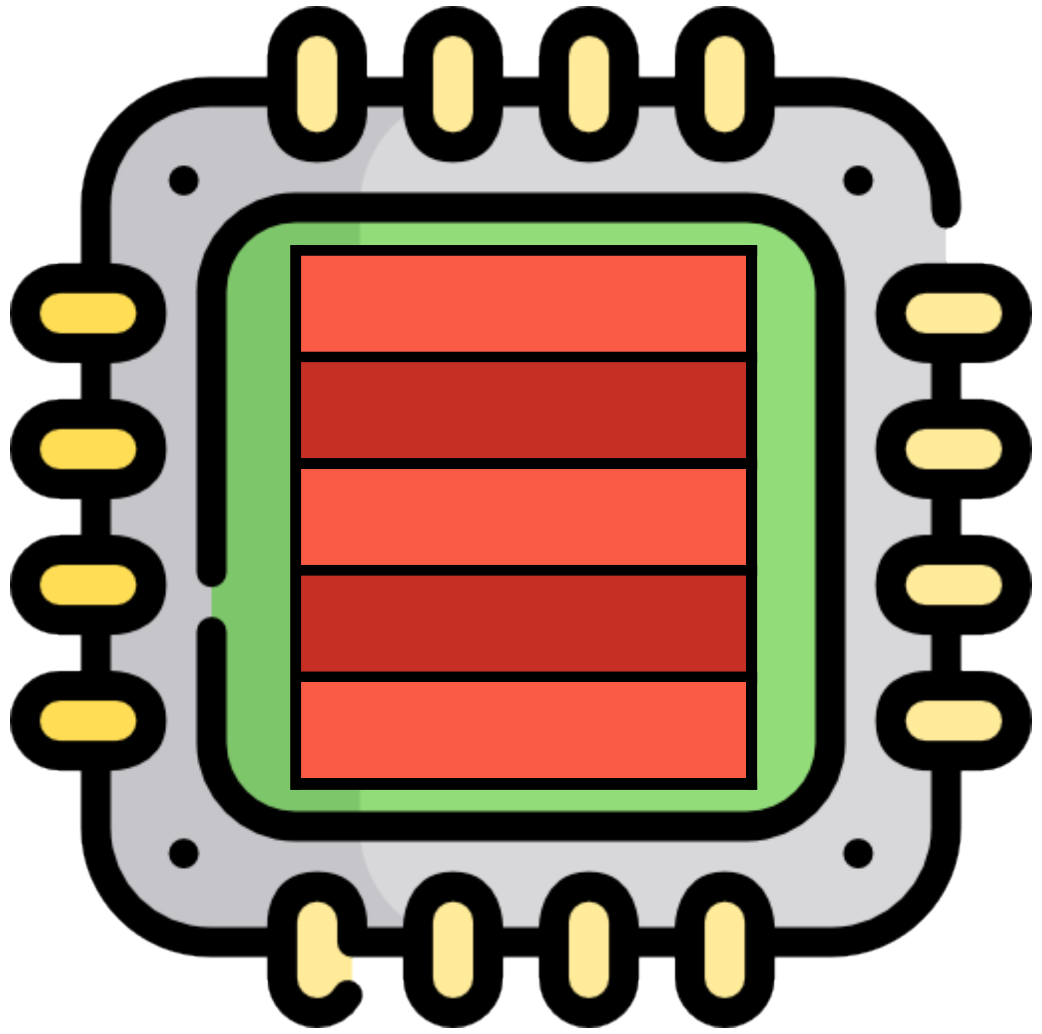


What is in A[128]?

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

1b) Prepare cache

2) Run f(128)



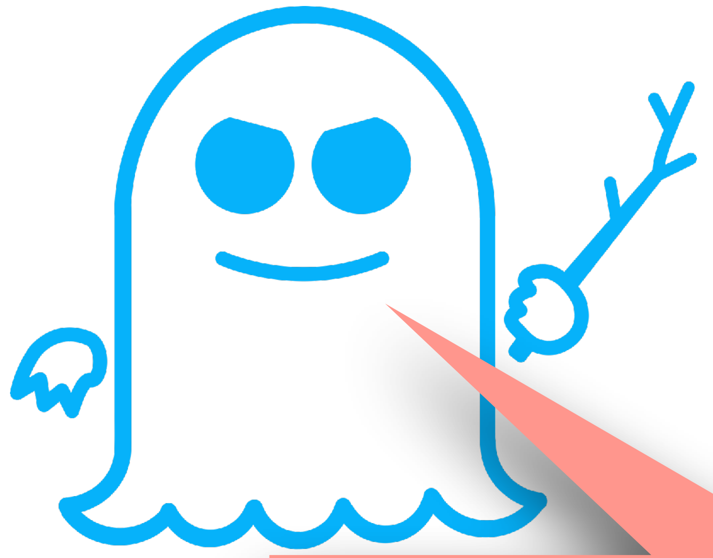
Cache state

Spectre V1

A_size=16

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

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

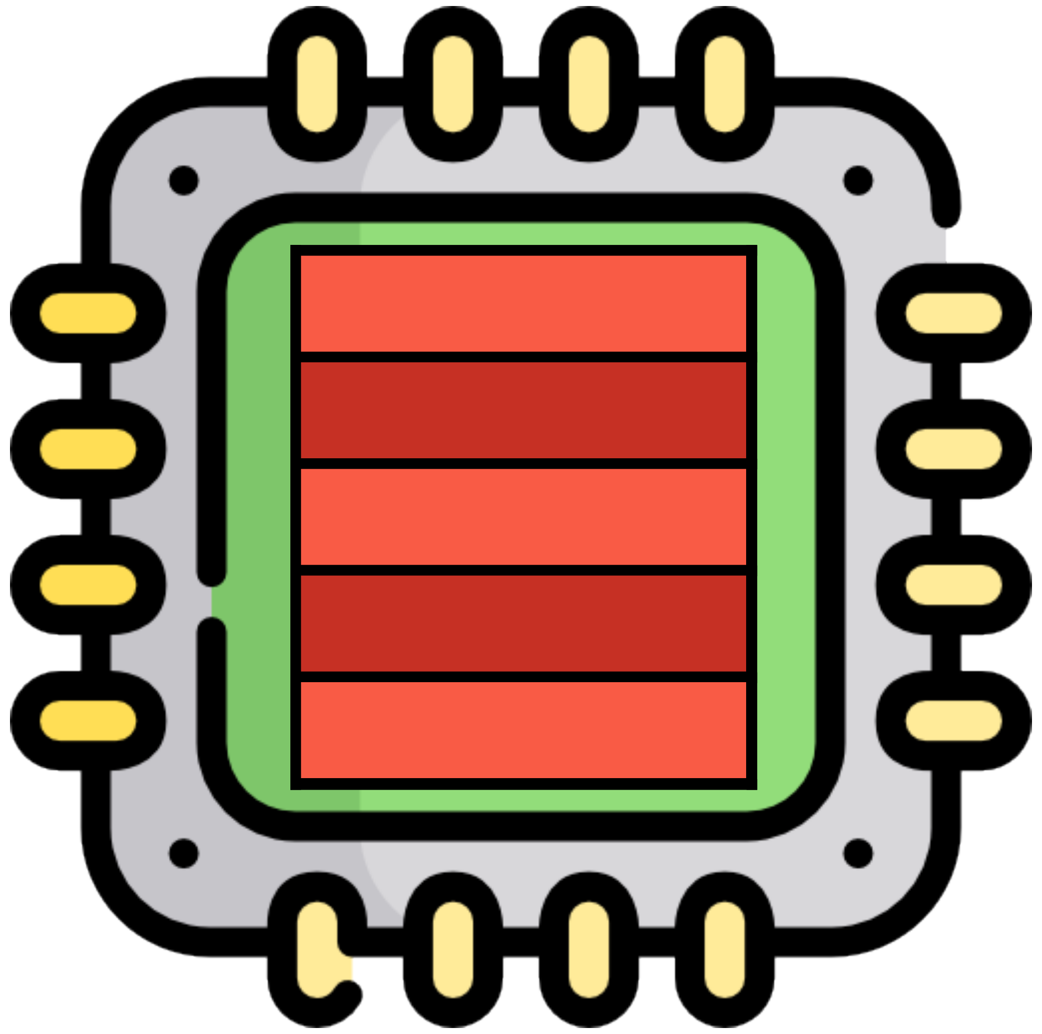


What is in A[128]?

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

1b) Prepare cache

2) Run f(128)



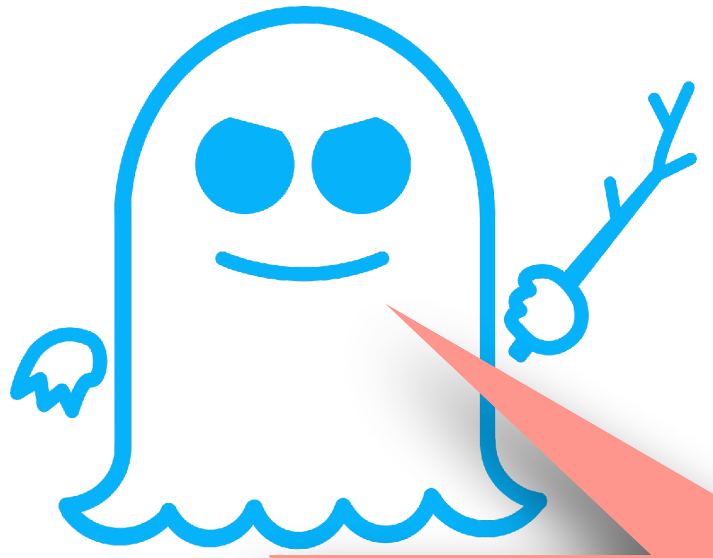
Cache state

Spectre V1

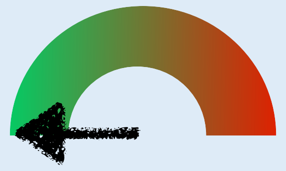
`A_size`=16

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

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

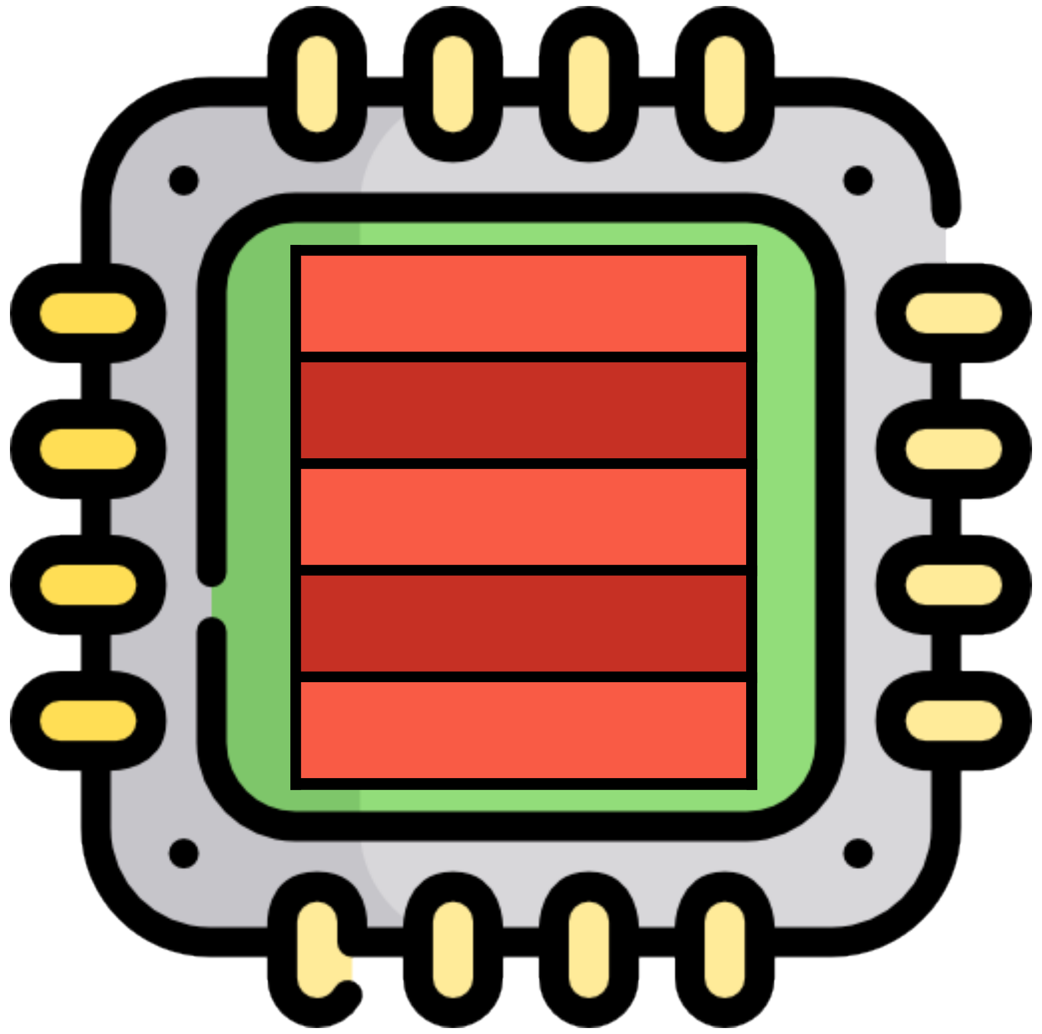


What is in `A[128]`?

1a) Training  `f(0);f(1);f(2);` …

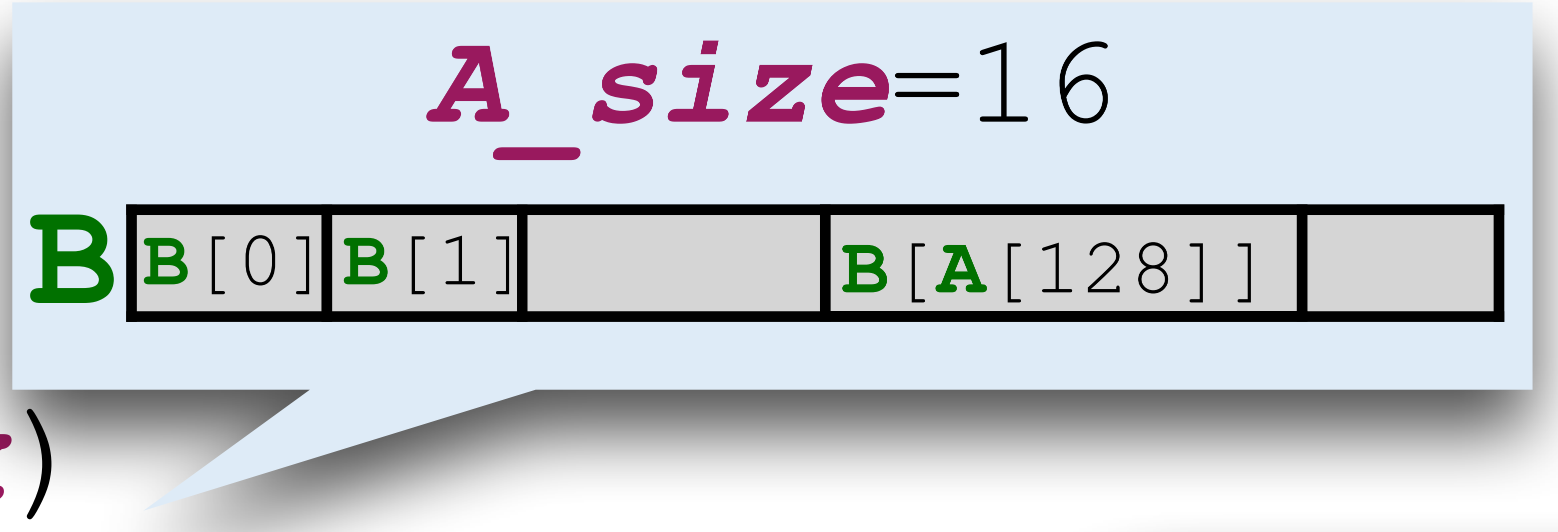
1b) Prepare cache

2) Run `f(128)`

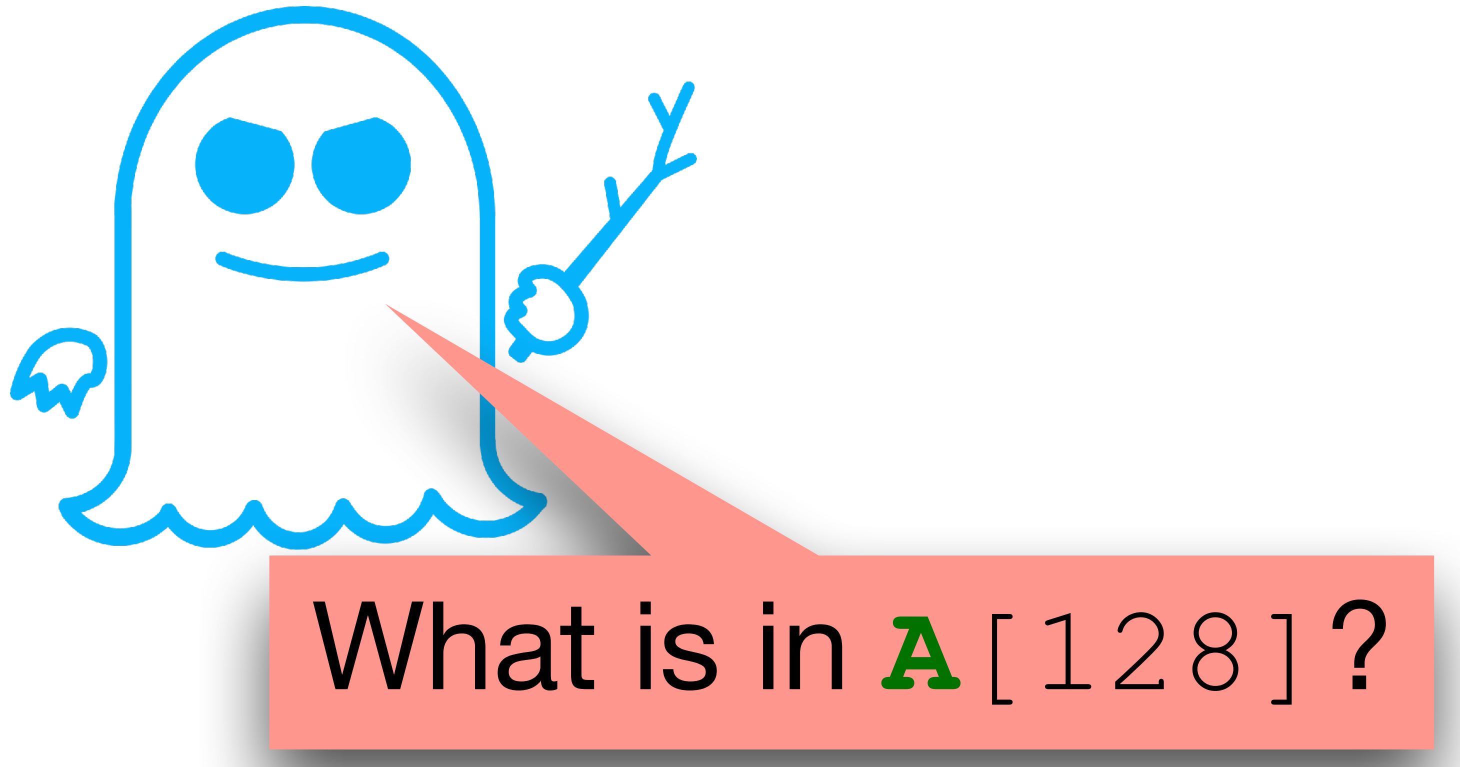


Cache state

Spectre V1



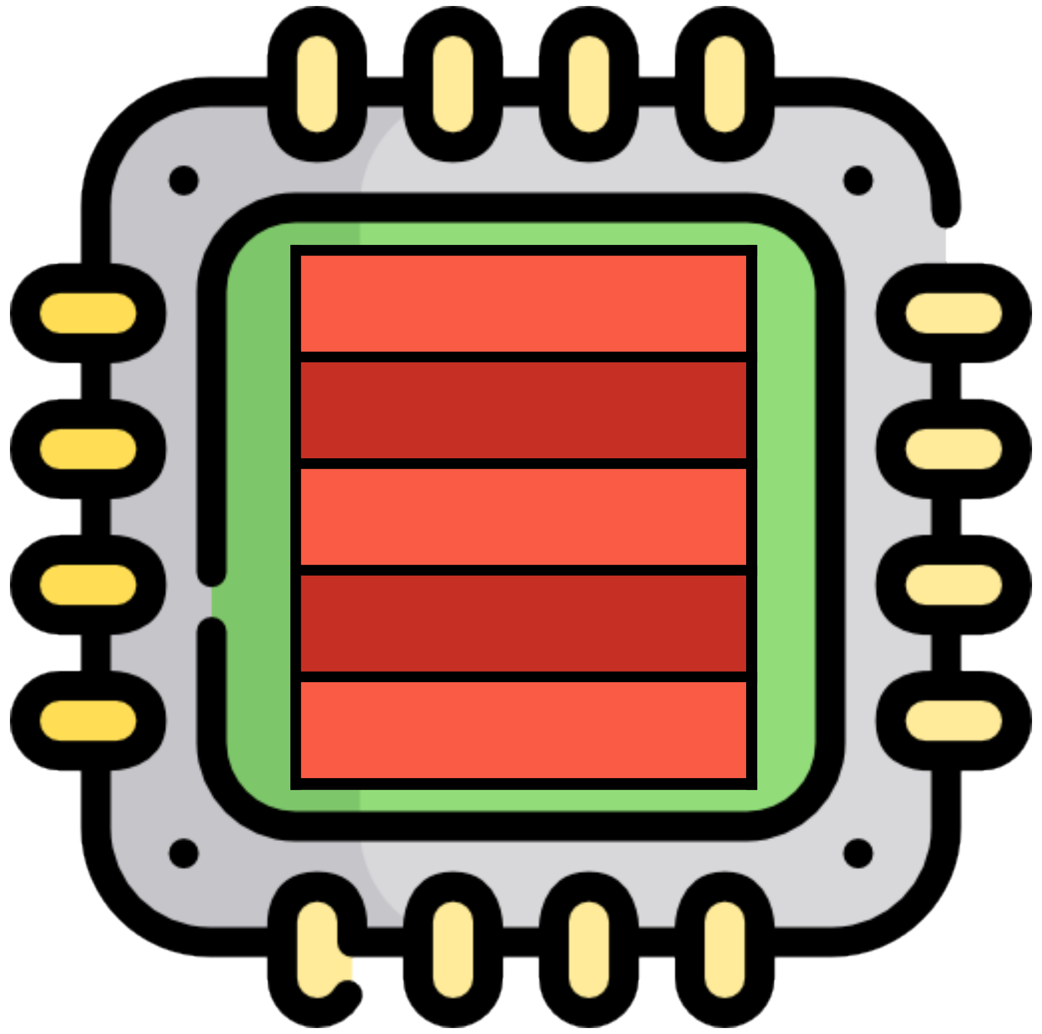

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



1a) Training f(0);f(1);f(2); …

1b) Prepare cache

2) Run f(128)



Cache state

Spectre V1

A_size=16

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

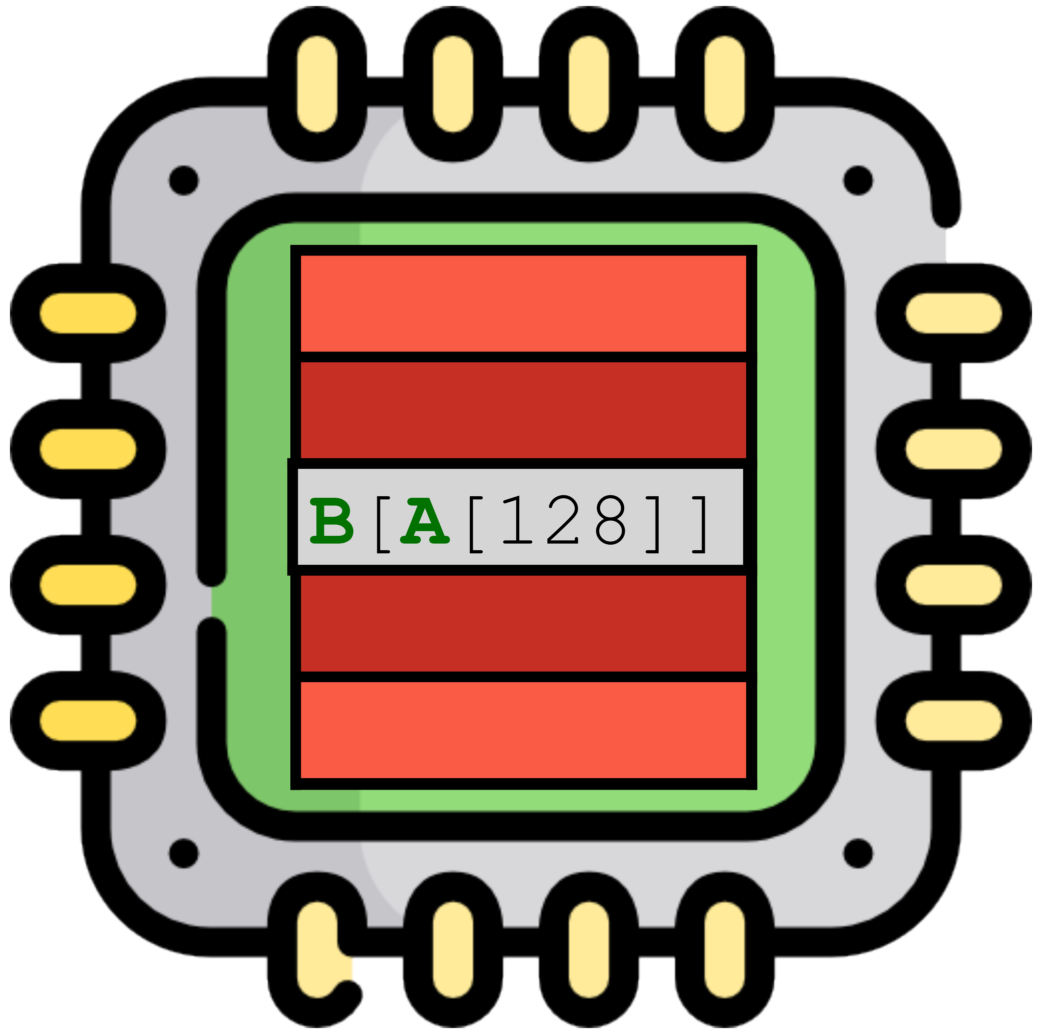
What is in A[128]?

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

1a) Training f(0);f(1);f(2); …

1b) Prepare cache

2) Run f(128)



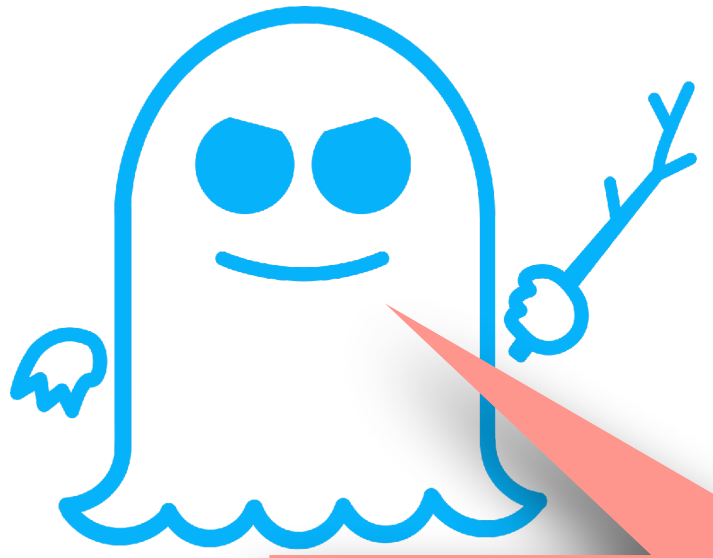
Cache state

Spectre V1

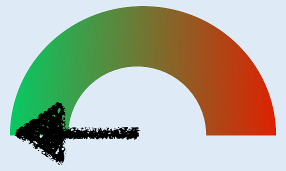
A_size=16

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

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



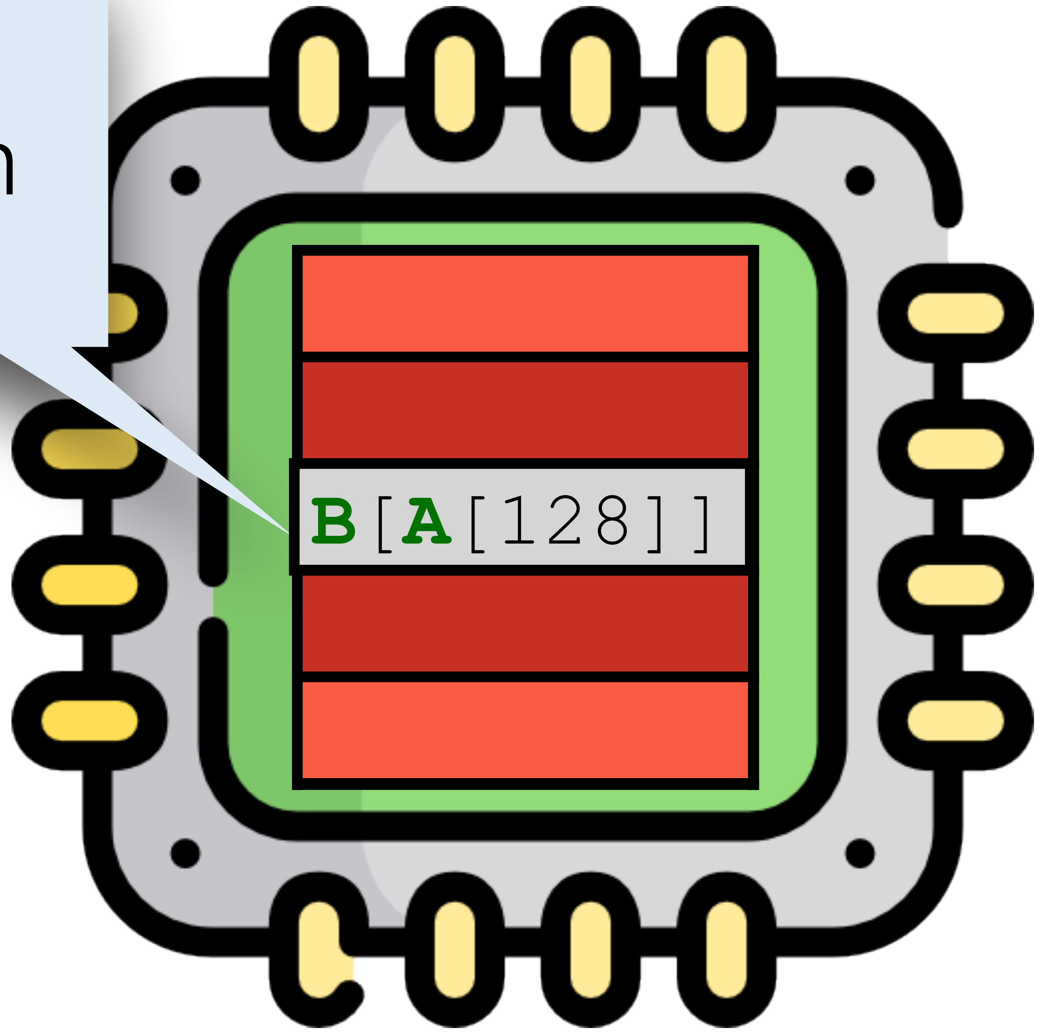
What is in A[128]?

1a) Training  f(0);f(1);f(2); …

1b) Prepare cache

2) Run f(128)

Address depends on A[128]



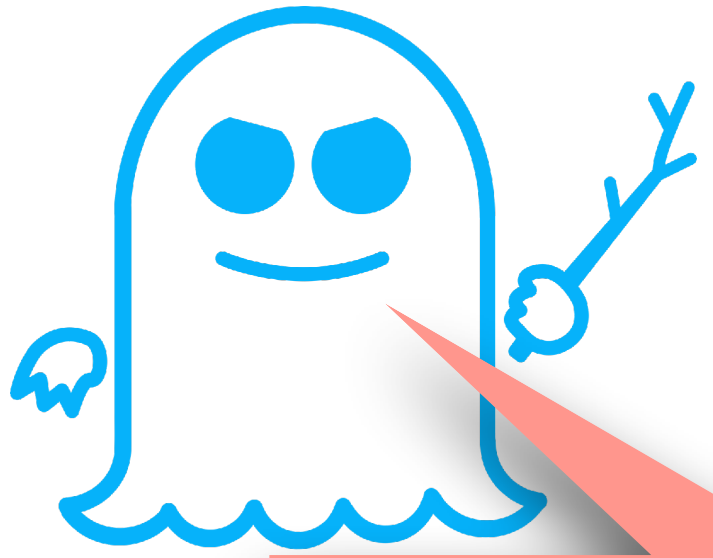
Cache state

Spectre V1

A_size=16

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

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



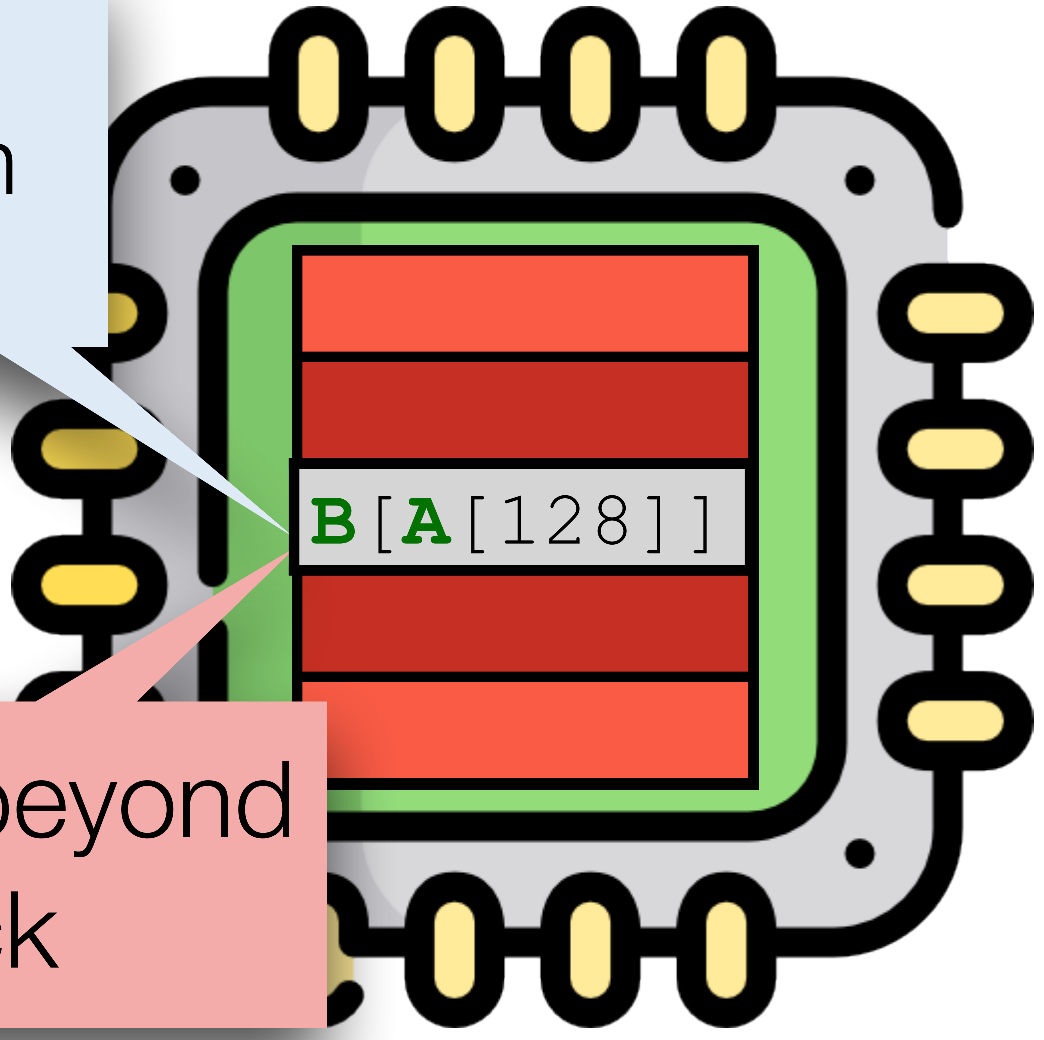
What is in A[128]?

1a) Training f(0);f(1);f(2); …

1b) Prepare cache

2) Run f(128)

Address depends on A[128]



Persistent beyond rollback

Cache state

Spectre V1

A_size=16

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

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

What is in A[128]?

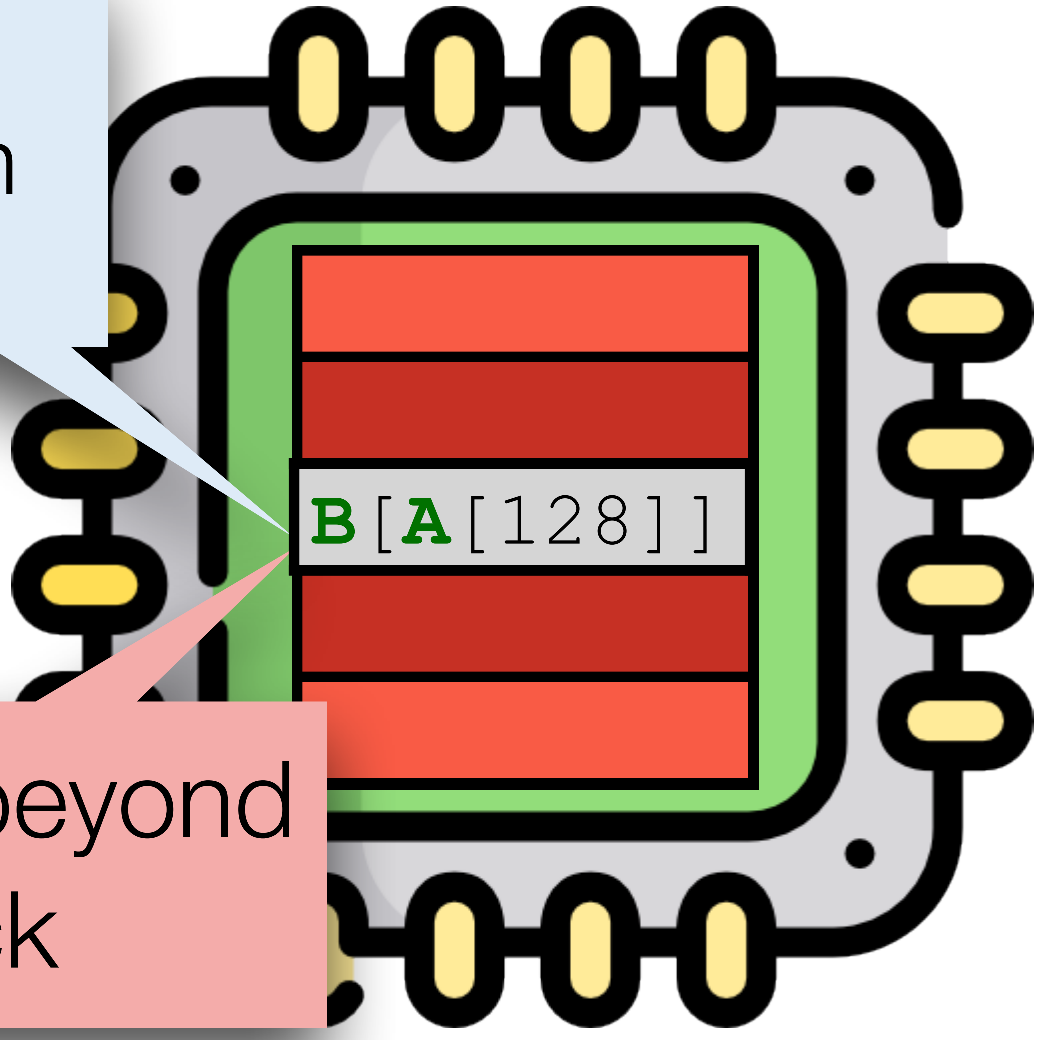
1a) Training f(0);f(1);f(2); …

1b) Prepare cache

2) Run f(128)

3) Extract from cache

Address depends on A[128]

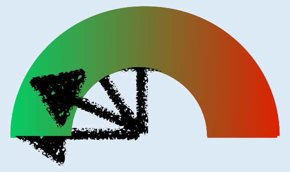


Persistent beyond rollback

Cache state

2. Speculative non-interference

Generalizing the Spectre V1 example

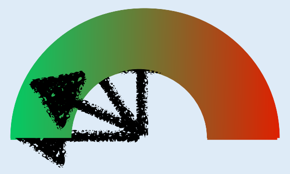
1a) Training  `f(0);f(1);f(2); …`

1b) Prepare cache

2) Run `f` `(128)`

3) Extract from cache

Generalizing the Spectre V1 example

1a) Training  `f(0);f(1);f(2); …`

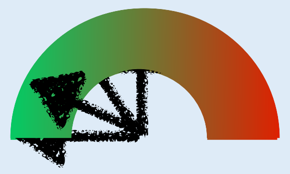
1b) Prepare cache

} Attacker

2) Run `f``(128)`

3) Extract from cache

Generalizing the Spectre V1 example

1a) Training  `f(0);f(1);f(2); …`

1b) Prepare cache

Attacker (1a, 1b)

2) Run `f` `(128)`

Victim (2)

3) Extract from cache

Generalizing the Spectre V1 example

1a) Training `f(0);f(1);f(2); …`

1b) Prepare cache

} Attacker

Victim {

2) Run `f(128)`

3) Extract from cache

} Attacker

Generalizing the Spectre V1 example

1) Prepares microarchitectural state } Attacker

Victim { **2) Leaks information into microarchitectural state**

3) Extracts information from microarchitecture } Attacker

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	$\stackrel{?}{=}$	Leakage of **P** in **speculative** execution

Speculative non-interference

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

Informally:

Leakage of **P** in **non-speculative** execution	$\stackrel{?}{=}$	Leakage of **P** in **speculative** execution

More formally:

Speculative non-interference

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

Informally:

Leakage of **P** in **non-speculative** execution	$\stackrel{?}{=}$	Leakage of **P** in **speculative** execution

More formally:

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

Speculative non-interference

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

Informally:

Leakage of **P** in **non-speculative** execution	$\stackrel{?}{=}$	Leakage of **P** in **speculative** execution

More formally:

For all program states s and s':

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

Speculative non-interference

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

Informally:

Leakage of **P** in **non-speculative** execution	$\stackrel{?}{=}$	Leakage of **P** in **speculative** execution

More formally:

For all program states s and s':

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

Speculative non-interference

Extended with policies

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

Informally:

Leakage of **P** in **non-speculative** execution	$\stackrel{?}{=}$	Leakage of **P** in **speculative** execution

More formally:

For all program states s and s':

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

How to capture leakage into microarchitectural state?

How to capture leakage into microarchitectural state?

Non-speculative semantics

Speculative semantics

How to capture leakage into microarchitectural state?

Non-speculative semantics

Speculative semantics

+

Attacker/Observer model

µAssembly

```
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

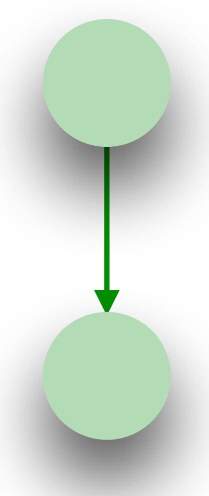
```
if (x < A_size)
  y = B[A[x]]
```

```
       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]]
```

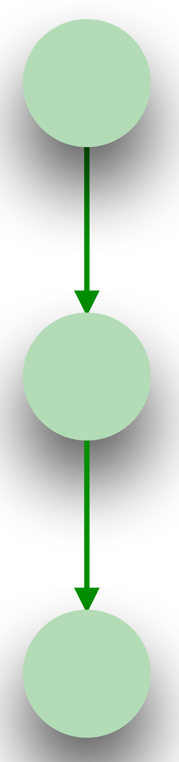
```
        rax <- A_size
        rcx <- x
        jmp rcx≥rax, END
L1:     load rax, A + rcx
        load rax, B + rax
END:
```



µAssembly + Non-speculative semantics

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

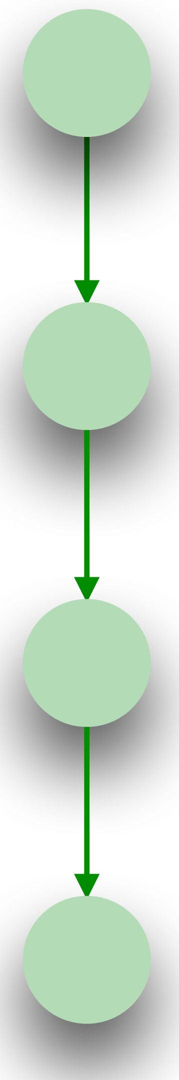
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```



µAssembly + Non-speculative semantics

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

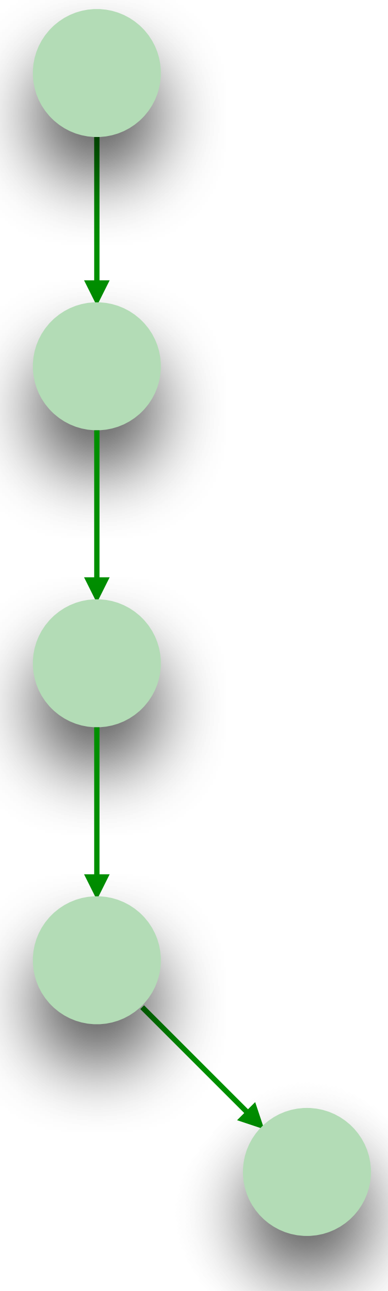
```
        rax <- A_size
        rcx <- x
        jmp rcx≥rax, END
L1:     load rax, A + rcx
        load rax, B + rax
END:
```



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:
```



Non-speculative semantics: Inference Rules

Expression evaluation

$[\![n]\!](a) = n \qquad [\![x]\!](a) = a(x) \qquad [\![\ominus e]\!](a) = \ominus [\![e]\!](a) \qquad [\![e_1 \otimes e_2]\!](a) = [\![e_1]\!](a) \otimes [\![e_2]\!](a)$

Instruction evaluation

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

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

Assign
$$\frac{p(a(\mathbf{pc})) = x \leftarrow e \qquad x \neq \mathbf{pc}}{\langle m, a\rangle \to \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1, x \mapsto [\![e]\!](a)]\rangle}$$

ConditionalUpdate-Sat
$$\frac{p(a(\mathbf{pc})) = x \xleftarrow{e'} e \qquad [\![e']\!](a) = 0 \qquad x \neq \mathbf{pc}}{\langle m, a\rangle \to \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1, x \mapsto [\![e]\!](a)]\rangle}$$

ConditionalUpdate-Unsat
$$\frac{p(a(\mathbf{pc})) = x \xleftarrow{e'} e \qquad [\![e']\!](a) \neq 0 \qquad x \neq \mathbf{pc}}{\langle m, a\rangle \to \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1]\rangle}$$

Terminate
$$\frac{p(a(\mathbf{pc})) = \bot}{\langle m, a\rangle \to \langle m, a[\mathbf{pc} \mapsto \bot]\rangle}$$

Load
$$\frac{p(a(\mathbf{pc})) = \mathbf{load}\ x, e \qquad x \neq \mathbf{pc} \qquad n = [\![e]\!](a)}{\langle m, a\rangle \xrightarrow{\texttt{load}\ n} \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1, x \mapsto m(n)]\rangle}$$

Store
$$\frac{p(a(\mathbf{pc})) = \mathbf{store}\ x, e \qquad n = [\![e]\!](a)}{\langle m, a\rangle \xrightarrow{\texttt{store}\ n} \langle m[n \mapsto a(x)], a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1]\rangle}$$

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

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

Jmp
$$\frac{p(a(\mathbf{pc})) = \mathbf{jmp}\ e \qquad \ell = [\![e]\!](a)}{\langle m, a\rangle \xrightarrow{\texttt{pc}\ \ell} \langle m, a[\mathbf{pc} \mapsto \ell]\rangle}$$

Speculative semantics

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

Speculative semantics

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

Starts **speculative transactions** upon branches

Speculative semantics

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

Starts **speculative transactions** upon branches

Committed upon correct speculation

Speculative semantics

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

Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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

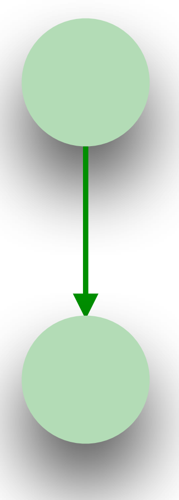
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



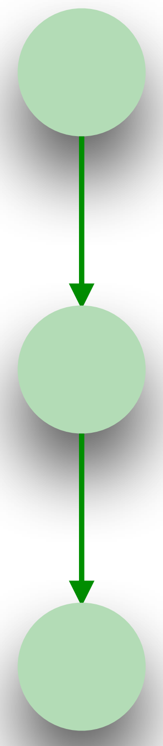
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



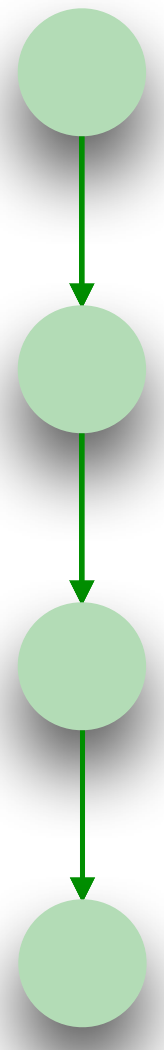
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



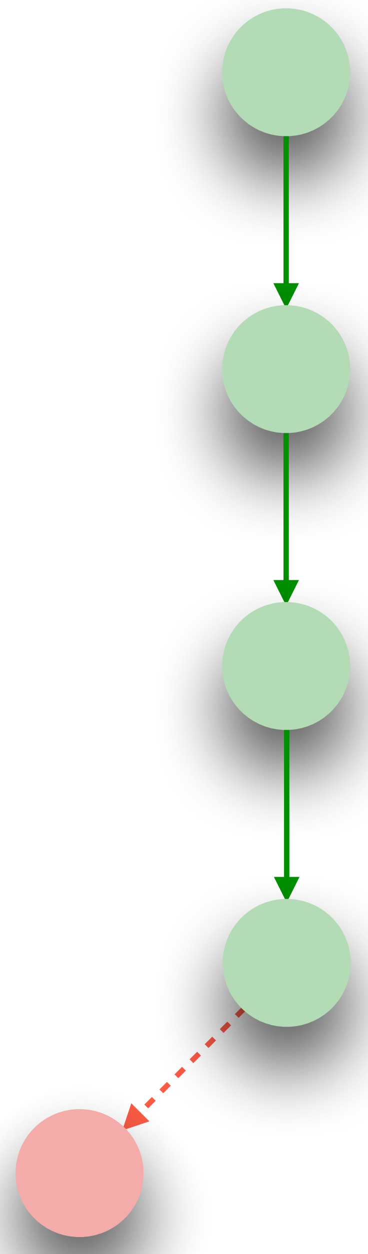
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



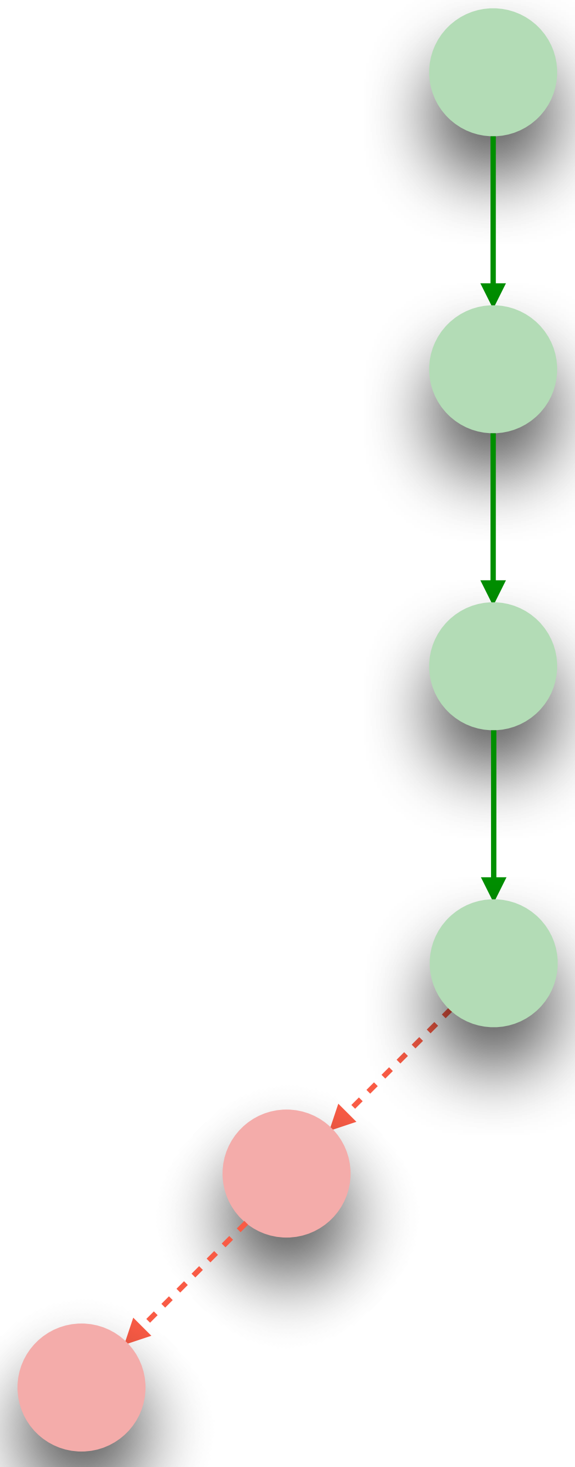
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



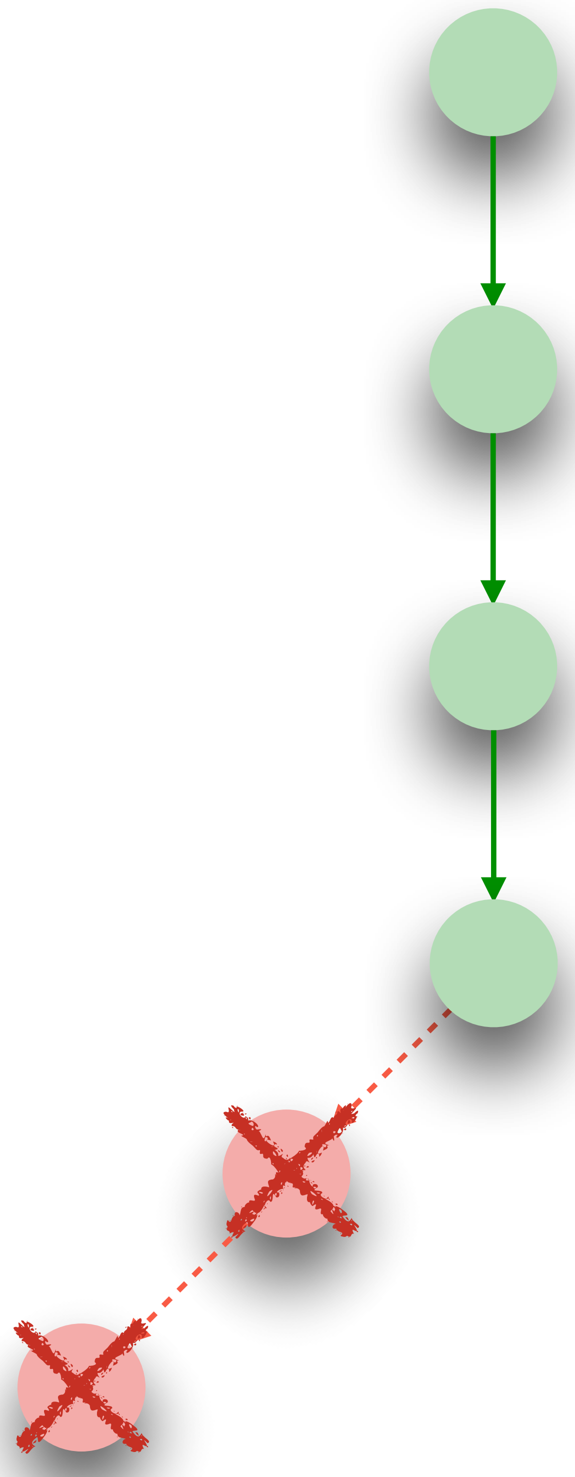
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



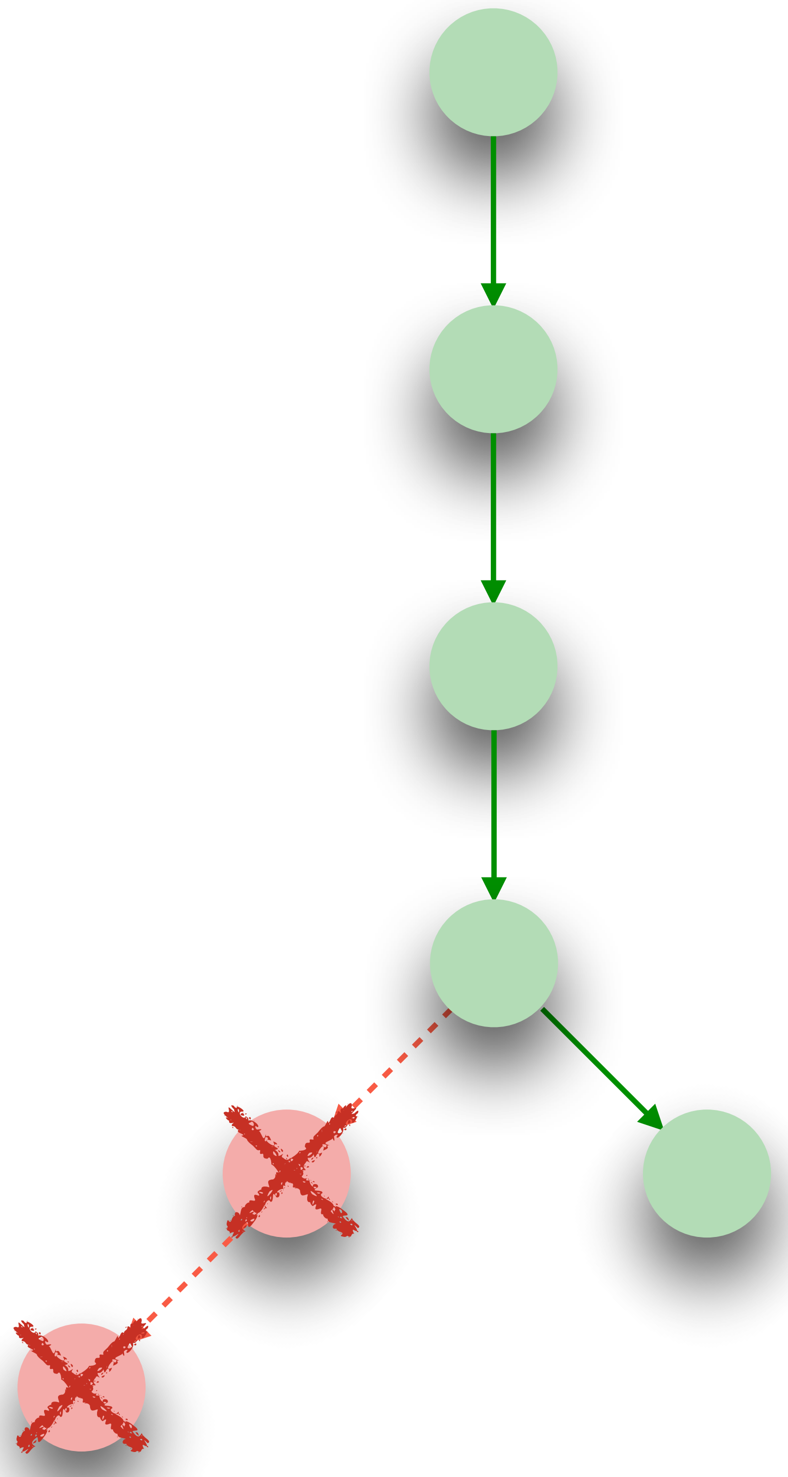
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



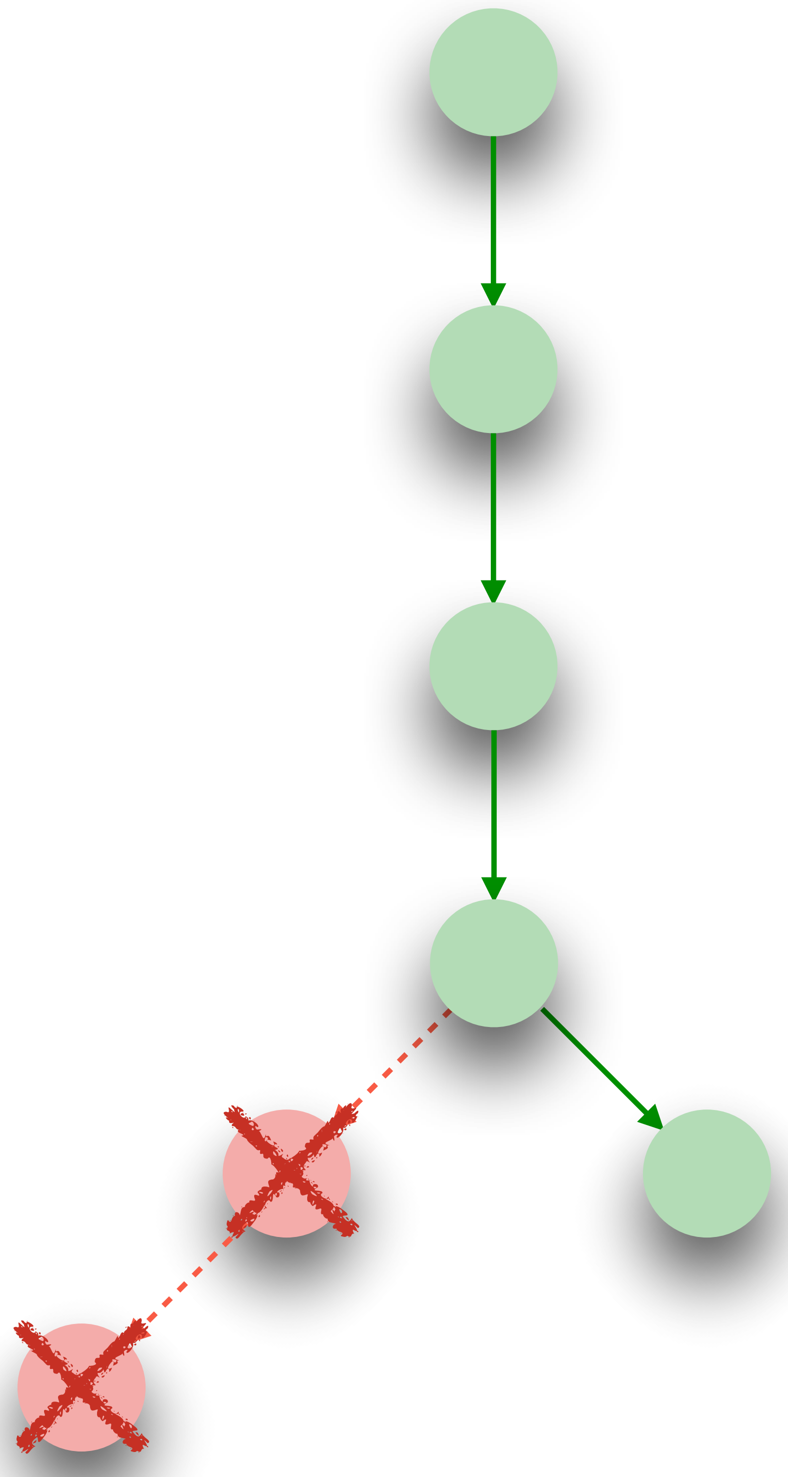
Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Speculative semantics

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



Starts **speculative transactions** upon branches

Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle O determines branch prediction + length of speculative window

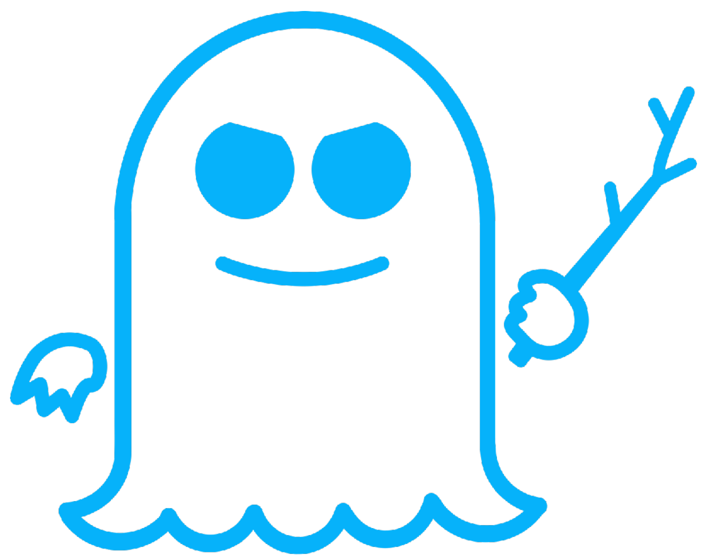
Observer model: Leakage into μarchitectural state

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

Observer model: Leakage into μarchitectural state

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

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



Observer model: Leakage into μarchitectural state

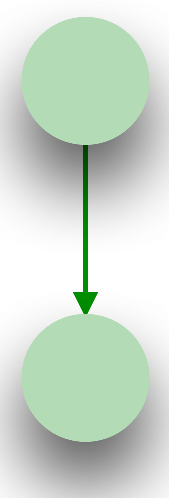
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```

Attacker can observe:

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

Observer model: Leakage into μarchitectural state

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

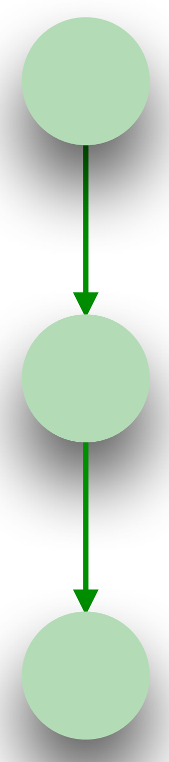


Attacker can observe:

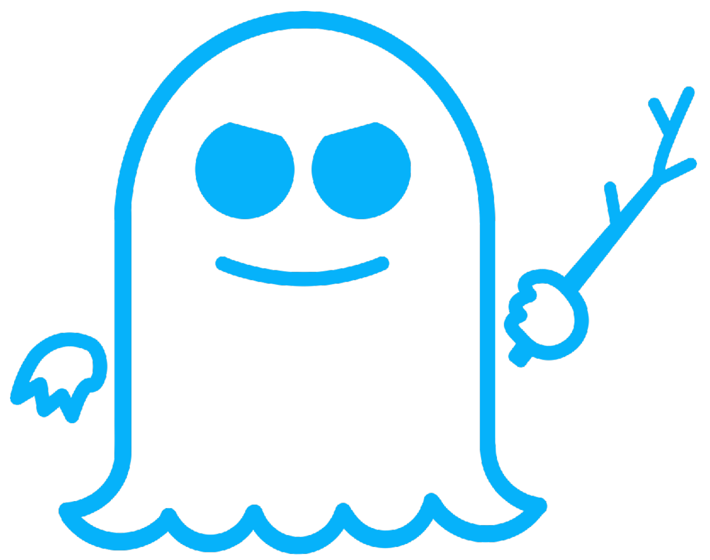
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Observer model: Leakage into µarchitectural state

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

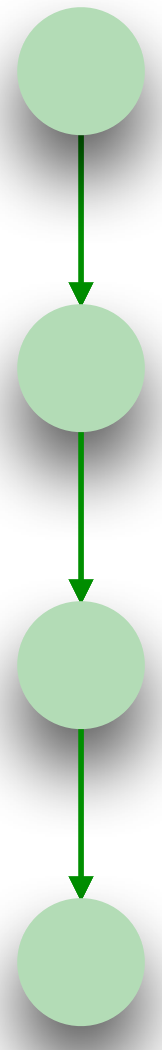


Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution



Observer model: Leakage into μarchitectural state

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

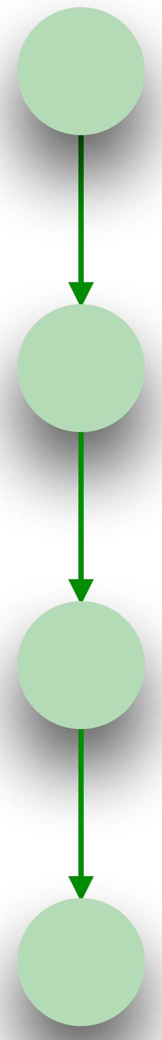


Attacker can observe:

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

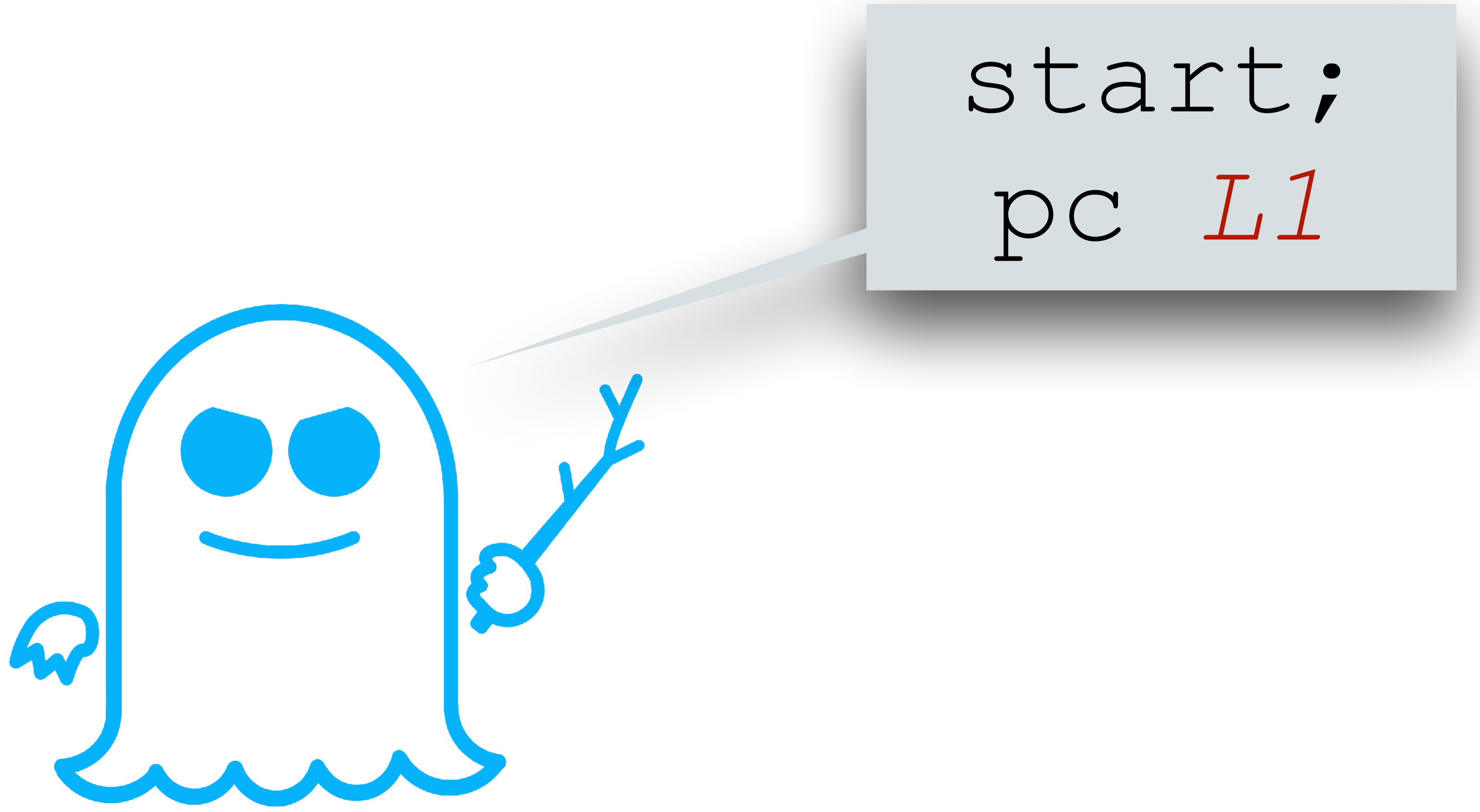
Observer model: Leakage into µarchitectural state

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



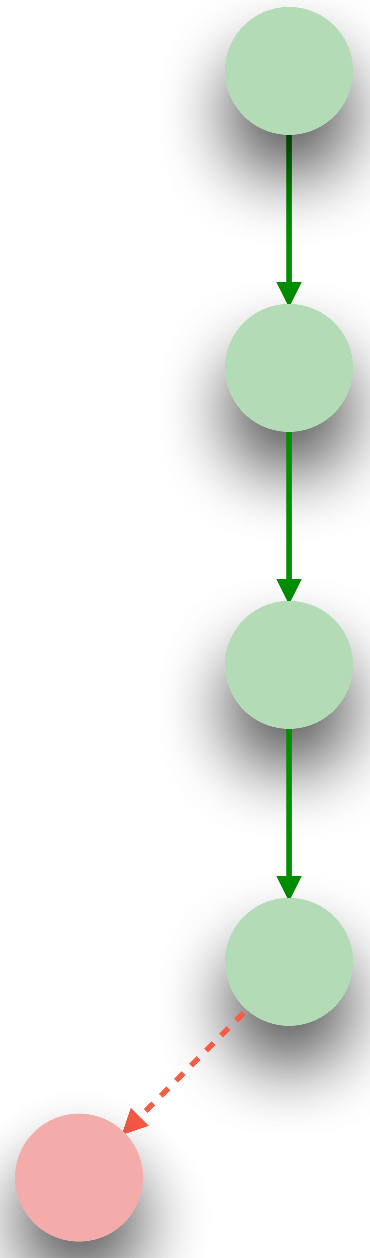
Attacker can observe:

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



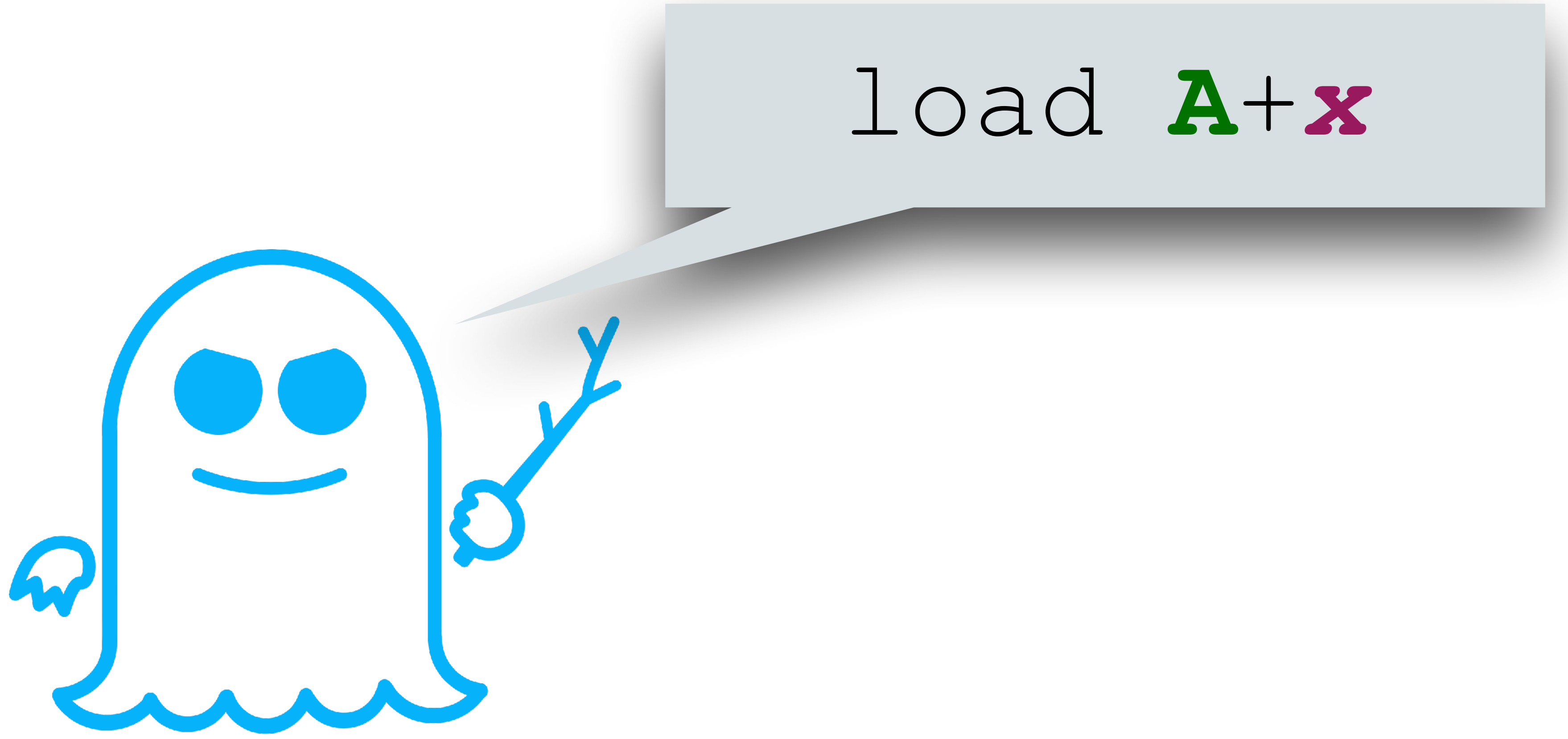
Observer model: Leakage into μarchitectural state

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



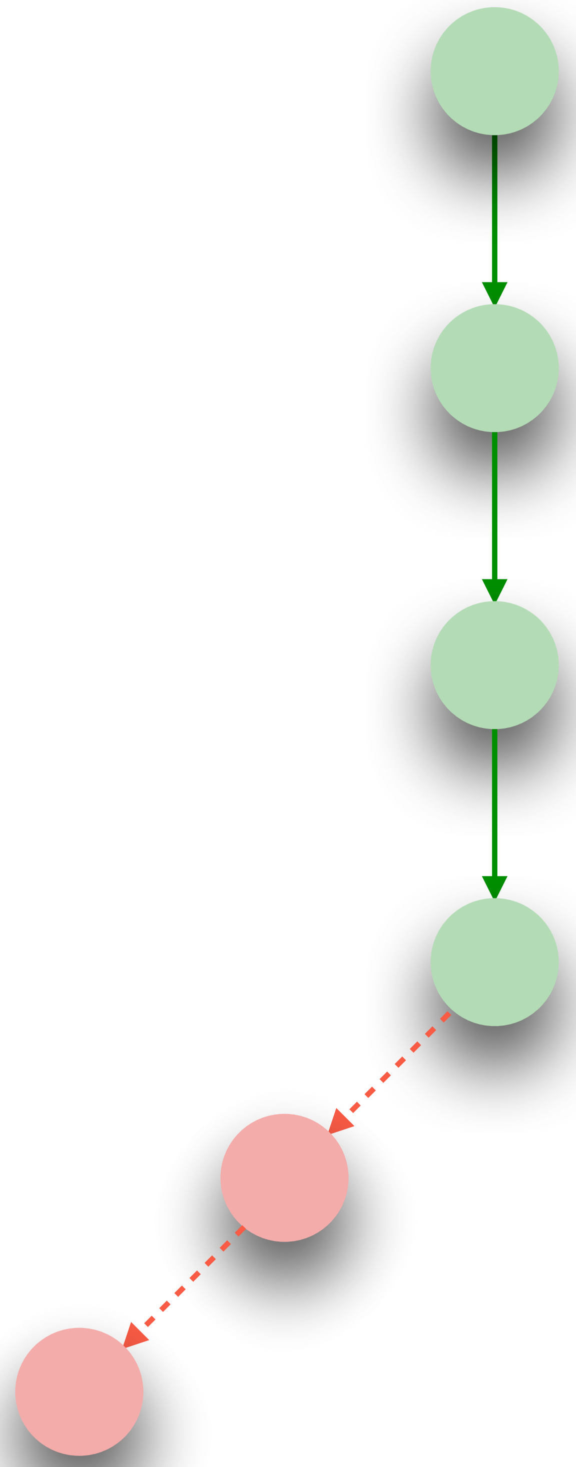
Attacker can observe:

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



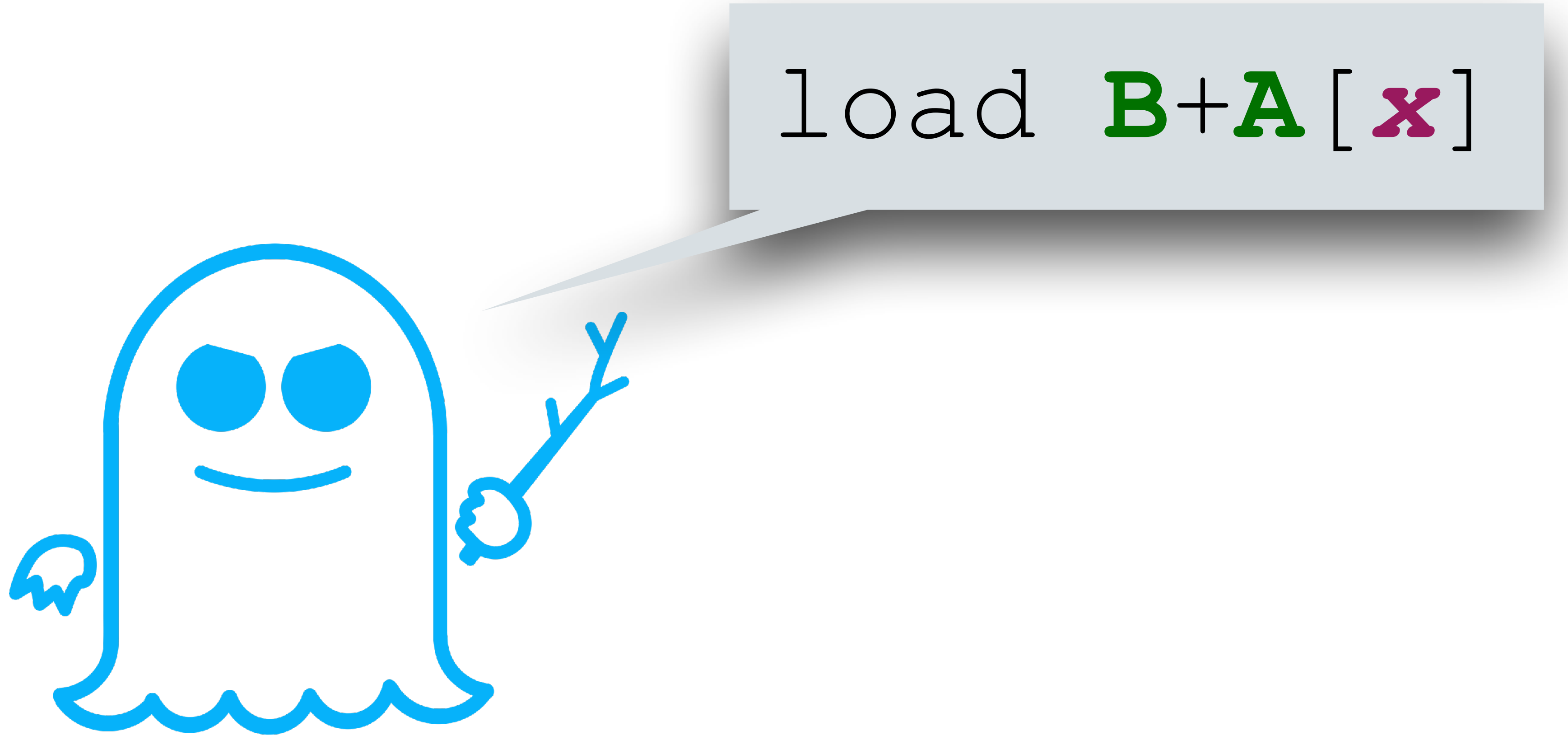
Observer model: Leakage into μarchitectural state

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



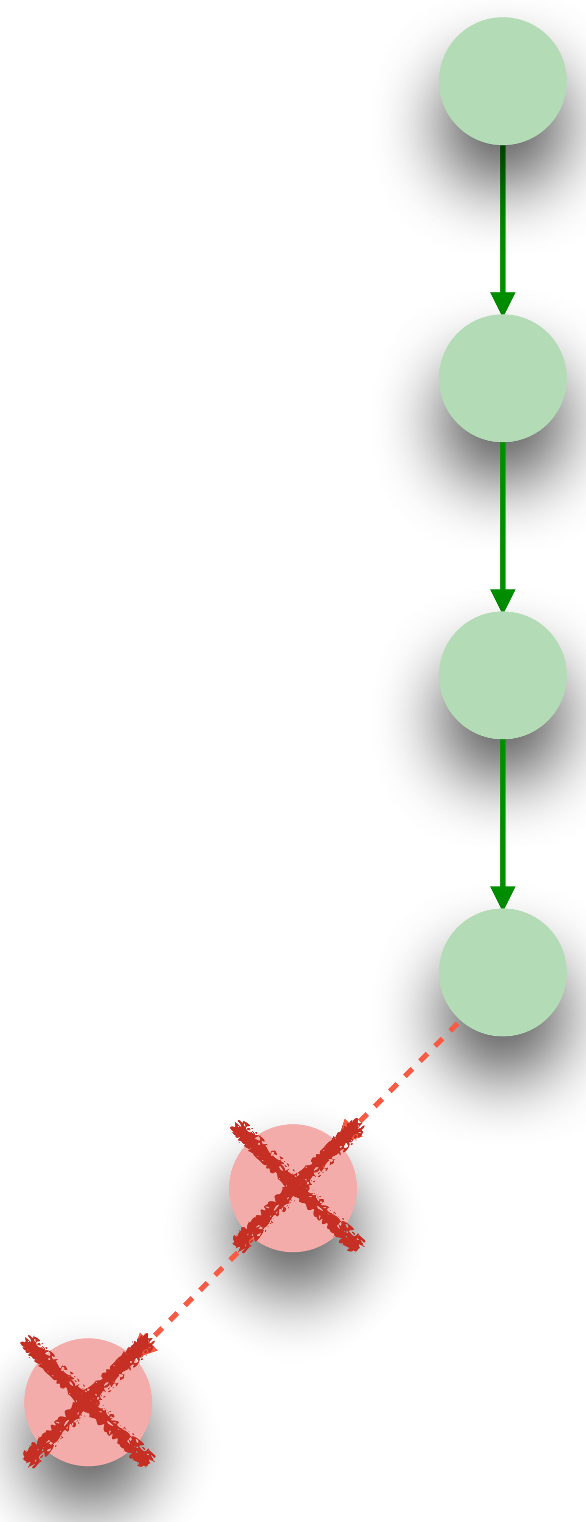
Attacker can observe:

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



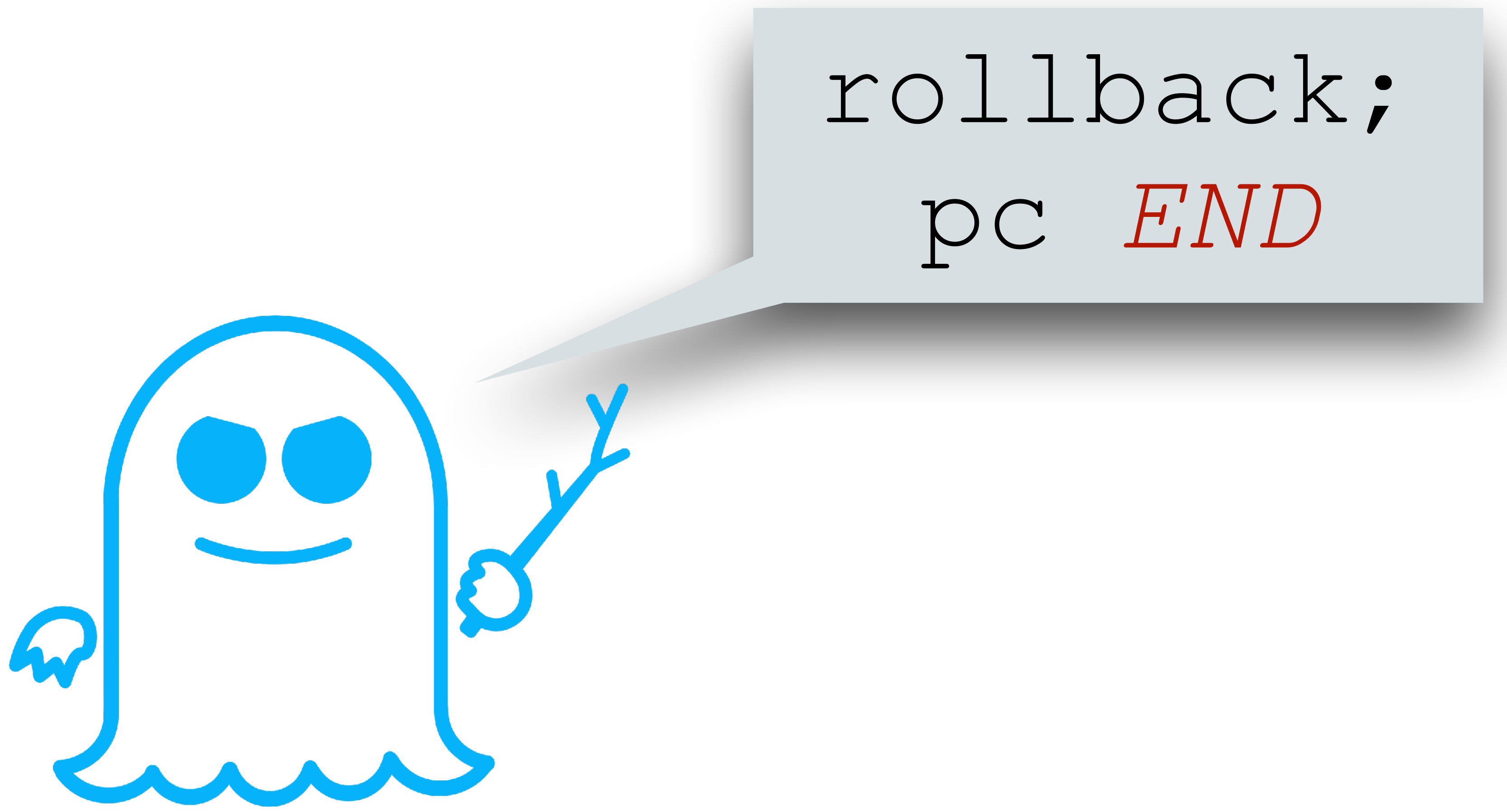
Observer model: Leakage into μarchitectural state

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



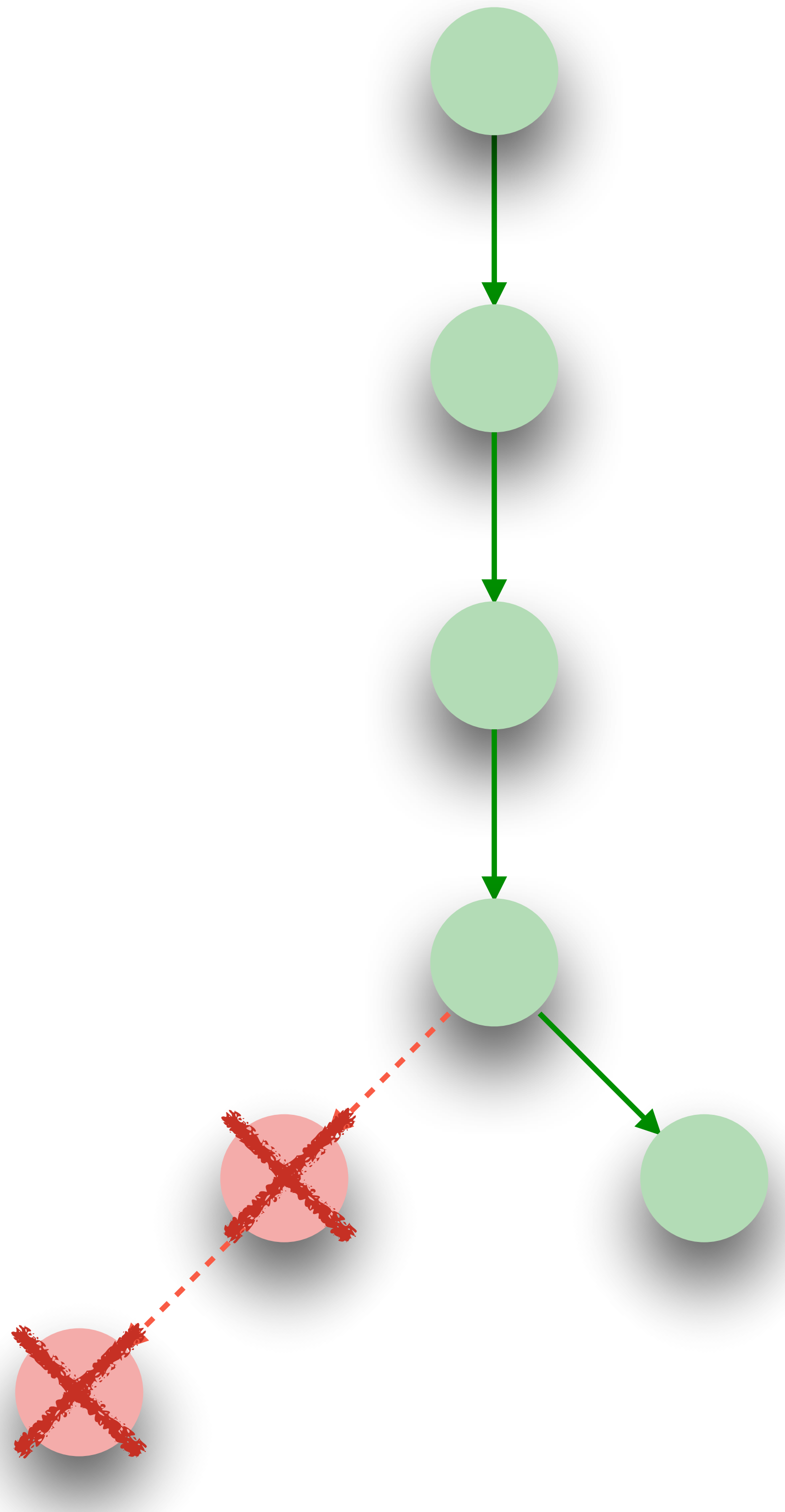
Attacker can observe:

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



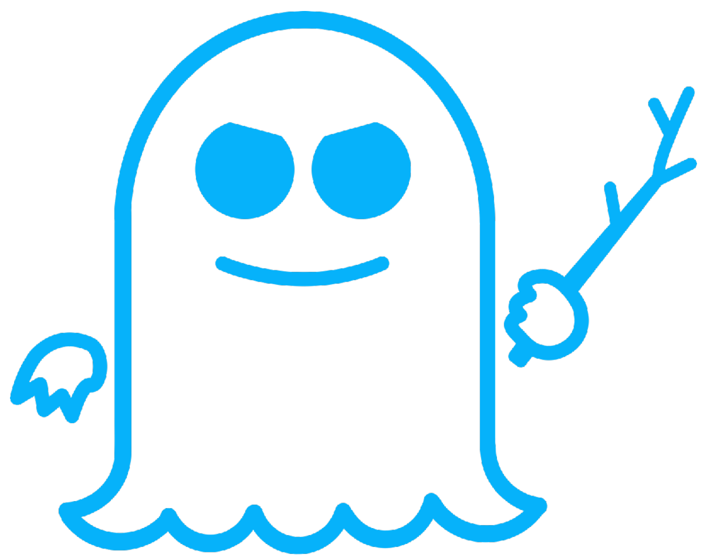
Observer model: Leakage into μarchitectural state

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



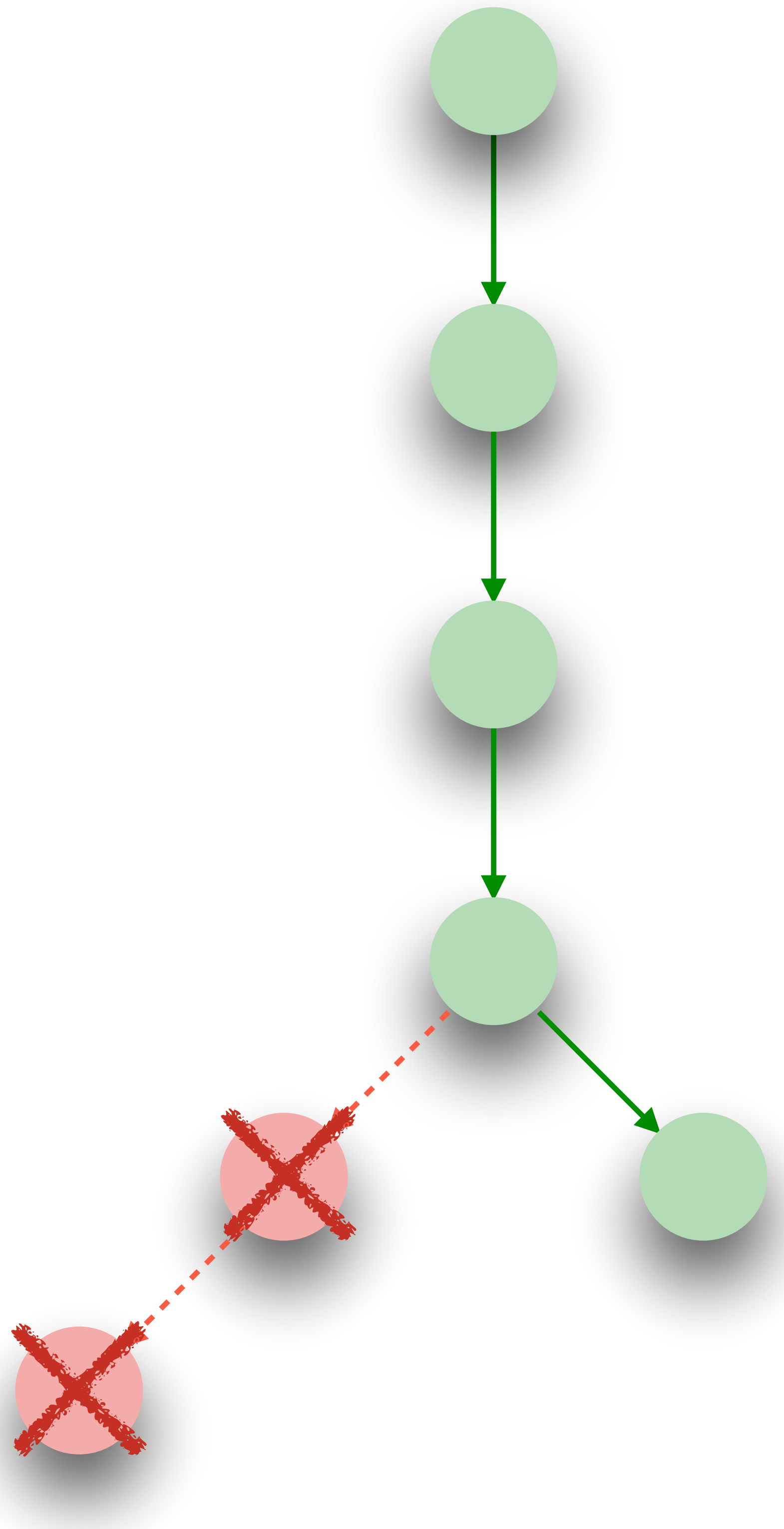
Attacker can observe:

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



Observer model: Leakage into μarchitectural state

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

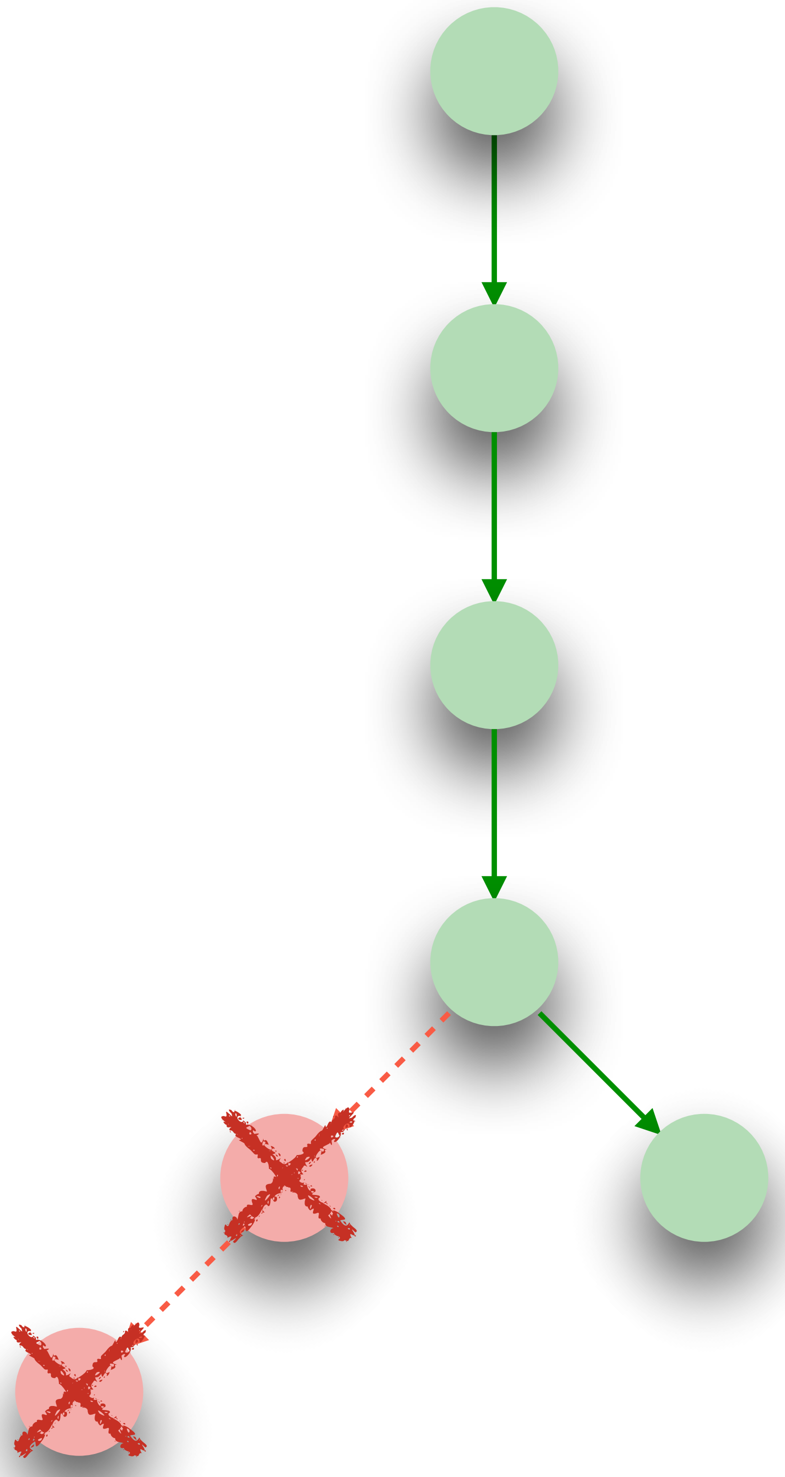


Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by “constant-time” programming requirements

Observer model: Leakage into μarchitectural state

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



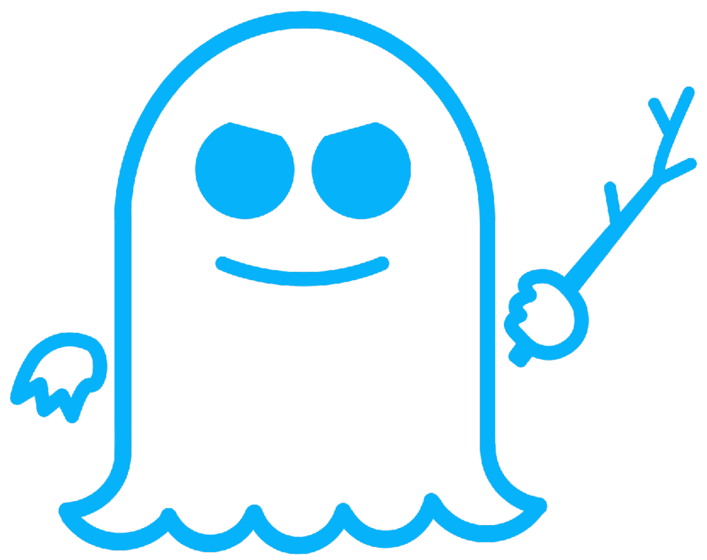
Attacker can observe:

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

Inspired by "constant-time" programming requirements

No need for detailed model of memory hierarchy:

- possibly pessimistic
- more robust



Reasoning about arbitrary prediction oracles

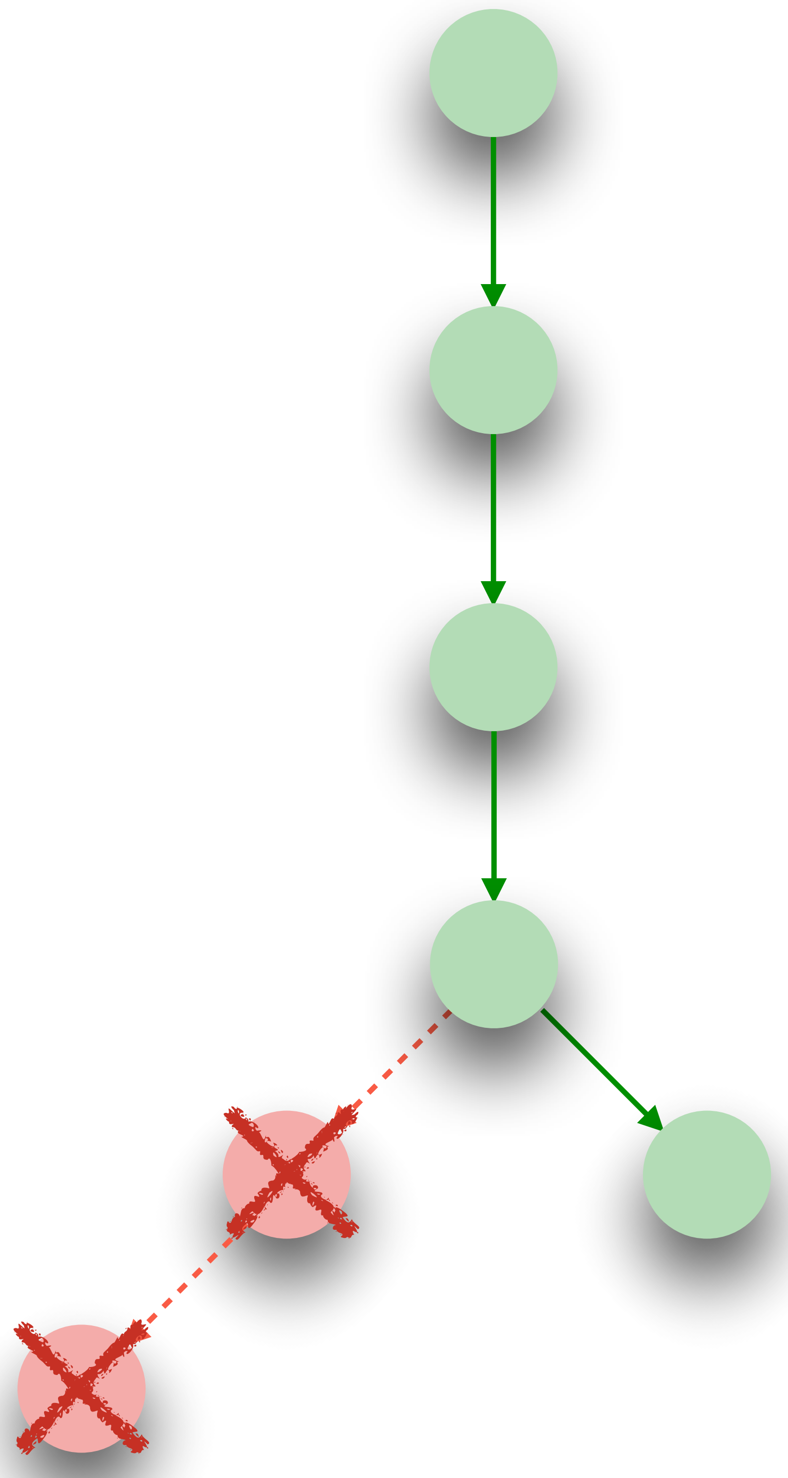
Speculative semantics
+
Prediction oracle

→

Always-mispredict speculative semantics

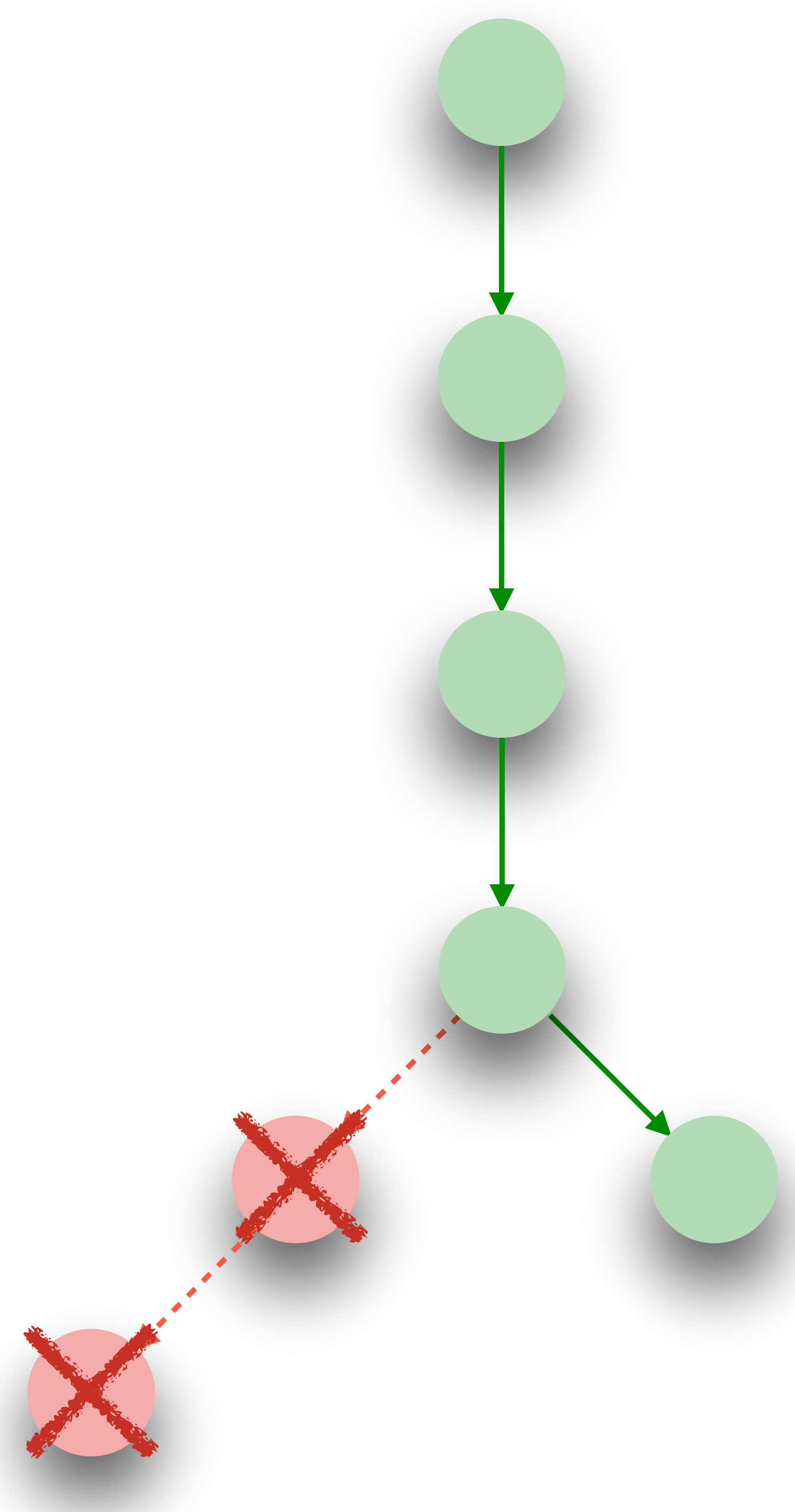
Always-mispredict speculative semantics

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



Always-mispredict speculative semantics

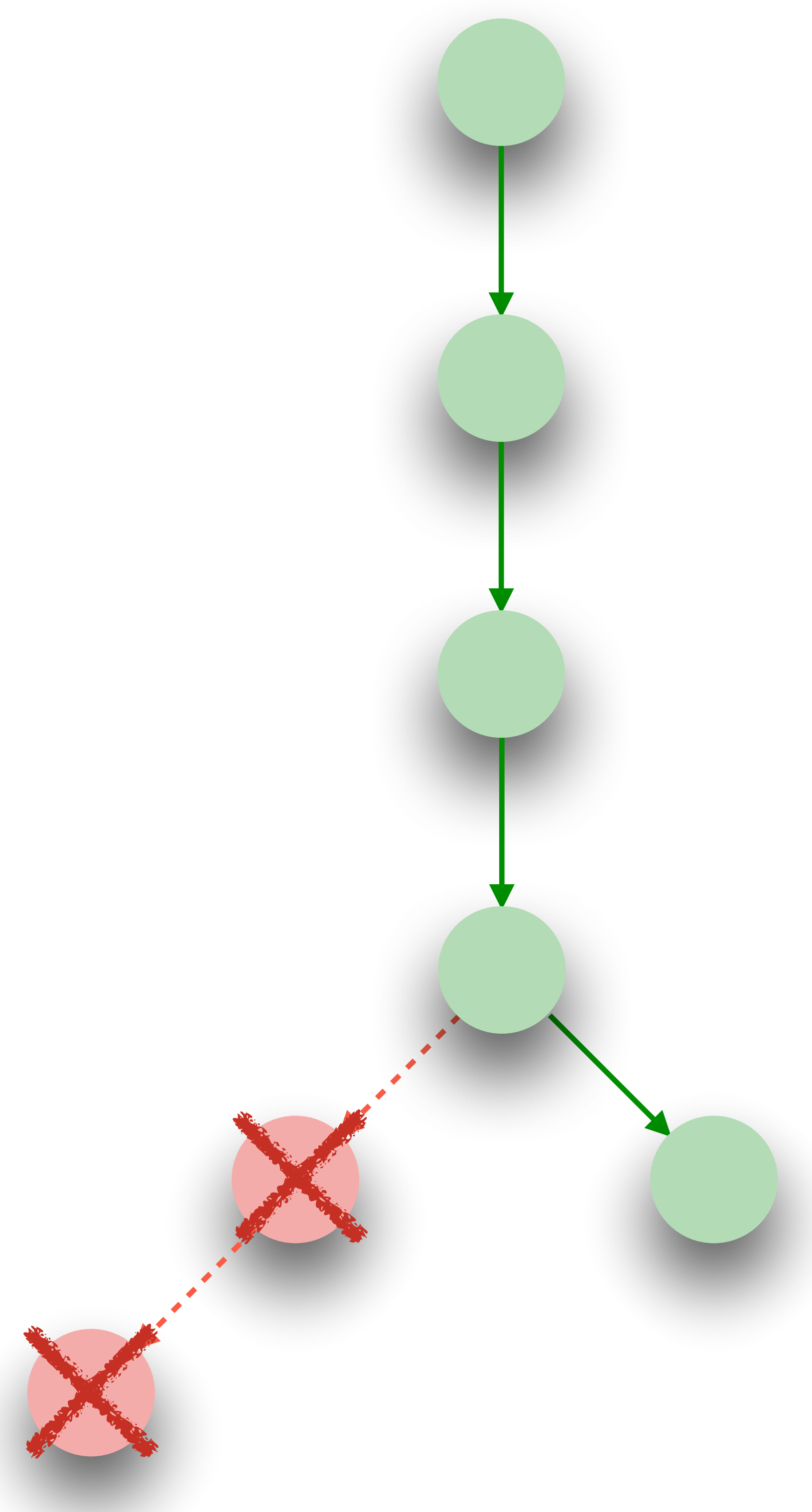
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```



Always mispredict branch instructions' outcomes

Always-mispredict speculative semantics

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

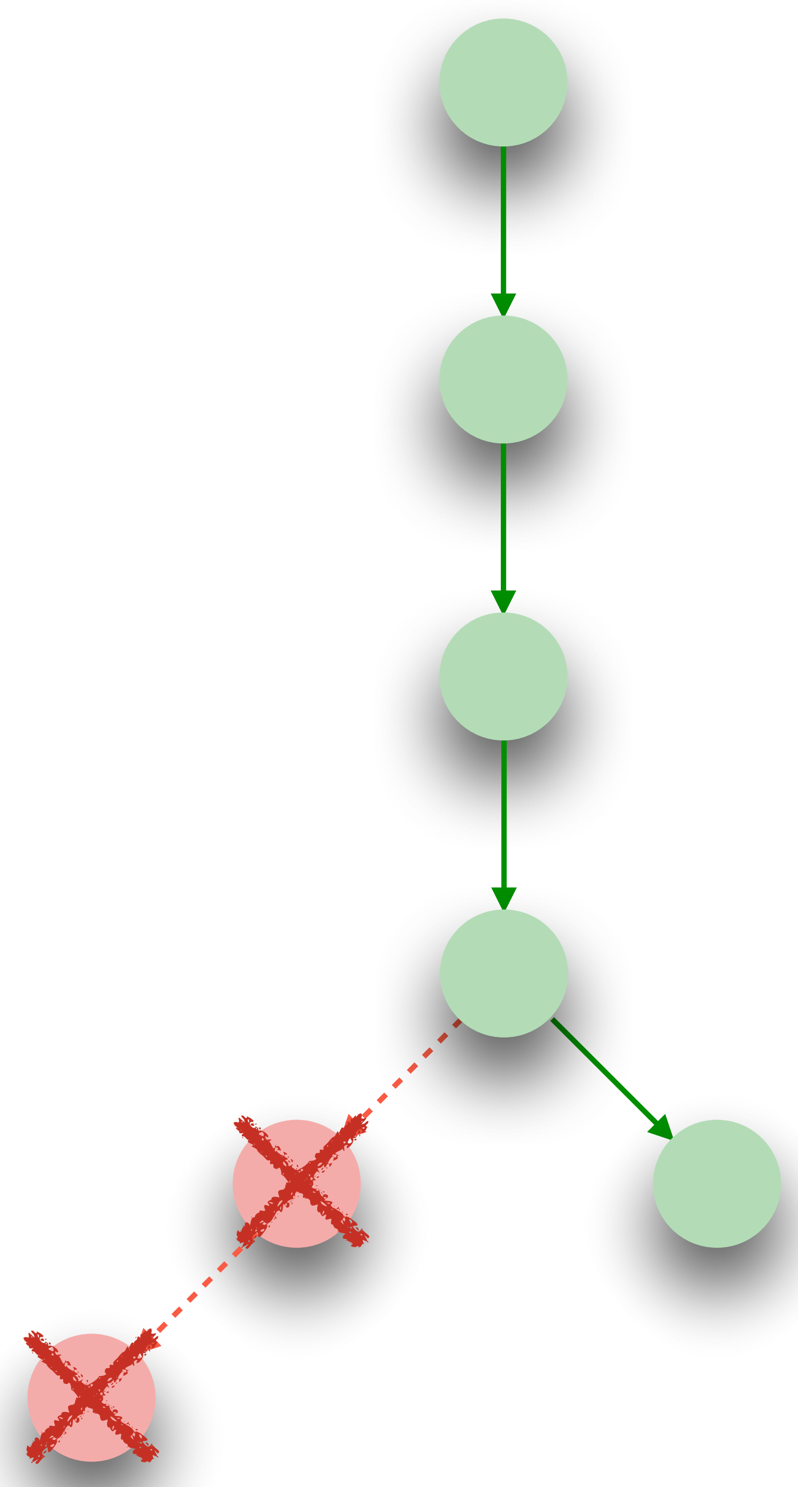


Always mispredict branch instructions' outcomes

Fixed speculative window

Always-mispredict speculative semantics

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



Always mispredict branch instructions' outcomes

Fixed speculative window

Rollback of every transaction

Always-mispredict speculative semantics: Inference Rules

SE-NOBRANCH

$$\frac{p(\sigma(\mathbf{pc})) \neq \mathbf{beqz}\ x, \ell \qquad \sigma \xrightarrow{\tau}_s \sigma' \qquad \mathit{enabled}'(s) \qquad s' = \begin{cases} \mathit{decr}'(s) & \text{if } p(\sigma(\mathbf{pc})) \neq \mathbf{spbarr} \\ \mathit{zeroes}'(s) & \text{otherwise} \end{cases}}{\langle \mathit{ctr}, \sigma, s\rangle \overset{\tau}{\Longrightarrow}_s \langle \mathit{ctr}, \sigma', s'\rangle}$$

SE-BRANCH-SYMB

$$\frac{p(\sigma(\mathbf{pc})) = \mathbf{beqz}\ x, \ell'' \qquad \mathit{enabled}'(s) \qquad \sigma \xrightarrow{\texttt{symPc}(se)\cdot\texttt{pc}\ \ell'}_s \sigma' \qquad \ell = \begin{cases} \sigma(\mathbf{pc})+1 & \text{if } \ell' \neq \sigma(\mathbf{pc})+1 \\ \ell'' & \text{if } \ell' = \sigma(\mathbf{pc})+1 \end{cases} \qquad s' = s \cdot \langle \sigma, \mathit{ctr}, \mathit{min}(w, \mathit{wndw}(s)-1), \ell\rangle \qquad \mathit{id} = \mathit{ctr}}{\langle \mathit{ctr}, \sigma, s\rangle \xRightarrow{\texttt{symPc}(se)\cdot\texttt{start}\ \mathit{id}\cdot\texttt{pc}\ \ell}_s \langle \mathit{ctr}+1, \sigma[\mathbf{pc} \mapsto \ell], s'\rangle}$$

SE-ROLLBACK

$$\frac{\sigma' \xrightarrow{\tau}_s \sigma''}{\langle \mathit{ctr}, \sigma, s \cdot \langle \sigma', \mathit{id}, 0, \ell\rangle\rangle \xRightarrow{\texttt{rollback}\ \mathit{id}\cdot\texttt{pc}\ \sigma''(\mathbf{pc})}_s \langle \mathit{ctr}, \sigma'', s\rangle}$$

Always-mispredict leaks maximally

Speculative semantics
+
Prediction oracle

→

Always-mispredict speculative semantics

For all program states s and s':

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

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

Recap: Speculative non-interference

Program $\texttt{P}$ is **speculatively non-interferent** if

For all program states s and s':

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

Speculative non-interference: Example

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

Speculative non-interference: Example

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

Speculative non-interference: Example

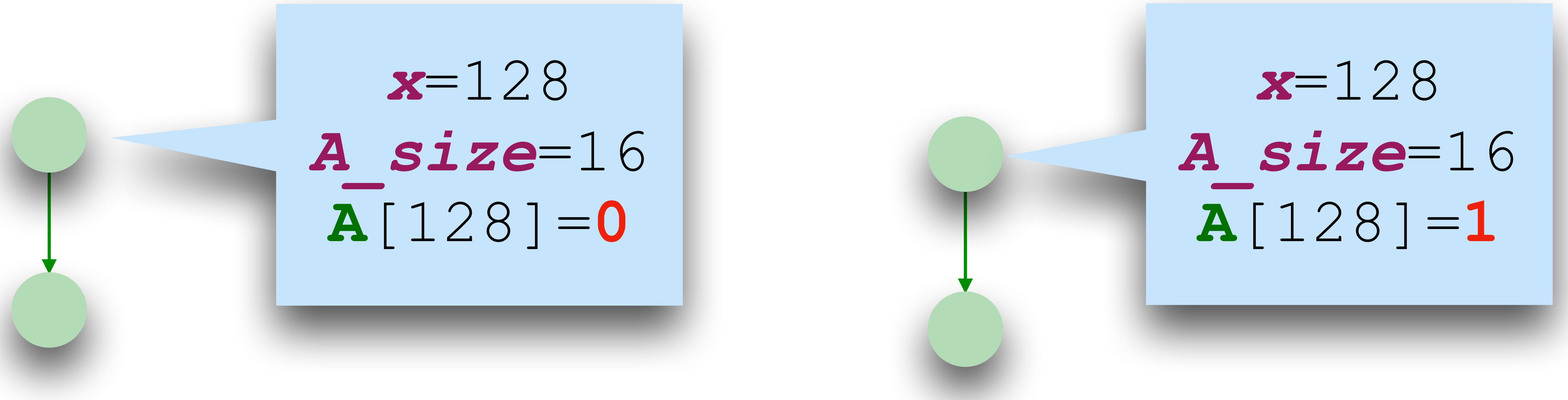
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```

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

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

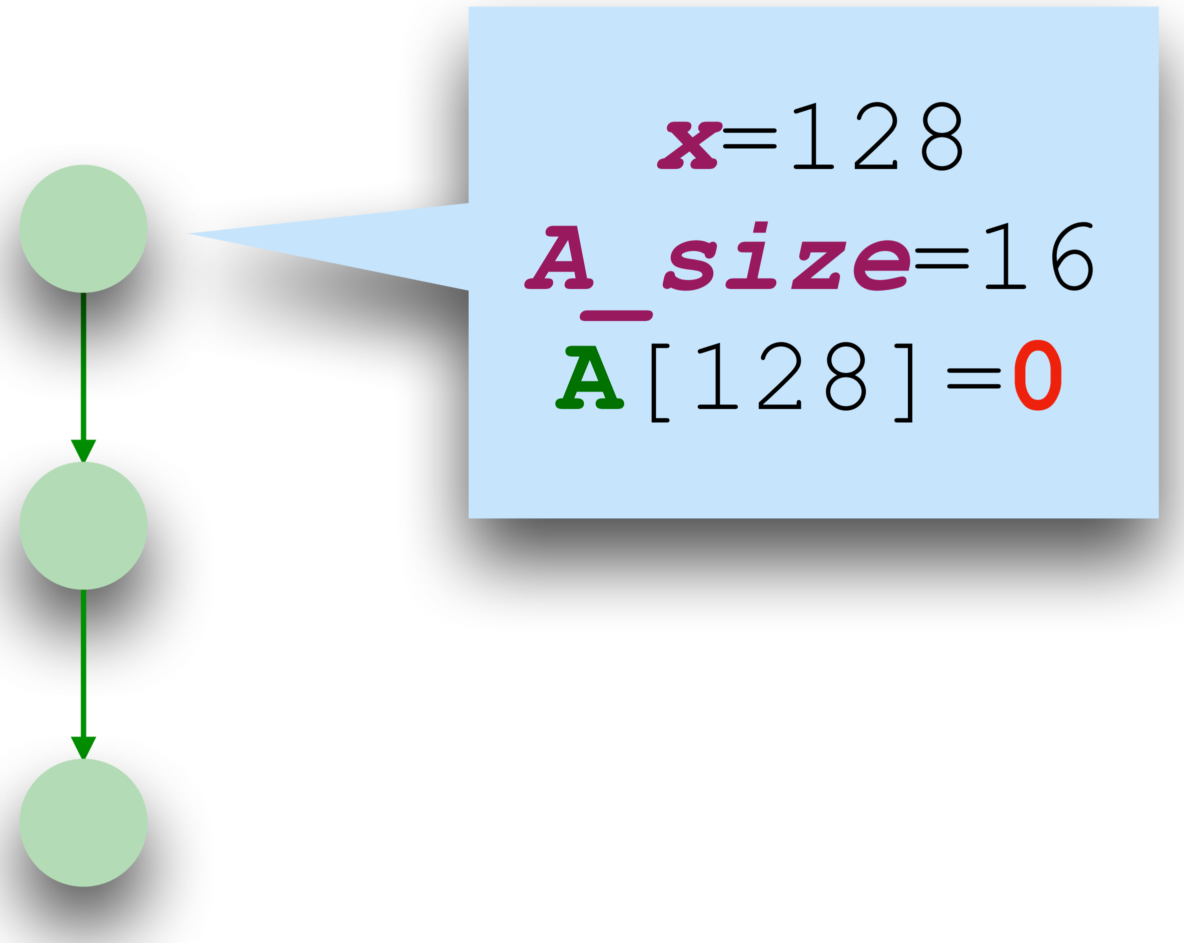
Speculative non-interference: Example

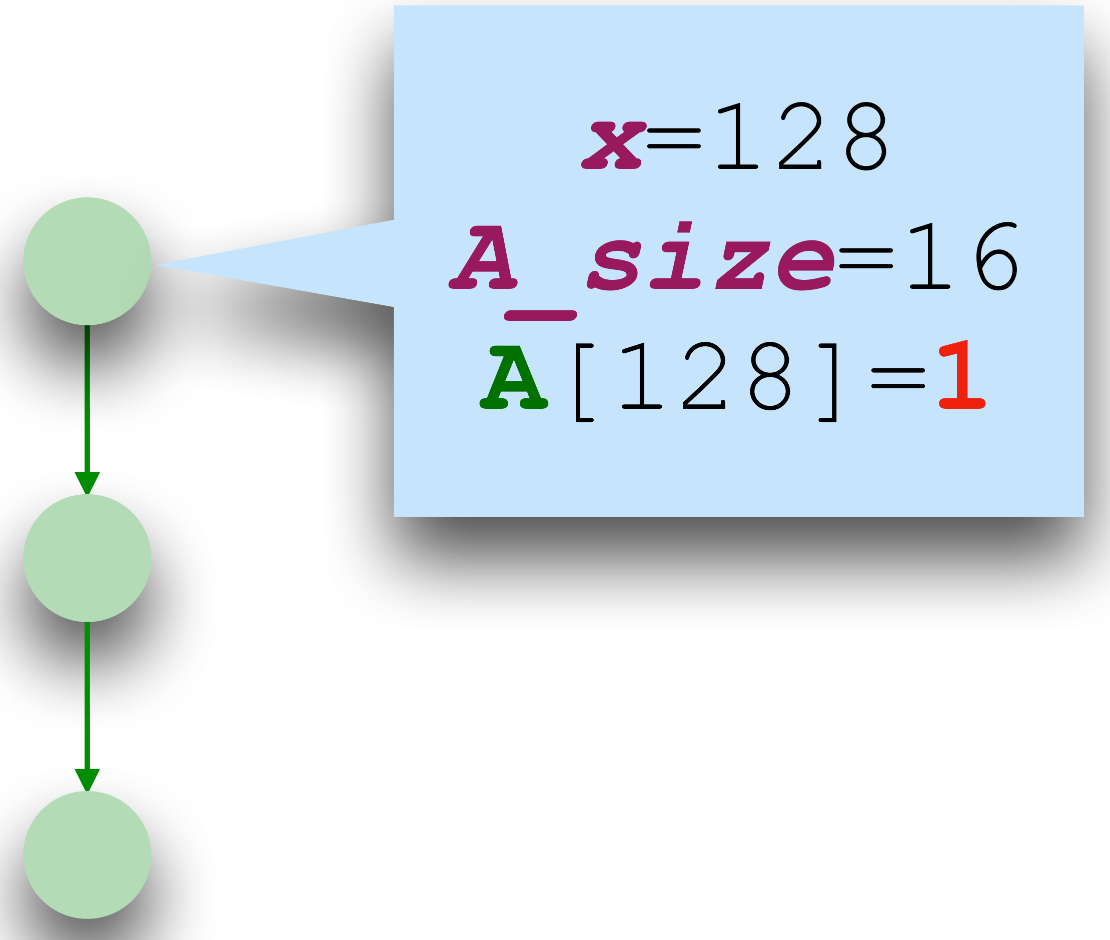
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Speculative non-interference: Example

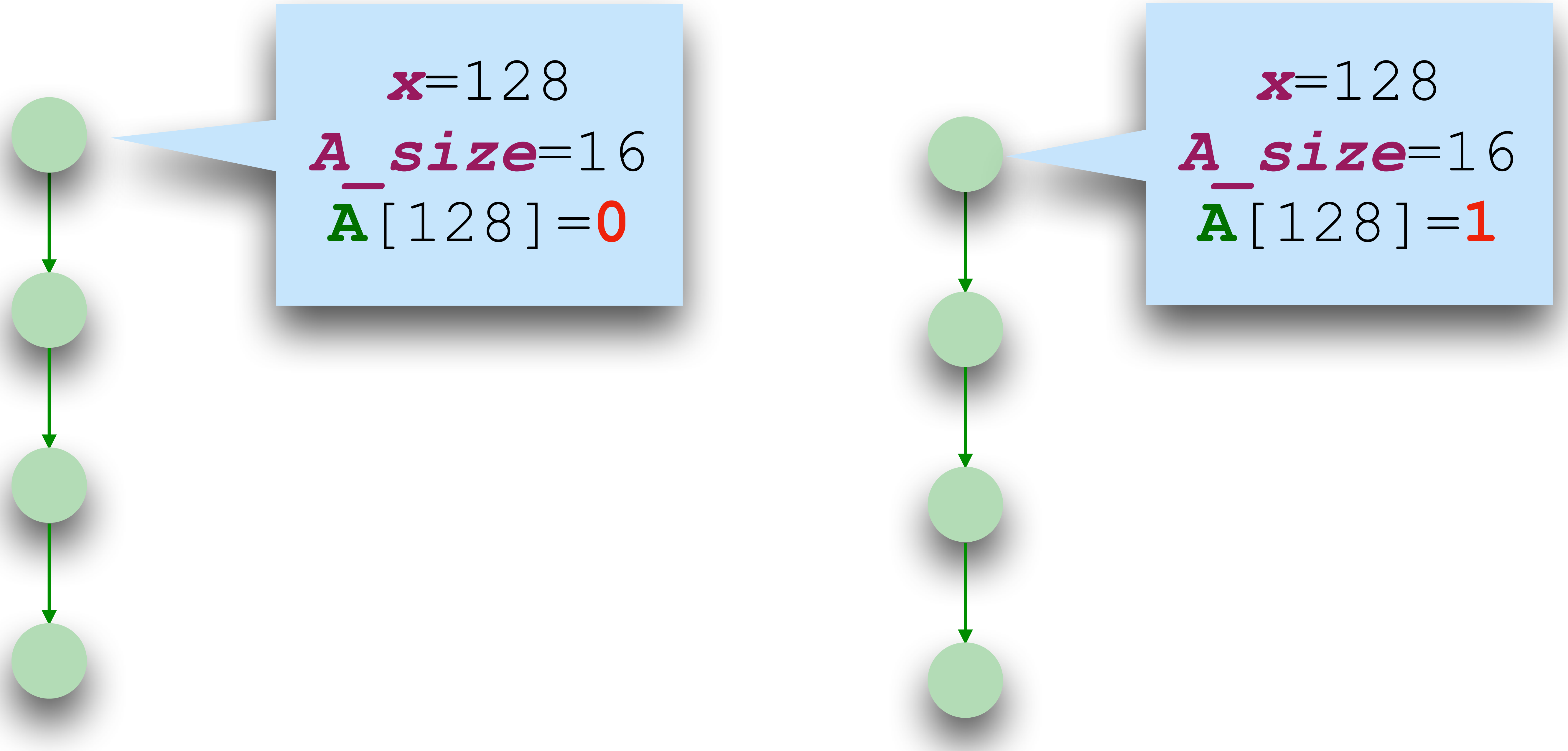
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```





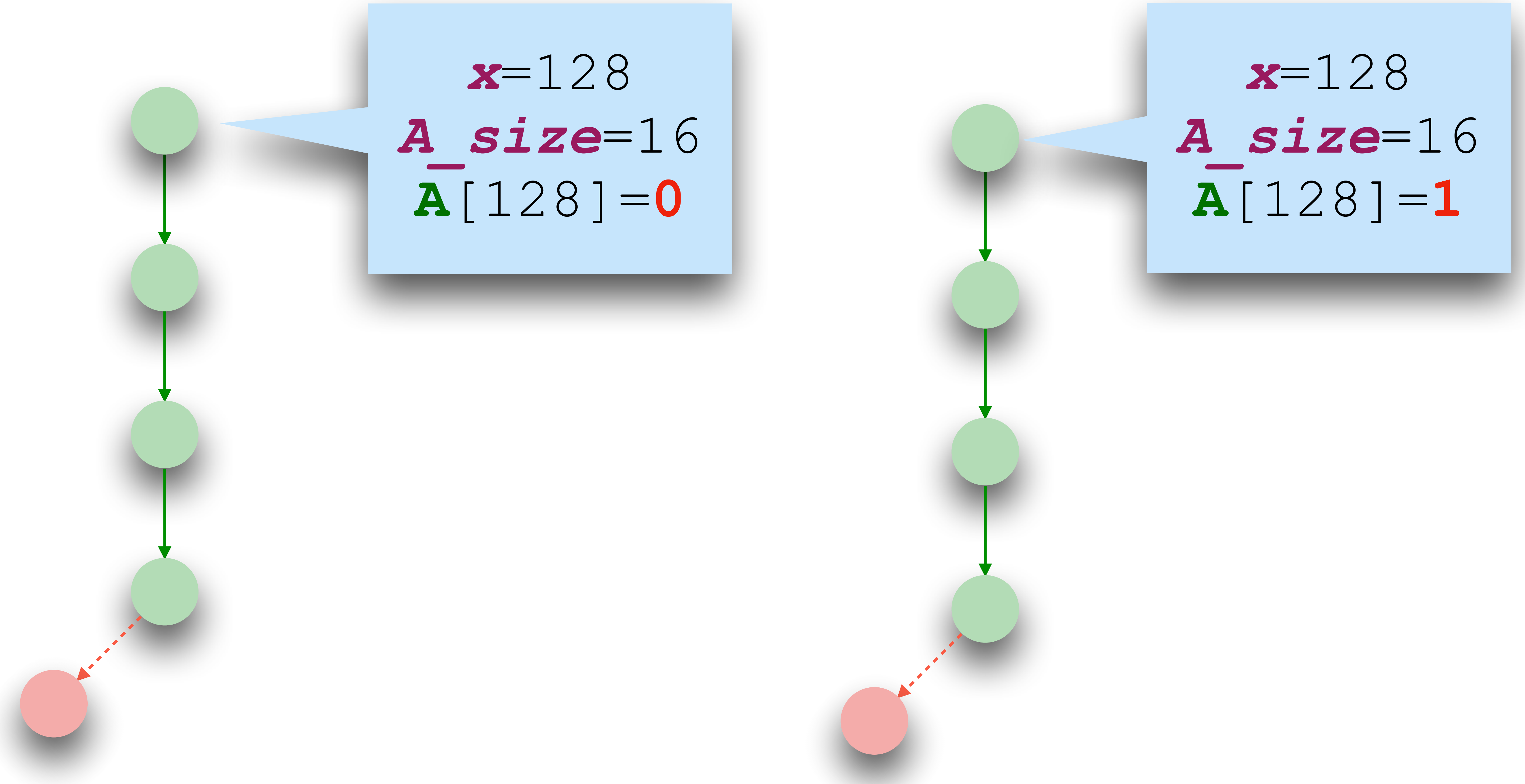
Speculative non-interference: Example

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



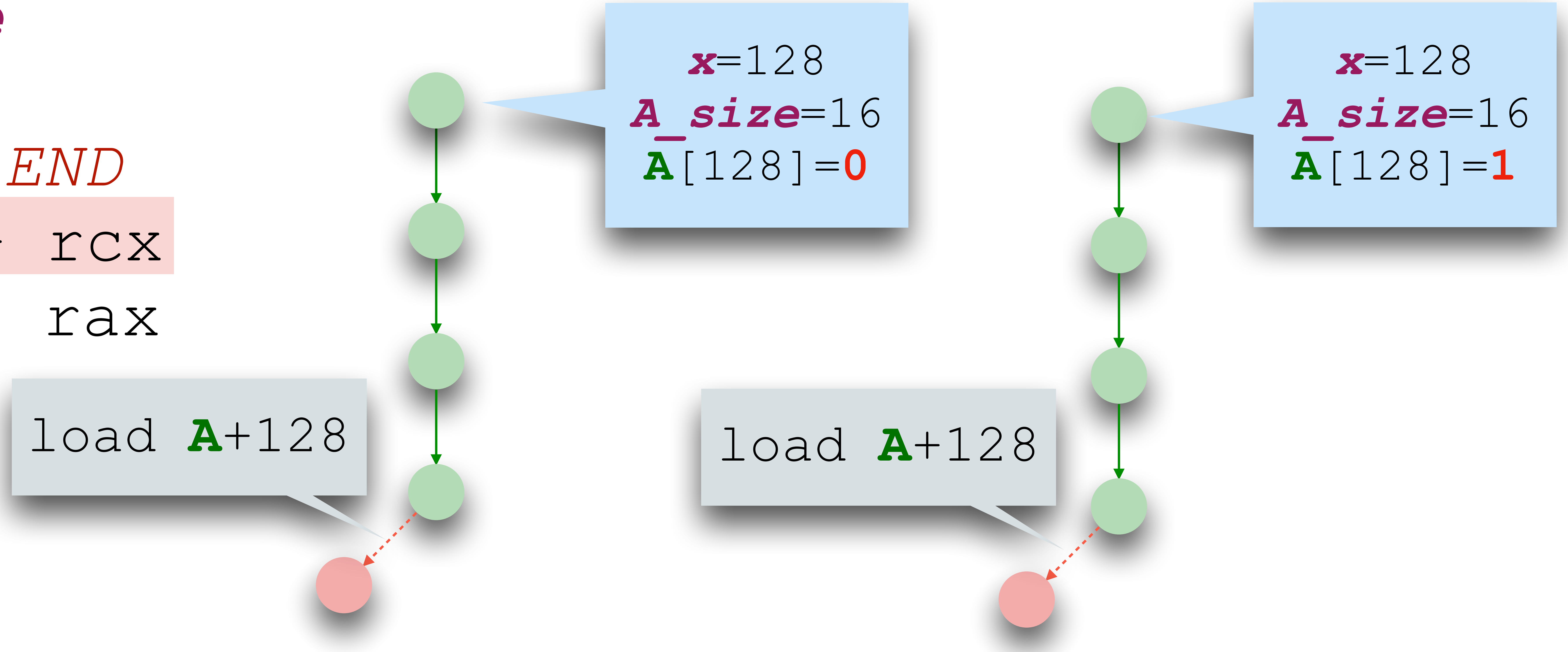
Speculative non-interference: Example

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



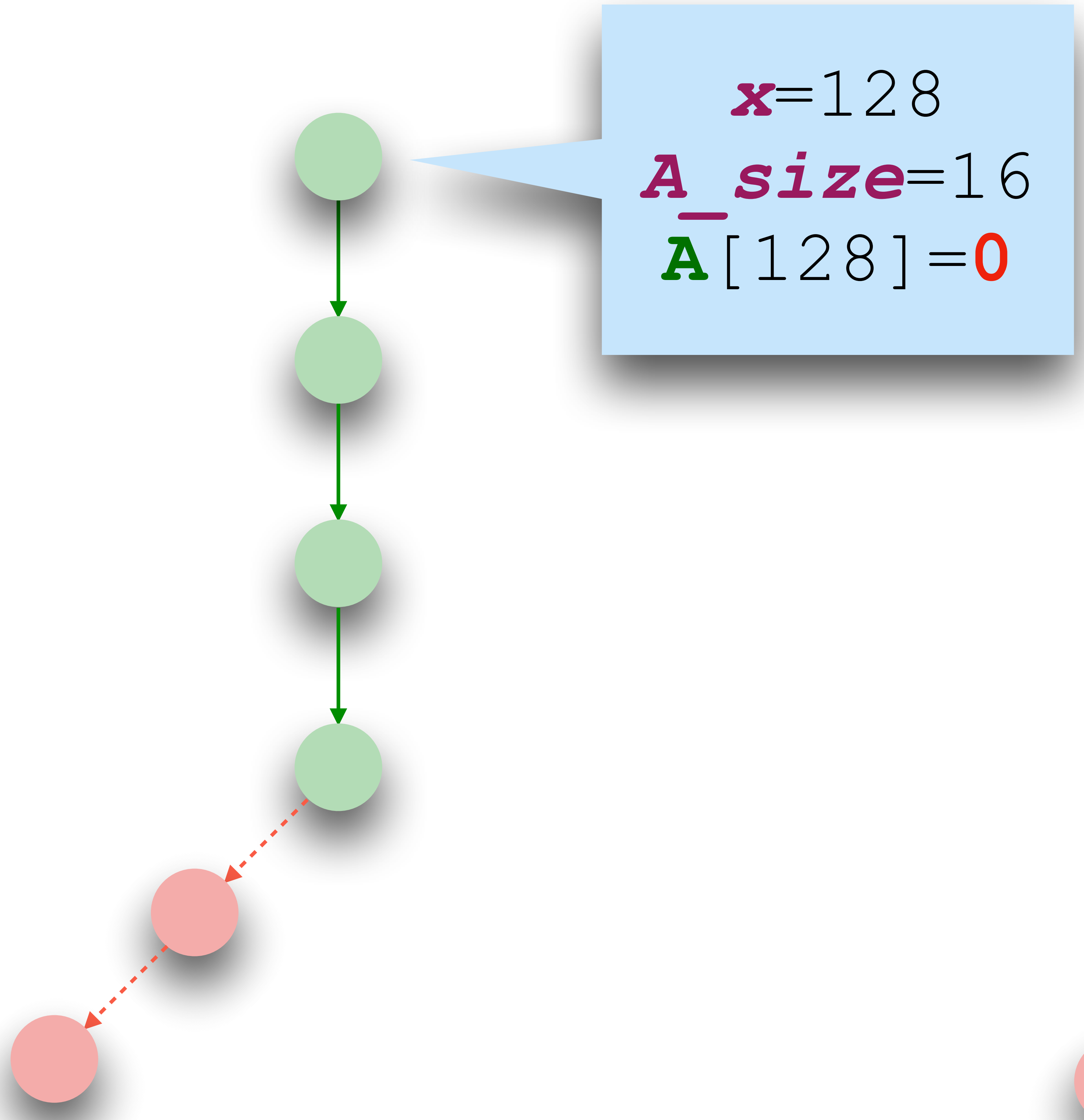
Speculative non-interference: Example

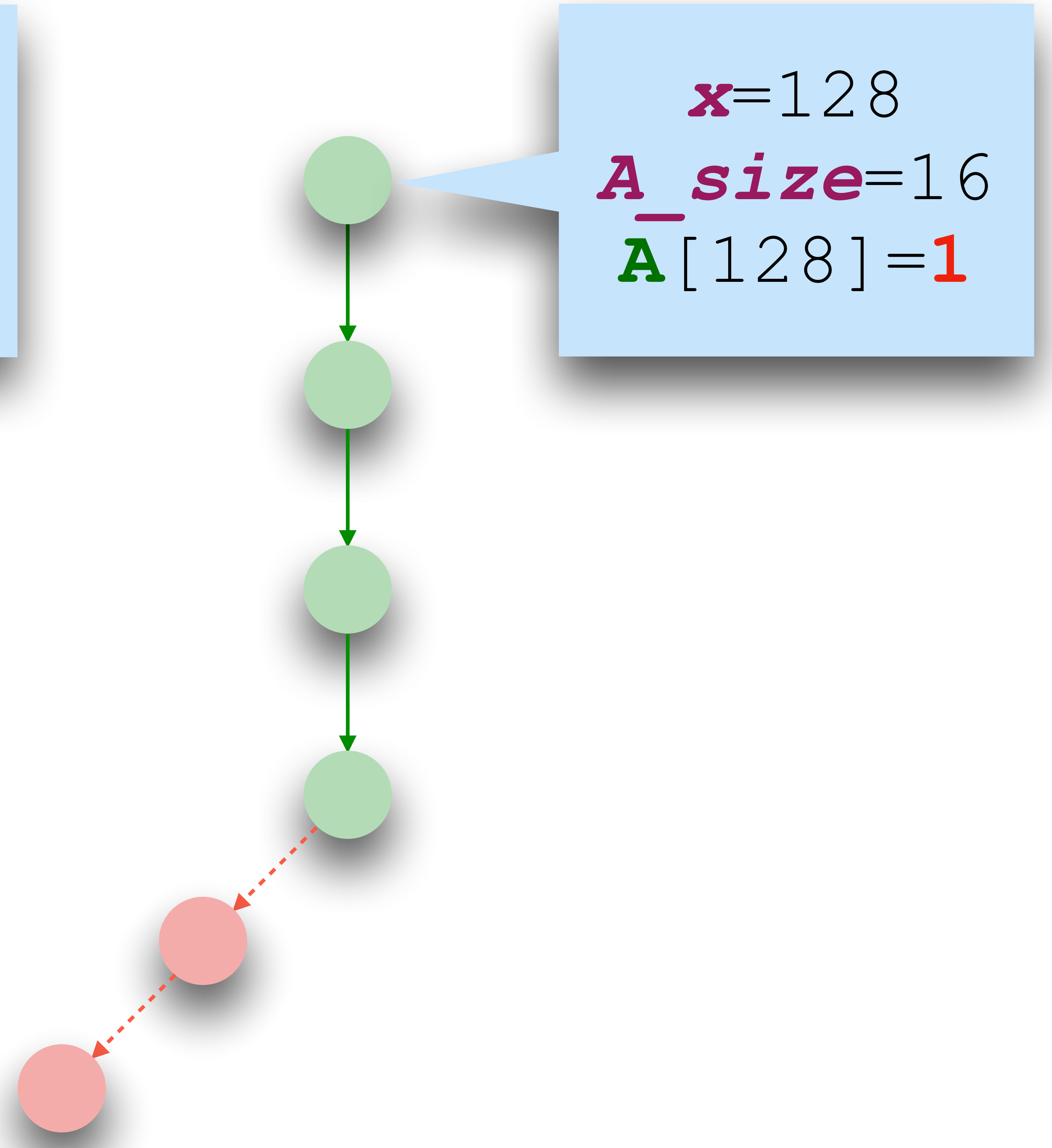
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Speculative non-interference: Example

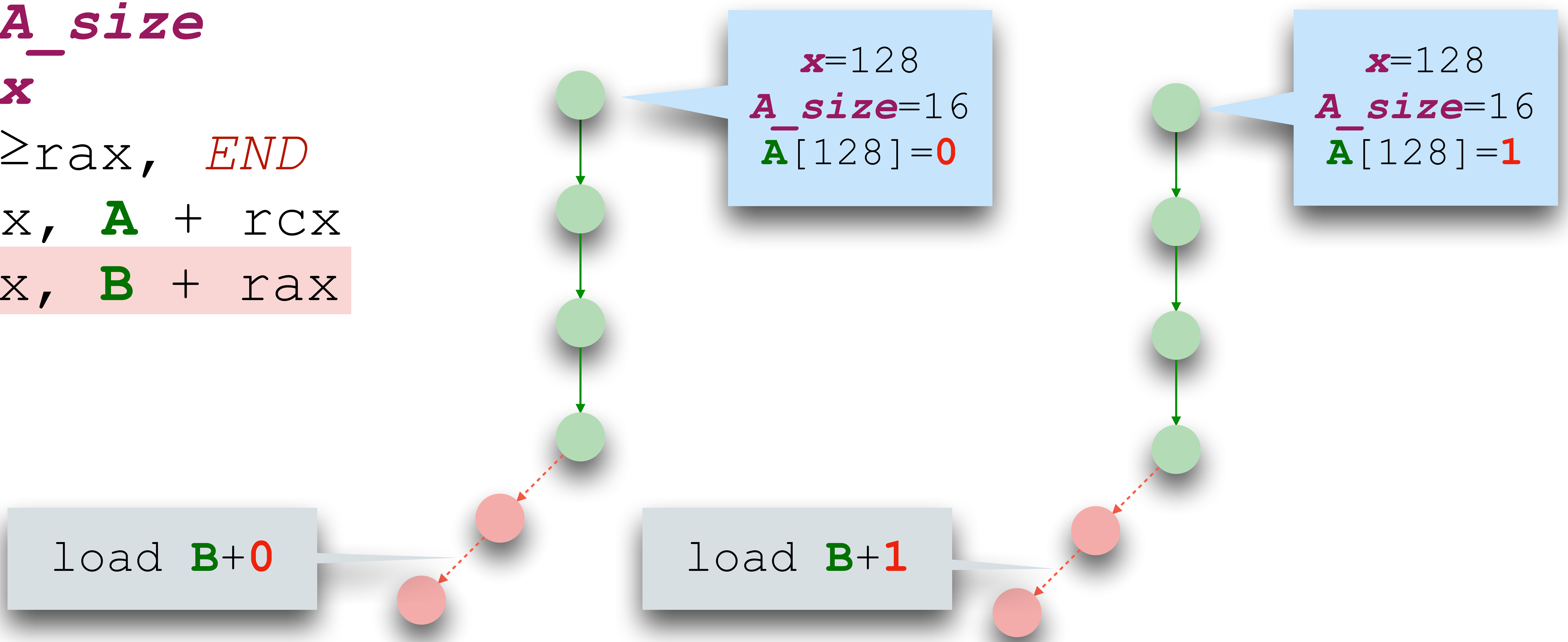
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```





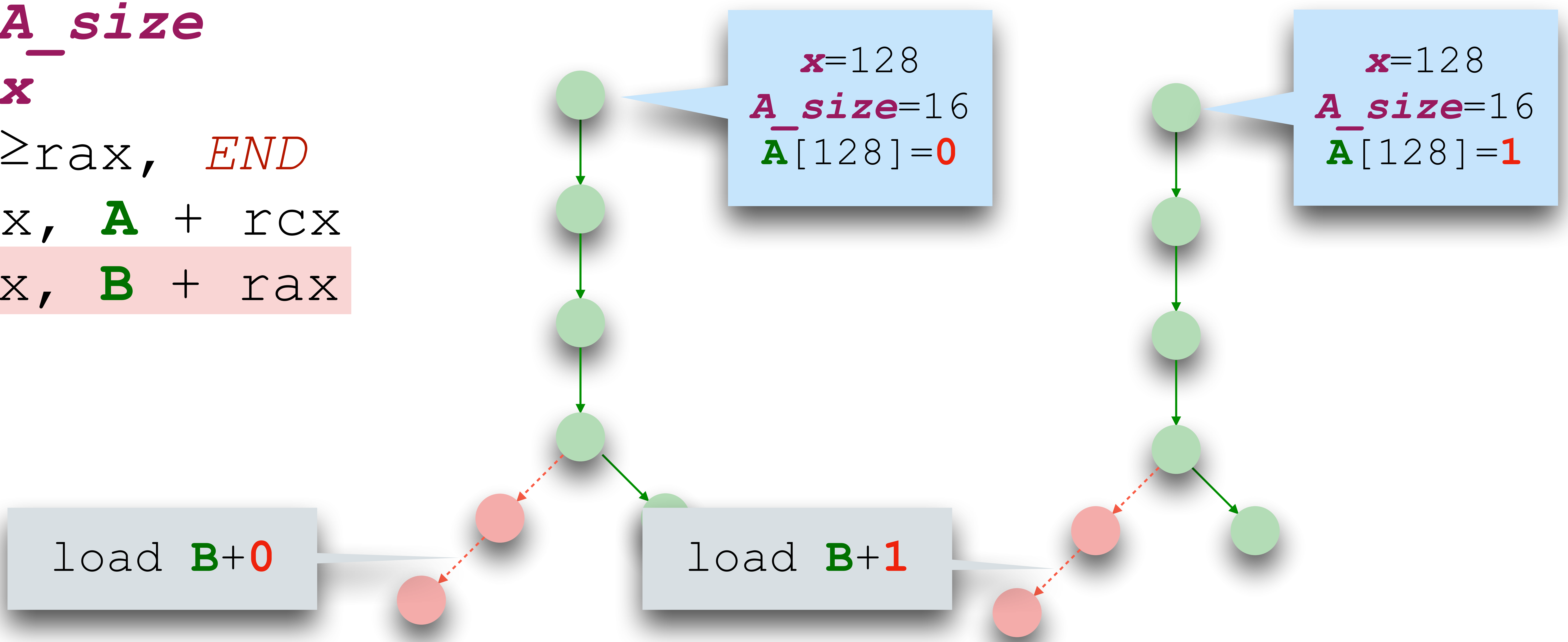
Speculative non-interference: Example

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



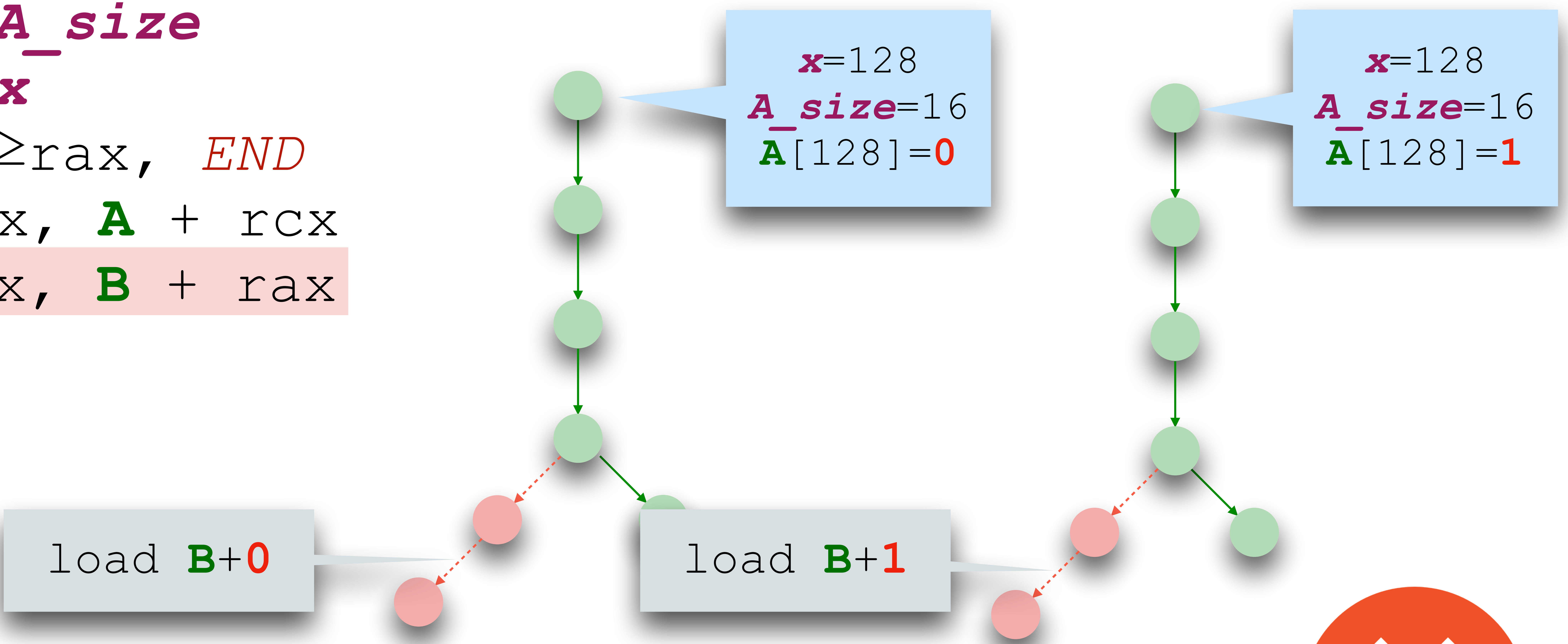
Speculative non-interference: Example

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



Speculative non-interference: Example

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



3. Spectector: Detecting speculative leaks

Spectector: Detecting speculative leaks

Spectector: Detecting speculative leaks

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

Spectector: Detecting speculative leaks

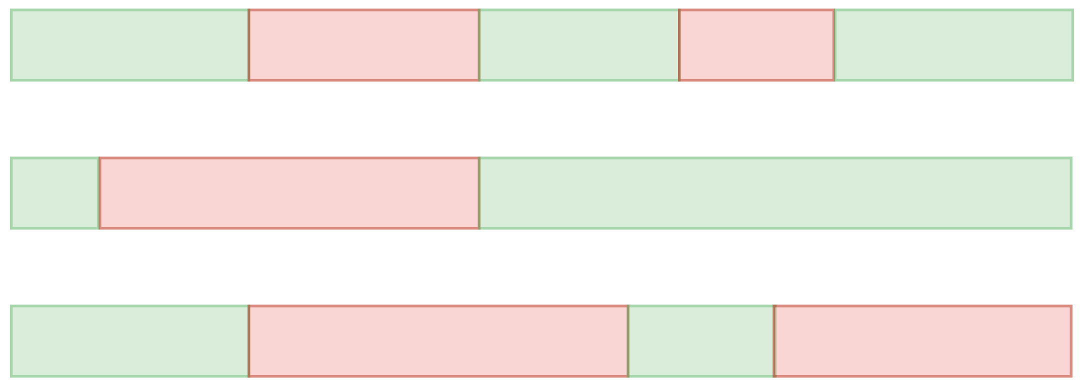
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```

Symbolic execution

Spectector: Detecting speculative leaks

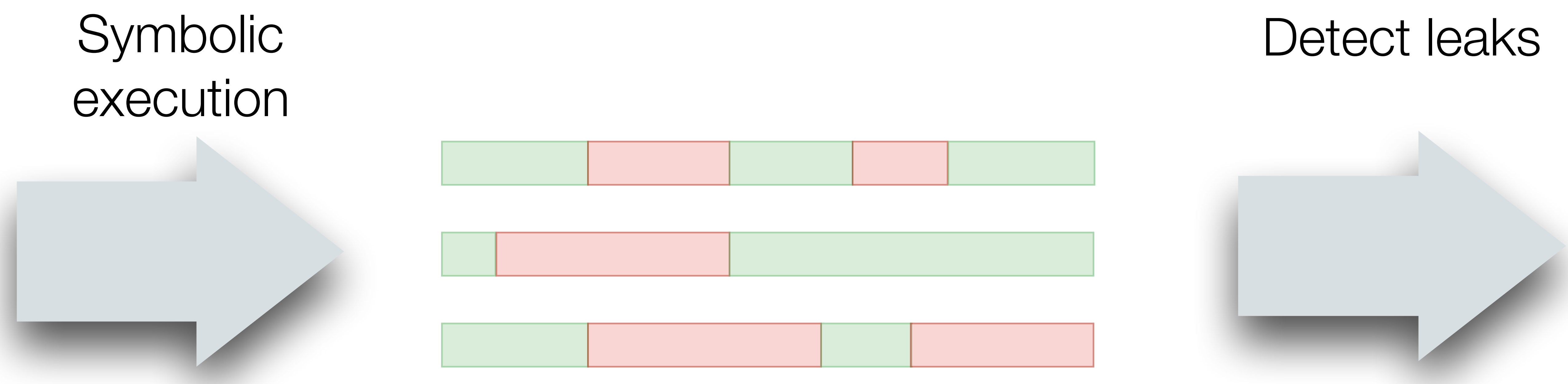
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```

Symbolic execution



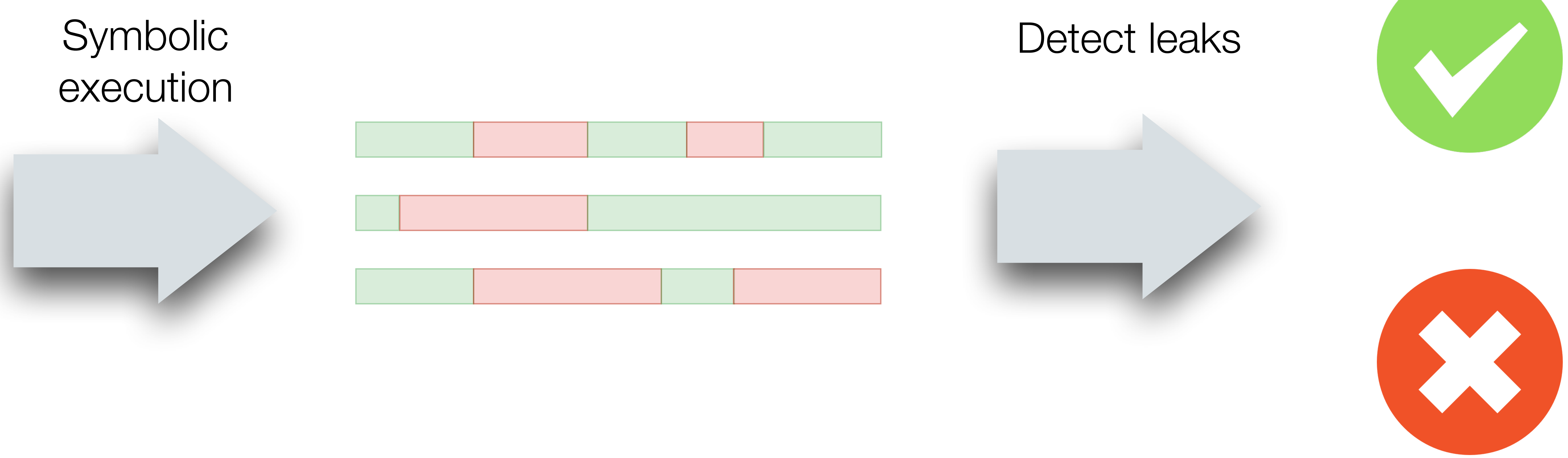
Spectector: Detecting speculative leaks

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



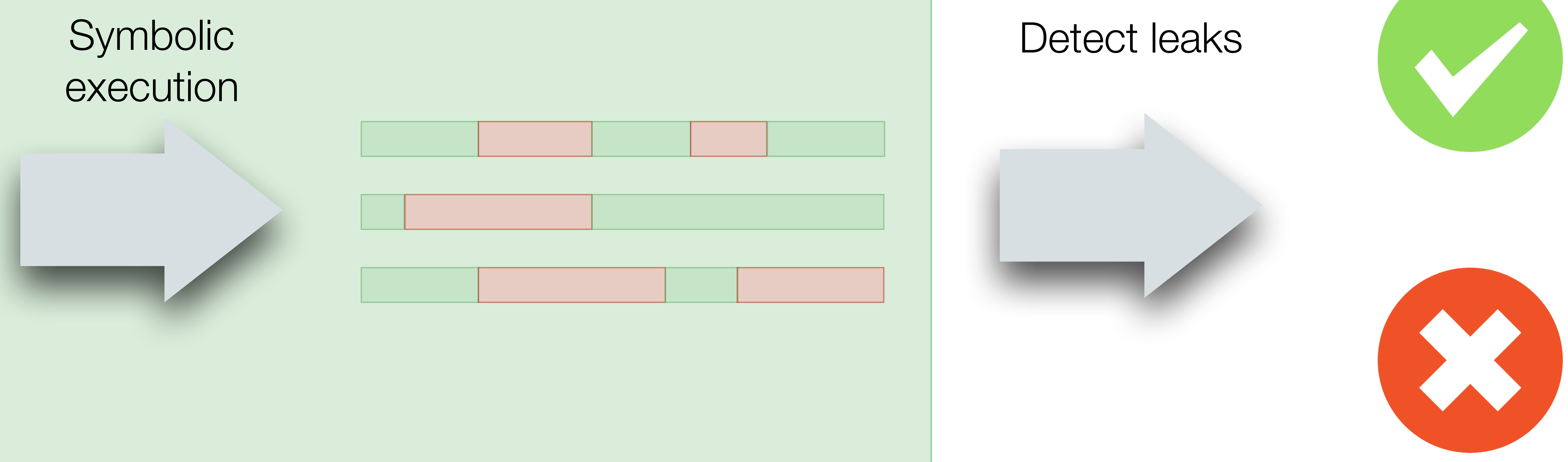
Spectector: Detecting speculative leaks

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



Spectector: Detecting speculative leaks

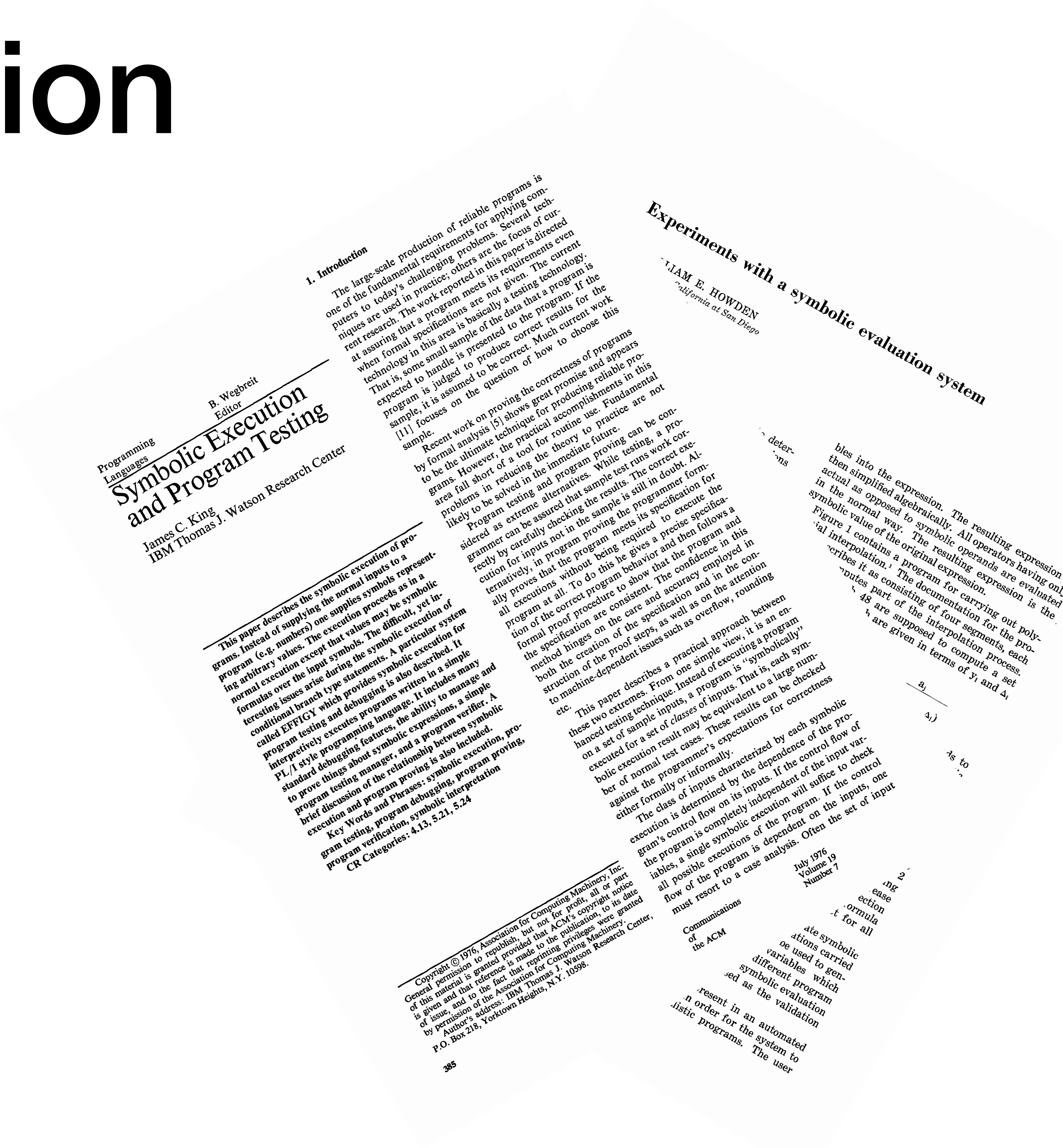
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```



Symbolic execution

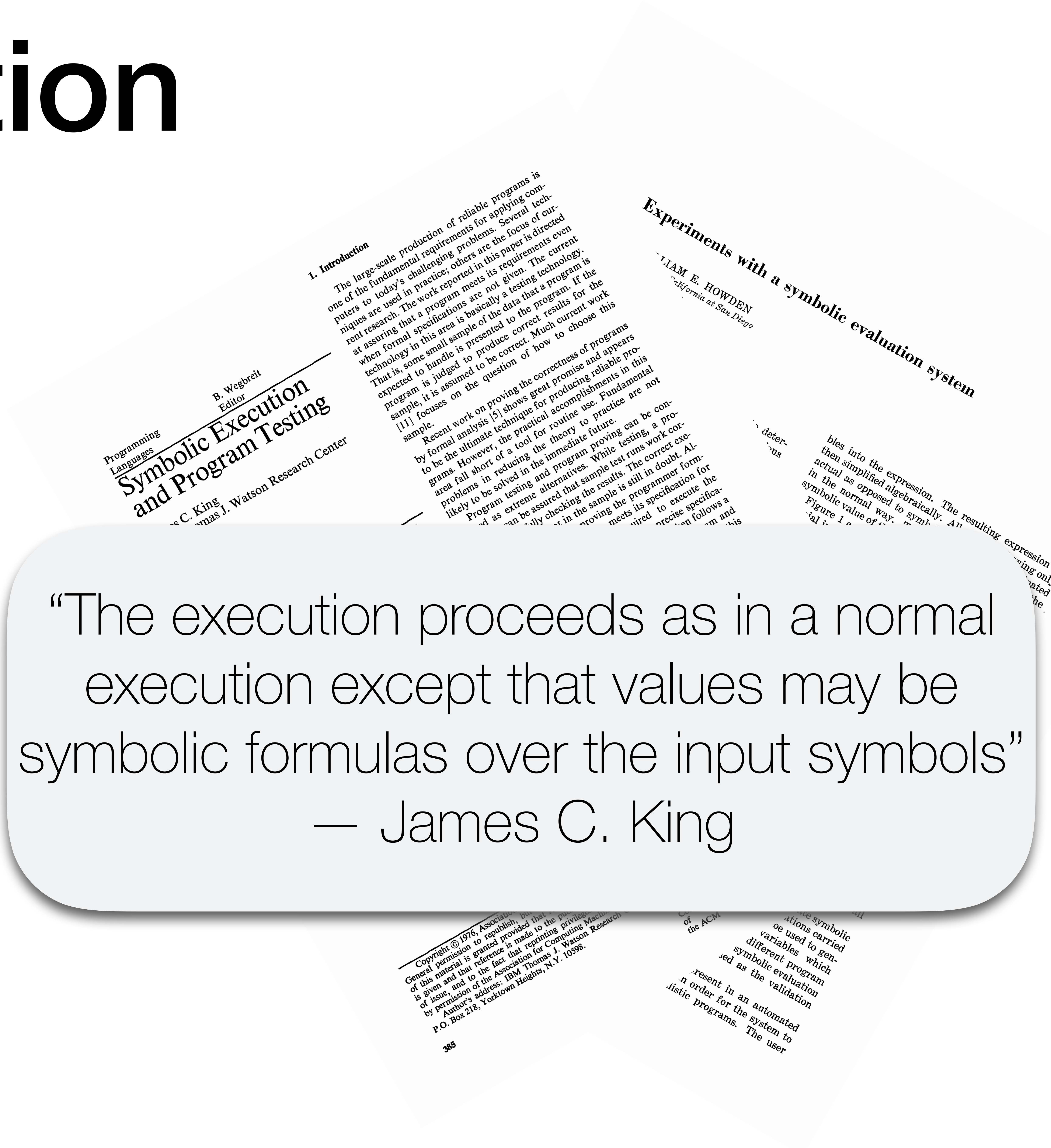
Symbolic execution

- Program analysis technique



Symbolic execution

- Program analysis technique

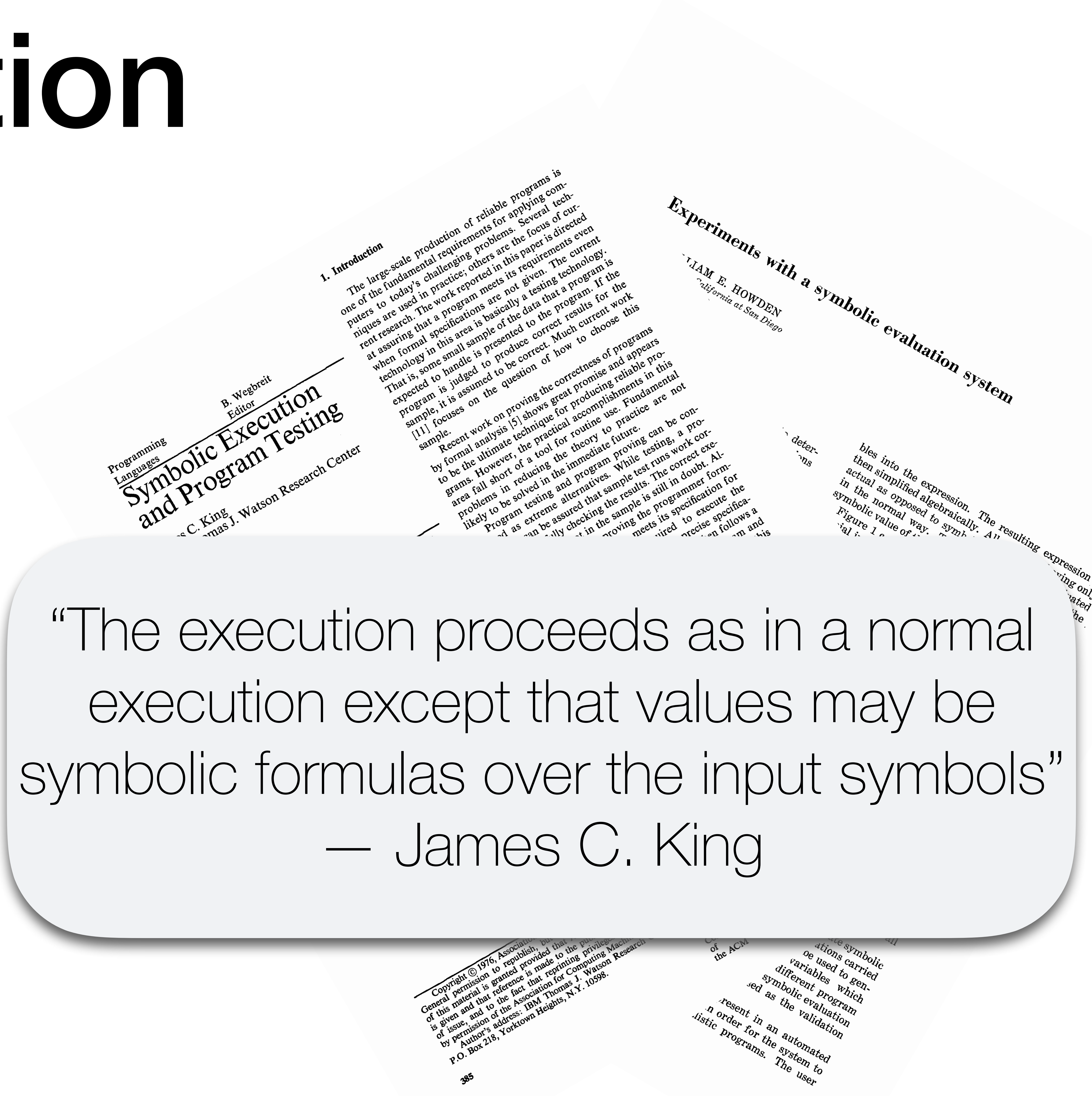


“The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols”
— James C. King

Symbolic execution

- Program analysis technique

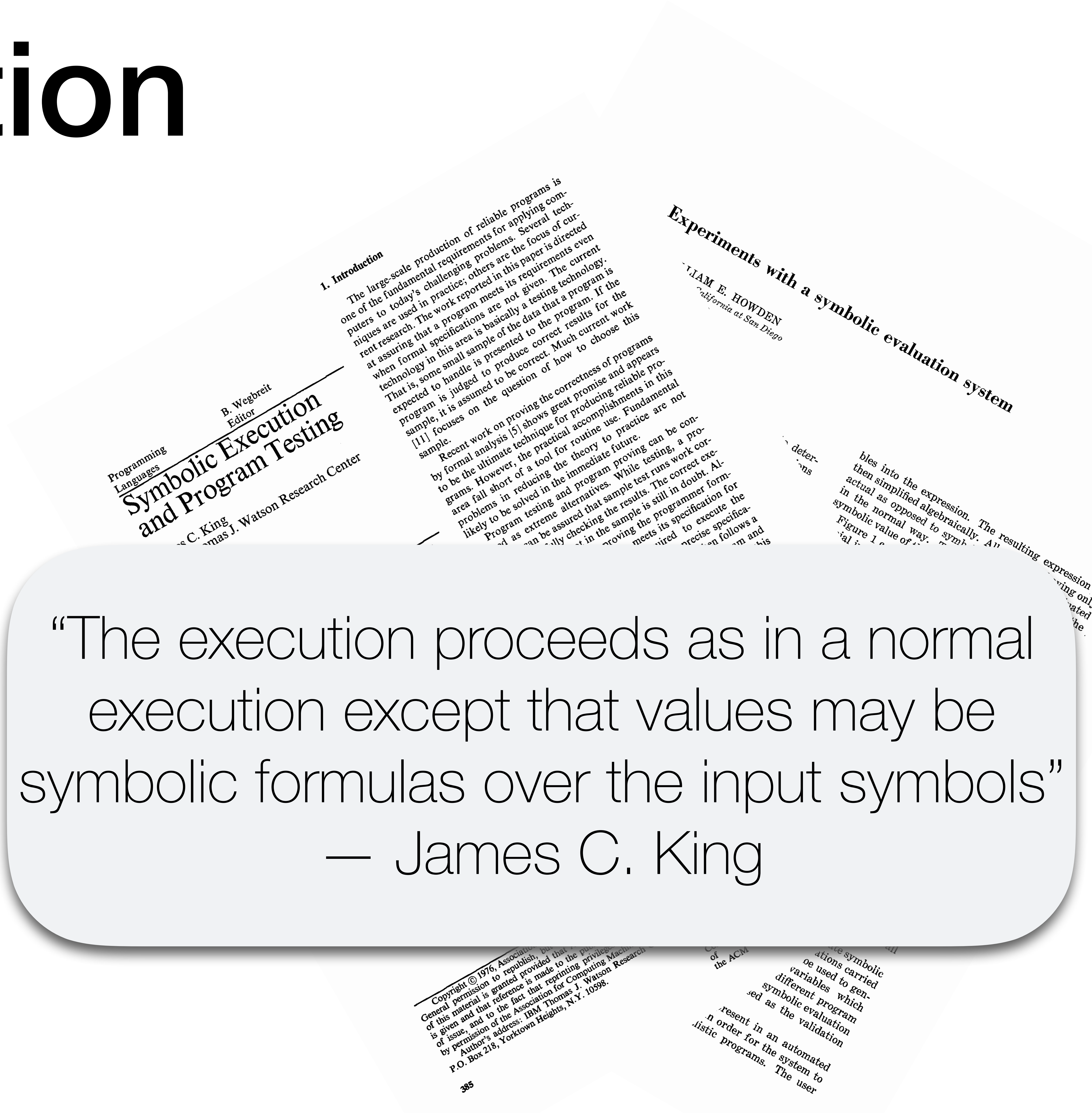
- Execute programs over symbolic values



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

Symbolic execution

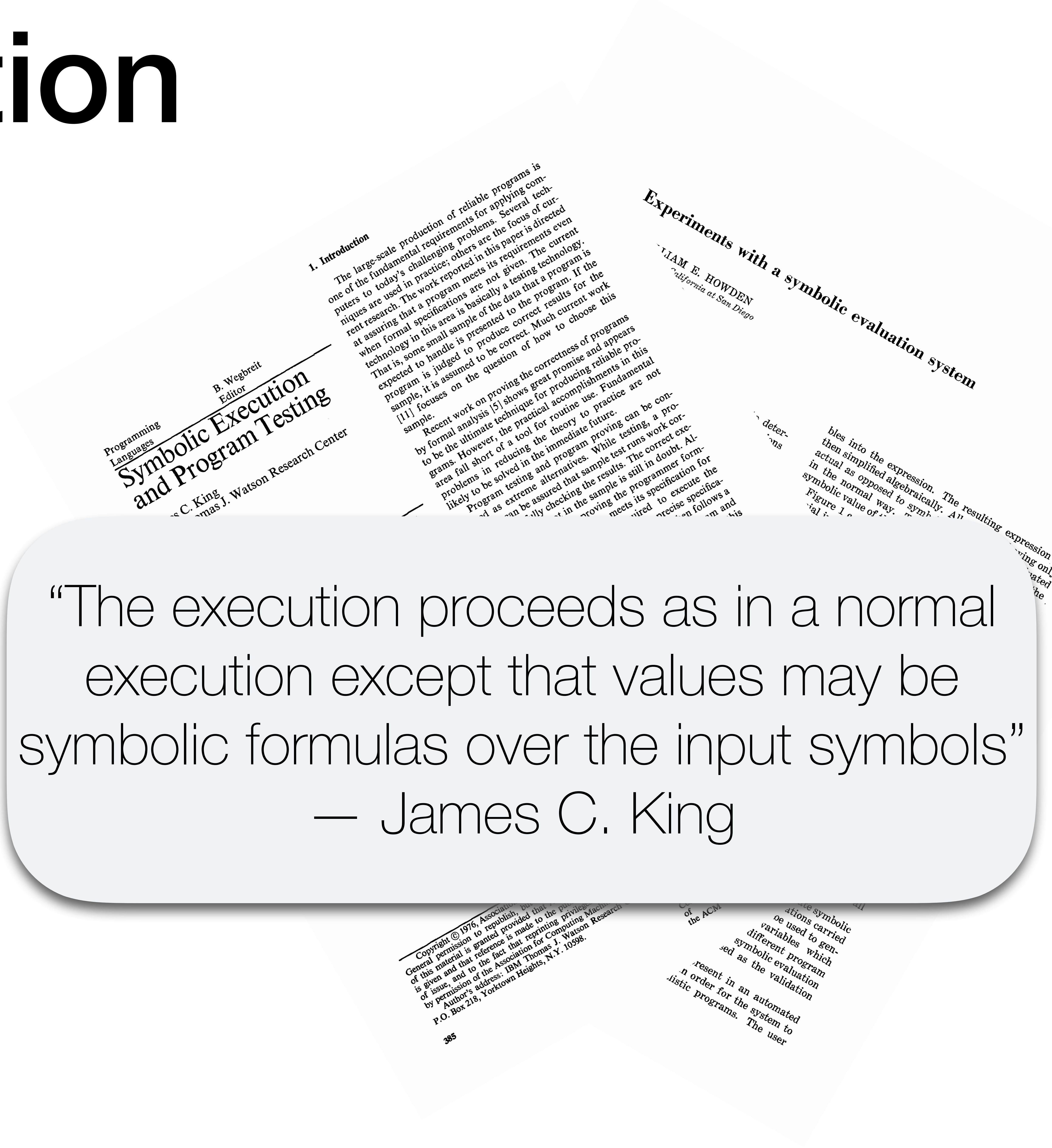
- Program analysis technique
- Execute programs over symbolic values
  - Explore all paths,
    each with its own path constraint



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

Symbolic execution

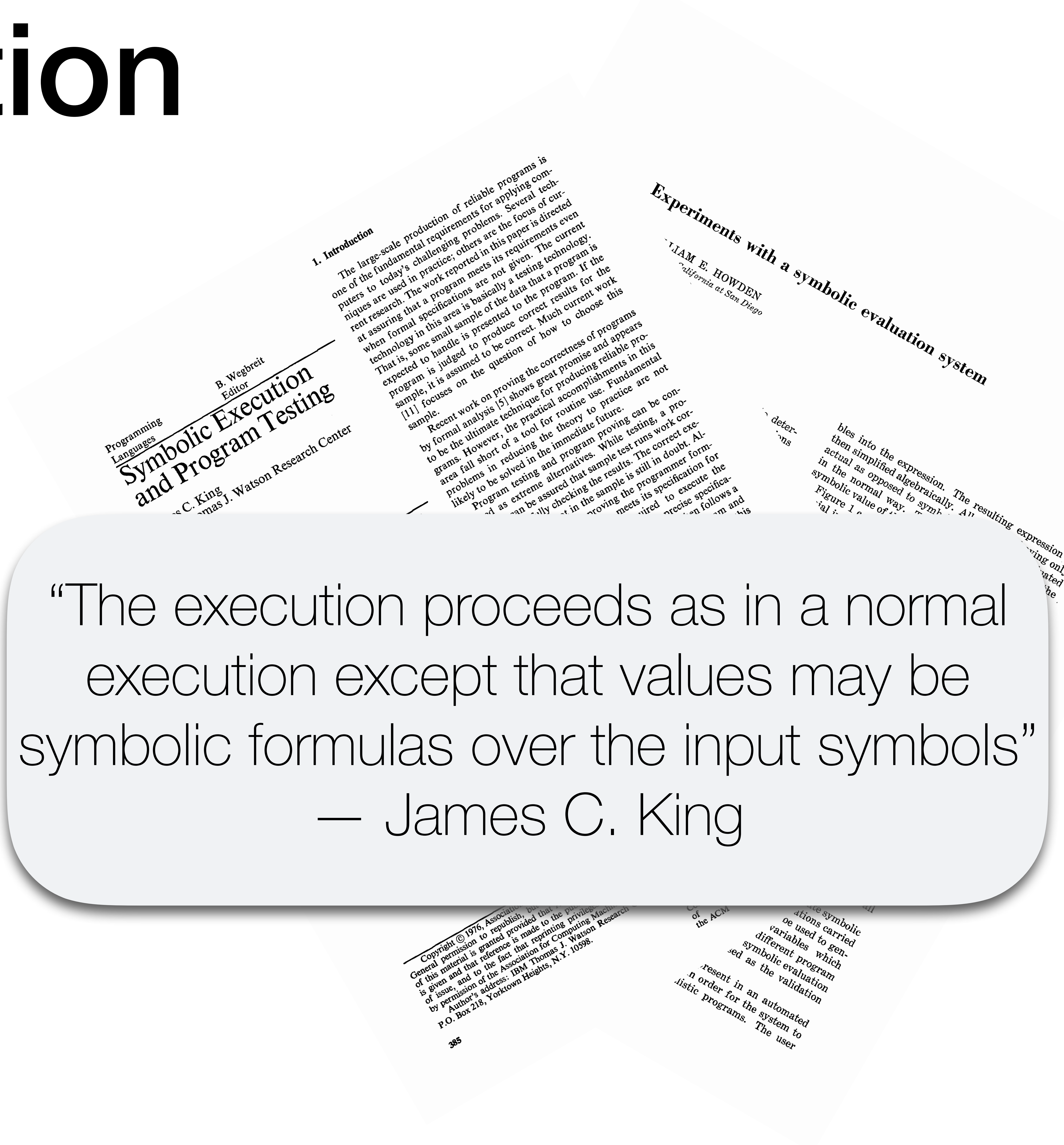
- Program analysis technique
- Execute programs over symbolic values
  - Explore all paths, each with its own path constraint
  - Each path represents all possible executions satisfying the constraints



"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 possible executions satisfying the constraints
    - Branch and jump instructions: fork paths and update path constraint



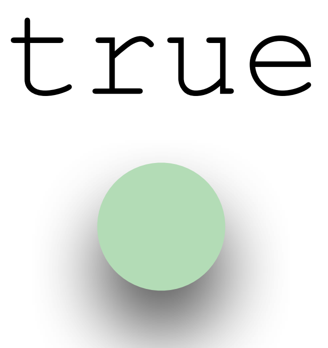
"The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols"
— James C. King

Symbolic execution

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

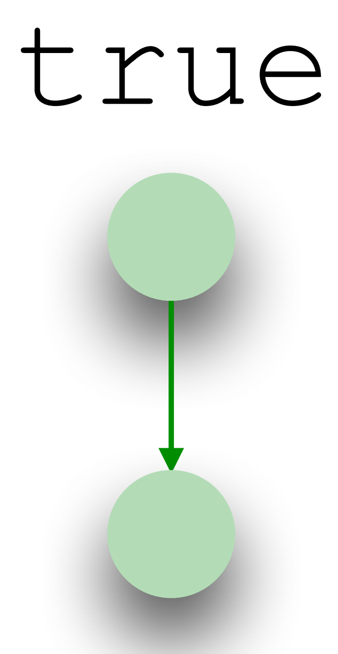
Symbolic execution

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



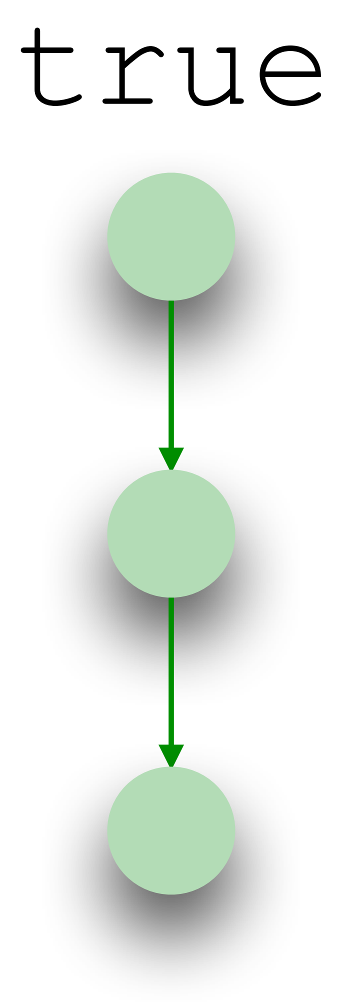
Symbolic execution

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



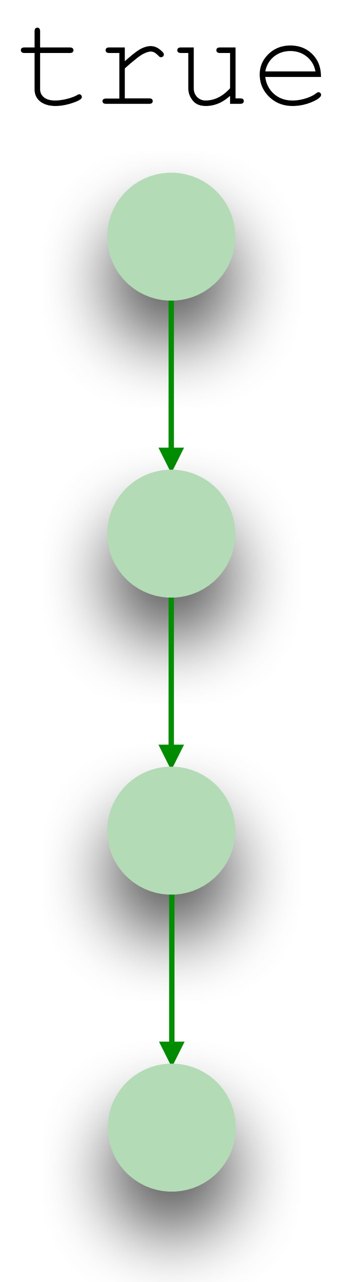
Symbolic execution

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



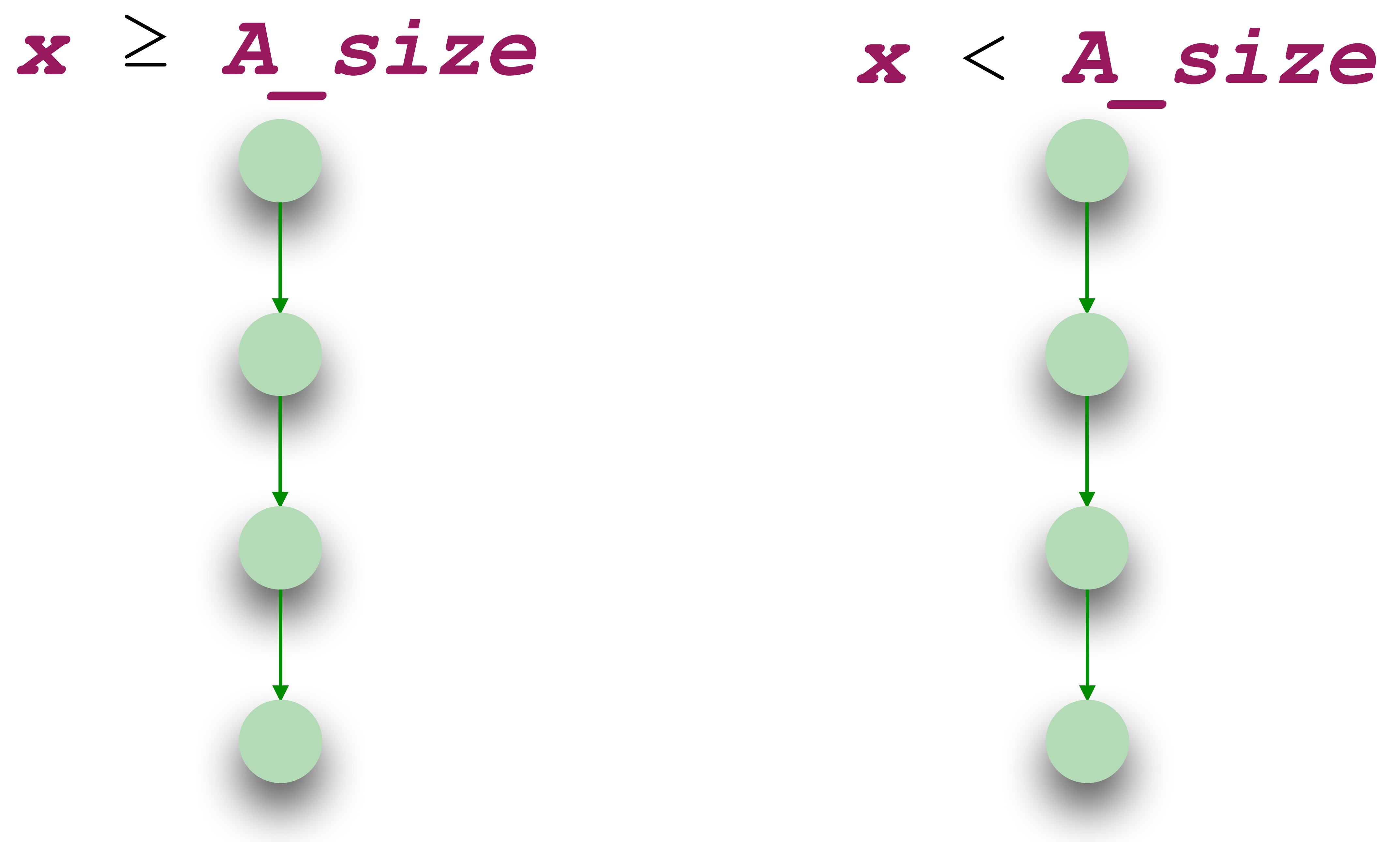
Symbolic execution

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



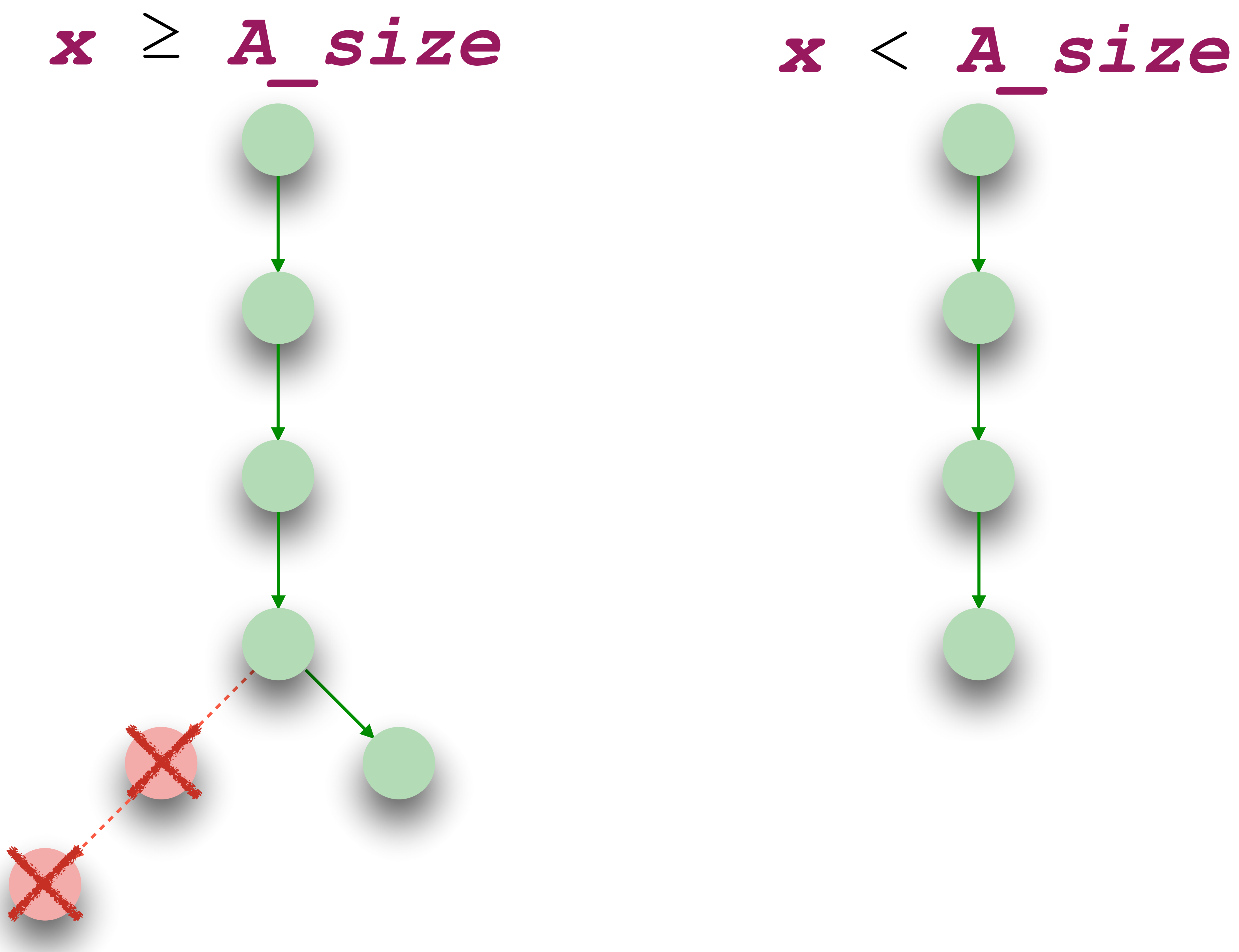
Symbolic execution

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



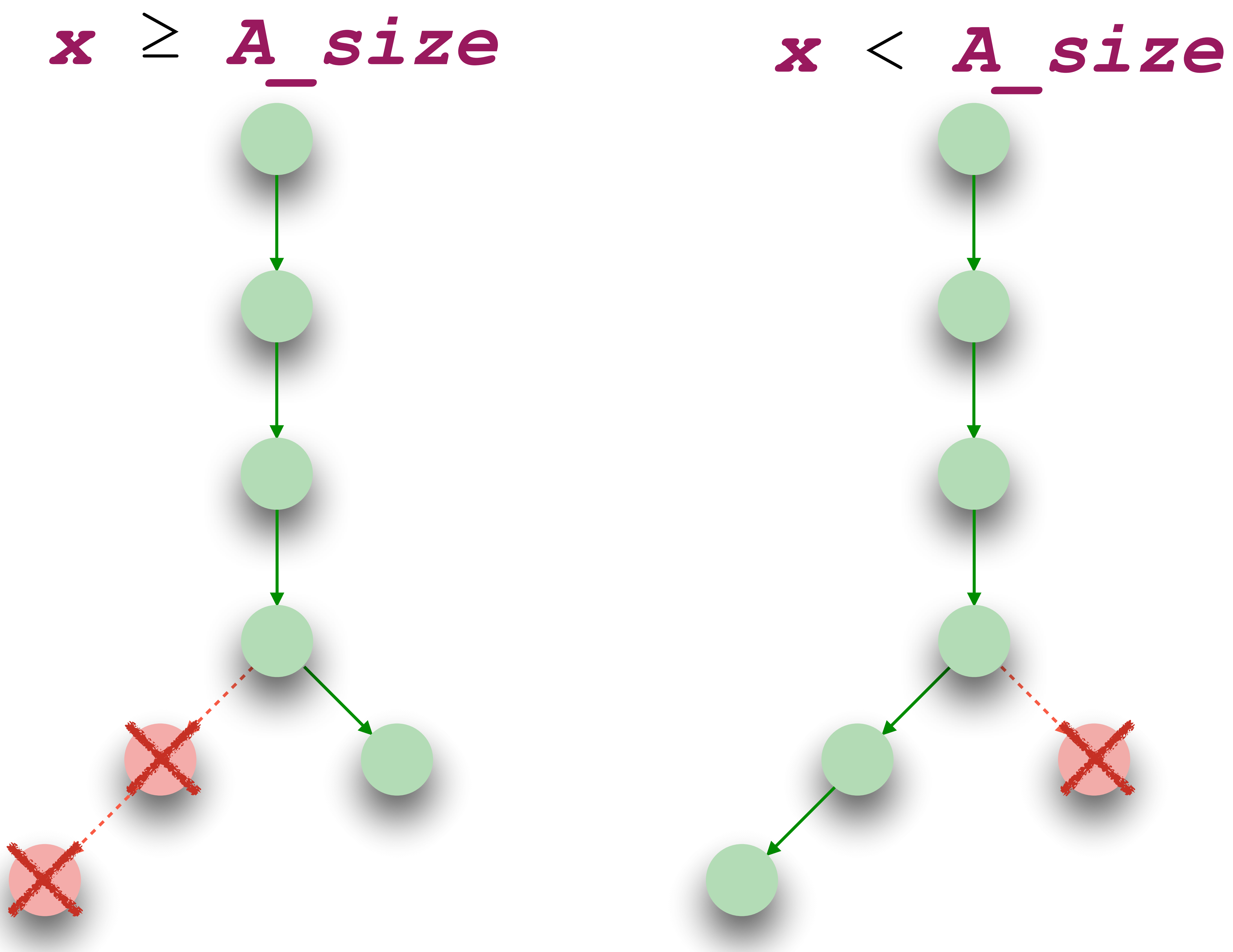
Symbolic execution

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



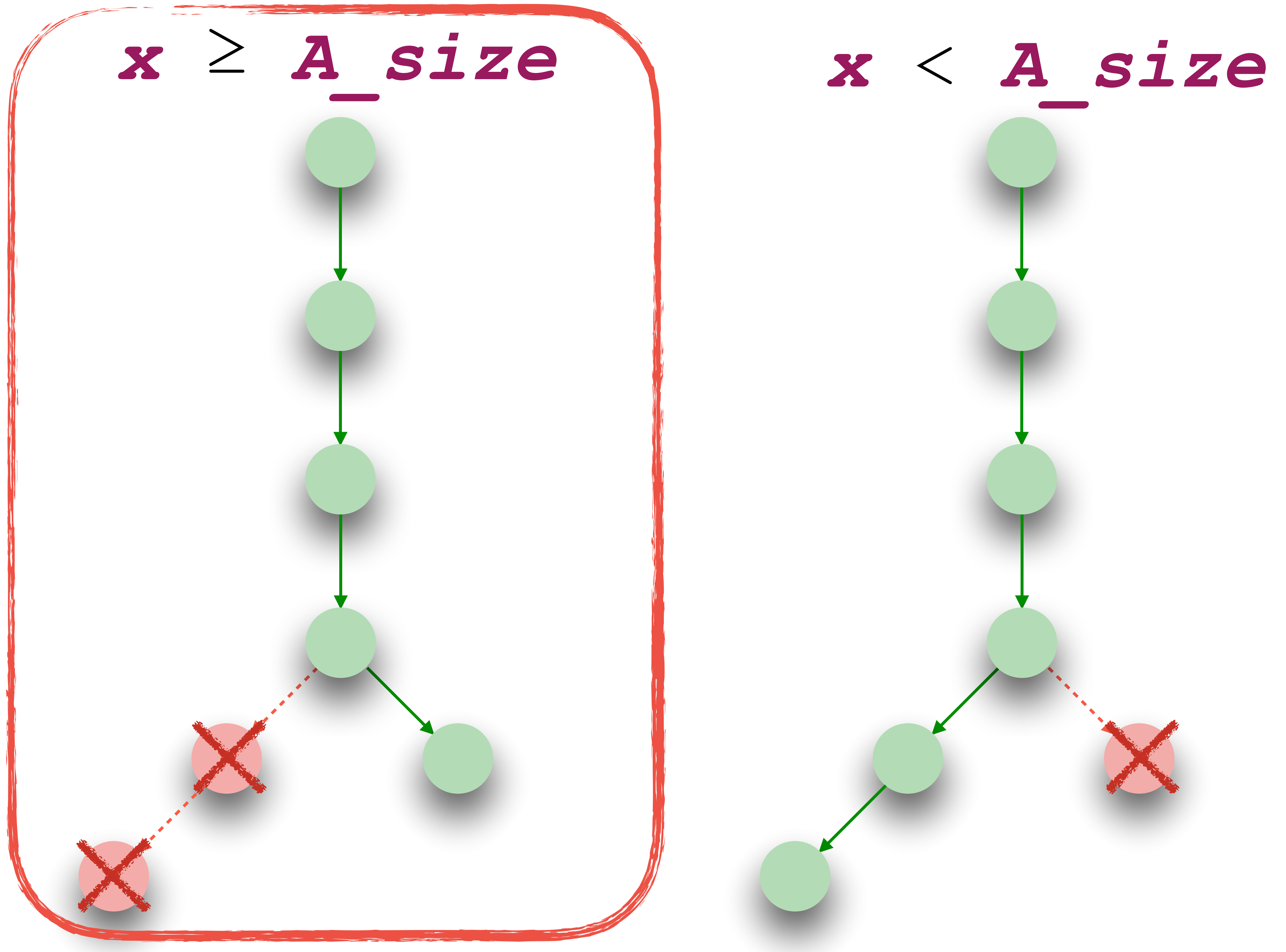
Symbolic execution

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



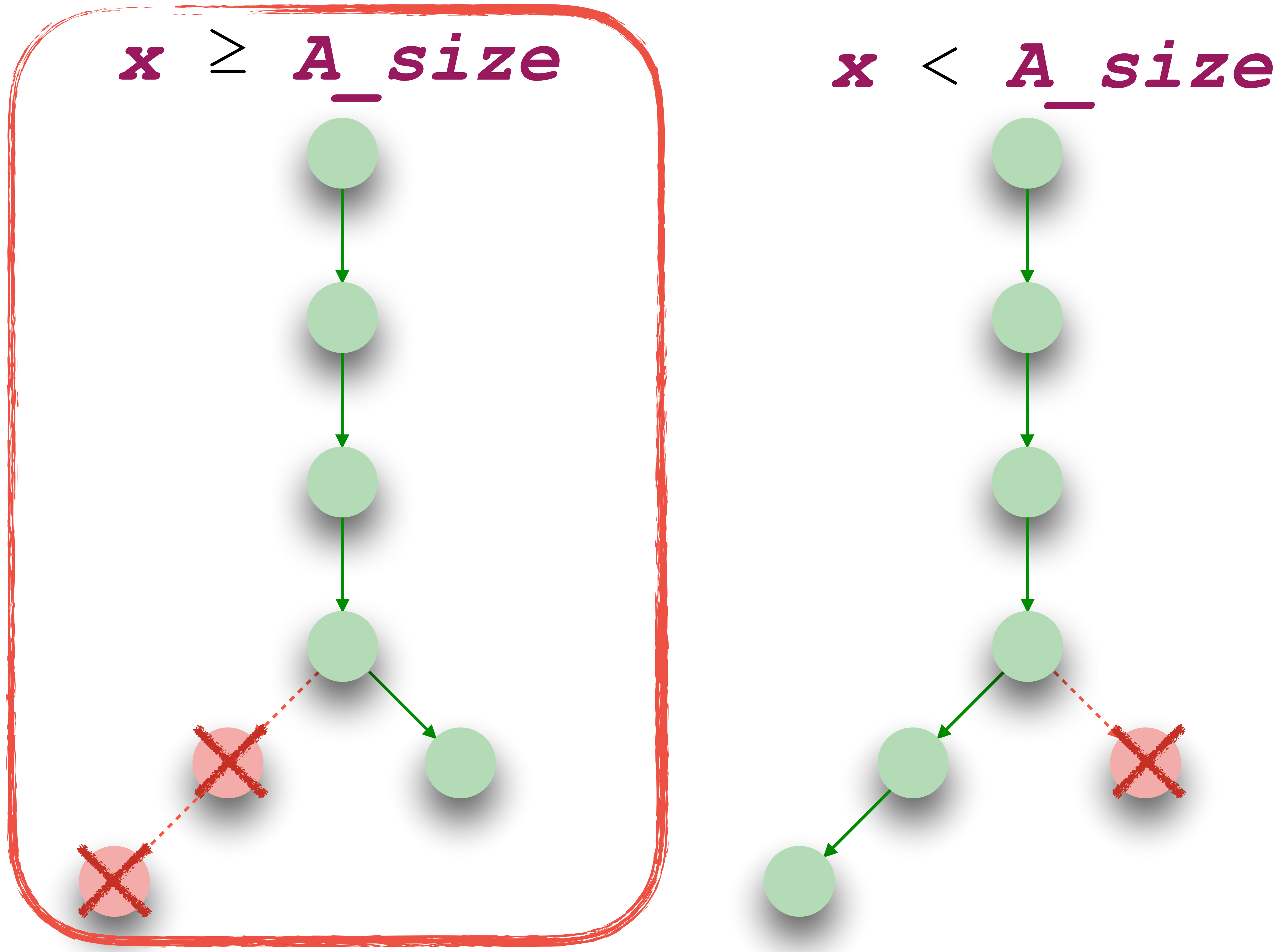
Symbolic execution

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



Symbolic execution

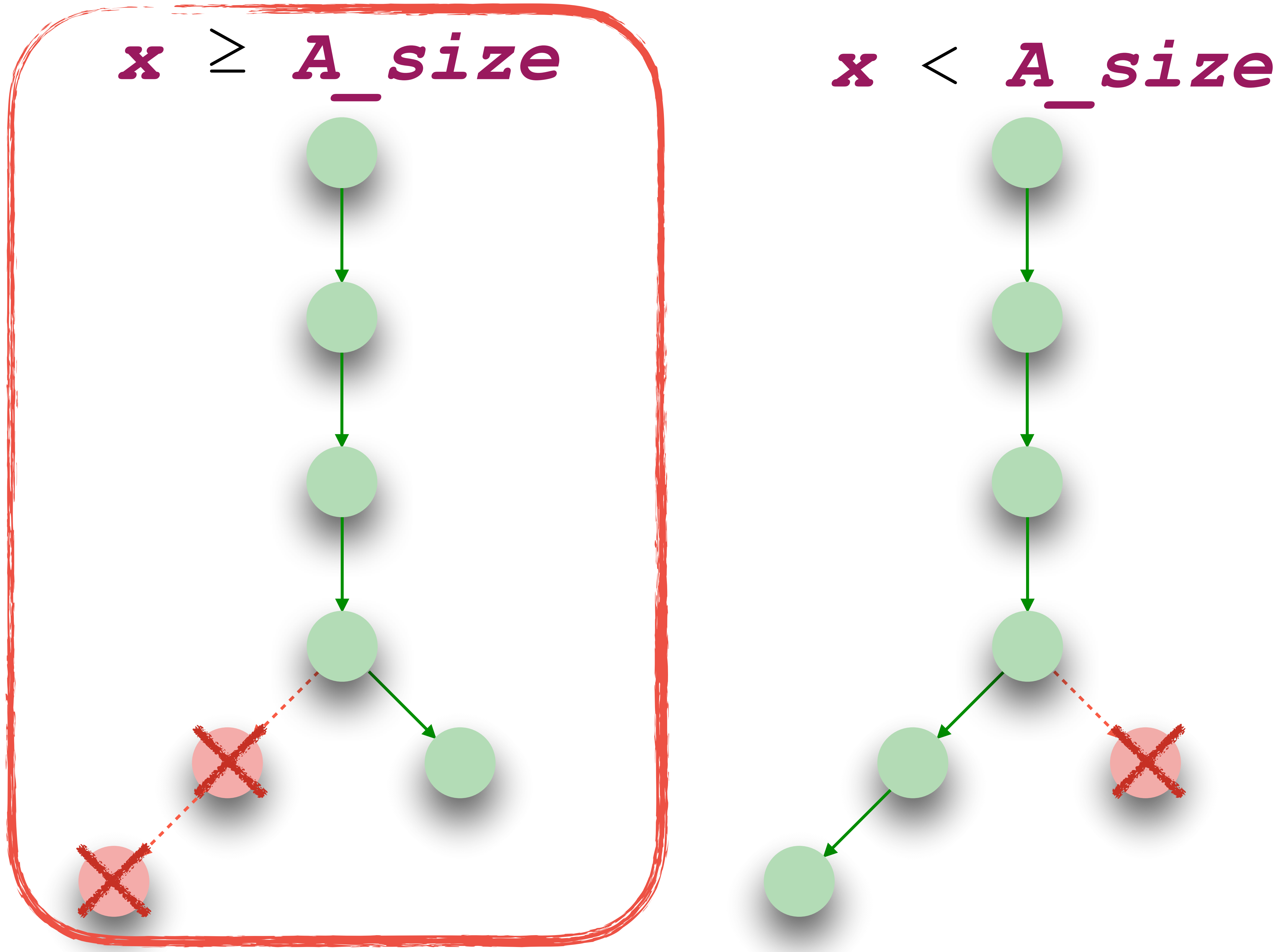
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



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

Symbolic execution

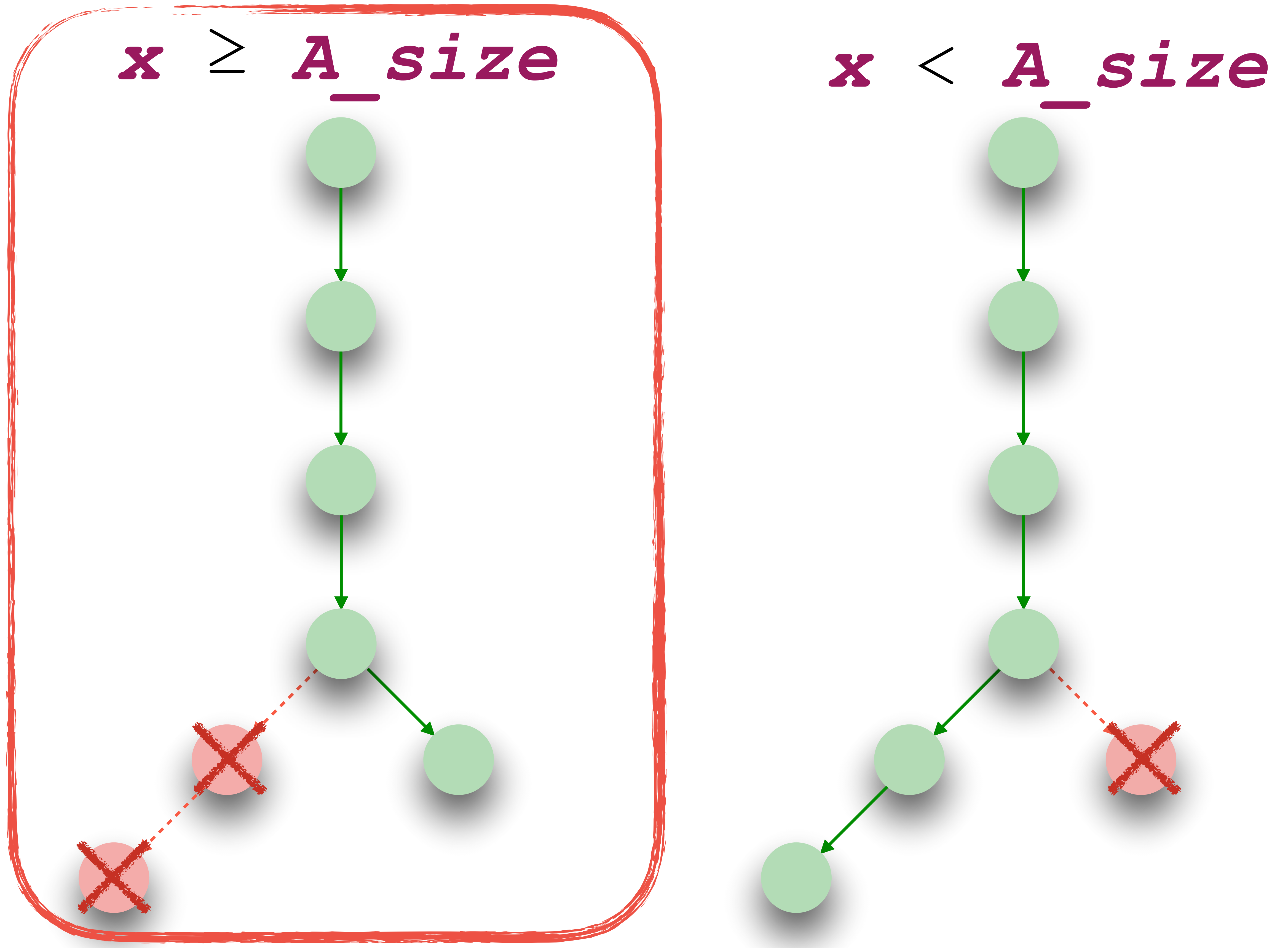
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



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

Symbolic execution

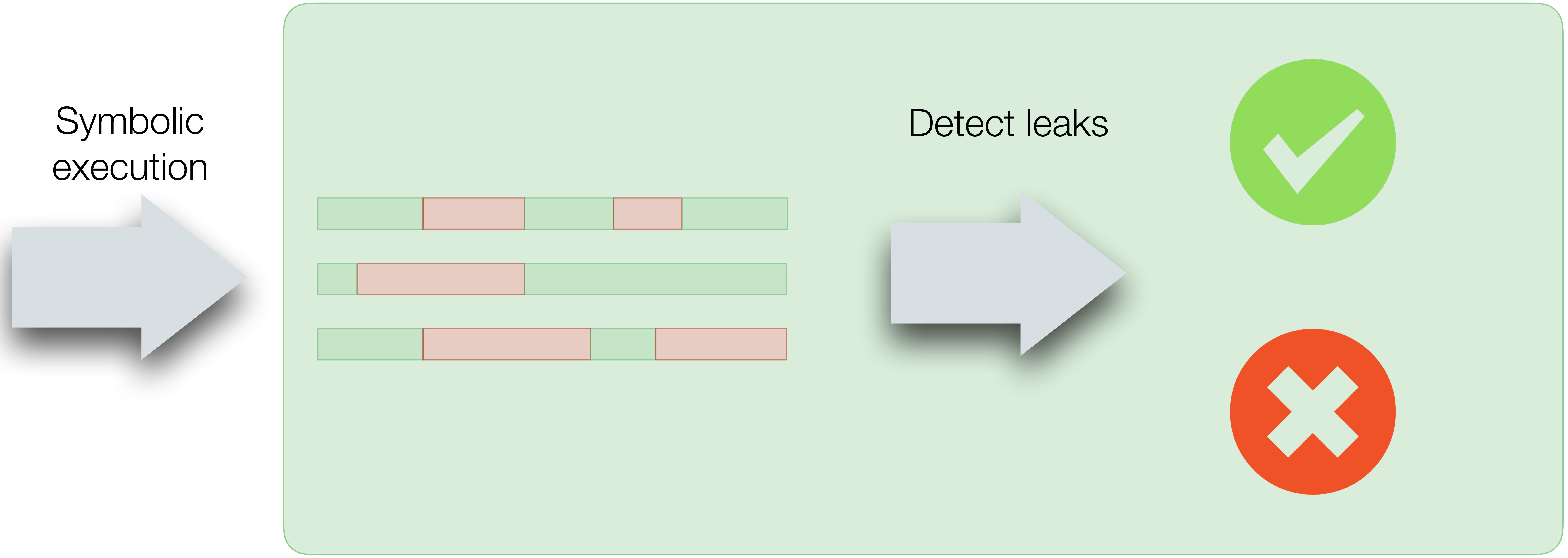
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



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

Detecting speculative leaks

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



Detecting speculative leaks

```
     rax <- A_s
     rcx <- x
     jmp rcx≥ra
L1:  load rax,
     load rax,
END:
```

For each $\tau \in$ **sym-traces**(P)

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

    return $INSECURE$

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

    return $INSECURE$

return $SECURE$

Detecting speculative leaks

```
     rax <- A_s
     rcx <- x
     jmp rcx≥ra
L1:  load rax,
     load rax,
END:
```

For each $\tau \in$ **sym-traces**(P)

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

    return $INSECURE$

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

    return $INSECURE$

return $SECURE$

Memory leaks

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

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

Memory leaks

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

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

τ

Memory leaks

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

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

τ

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

Memory leaks

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

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

τ

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

s_1

s_2

Memory leaks

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

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

τ

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

$$s_1 \models \varphi$$

$$s_2 \models \varphi$$

Memory leaks

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

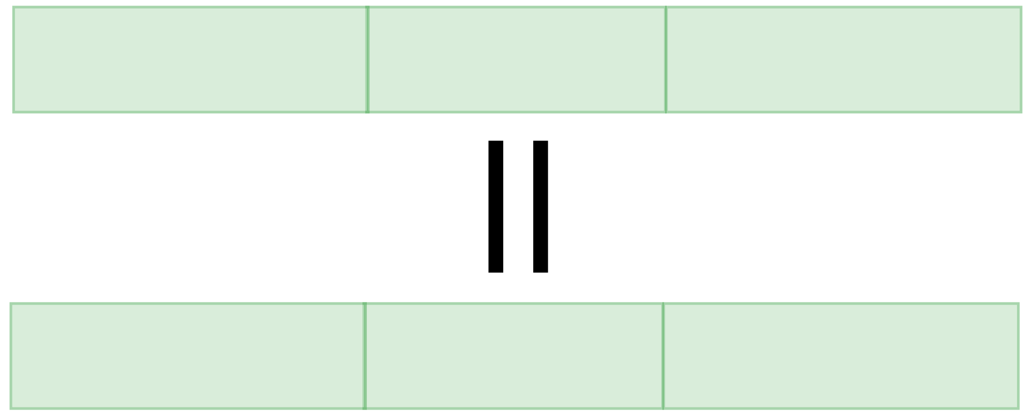
- Non-sensitive information (determined by policy), *or*
- be determined by non-speculative observations

τ

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

$s_1 \models \varphi$

$s_2 \models \varphi$



Memory leaks

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

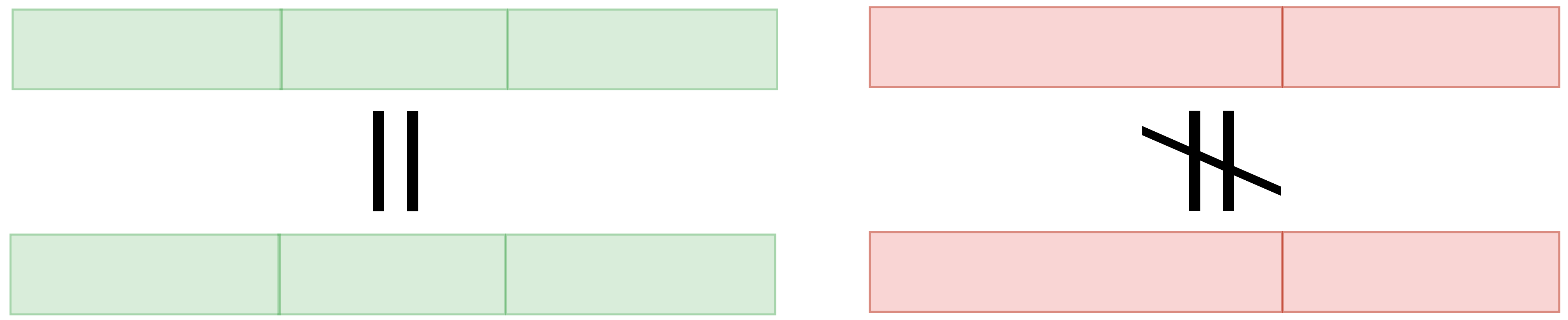
- Non-sensitive information (determined by policy), *or*
- be determined by non-speculative observations

τ 

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

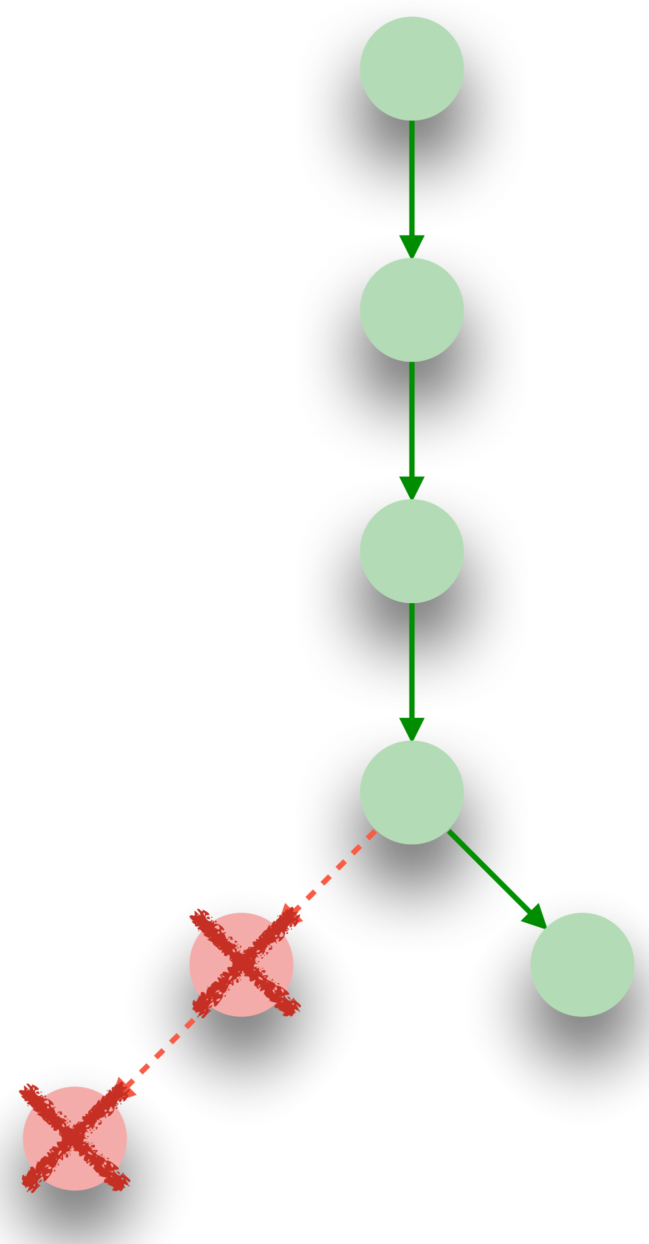
$s_1 \models \varphi$

$s_2 \models \varphi$



Memory leaks

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



Policy

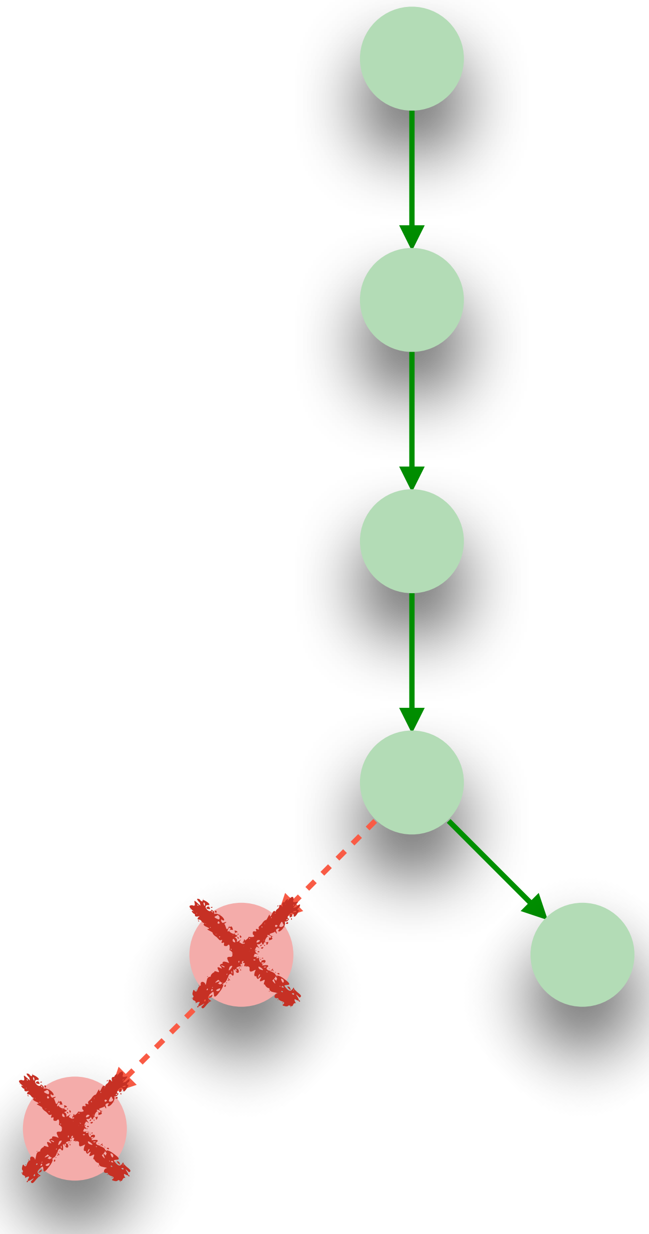
x, ***A_size***, **A**, **B**

are public

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

Memory leaks

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



Policy

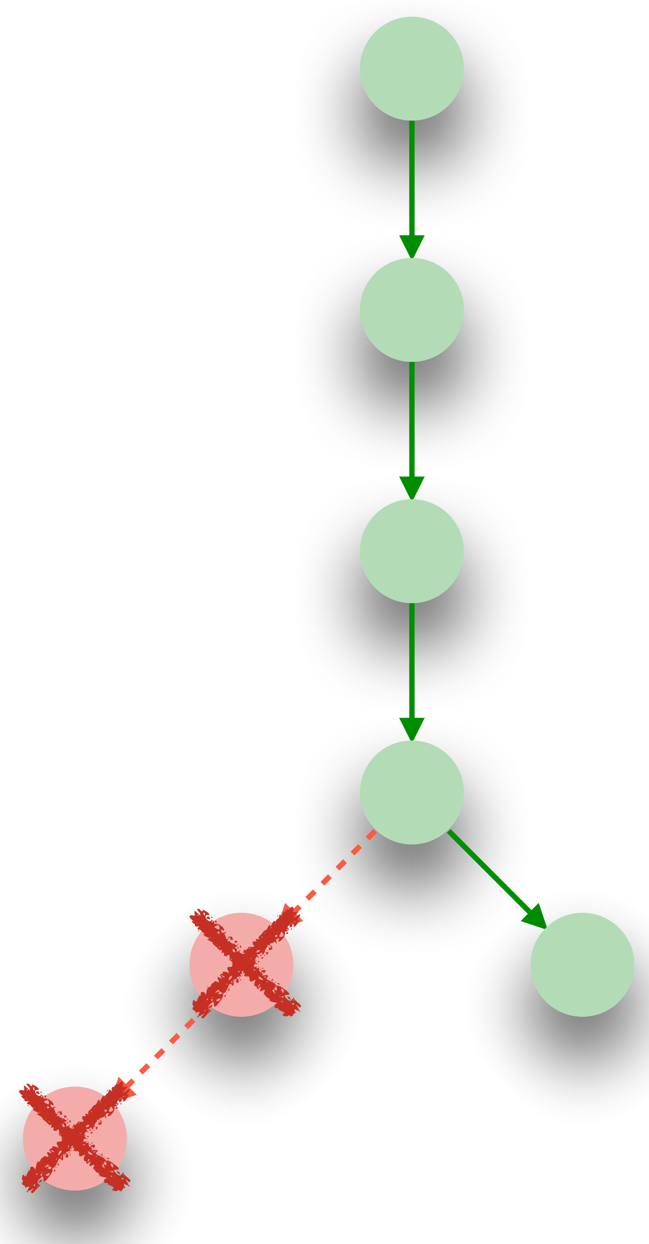
x, *A_size*, **A**, **B**
are public

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

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

Memory leaks

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



Policy
x, A_size, A, B
are public

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

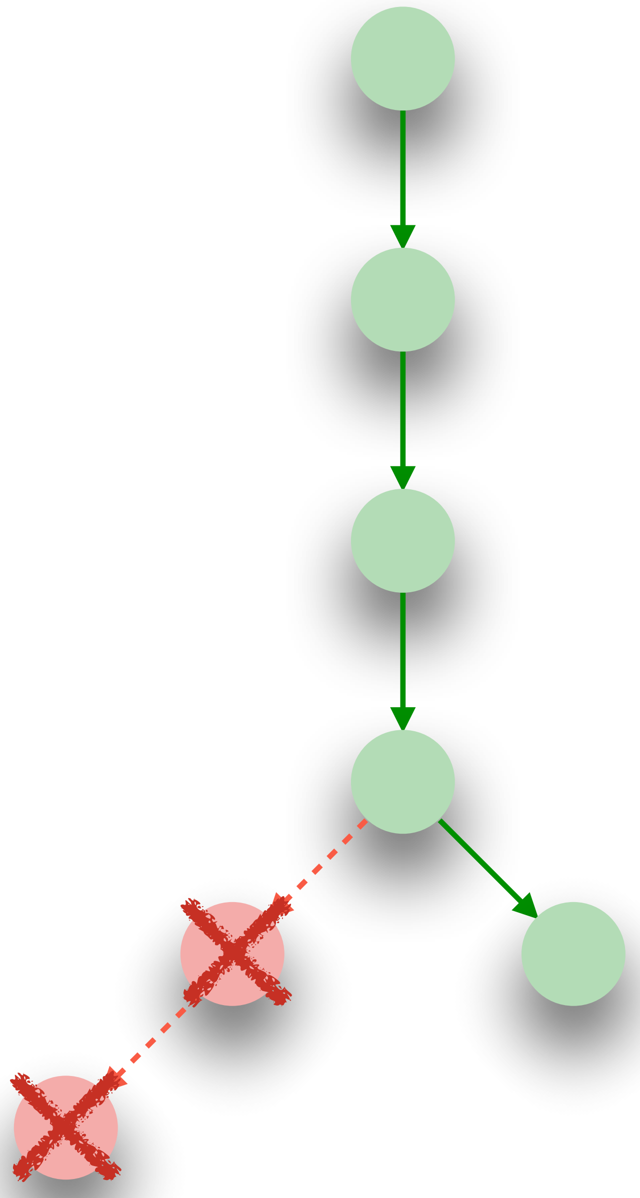
$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

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

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

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

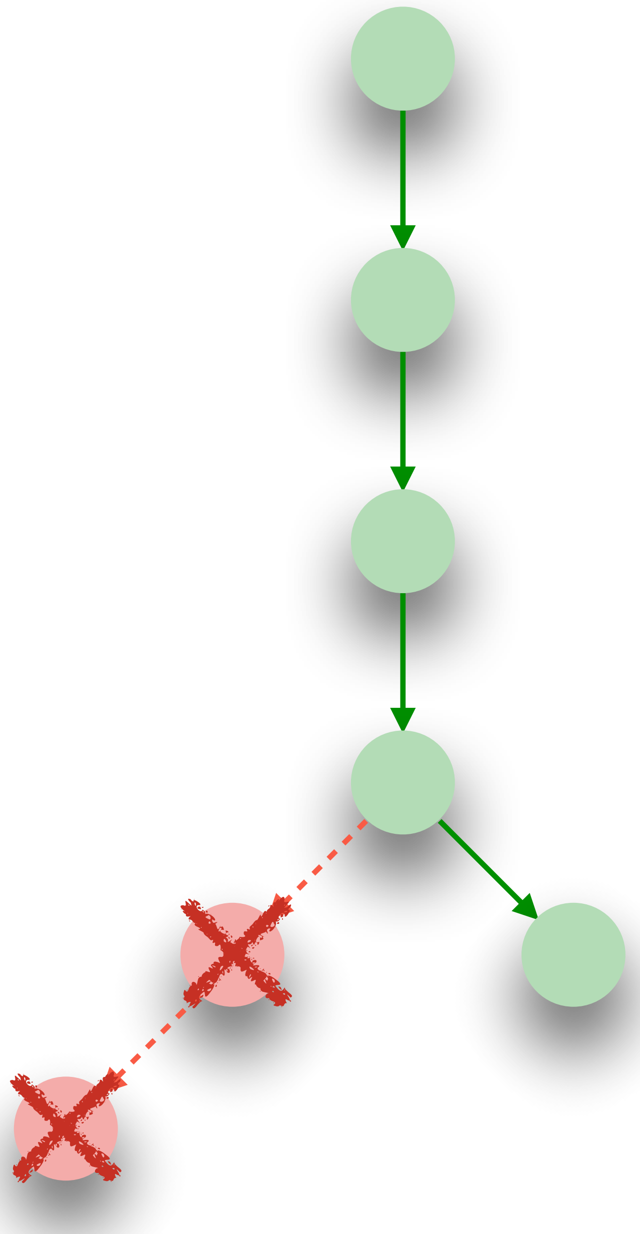
s_1

s_2

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

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

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

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

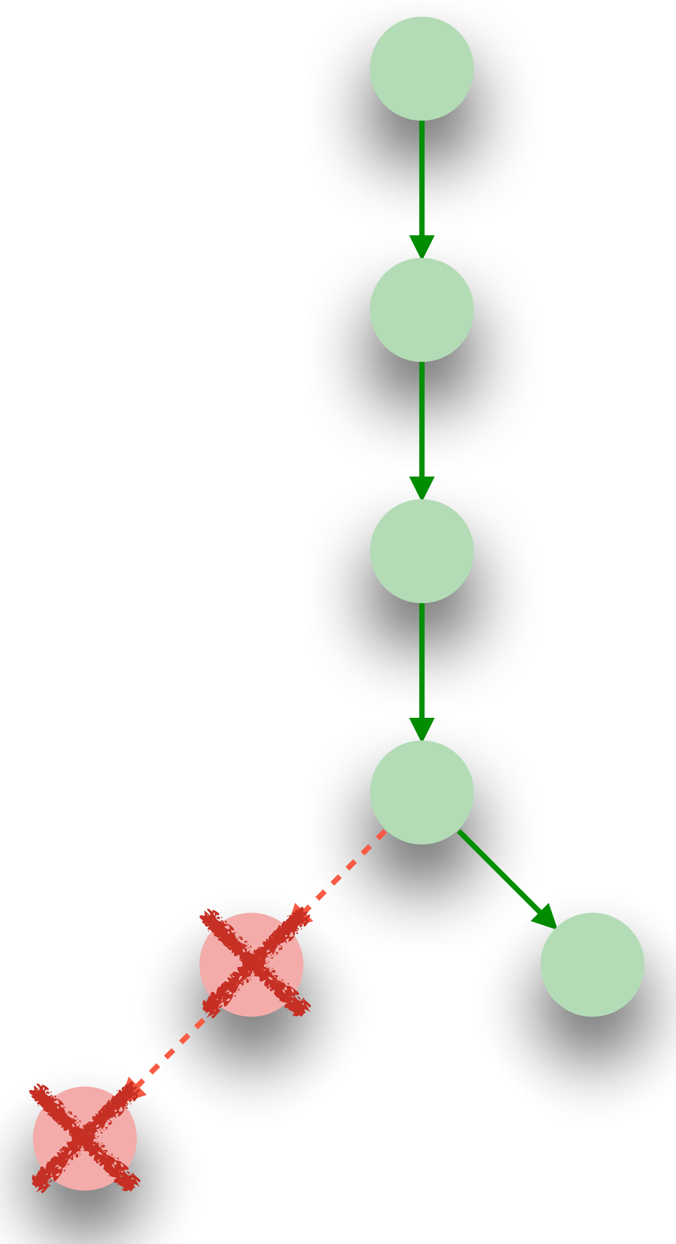
$s_1 \models$ $x_1 \geq A\_size_1$

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

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

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

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

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

$s_1 \models x_1 \geq A\_size_1$ — pc END

||

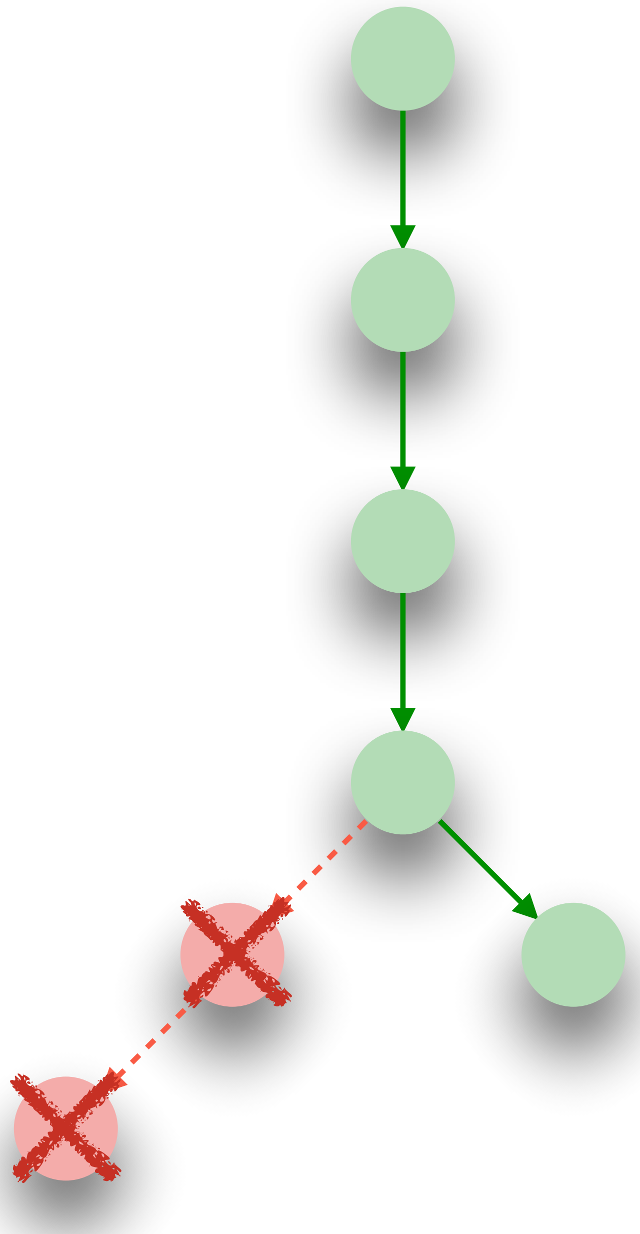
$s_2 \models x_2 \geq A\_size_2$ — pc END

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

Always true!

Memory leaks

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



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

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

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

$s_1 \models$ $x_1 \geq A\_size_1$ pc *END*

||

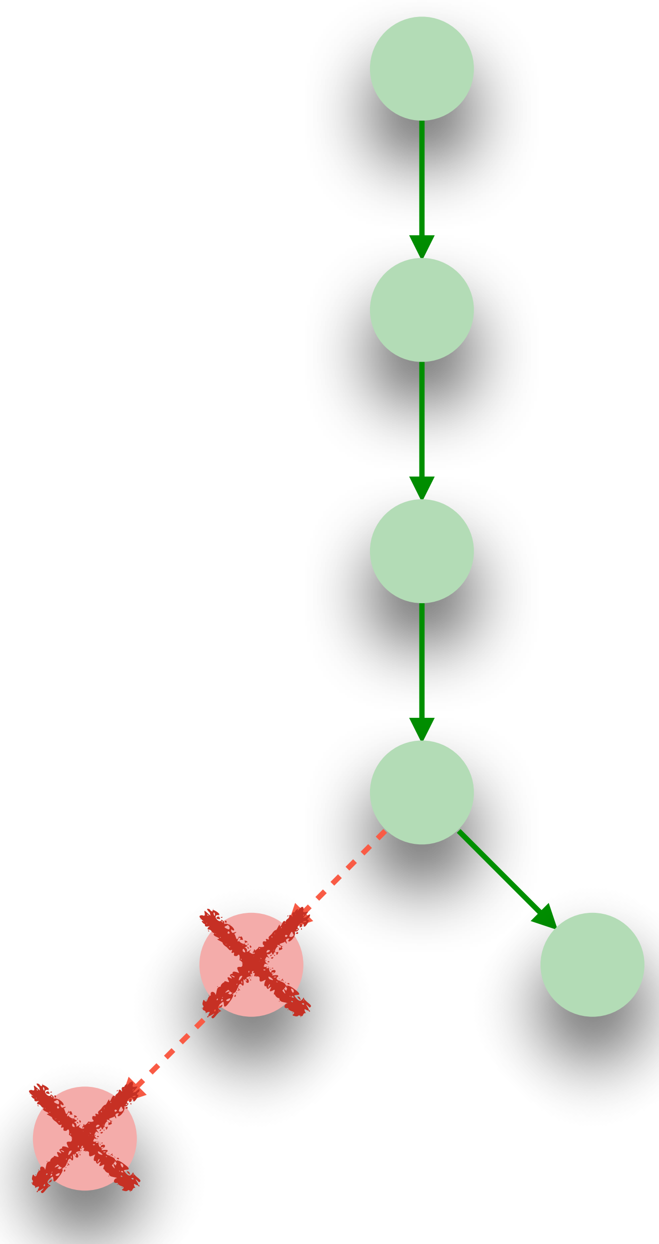
$s_2 \models$ $x_2 \geq A\_size_2$ pc *END*

Always true!

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

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

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

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

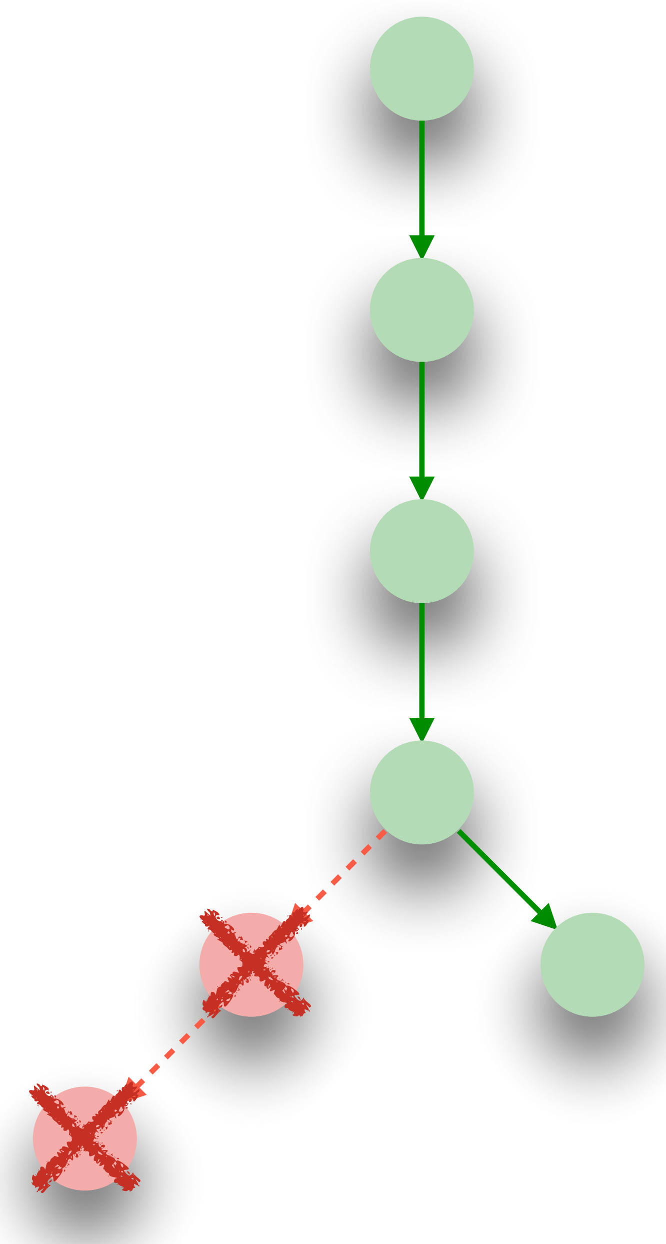
s_1	$\models$	$x_1 \geq A\_size_1$	pc END	$A_1 + x_1$
			$=$	$\neq$
s_2	$\models$	$x_2 \geq A\_size_2$	pc END	$A_2 + x_2$

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

Always true!

Memory leaks

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



Policy

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

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

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

$s_1 \models$ $x_1 \geq A\_size_1$ pc *END* $=$ A_1+x_1 $\neq$ $\lor$

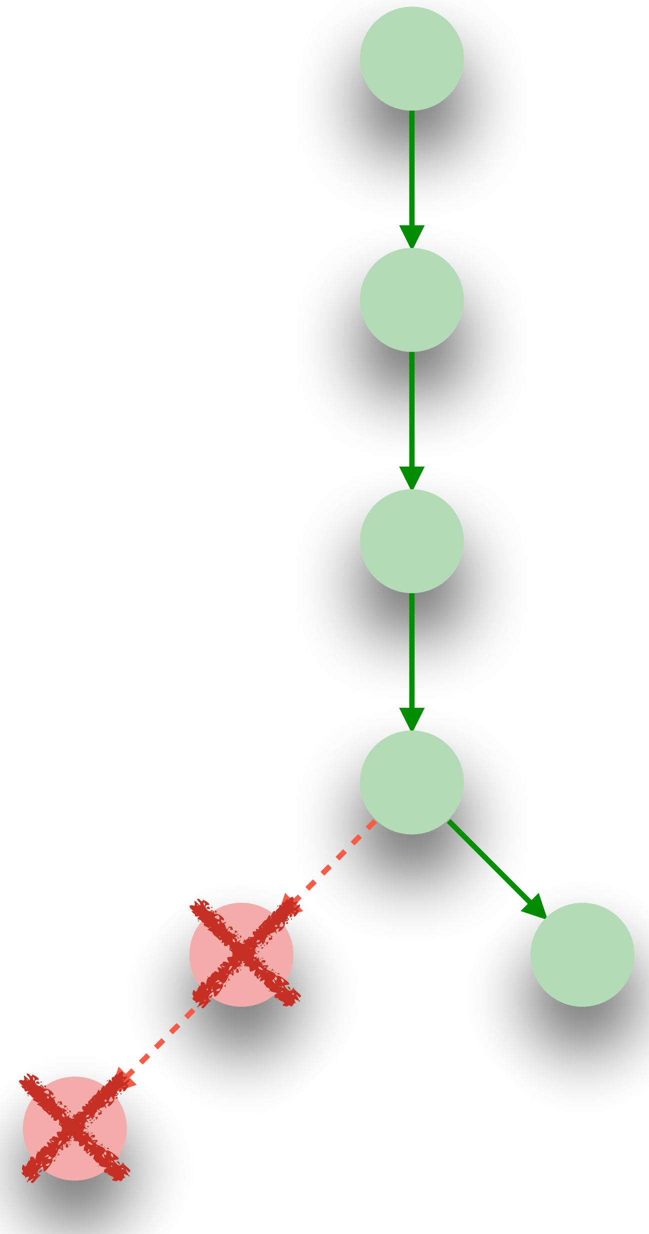
$s_2 \models$ $x_2 \geq A\_size_2$ pc *END* A_2+x_2

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

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

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

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

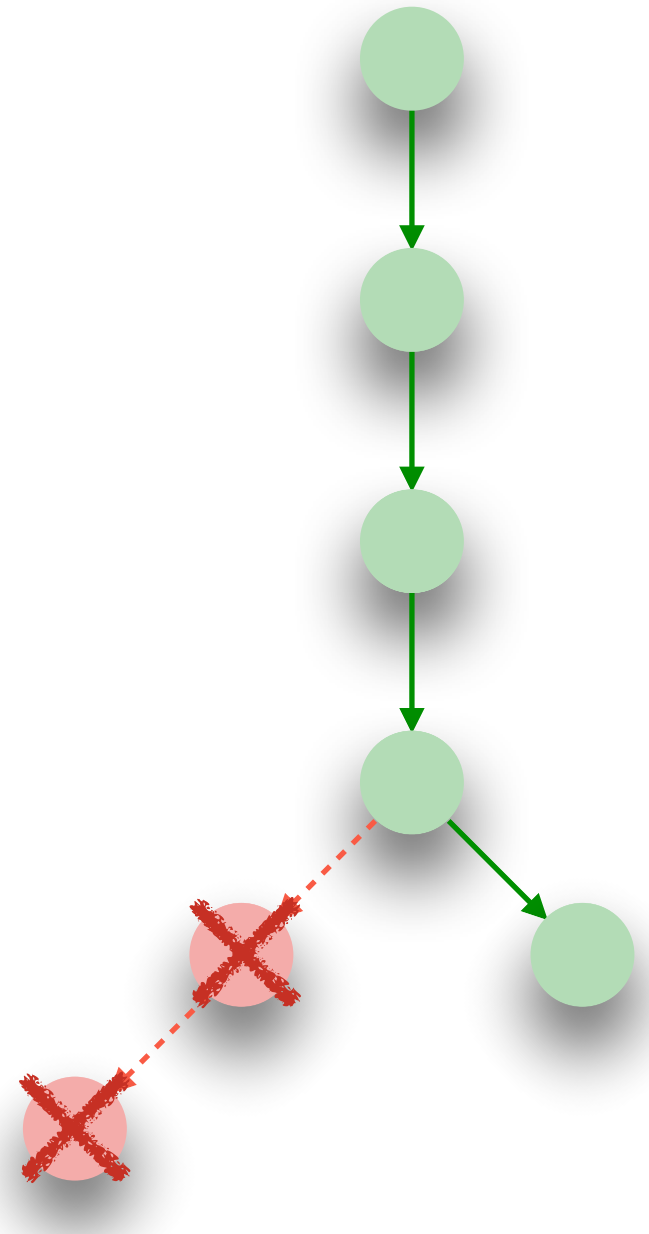
$s_1 \models$	$x_1 \geq A\_size_1$	pc END	A_1+x_1		$B_1+A_1[x_1]$
		$=$	$\neq$	$\vee$	$\neq$
$s_2 \models$	$x_2 \geq A\_size_2$	pc END	A_2+x_2		$B_2+A_2[x_2]$

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

Always true!

Memory leaks

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



Policy

x, A_size, A, B

are public

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

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

$s_1 \models$	$x_1 \geq A\_size_1$	pc END	A_1+x_1		$B_1+A_1[x_1]$	
		$\|$	$\not\Vdash$	$\vee$	$\not\Vdash$	✖
$s_2 \models$	$x_2 \geq A\_size_2$	pc END	A_2+x_2		$B_2+A_2[x_2]$	

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

Always true!

Experimental results

Ex.	VISUAL C++				ICC				CLANG					
	UNP		FEN		UNP		FEN		UNP		FEN		SLH	
	−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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

15 Spectre variants from Paul Kocher

F	UNP		FEN		ICC				CLANG					
					UNP		FEN		UNP		FEN		SLH	
	−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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

15 Spectre variants from Paul Kocher

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

					Icc				Clang					
					Unp		Fen		Unp		Fen		Slh	
	−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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

15 Spectre variants from Paul Kocher

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

[illegible]	[illegible]				ICC				CLANG					
	UNP		FEN		UNP		FEN		UNP		FEN		SLH	
	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2
01	○	○	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	○	○	●	●	○	○	●	●	●	●
04	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○	●	●	●	●
05	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○	●	●	●	●
06	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

15 Spectre variants from Paul Kocher

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

[illegible]	[illegible]				Icc				Clang					
	Unp		Fen		Unp		Fen		Unp		Fen		Slh	
	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2
01	○	○	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○	●	●	●	●
07	○	○	○	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○	●	●	●	●
08	○	●	○	●	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●	●
09	○	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○	●	●	●	●
10	○	○	○	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	○	●	●	●	○
11	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

Ex.	Visual C++				Icc				Clang					
	Unp		Fen		Unp		Fen		Unp		Fen		Slh	
	-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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

Ex.	Visual C++				Icc				Clang					
	Unp		Fen		Unp		Fen		Unp		Fen		Slh	
	−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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental r

No countermeasures

Ex.	Visual C++				Icc				Clang					
	Unp		Fen		Unp		Fen		Unp		Fen		Slh	
	−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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

Automated insertion of fences

Ex.	Visual C++				Icc				Clang					
	Unp		Fen		Unp		Fen		Unp		Fen		Slh	
	-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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

Speculative load hardening

Ex.	Visual C++				Icc				Clang					
	Unp		Fen		Unp		Fen		Unp		Fen		Slh	
	−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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

Ex.	Visual C++				Icc				Clang					
	Unp		Fen		Unp		Fen		Unp		Fen		Slh	
	-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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

Ex.	VISUAL C++				ICC				CLANG					
	UNP		FEN		UNP		FEN		UNP		FEN		SLH	
	-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	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Experimental results

Ex.	Visual C++				Icc				Clang					
	Unp		Fen		Unp		Fen		Unp		Fen		SLH	
	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2	−O0	−O2
01	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
02	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
03	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
04	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
05	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
06	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
07	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
08	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
09	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
10	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	○
11	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
12	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●
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
  - But: Unnecessary fences
- Programs with SLH are secure except #10 and #15

Experimental results

Ex.	VISUAL C++				ICC				CLANG					
	UNP		FEN		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●
02	○	○	●	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●
03	○	○	●	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●
04	○	○	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●
05	○	○	●	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●
06	○	○	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●
07	○	○	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●
08	○	●	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●
09	○	○	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	●
10	○	○	○	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	●	●	●	○
11	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Performance

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

4. Challenges

Scalable analysis

Goal:
Analysis of large, security-critical applications:

- Intel SGX SDK
- Xen hypervisor
- microkernels

Need: Scalable analysis of speculative non-interference

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

Verifying compiler-level countermeasures

```
mov     rax, A_size
mov     rcx, x
cmp     rcx, rax
jae     END
mov     rax, A[rcx]
shl     rax, 9
mov     rax, B[rax]
```

Inserting fences

```
mov     rax, A_size
mov     rcx, x
cmp     rcx, rax
jae     END
lfence
mov     rax, A[rcx]
shl     rax, 9
mov     rax, B[rax]
```

Speculative load hardening

```
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]
```

How can we **verify** such countermeasures?

A sound HW/SW security contract

Instruction-set architecture: to weak for security guarantees

Microarchitecture: not available publicly, and too detailed for analysis

A sound HW/SW security contract

Instruction-set architecture: to weak for security guarantees

HW/SW security contract

Microarchitecture: not available publicly, and too detailed for analysis

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

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

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

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

I am looking for PhD students and postdocs!

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

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

I am looking for PhD students and postdocs!

Thank you for your attention!

Backup

Example #01 - SLH

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

Example #01 - SLH

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

```
mov      rax, A_size
mov      rcx, x
mov      rdx, 0
cmp      rcx, rax
jae      END
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 $x \geq$ `A_size`

We can prove security

Example #10 - SLH

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

Example #10 - SLH

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

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

Example #10 - SLH

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

```
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