Spectector: Principled detection of speculative information flows

Marco Guarnieri
IMDEA Software Institute

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

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

To appear at IEEE Security & Privacy 2020

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***

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***

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

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!

Contributions

Contributions

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

Contributions

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

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

Outline

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

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

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

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

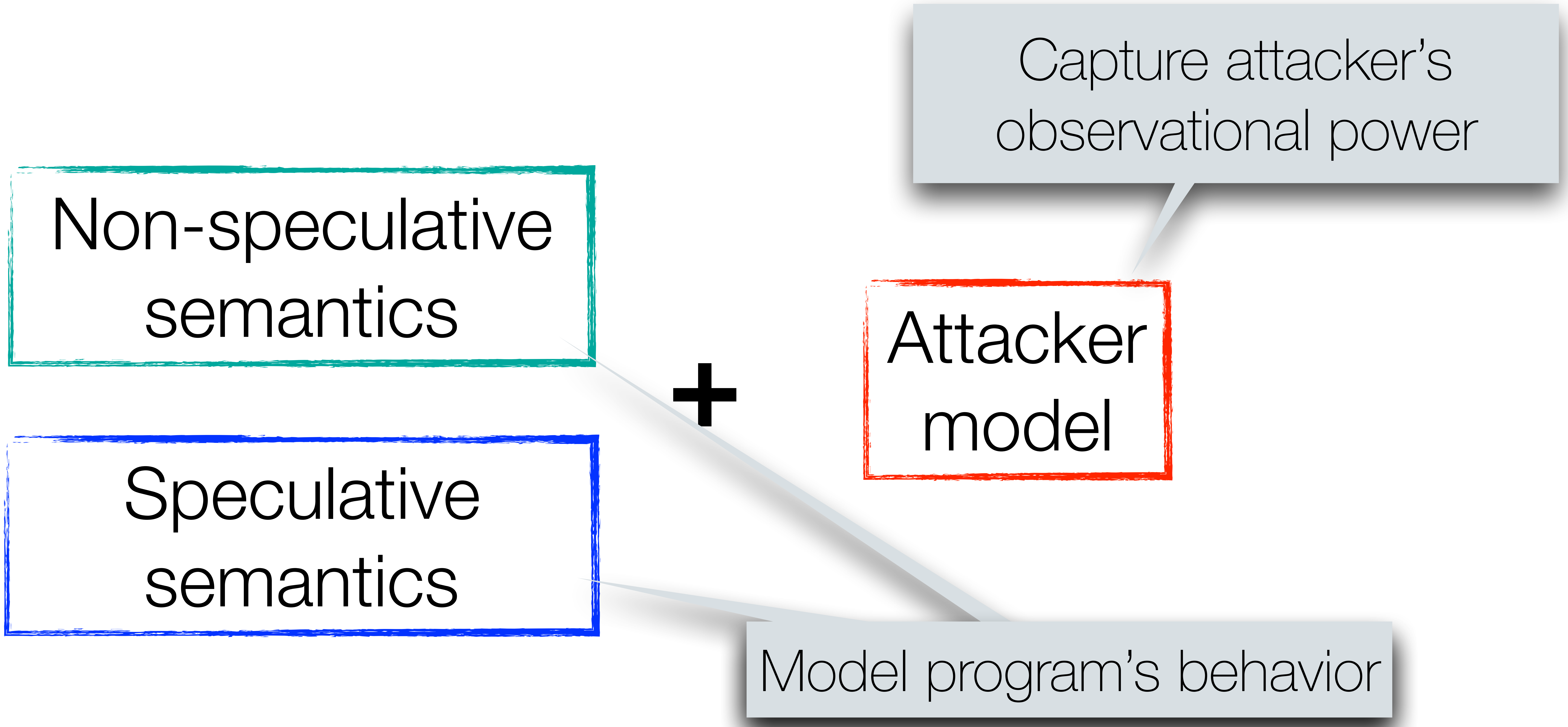
Speculative semantics

+

Attacker model

Model program's behavior

How to capture leakage?



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

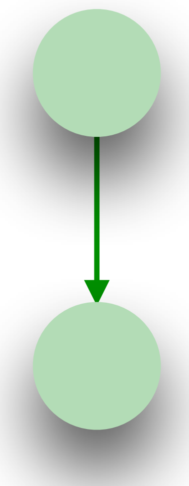
```
if (x < A_size)
    y = B[A[x]]
```

```
    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]]
```

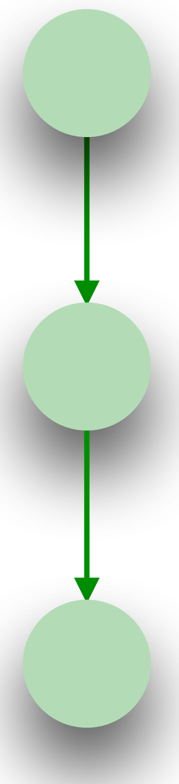
```
      rax <- A_size
      rcx <- x
      jmp rcx≥rax, END
L1:   load rax, A + rcx
      load rax, B + rax
END:
```



µAssembly + non-speculative semantics

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

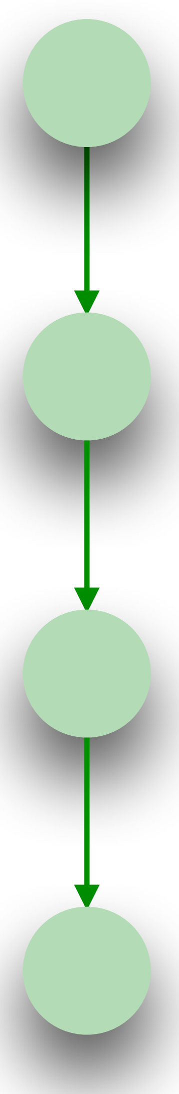
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```



µAssembly + non-speculative semantics

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

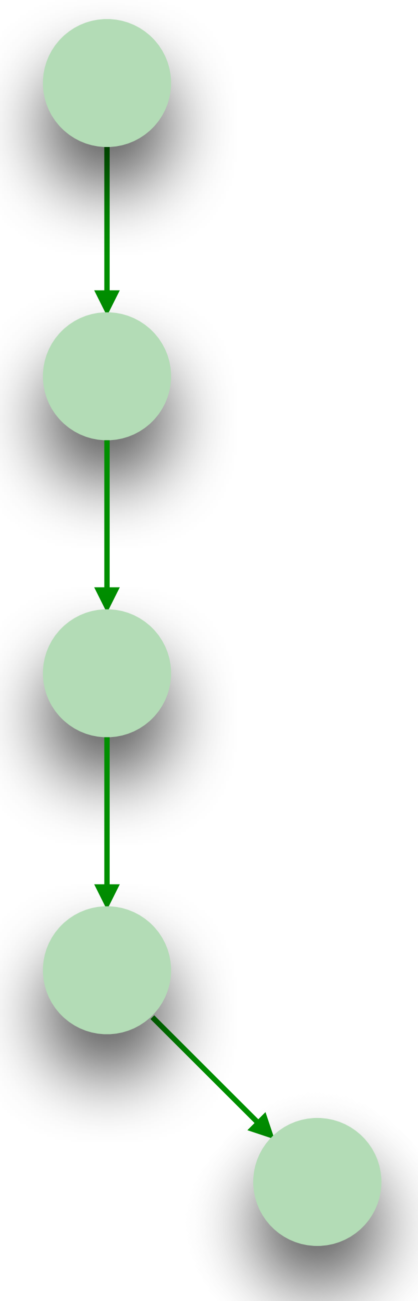
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```



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

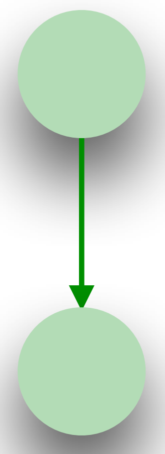
Committed upon correct speculation

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

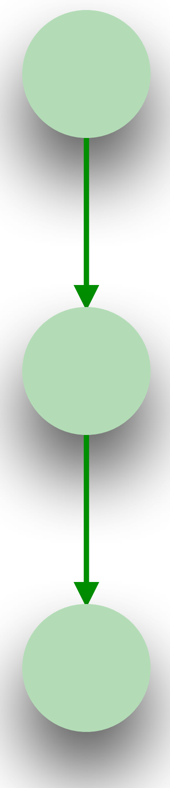
Committed upon correct speculation

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

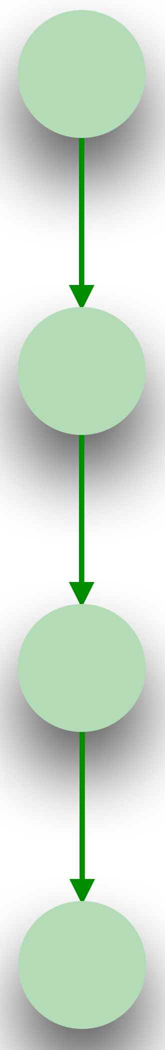
Committed upon correct speculation

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

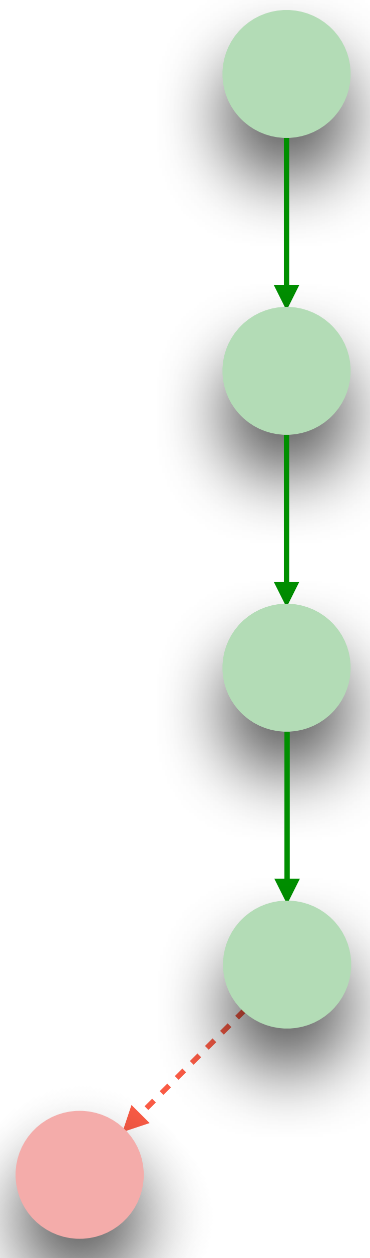
Committed upon correct speculation

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

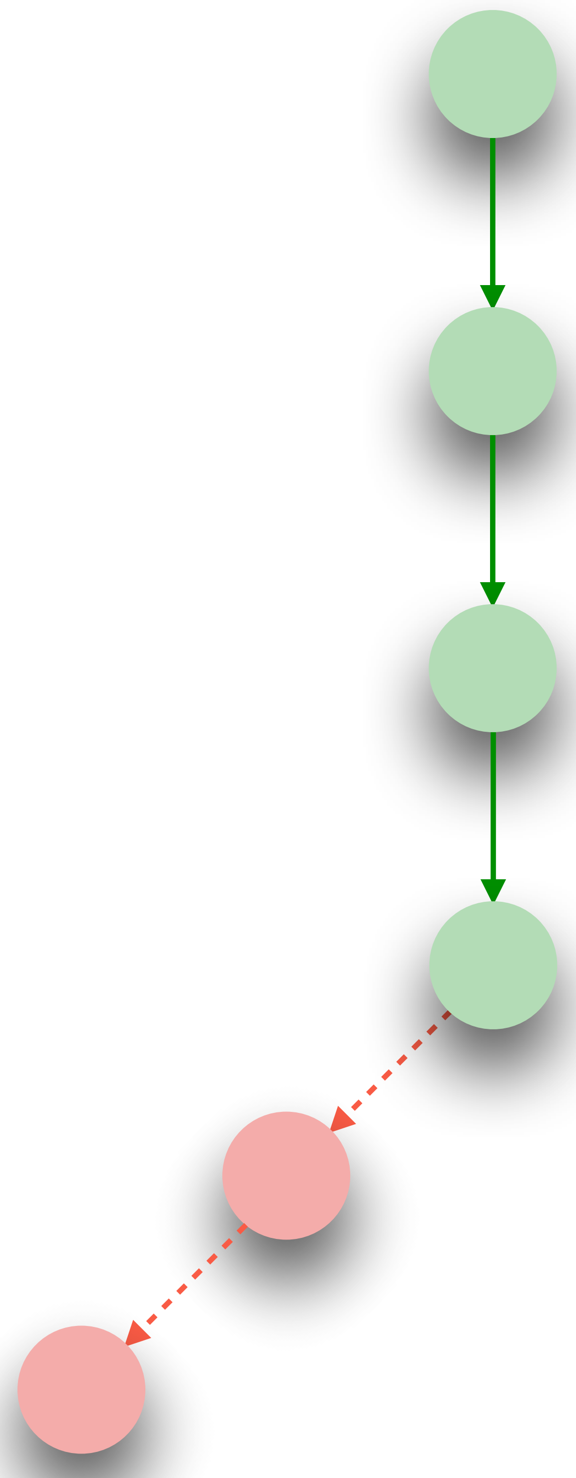
Committed upon correct speculation

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

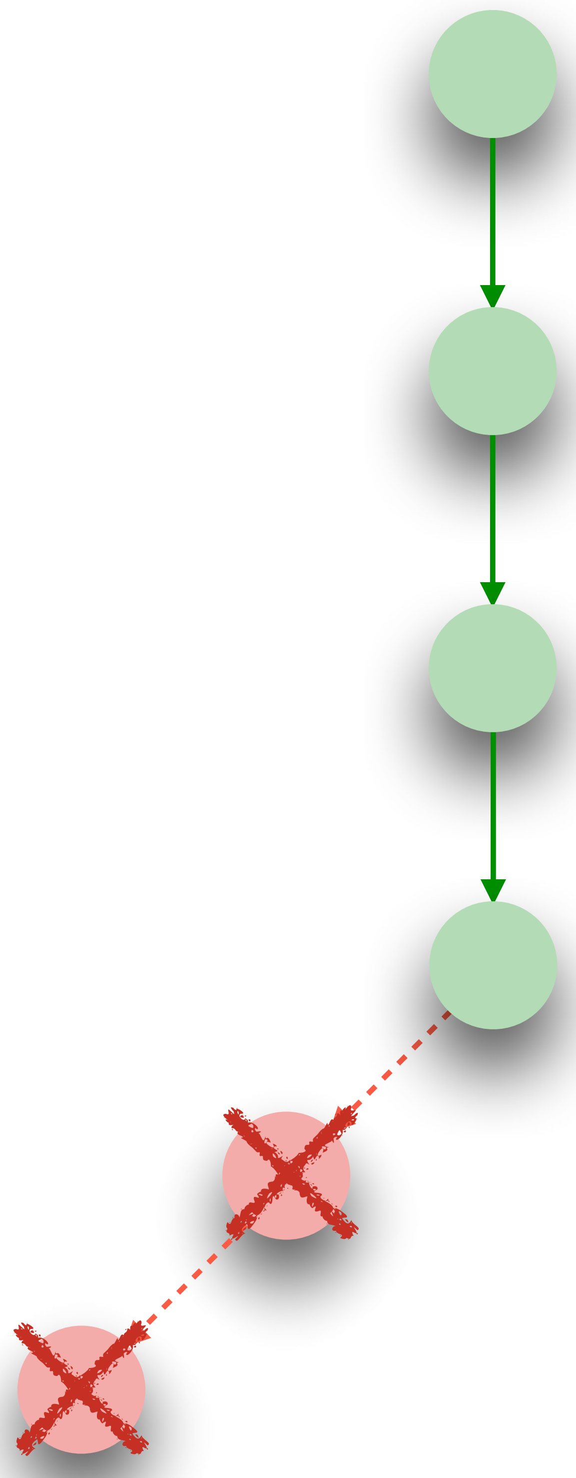
Committed upon correct speculation

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

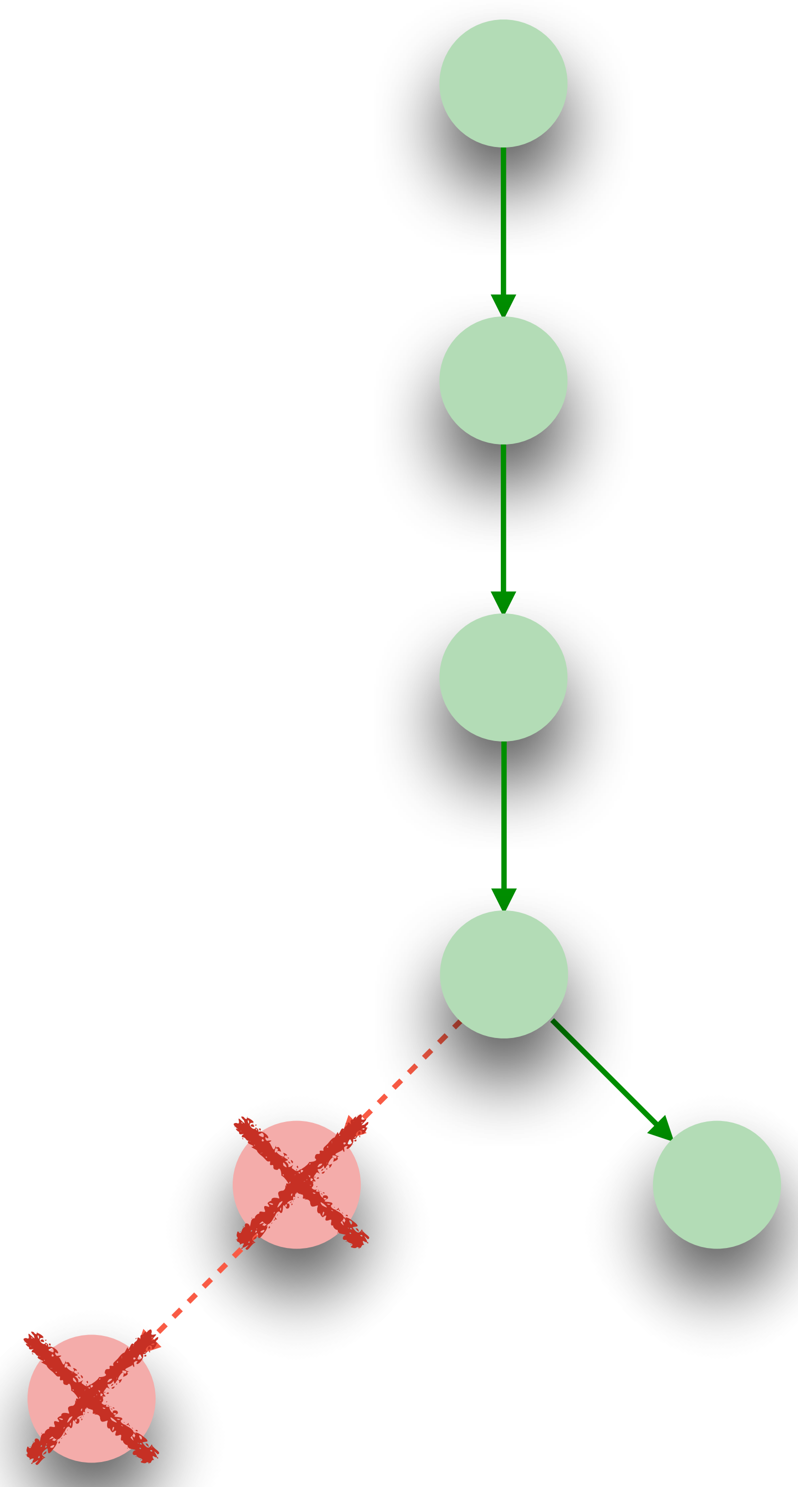
Committed upon correct speculation

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

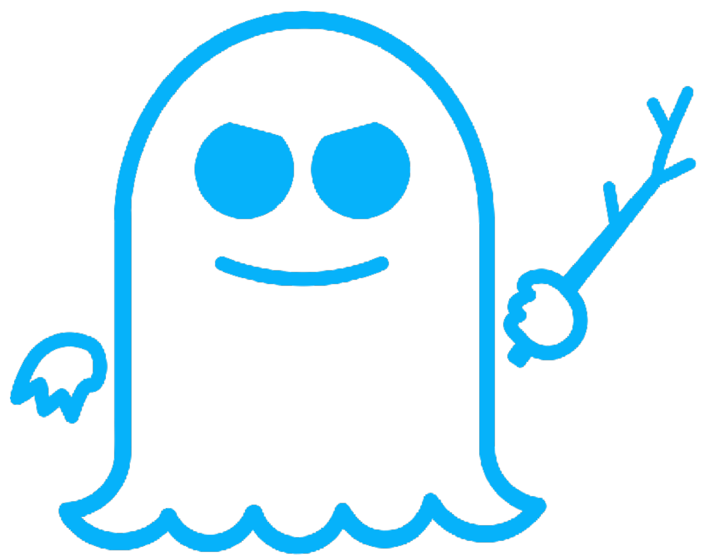
Committed upon correct speculation

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:
```

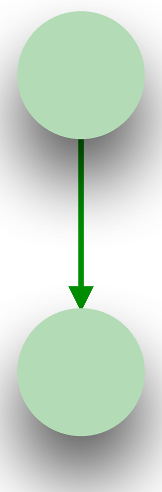
Attacker can observe:

- 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:
```



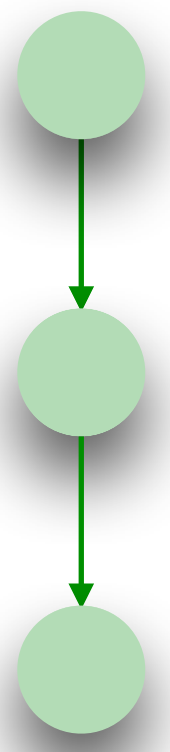
Attacker can observe:

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

Inspired by “constant-time” rqmts

Leakage into µarchitecture

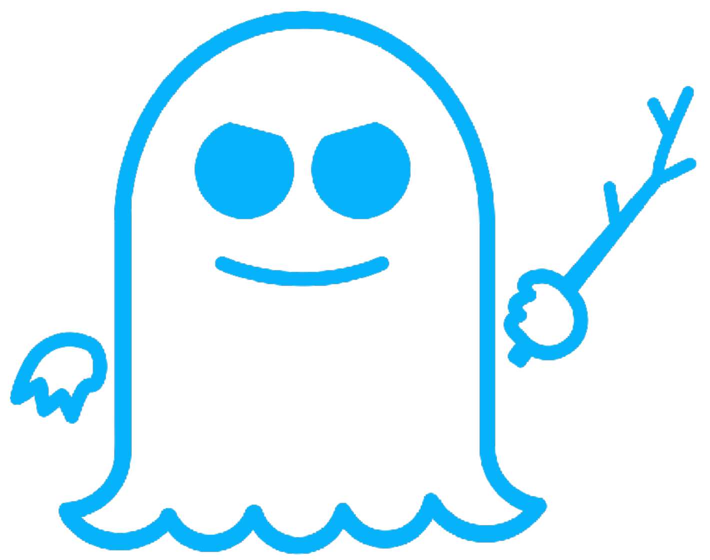
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Attacker can observe:

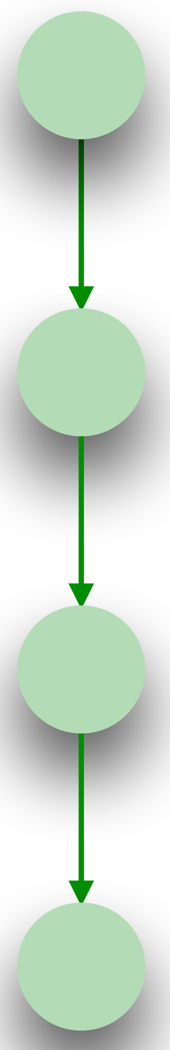
- locations of **memory accesses**
- **branch/jump** targets
- **start/end** speculative execution

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:
```



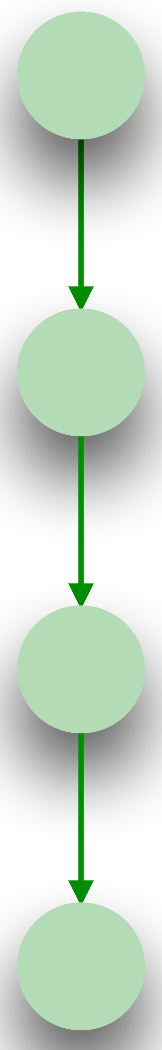
Attacker can observe:

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

Inspired by “constant-time” rqmts

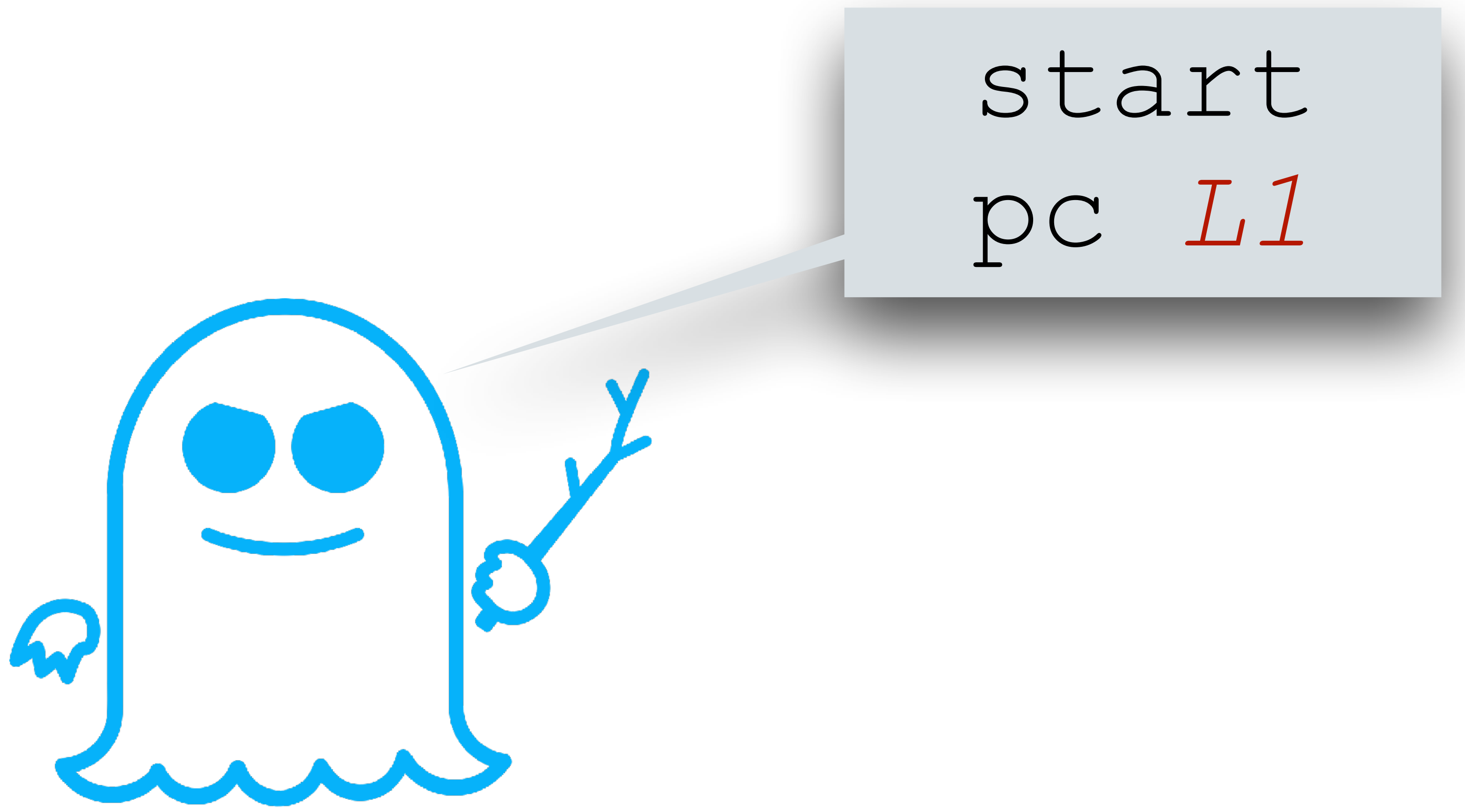
Leakage into μarchitecture

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



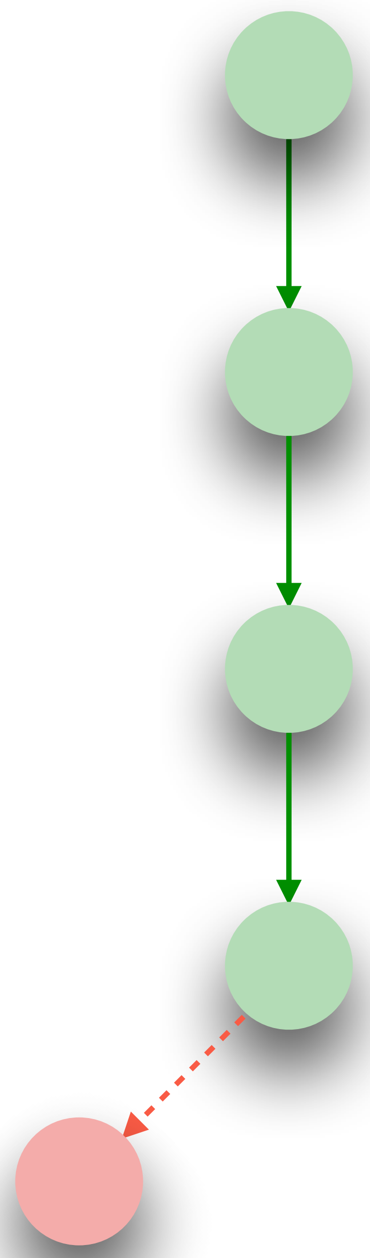
Attacker can observe:
- locations of **memory accesses**
- **branch/jump** targets
- **start/end** speculative execution

Inspired by “constant-time” rqmts



Leakage into µarchitecture

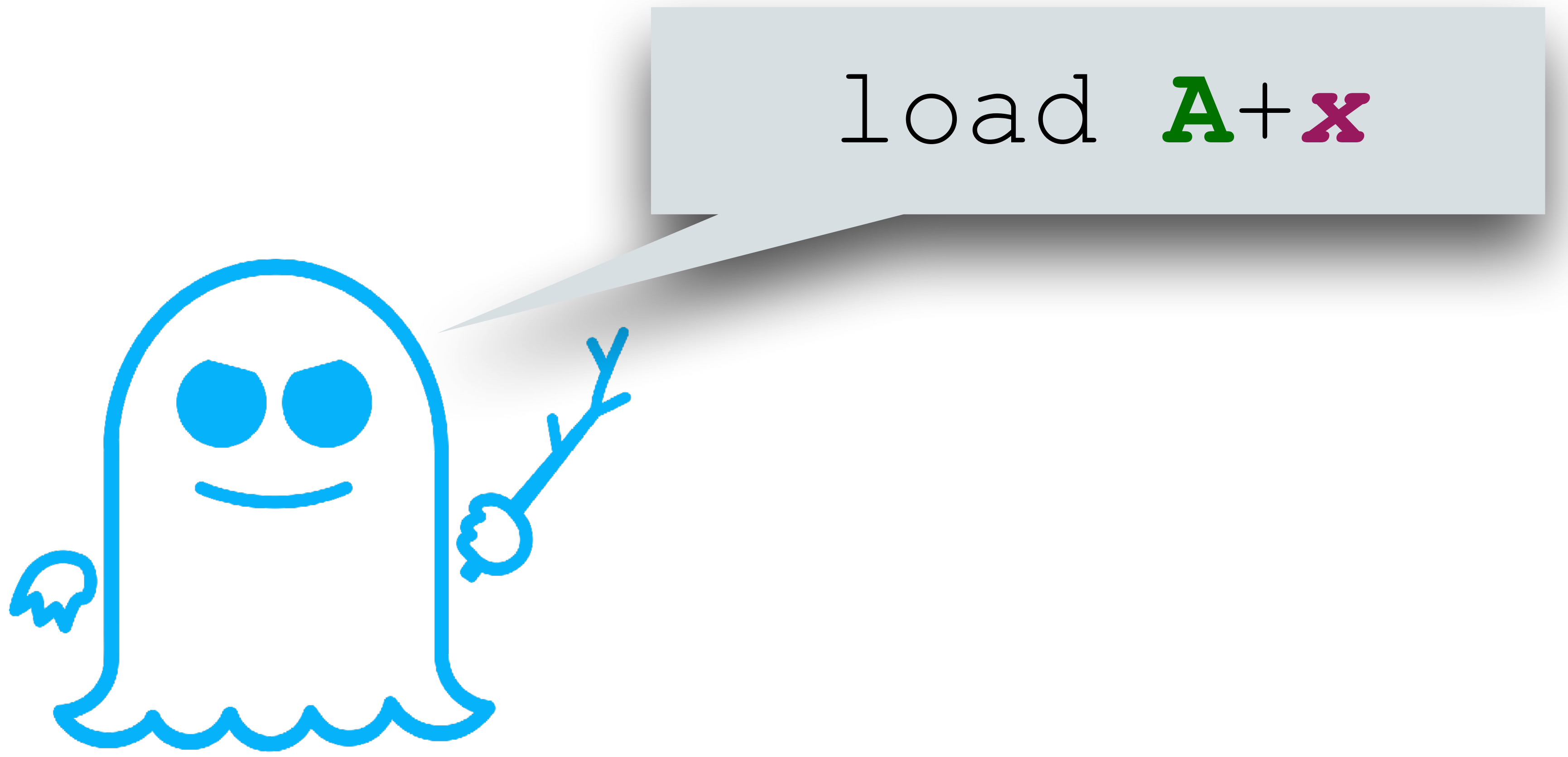
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Attacker can observe:

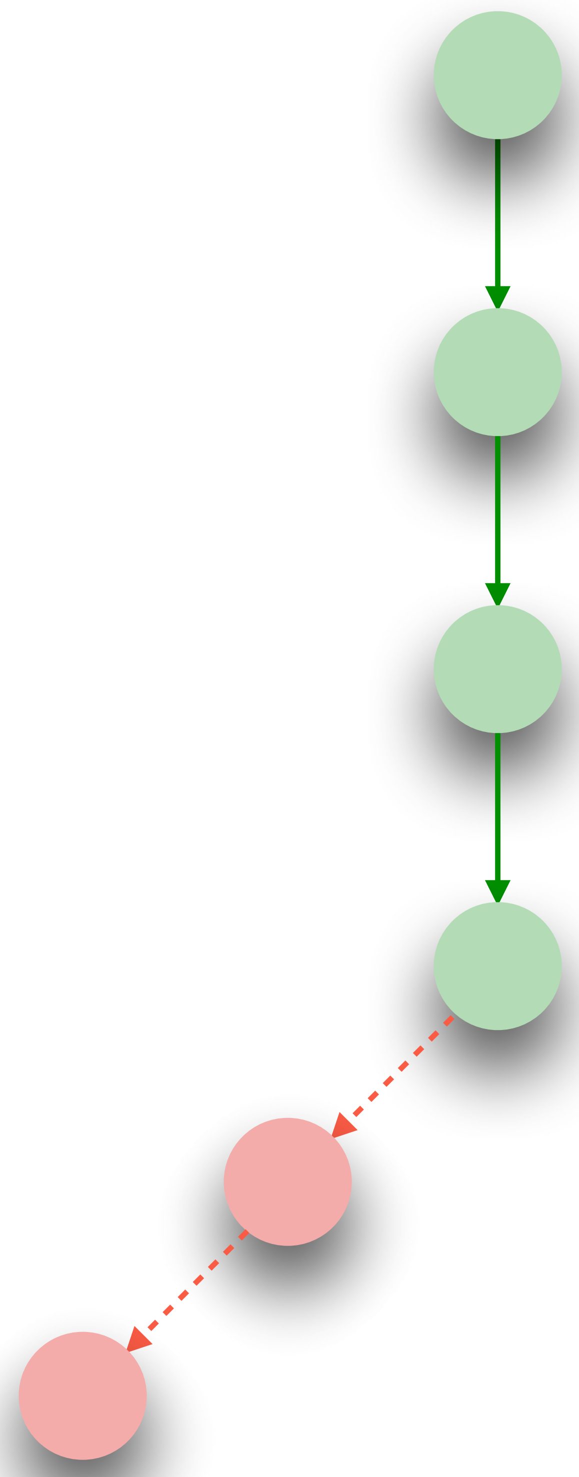
- locations of **memory accesses**
- **branch/jump** targets
- **start/end** speculative execution

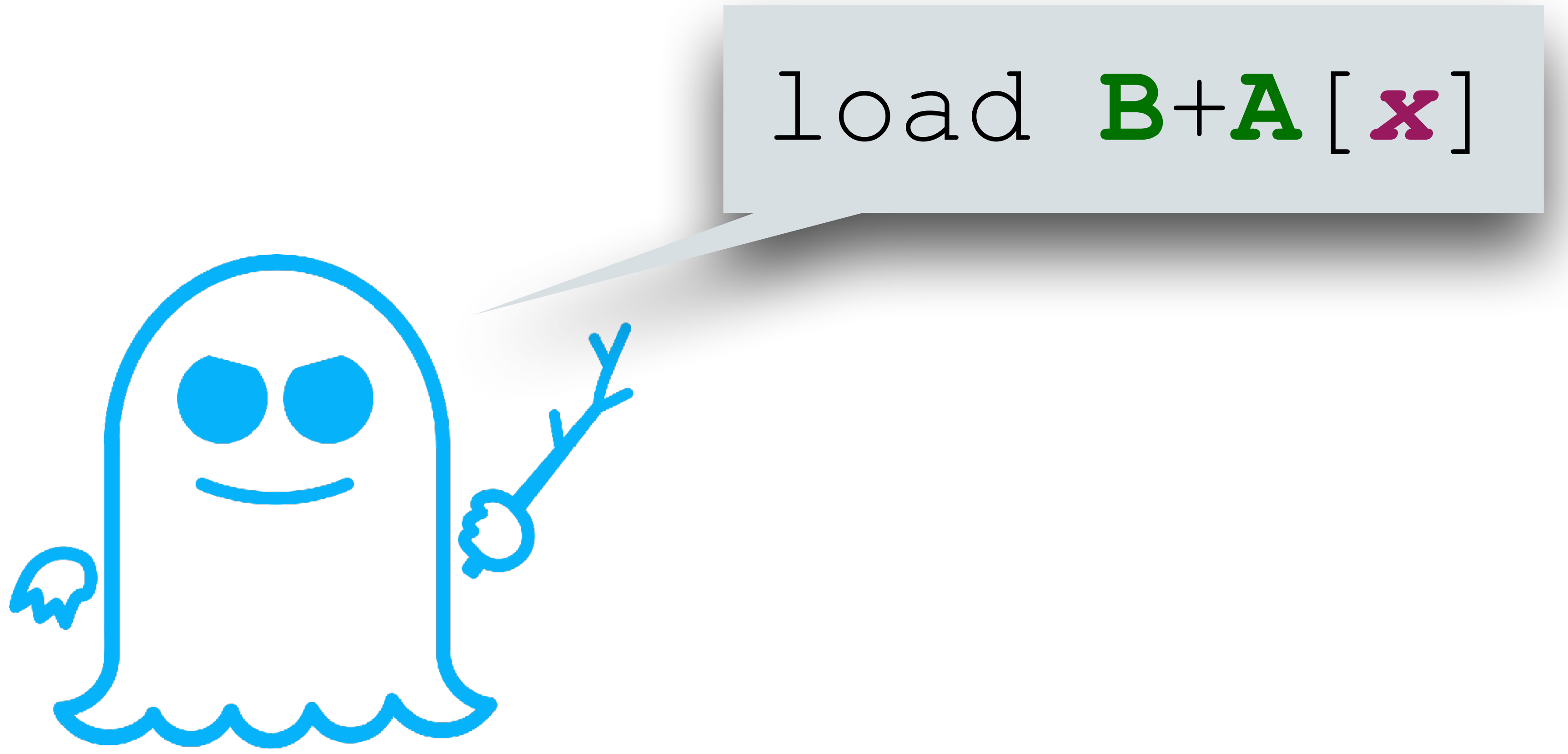
Inspired by “constant-time” rqmts



Leakage into μarchitecture

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





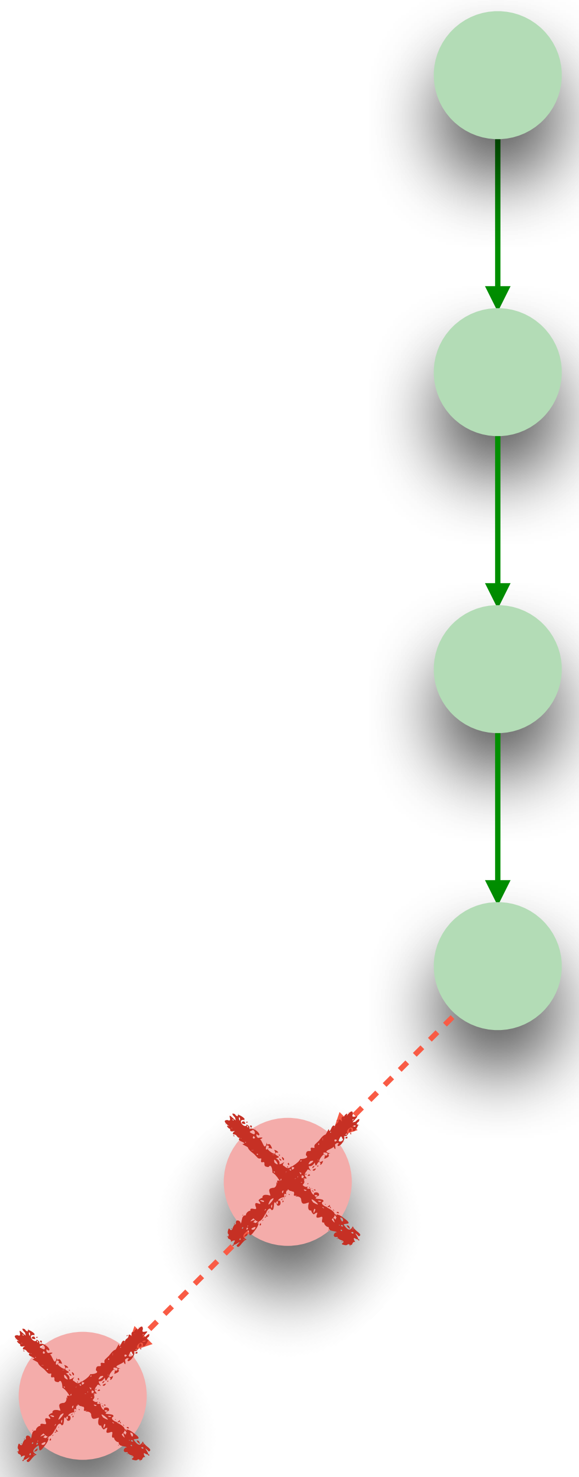
Attacker can observe:

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

Inspired by “constant-time” rqmts

Leakage into μarchitecture

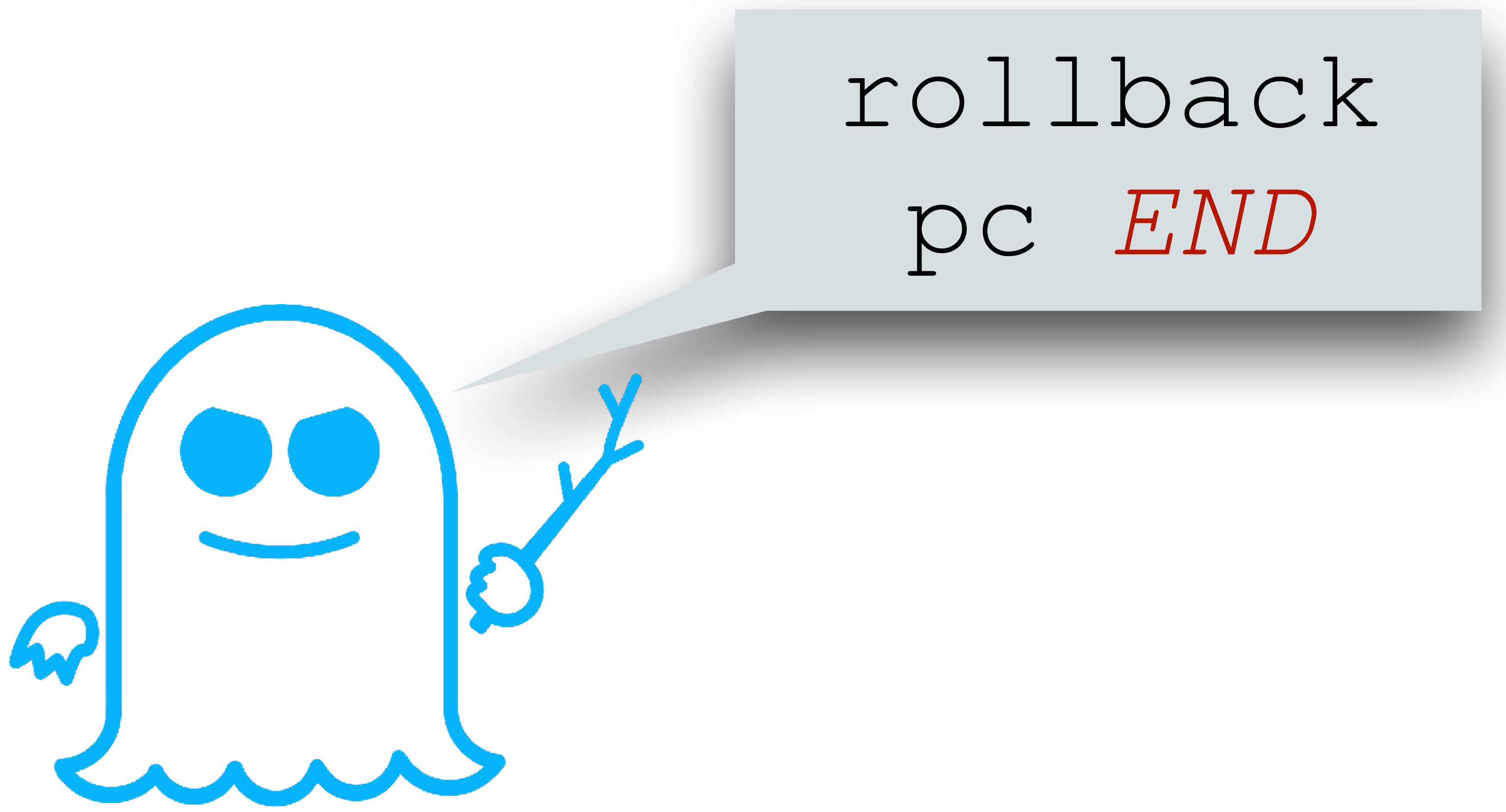
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



Attacker can observe:

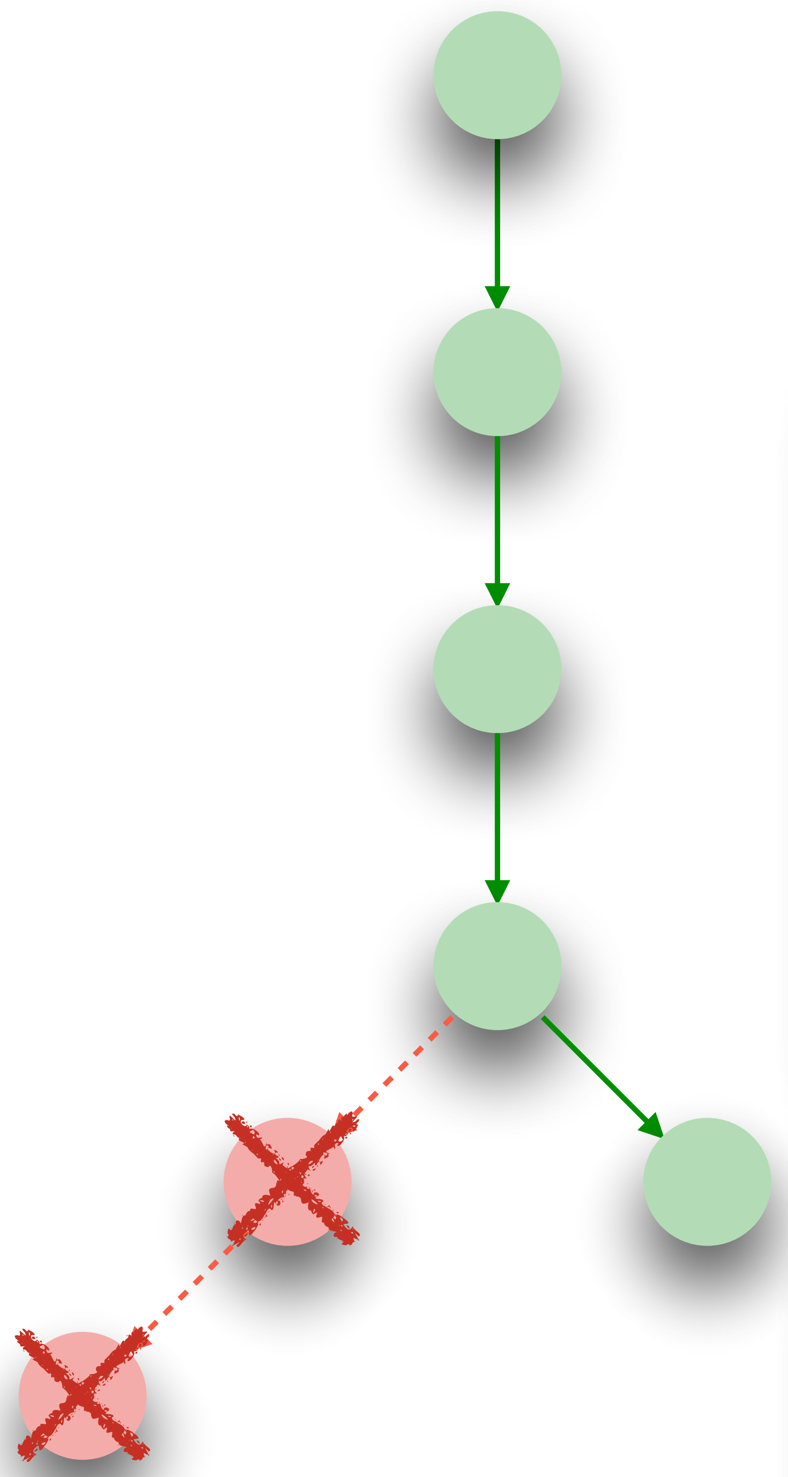
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

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

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':

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

Speculative non-interference

Formally!

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

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

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

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***

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

If program $\mathbf{P}$ satisfies

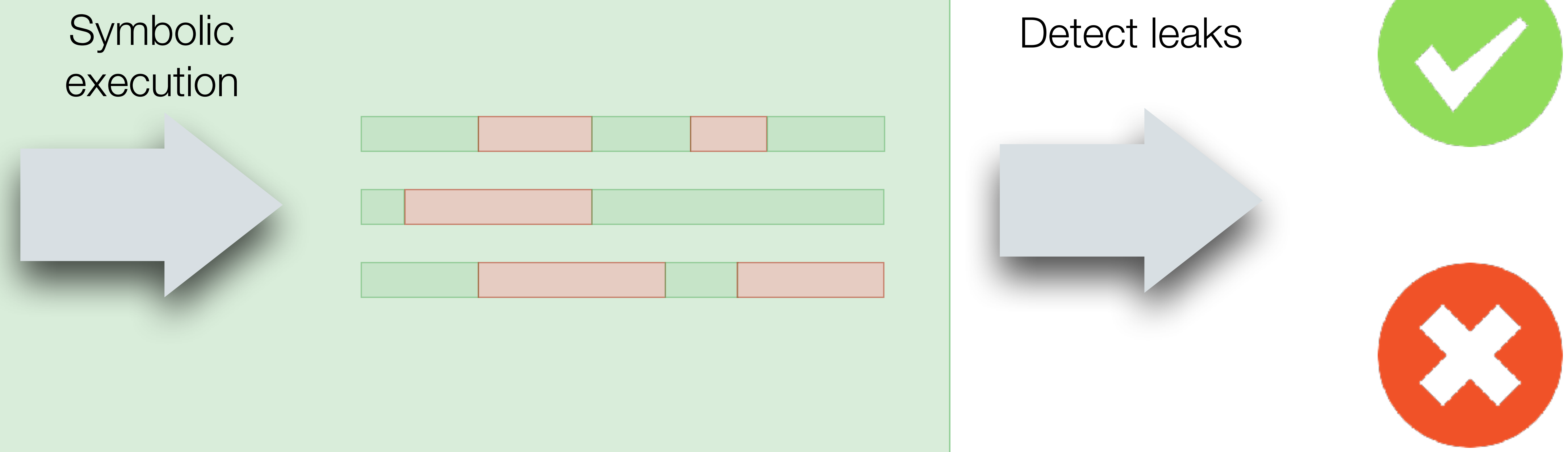
$$\forall s, s'.\ \mathbf{P}_{\mathtt{non\text{-}spec}}(s) = \mathbf{P}_{\mathtt{non\text{-}spec}}(s') \Rightarrow \mathbf{P}_{\mathtt{am}}(s) = \mathbf{P}_{\mathtt{am}}(s')$$

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

Detecting speculative leaks

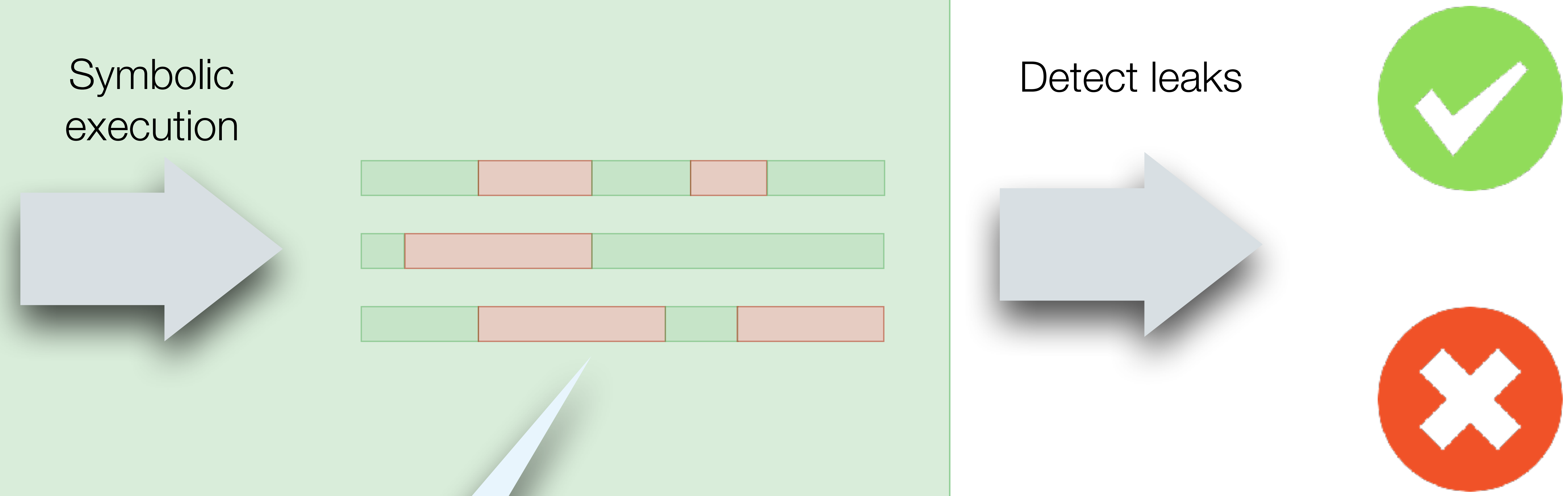
Detecting speculative leaks

```
     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

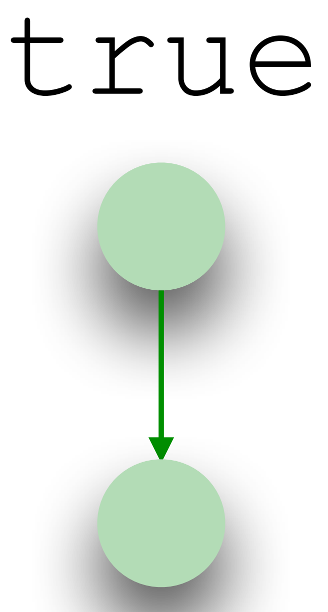
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```

true

Always mispredict
branch instructions

Symbolic execution

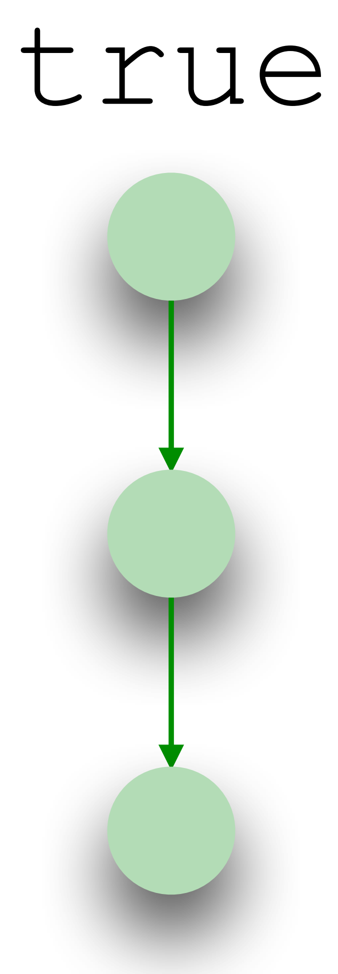
```
    rax <- A_size
    rcx <- x
    jmp rcx≥rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```



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:
```



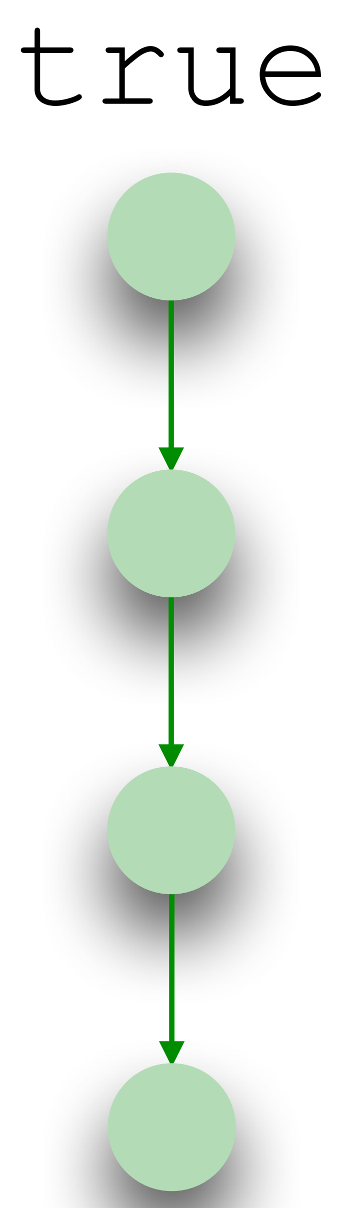
Always mispredict

branch instructions

Symbolic execution

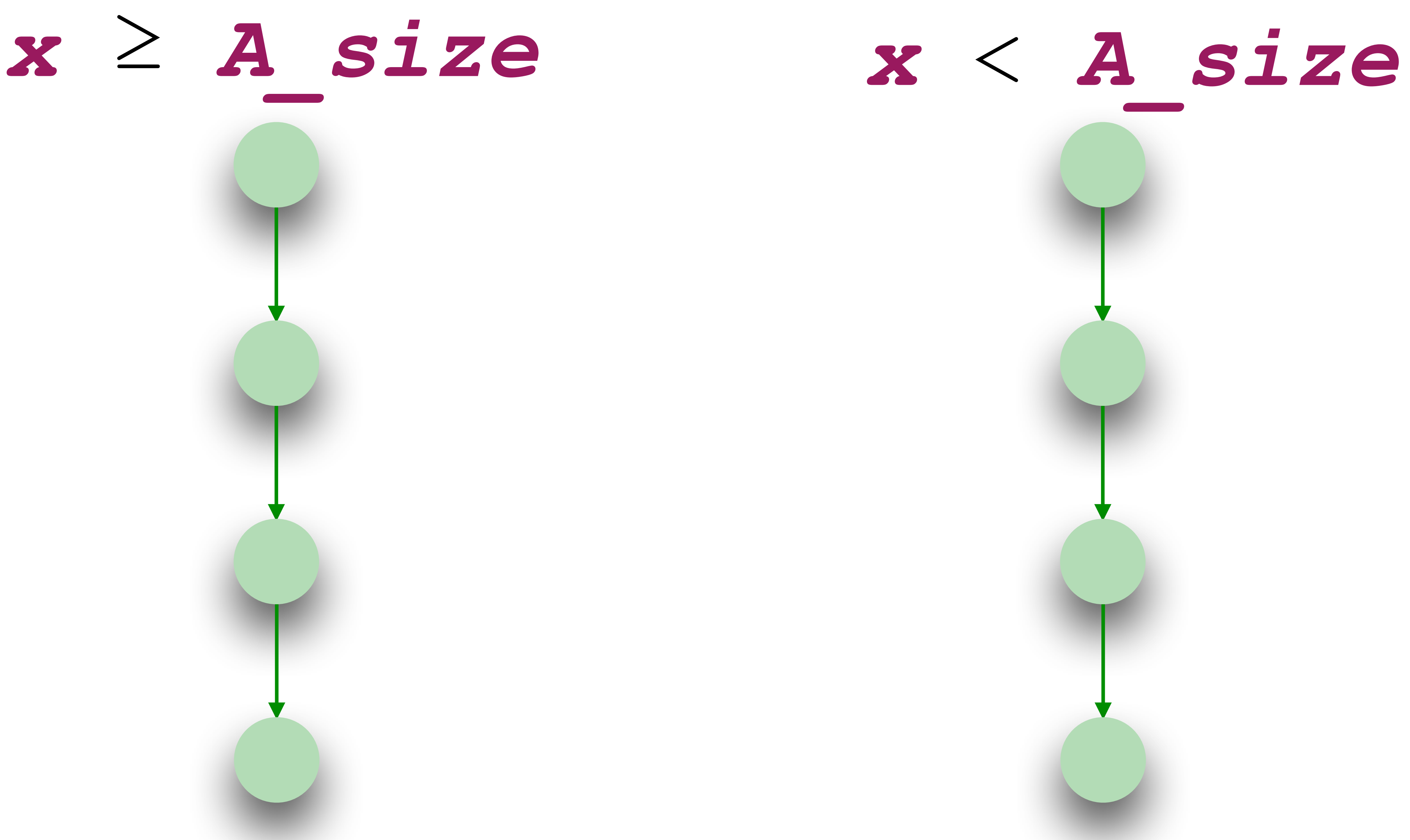
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```

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:
```

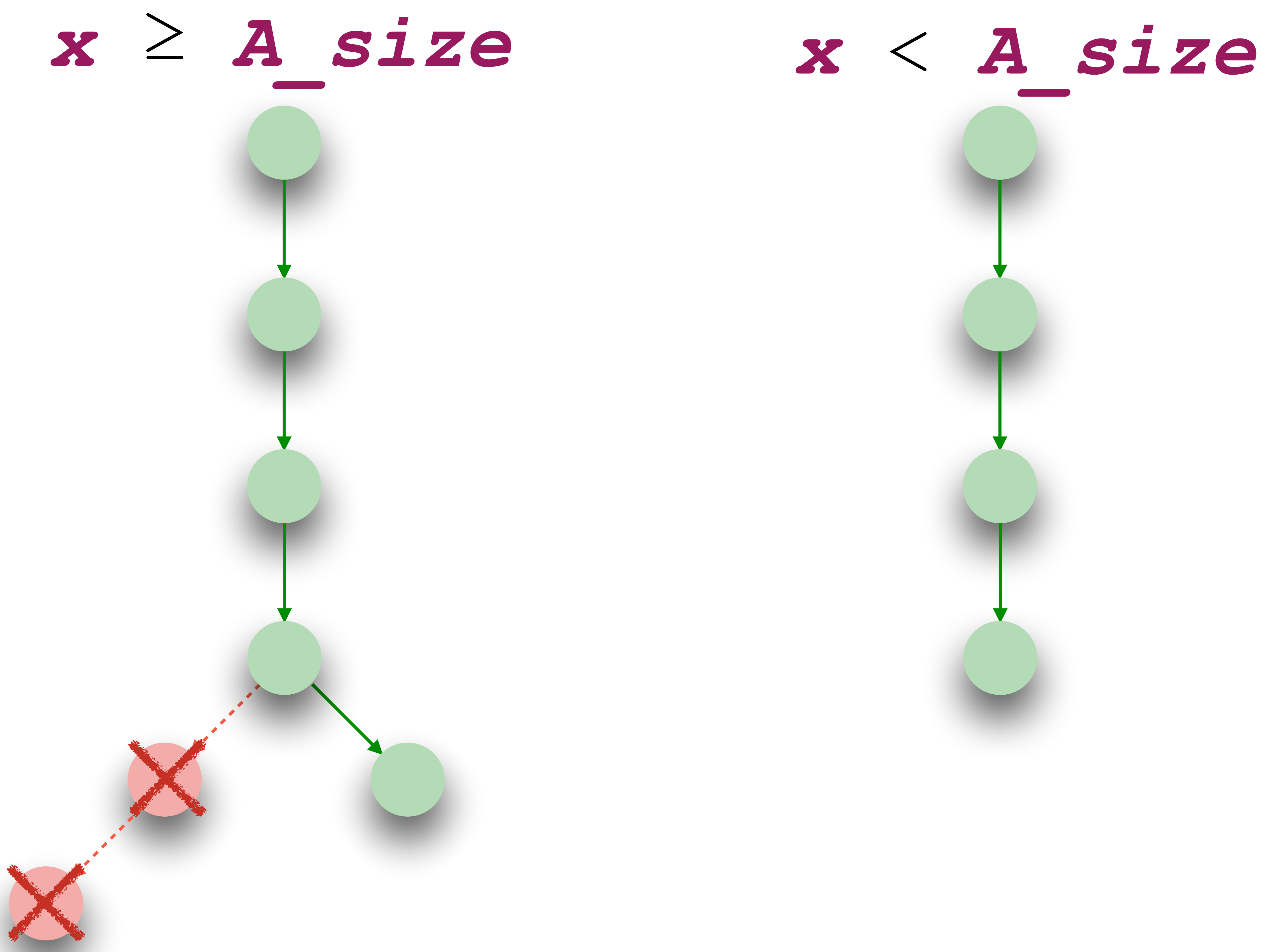


Always mispredict
branch instructions

Symbolic execution

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

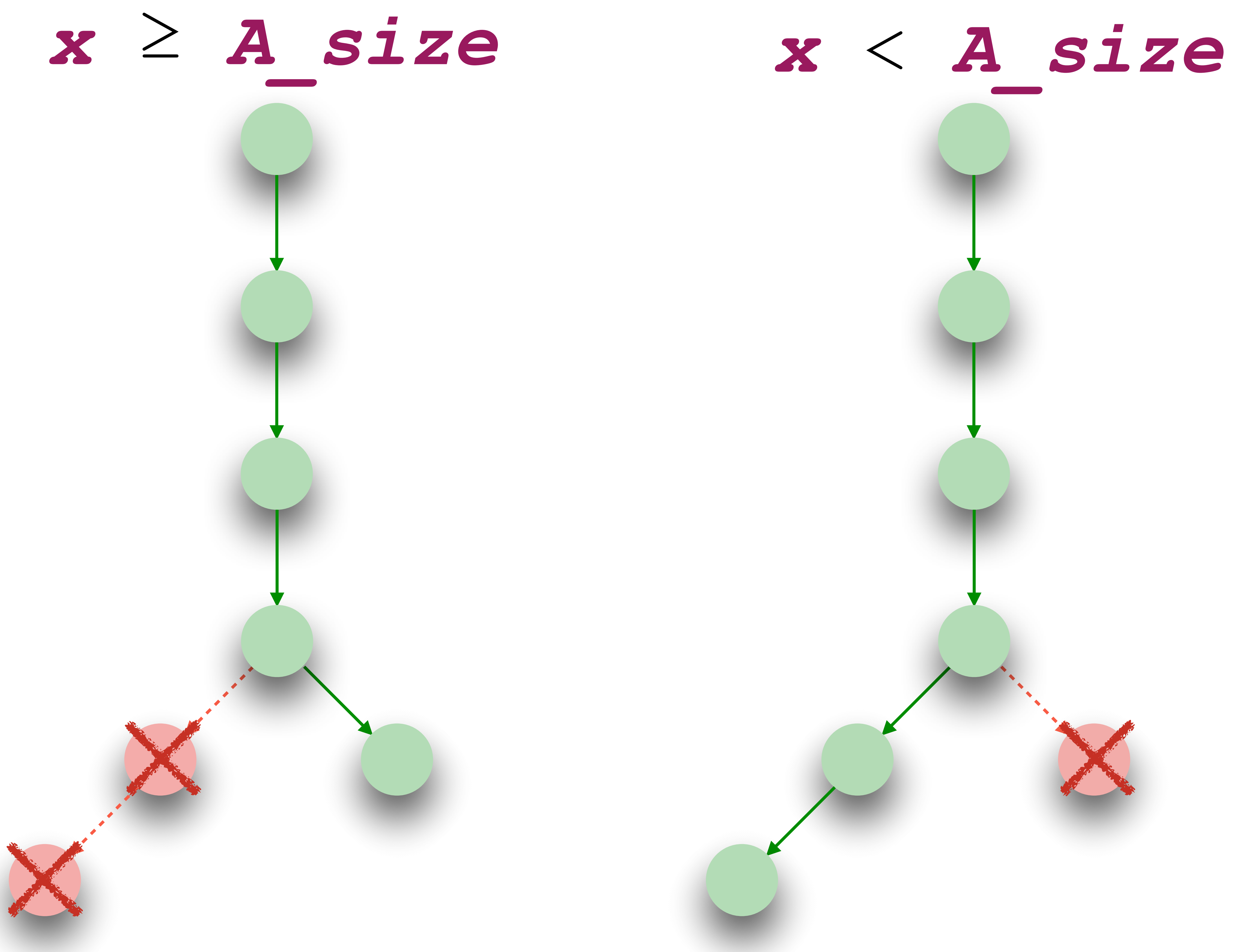
Always mispredict
branch instructions



Symbolic execution

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

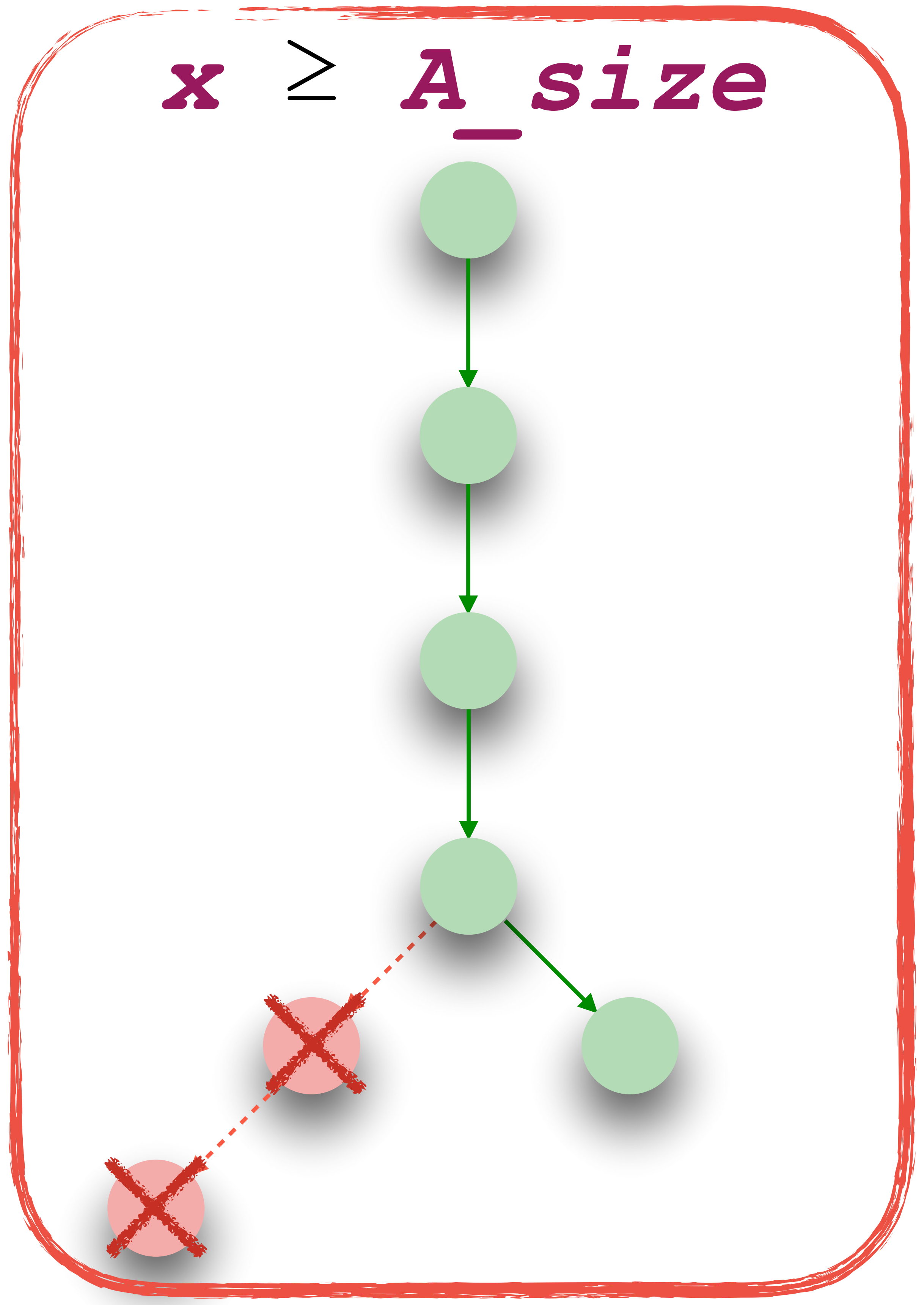
Always mispredict
branch instructions

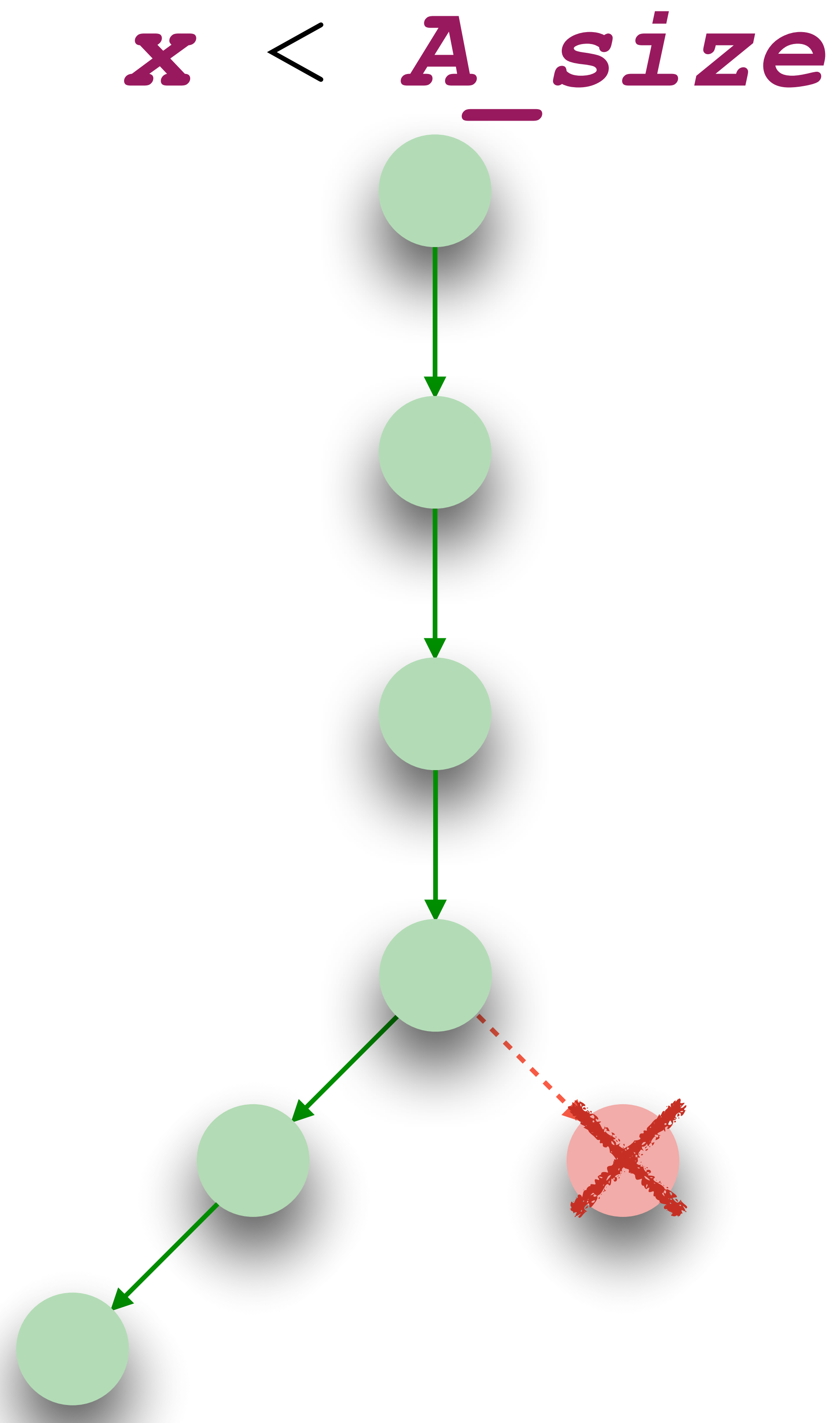


Symbolic execution

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

Always mispredict
branch instructions

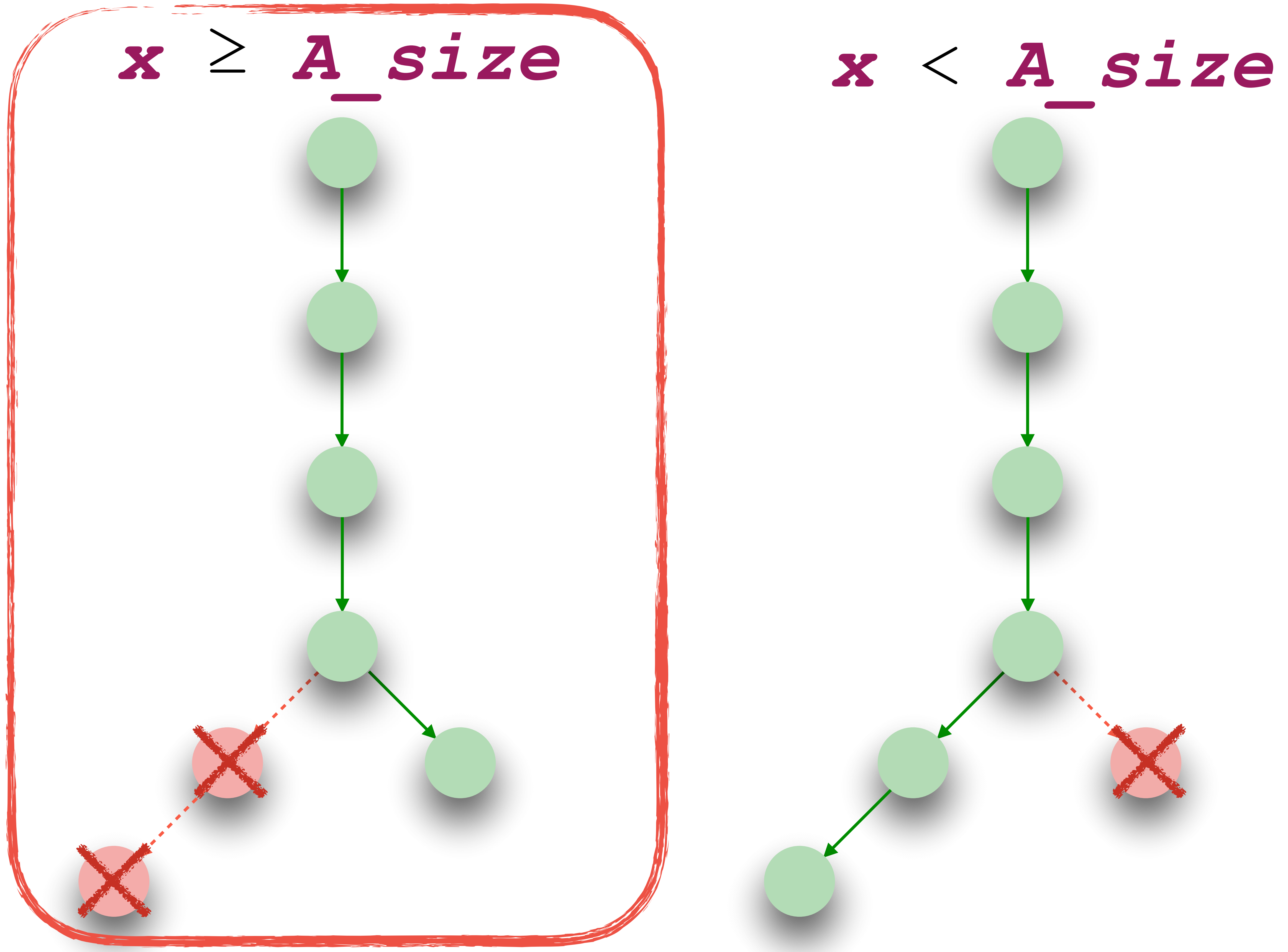




Symbolic execution

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

Always mispredict branch instructions

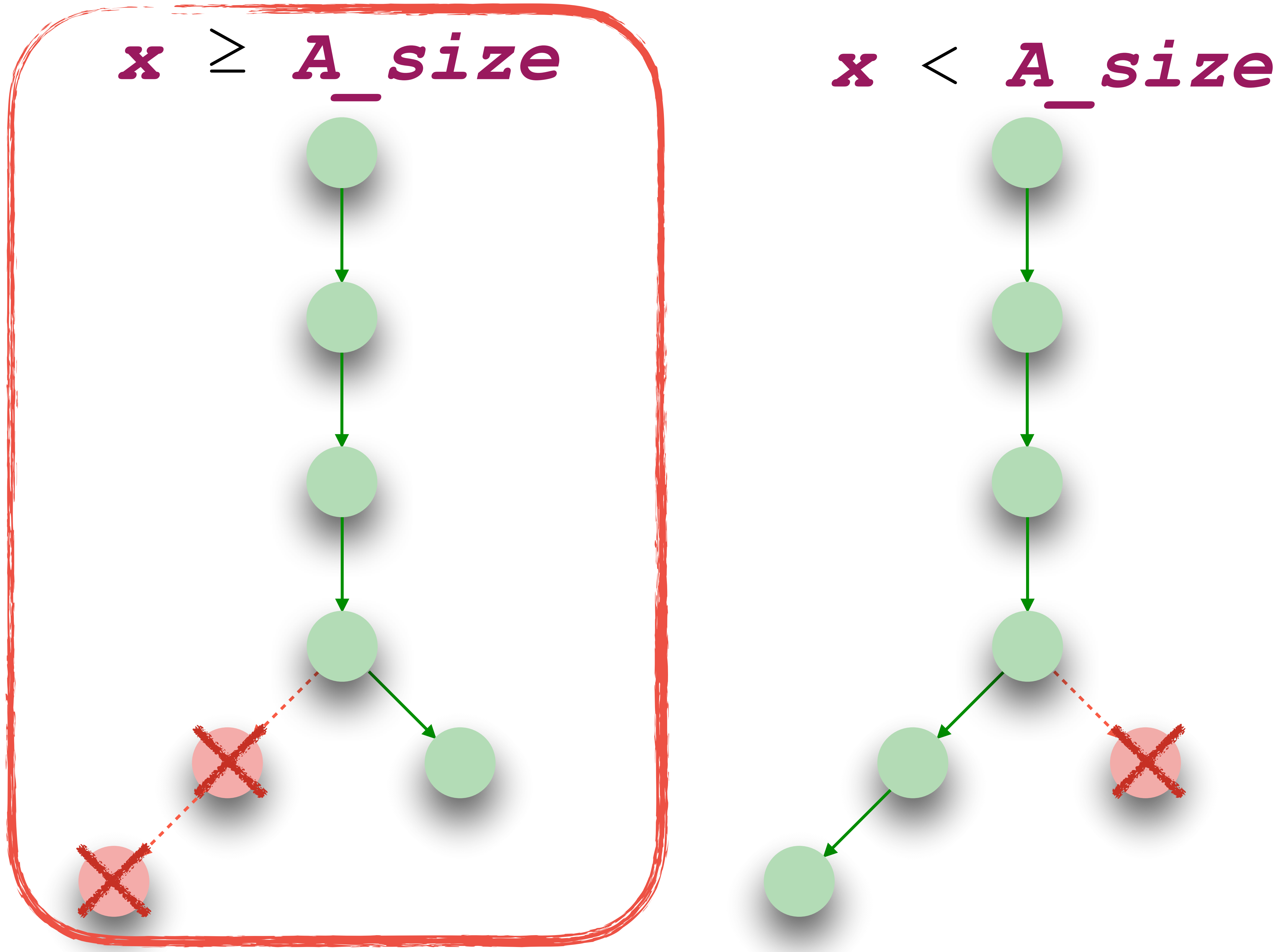


start pc *L1* load **A**+***x*** load **B**+**A**[***x***] rollback pc *END*

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

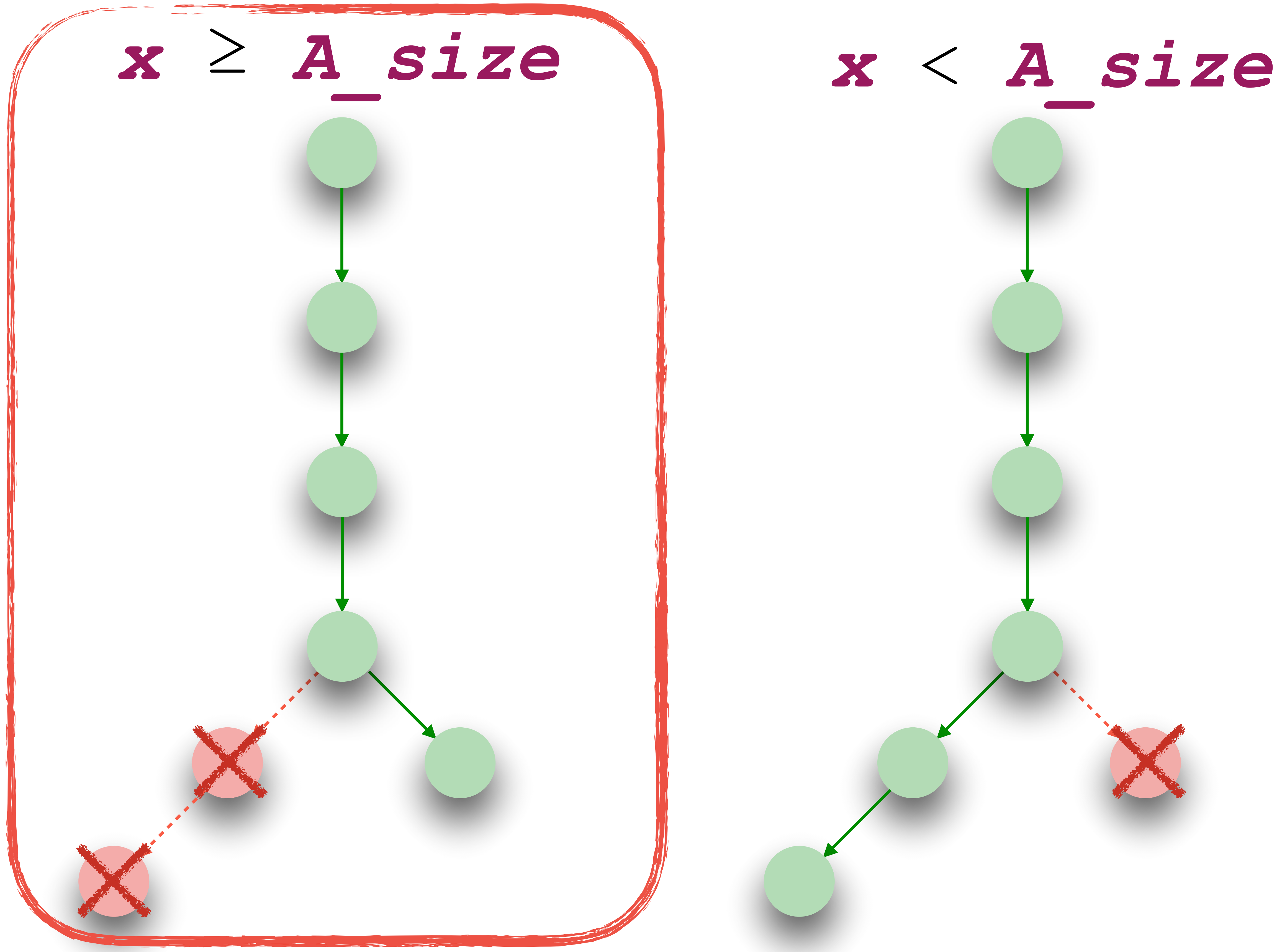


start pc *L1* load **A**+***x*** load **B**+**A**[***x***] rollback pc *END*

Symbolic execution

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

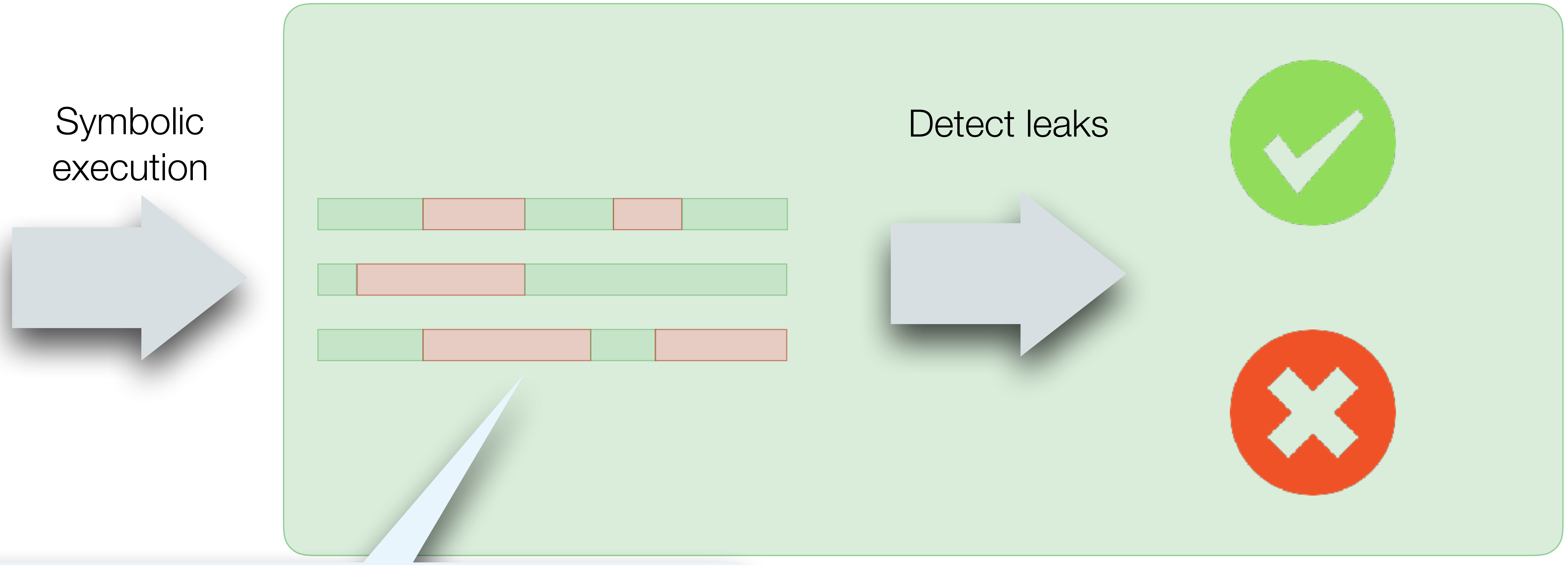
Always mispredict branch instructions



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 $MemLeak(\tau)$ **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

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

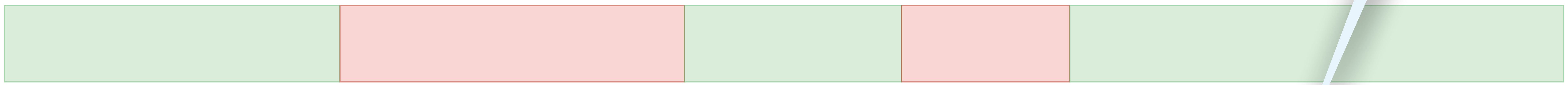
Memory leaks

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

- Non-sensitive information
- Non-speculative observations

Check with self-composition

τ



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

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

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

Equivalent wrt **`policy`**

s_1

s_2

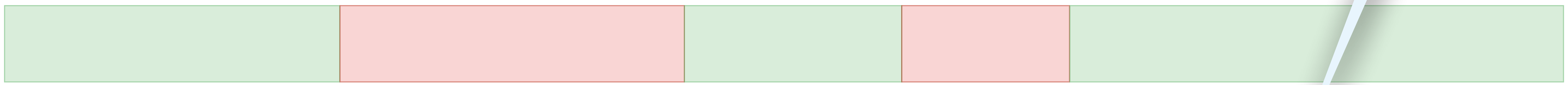
Memory leaks

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

- Non-sensitive information
- Non-speculative observations

Check with self-composition

τ



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

Equivalent wrt `policy`

$$s_1 \models \varphi$$

$$s_2 \models \varphi$$

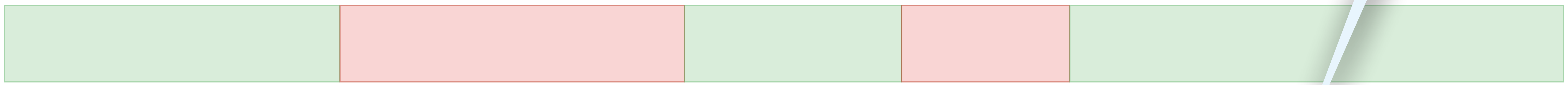
Memory leaks

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

- Non-sensitive information
- Non-speculative observations

Check with self-composition

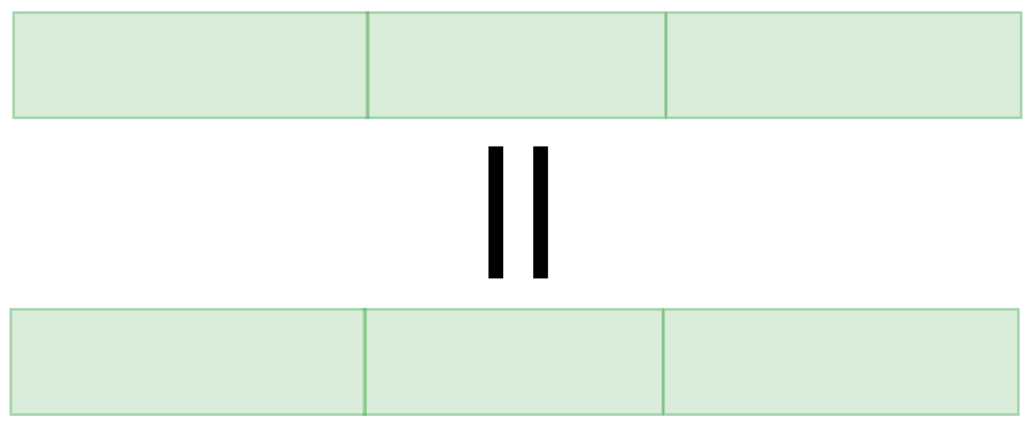
τ



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

Equivalent wrt **`policy`**

$s_1 \models \varphi$



$s_2 \models \varphi$

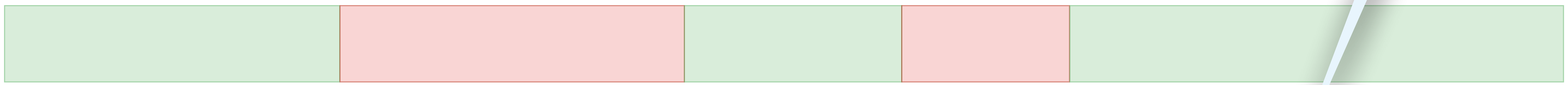
Memory leaks

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

- Non-sensitive information
- Non-speculative observations

Check with self-composition

τ

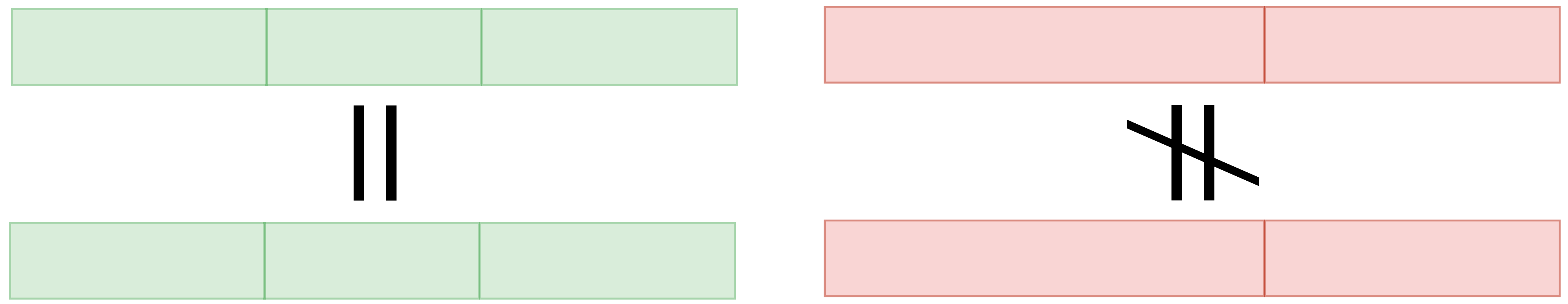


$$pathCnd(\tau) \land obsEqv(\tau|_{non-spec}) \land \neg obsEqv(\tau|_{spec})$$

Equivalent wrt `policy`

$s_1 \models \varphi$

$s_2 \models \varphi$



Spectector + Case studies

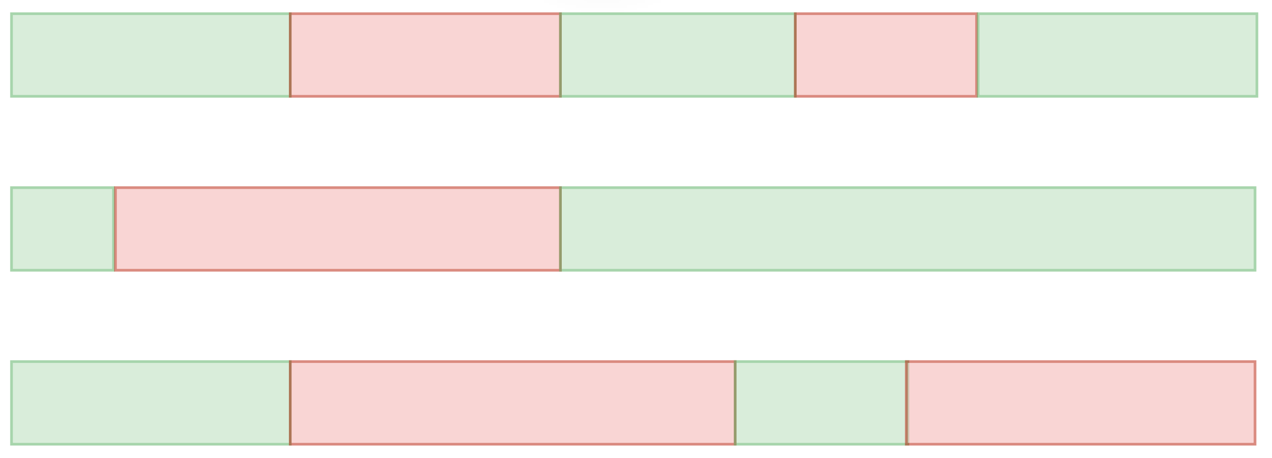
Spectector

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

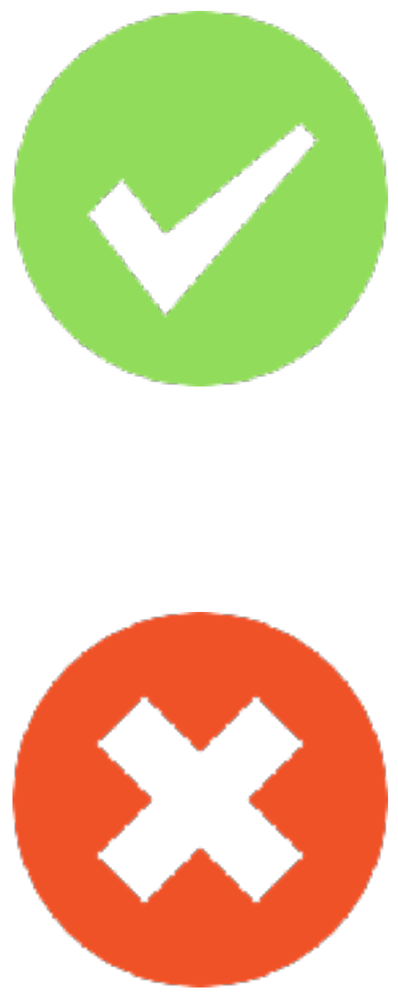
x64 to µASM

```
      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[
```

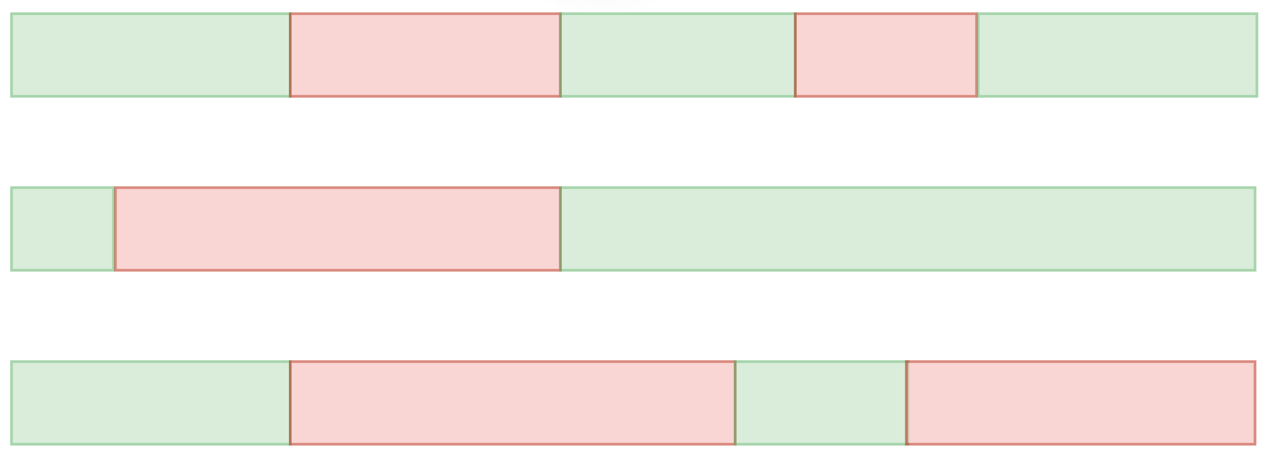
x64 to µASM

```
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 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

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:

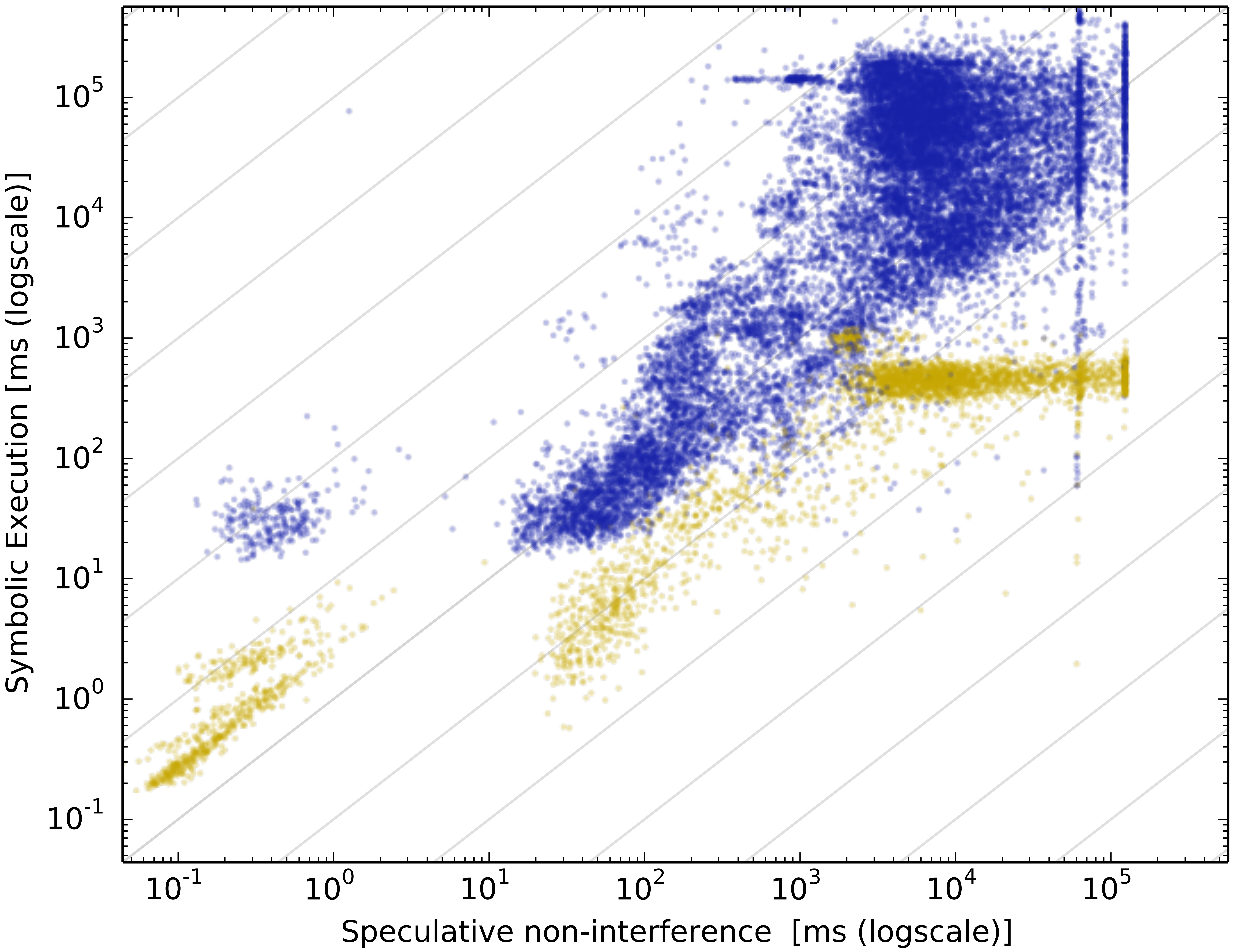
- Policy definition
- ISA coverage
- Path explosion

} 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)

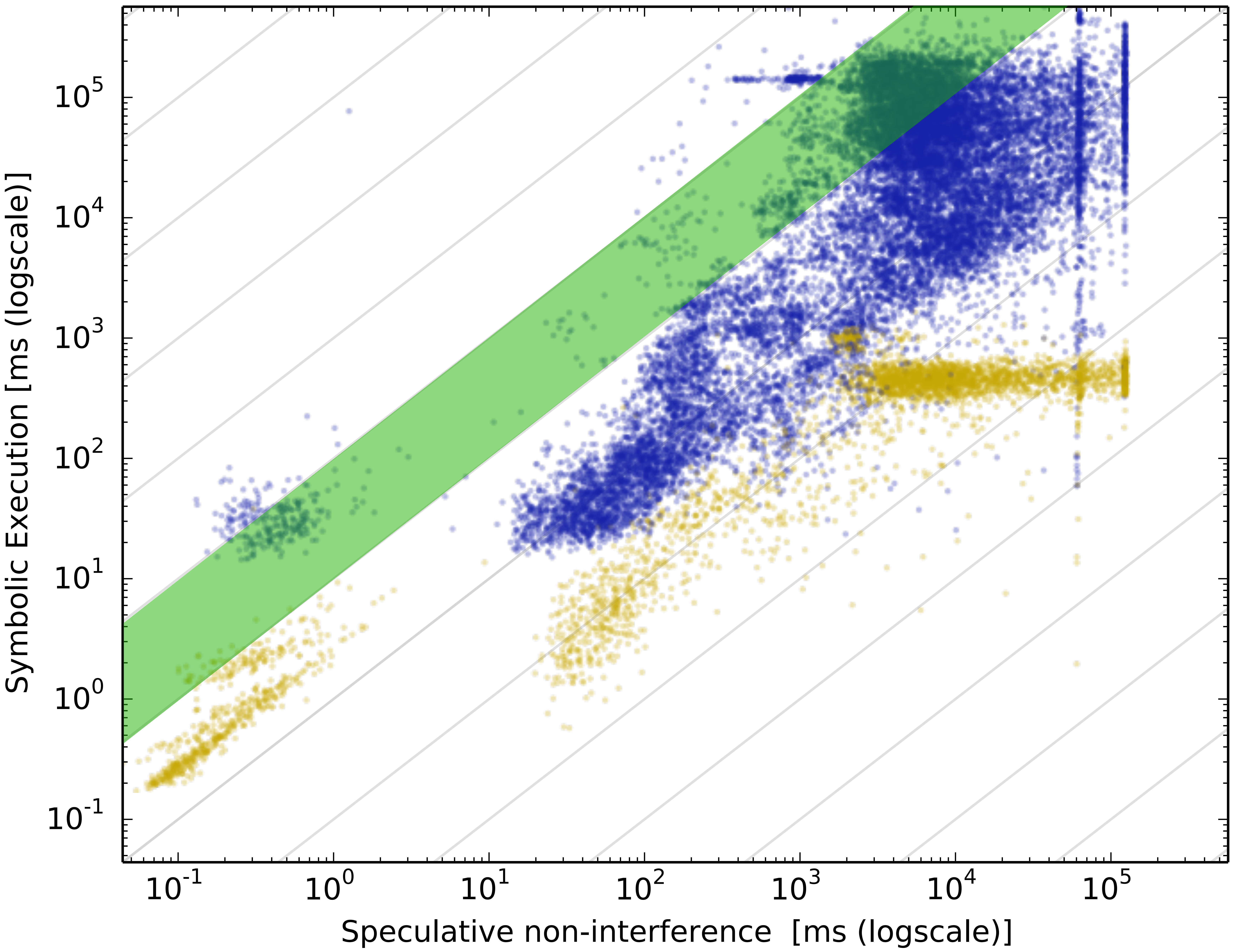
Results



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

Results

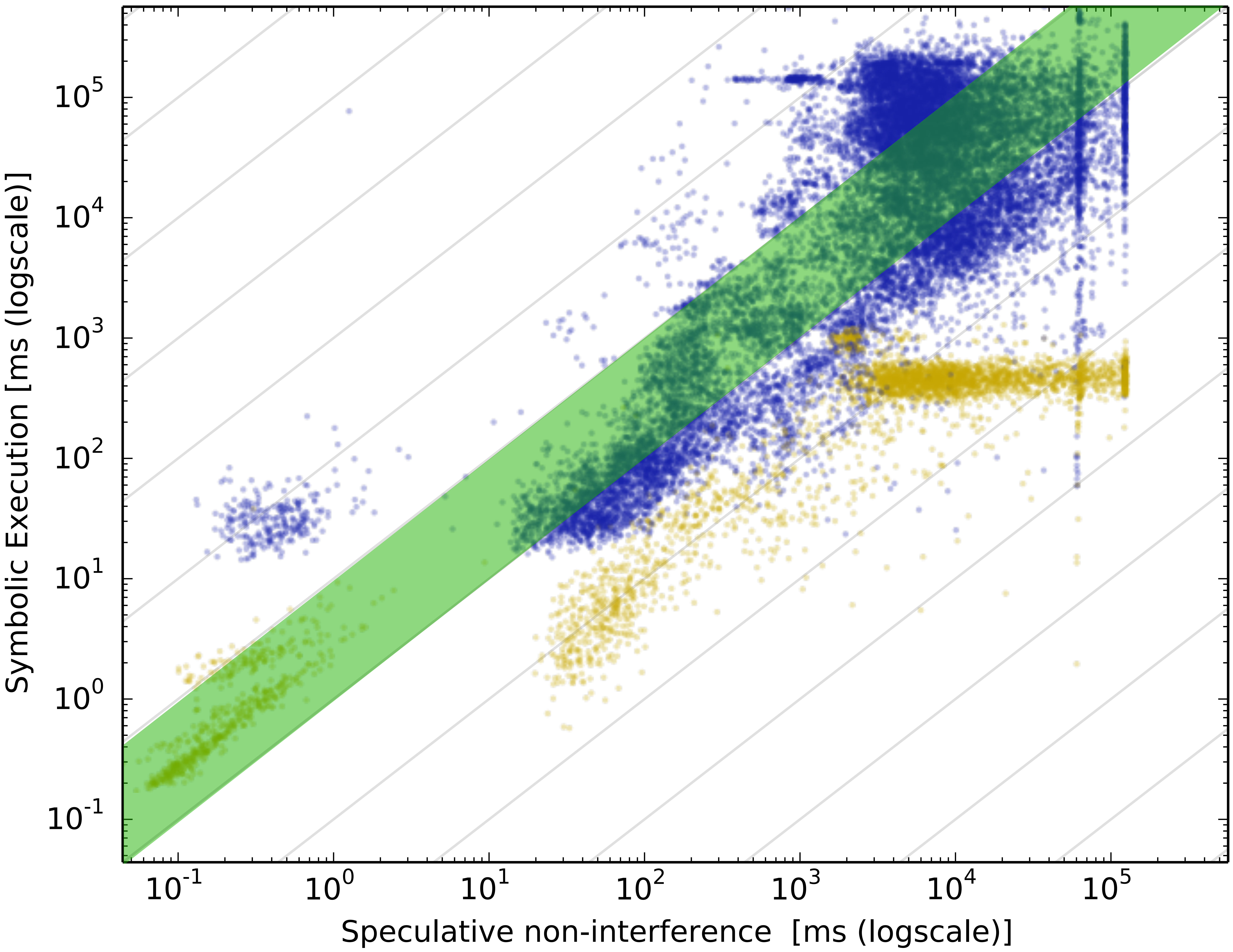
- SNI 10x-100x faster
  - 20.2% traces



Results

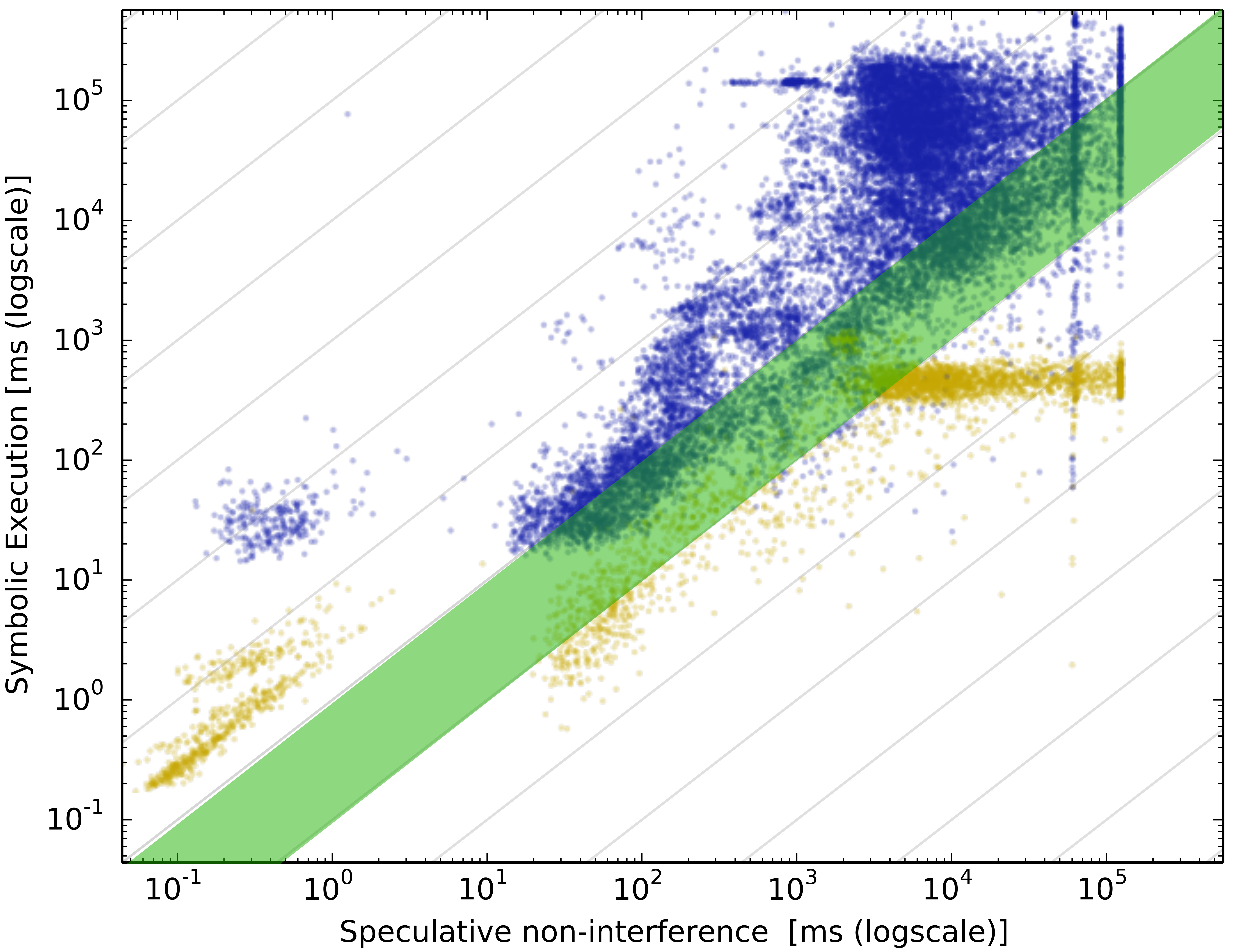
- SNI 10x-100x faster
  - 20.2% traces

- SNI ≤10x faster
  - 41.9% traces



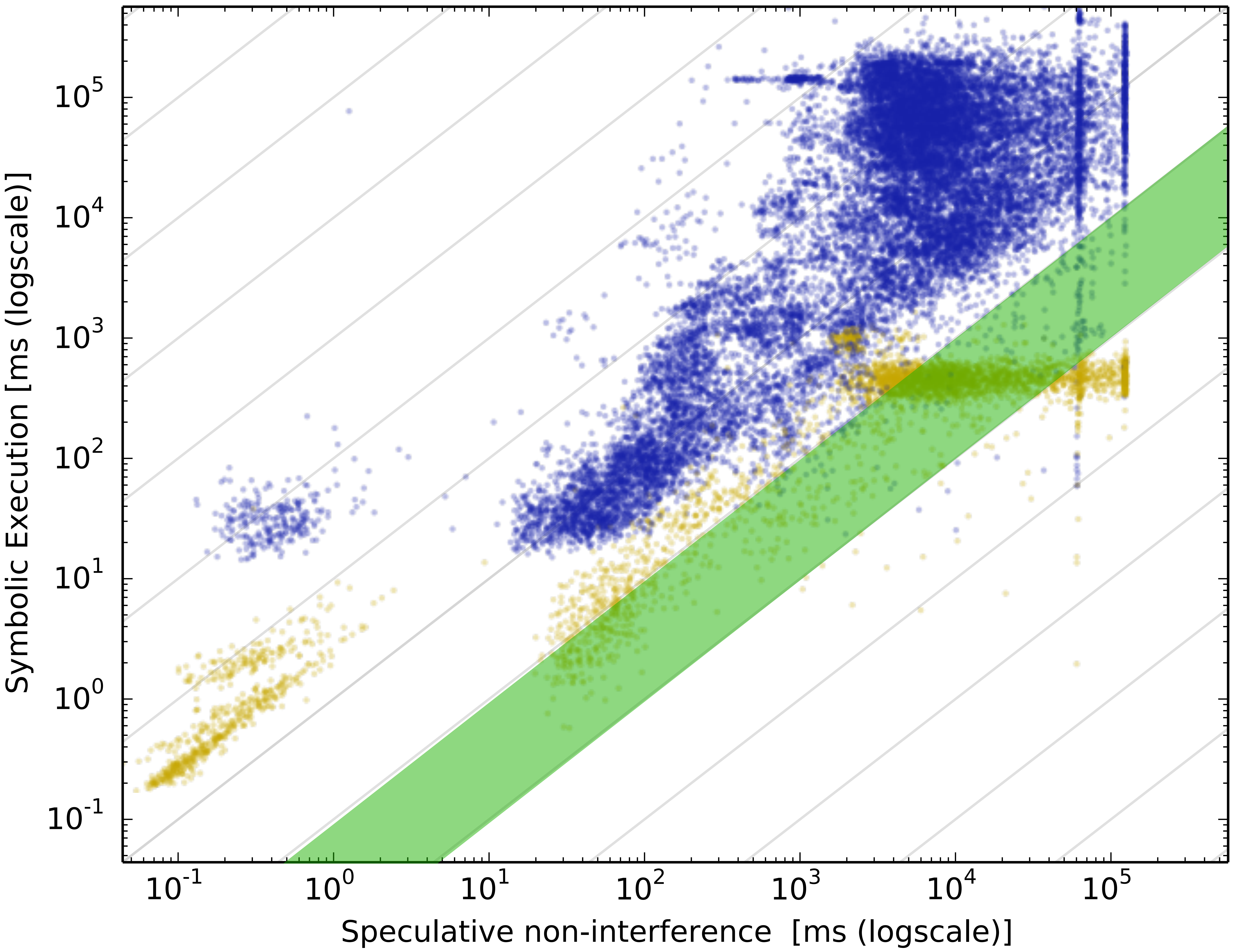
Results

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



Results

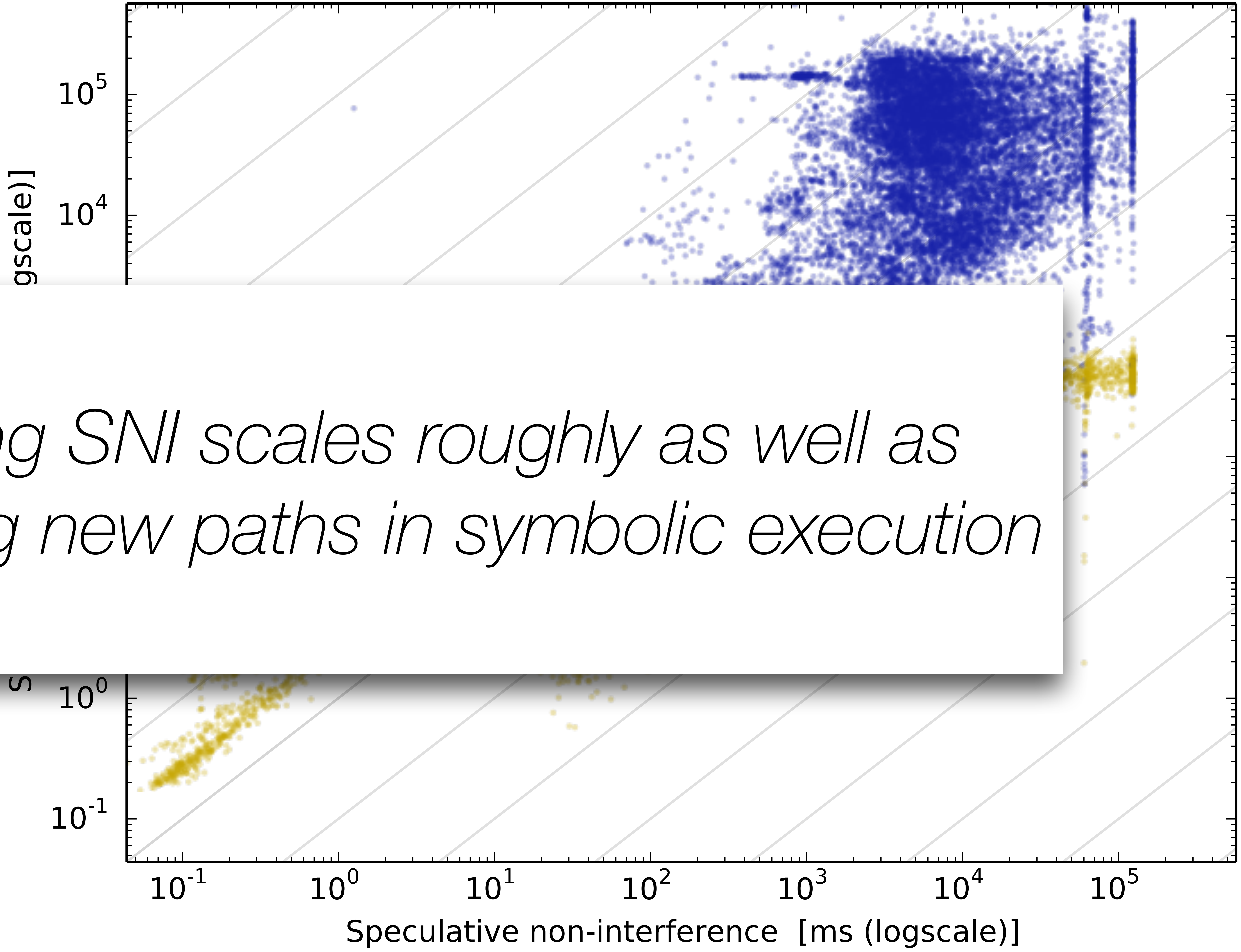
- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces
- SNI ≤10x slower
  - 26.9% traces
- SNI 10x-100x slower
  - 7.9% traces



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

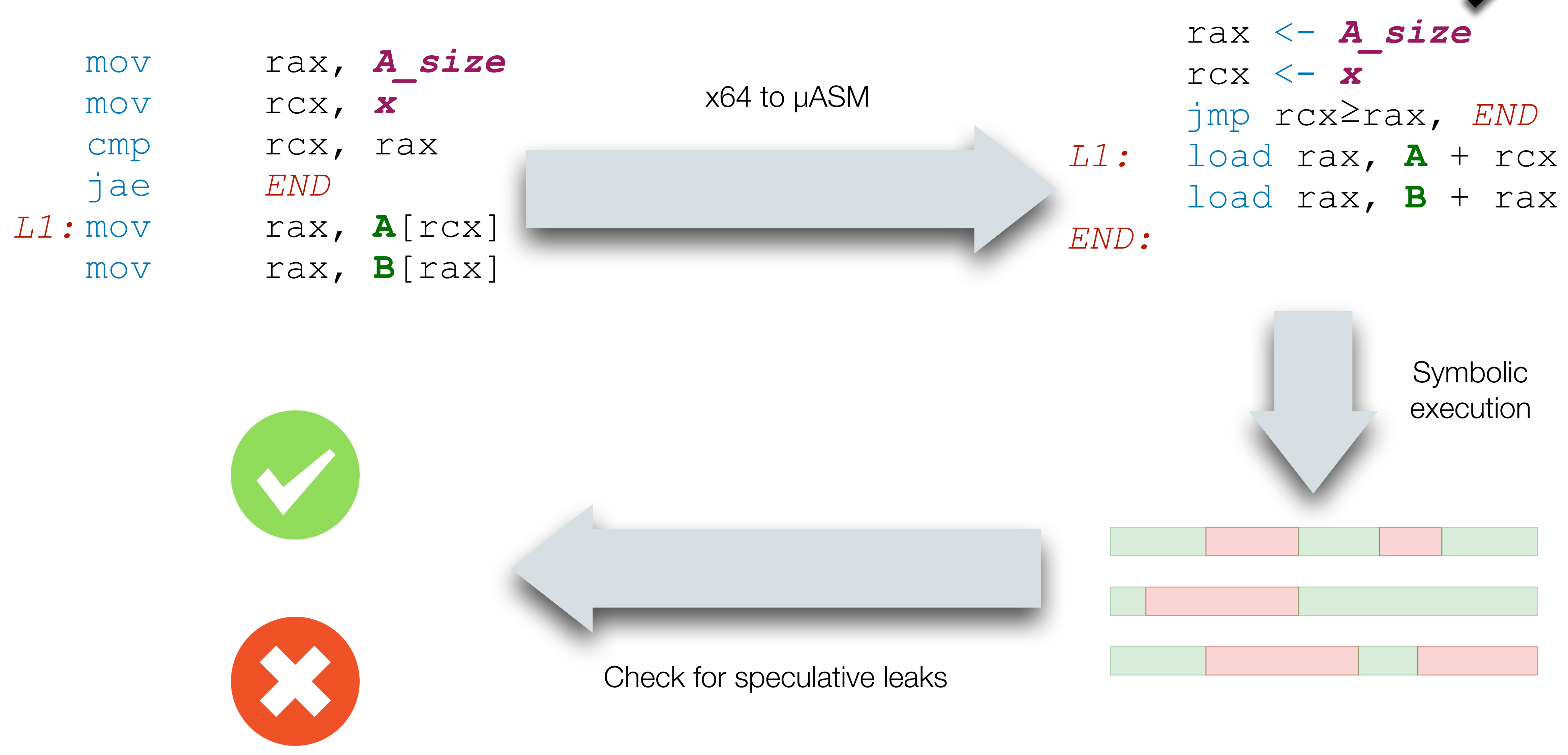
Formally!

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

For all program states s and s':

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

Spectector

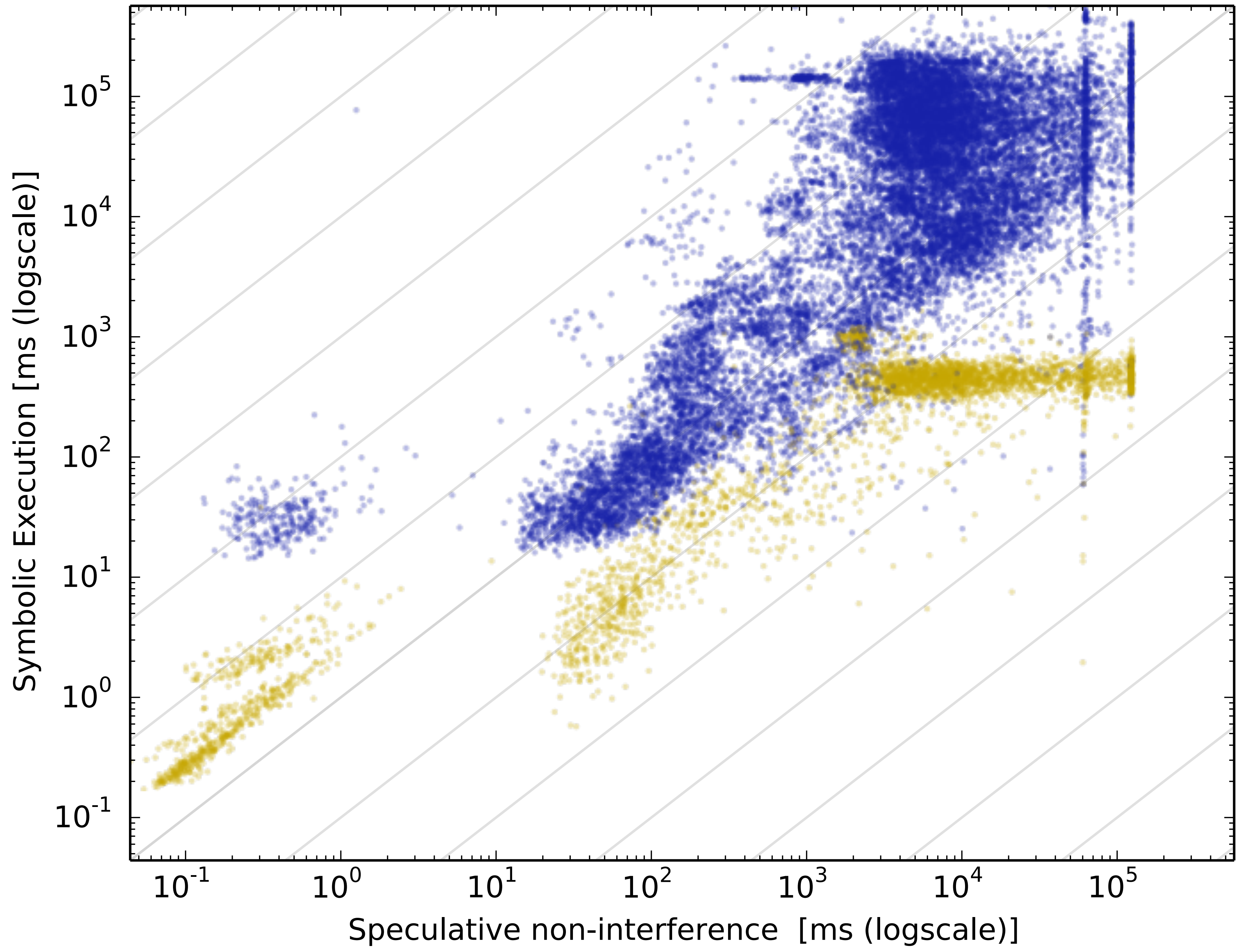


Results

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 all

P_{non}

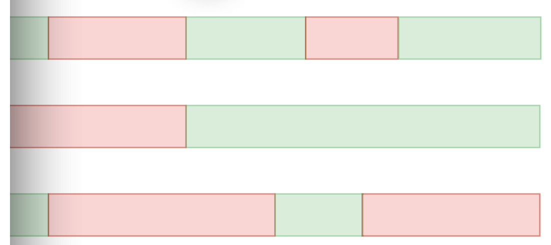
Spectector

```
    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