Seguridad Informatica - Fall 2024

Module IV: Physical security

Lecture 5

Reasoning about speculative execution attacks

Marco Guarnieri

IMDEA Software Institute

Module IV

- Lecture 1 — Introduction
- Lecture 2 — Cache-based side channel attacks
- Lecture 3 — Speculative execution attacks
- Lecture 4 — Non-interference
- Lecture 5 — Automated detection of speculative leaks

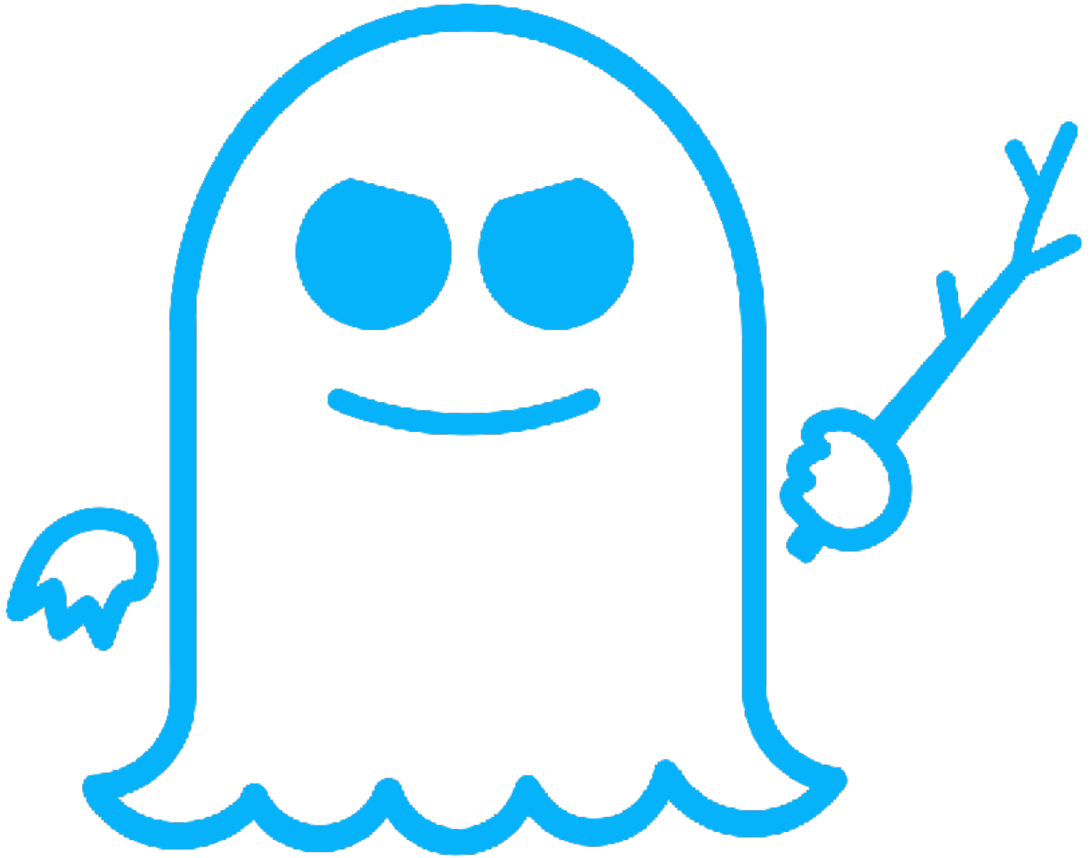
Module IV

- Lecture 1 — Introduction
- Lecture 2 — Cache-based side channel attacks
- Lecture 3 — Speculative execution attacks
- Lecture 4 — Non-interference
- Lecture 5 — Automated detection of speculative leaks

Recommended reading

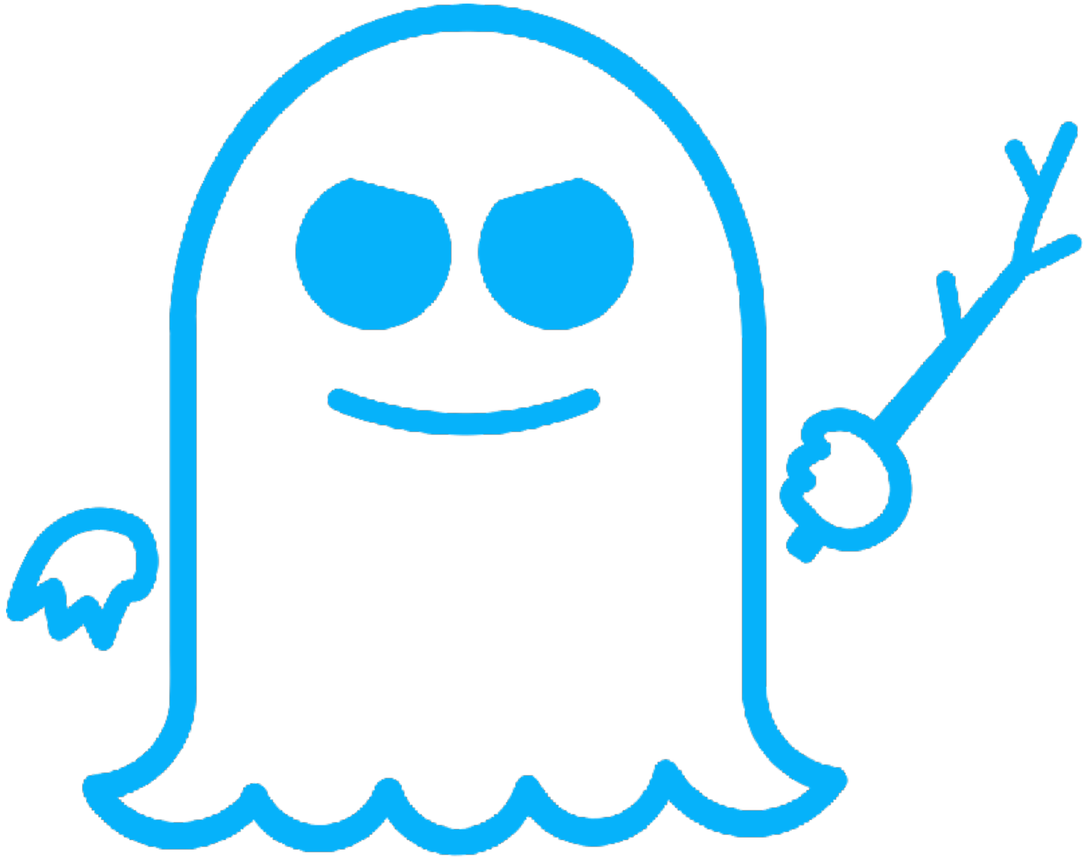
Spectector: Principled detection of speculative information flows by Marco Guarnieri, José F. Morales, Andrés Sánchez, Boris Köpf, Jan Reineke

Available at https://spectector.github.io/papers/spectector.pdf



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

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

Speculative execution attacks 101

Speculative execution + branch prediction

Speculative execution + branch prediction

Size of array **A**

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

Speculative execution + branch prediction

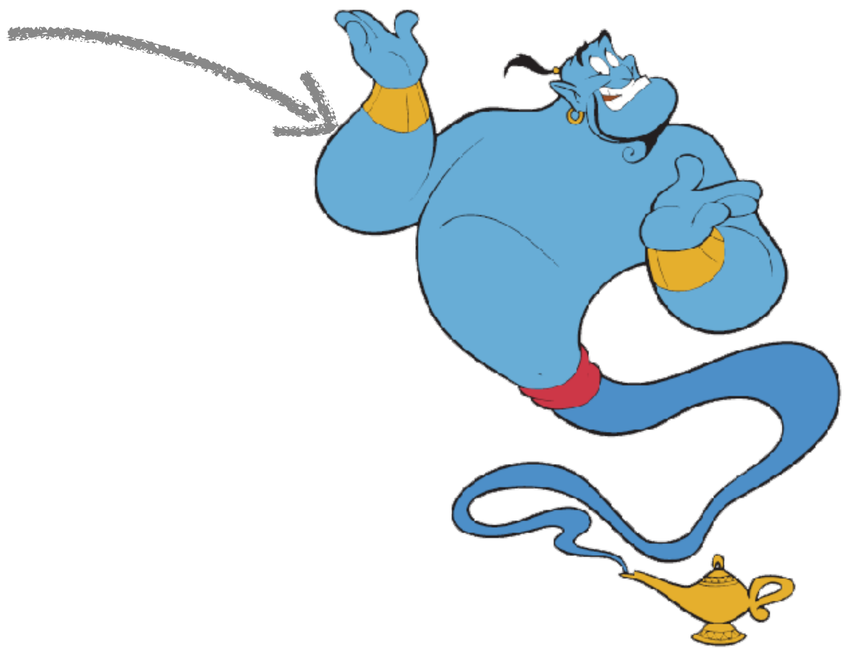
Size of array **A**

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

Speculative execution + branch prediction

Size of array **A**

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



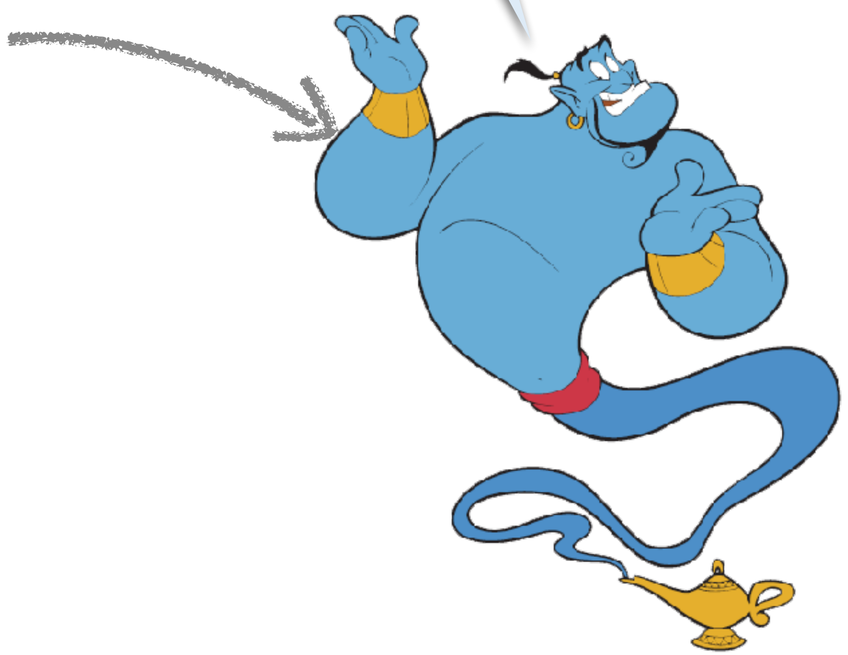
Branch predictor

Speculative execution + branch prediction

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

Size of array **A**

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



Branch predictor

Speculative execution + branch prediction

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

Size of array **A**

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

Branch predictor

Speculative execution + branch prediction

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

Size of array **A**

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

Branch predictor

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

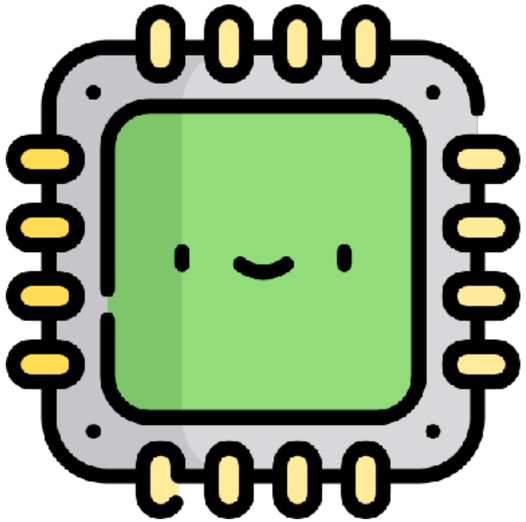
- ✓ Architectural (ISA) state
- ✗ Microarchitectural state

Spectre V1

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

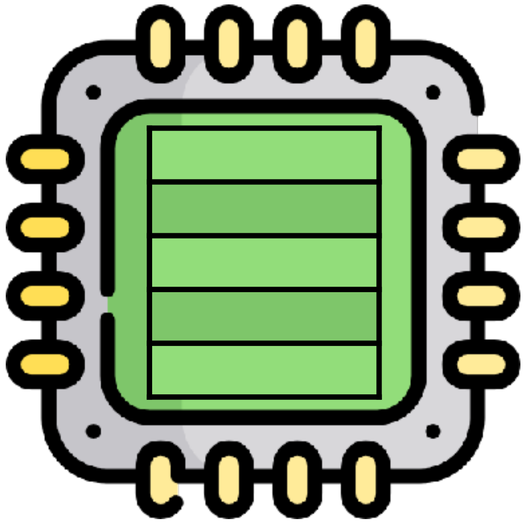
Spectre V1

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



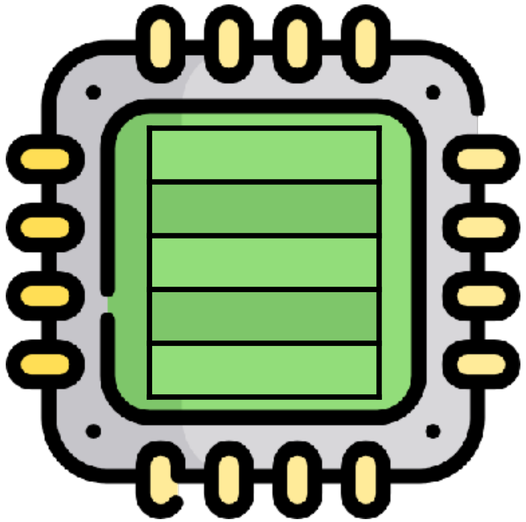
Spectre V1

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

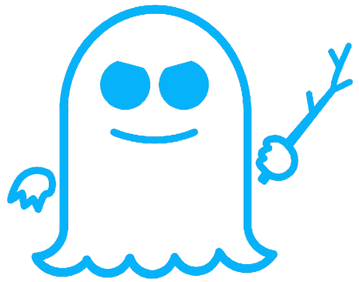


Spectre V1

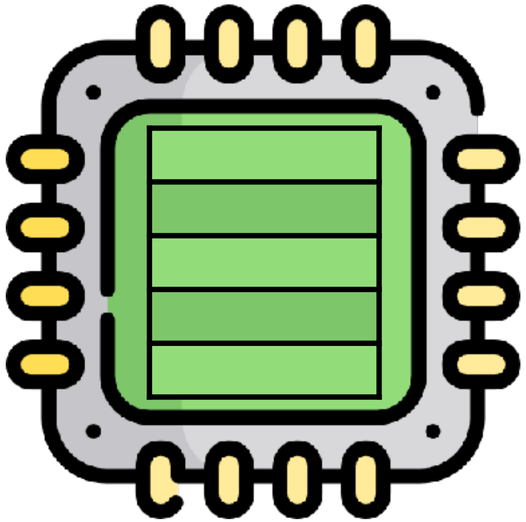
```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```



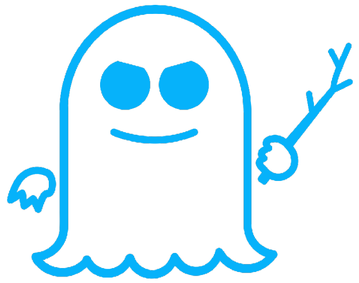
Spectre V1

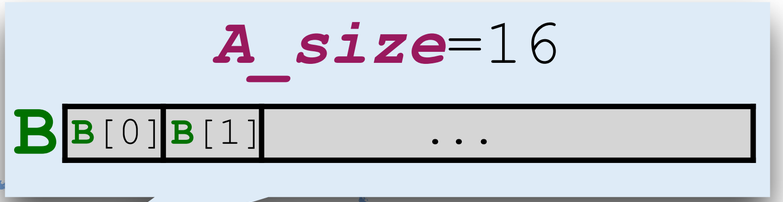


```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```

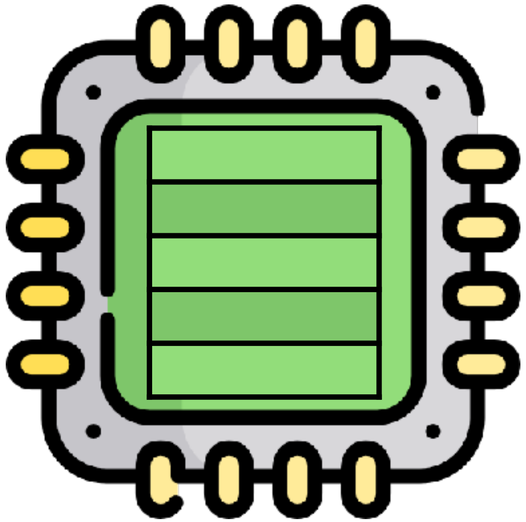


Spectre V1

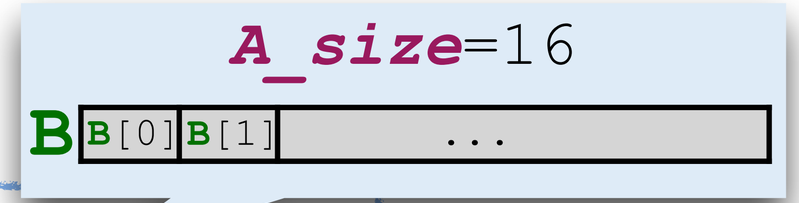




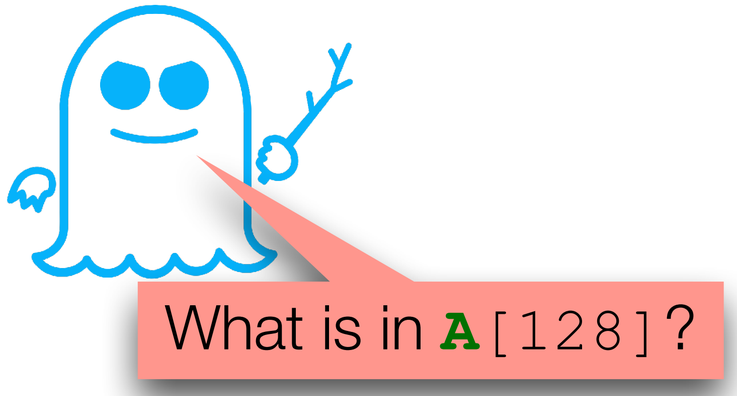

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

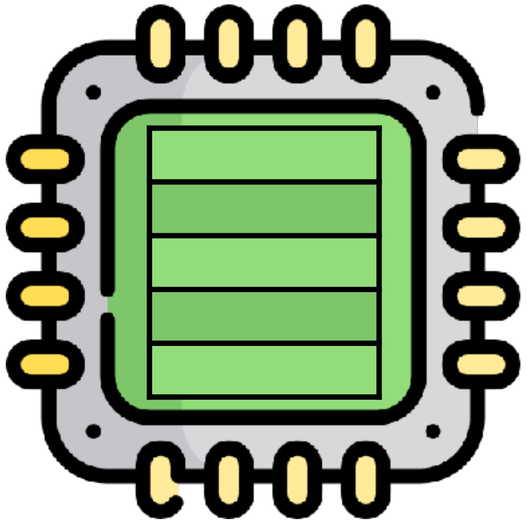


Spectre V1




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

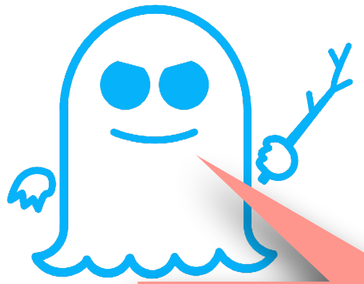




Spectre V1

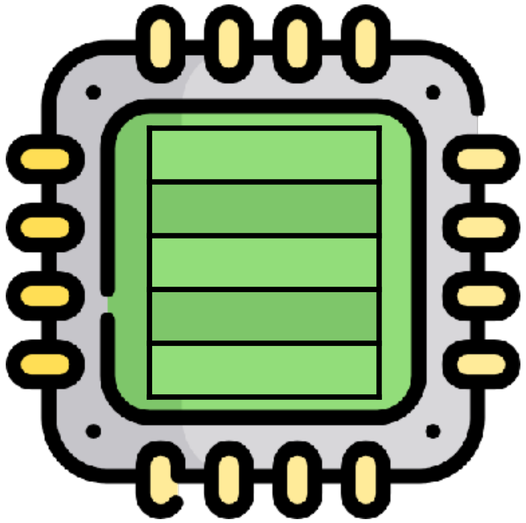
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

A_size=16

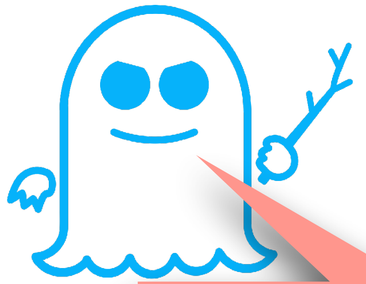


What is in A[128]?

1) Training



Spectre V1



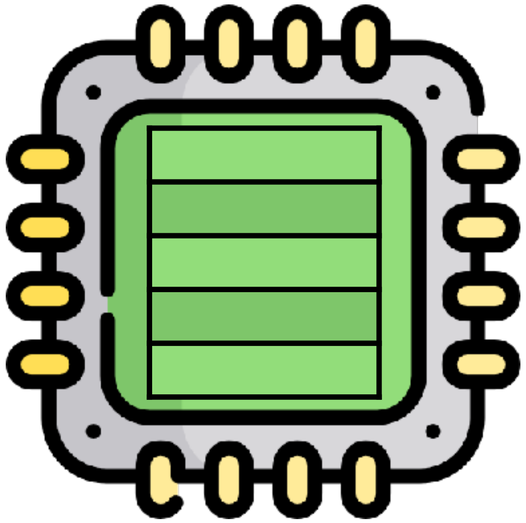
What is in `A[128]`?

`A_size`=16

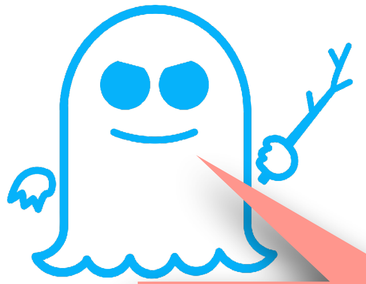
`B` `B[0]` `B[1]` `...`

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

1) Training 



Spectre V1

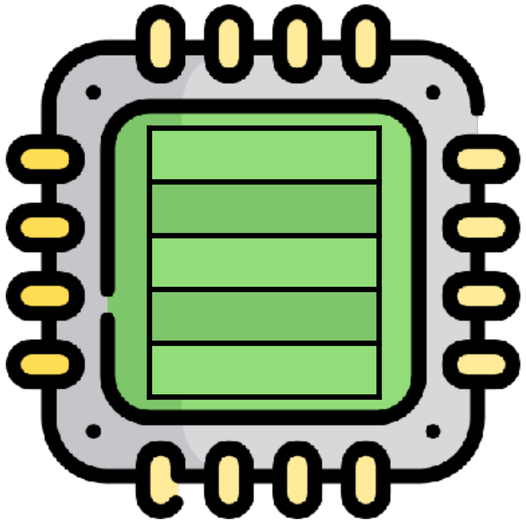


What is in `A[128]`?

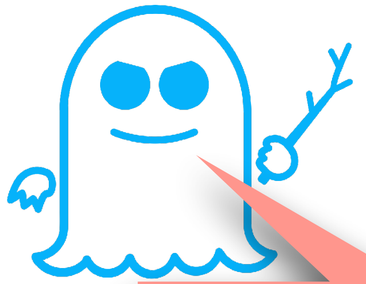
`A_size`=16


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

1) Training  `f(0);`

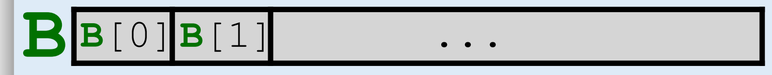


Spectre V1

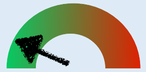


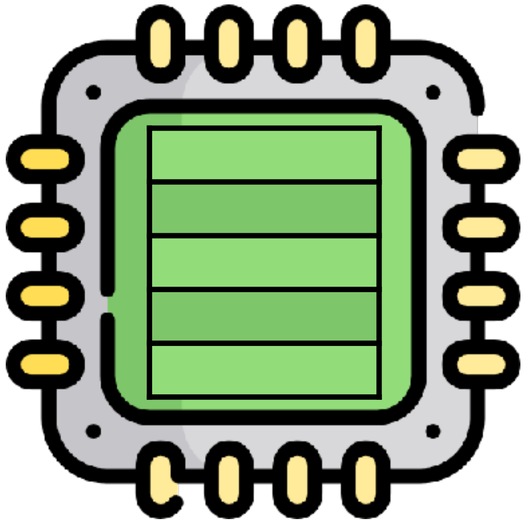
What is in `A[128]`?

`A_size`=16

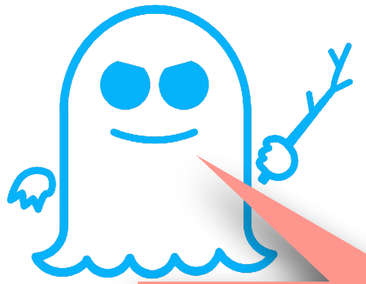



```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

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



Spectre V1

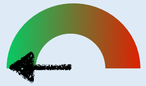


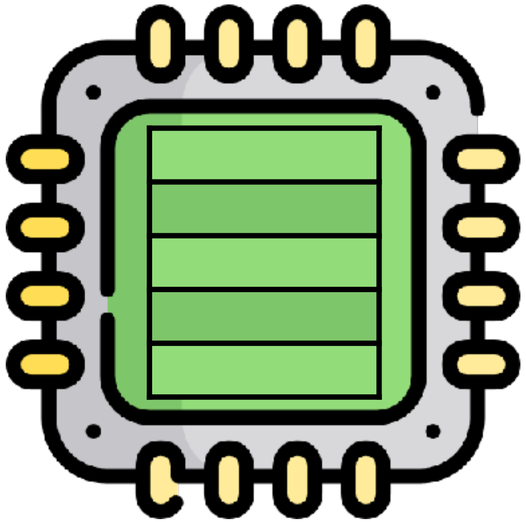
What is in A[128]?

A_size=16

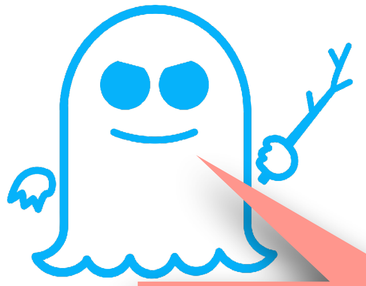
B B[0] B[1] ...

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

1) Training  f(0);f(1);f(2); …

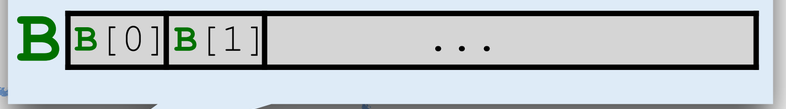


Spectre V1

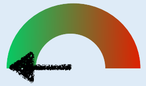


What is in A[128]?

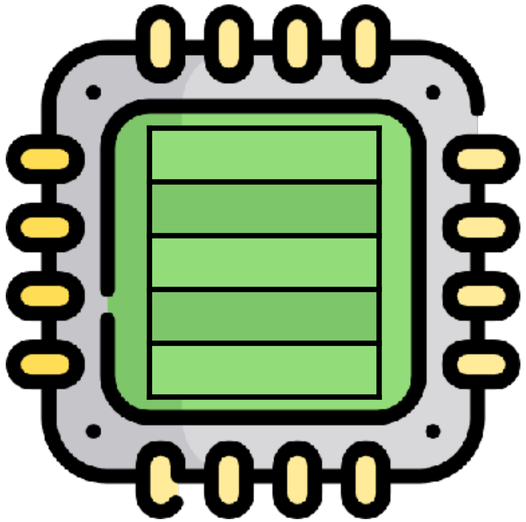
A_size=16



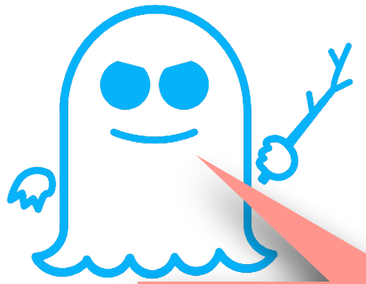

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

1) Training  f(0);f(1);f(2); …

2) Prepare cache



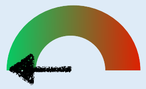
Spectre V1



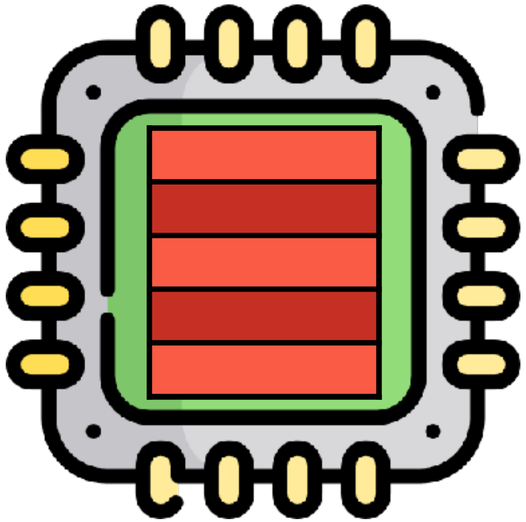
What is in `A[128]`?

`A_size`=16

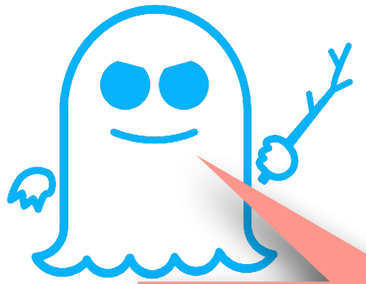

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Training  `f(0);f(1);f(2); …`

2) Prepare cache



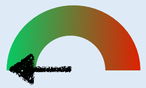
Spectre V1



What is in `A[128]`?

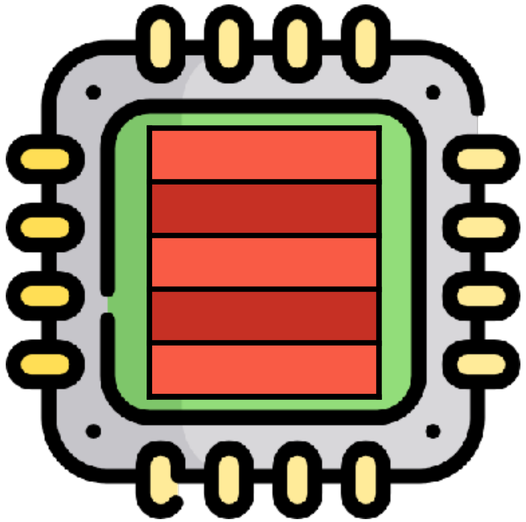
`A_size`=16


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

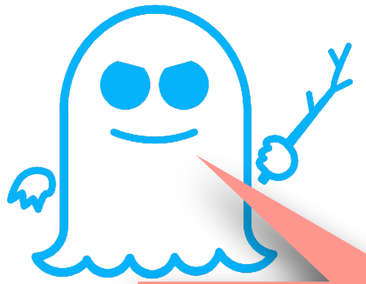
1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with `x = 128`



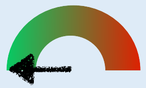
Spectre V1



What is in `A[128]`?

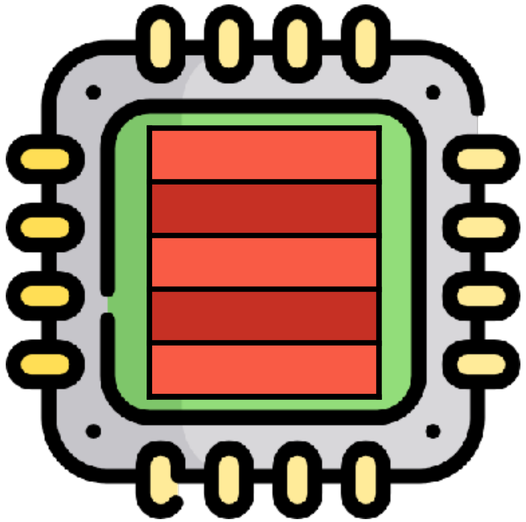
`A_size`=16


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

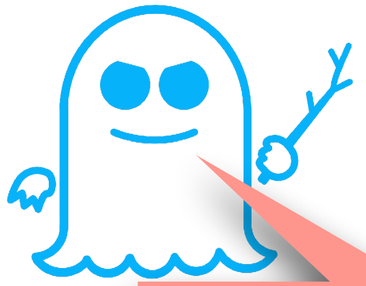
1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with `x = 128`



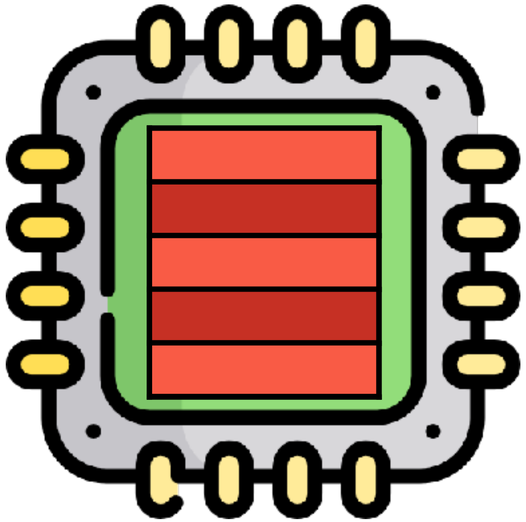
Spectre V1

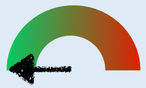


What is in `A[128]`?

`A_size`=16


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

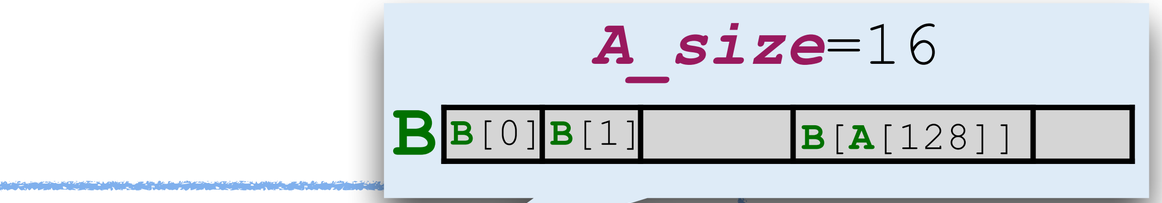


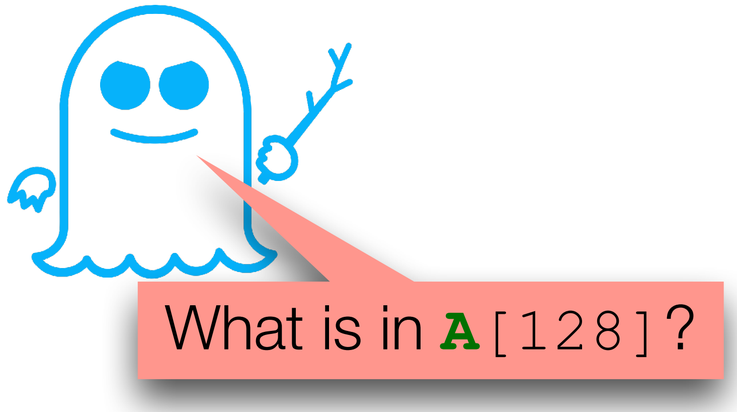
1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

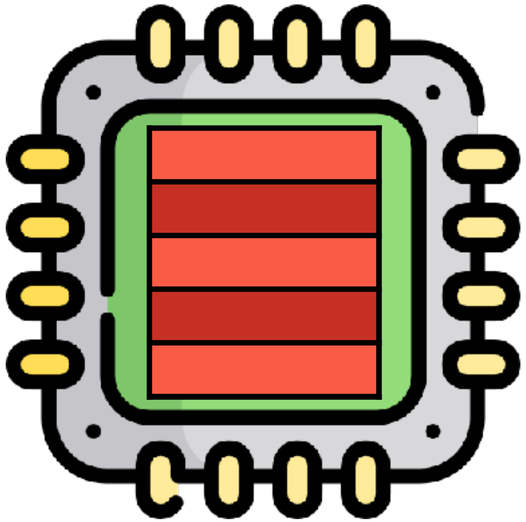
3) Run with `x` = `128`

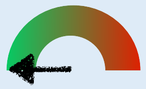
Spectre V1






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

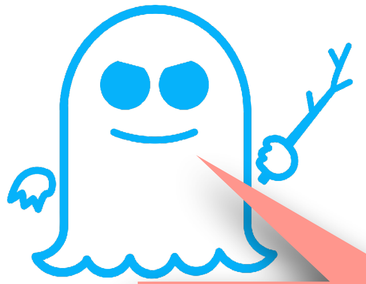


1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with `x = 128`

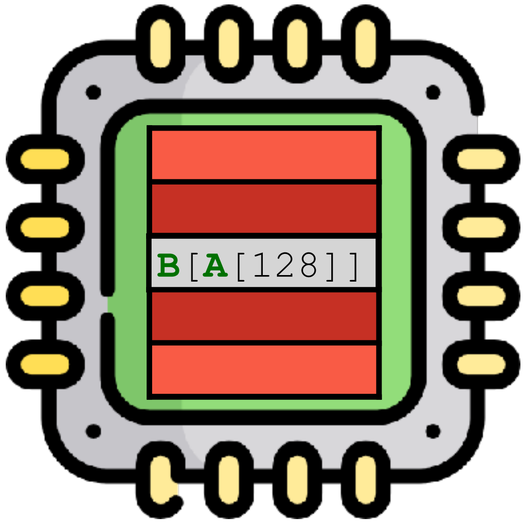
Spectre V1

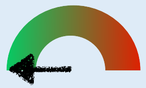


What is in `A[128]`?

`A_size`=16


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

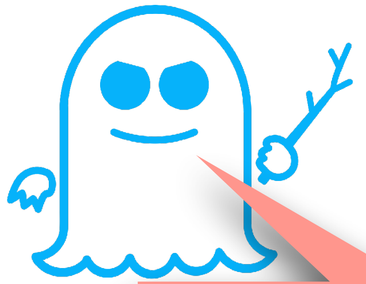


1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with `x` = 128

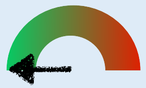
Spectre V1



What is in `A[128]`?

`A_size`=16

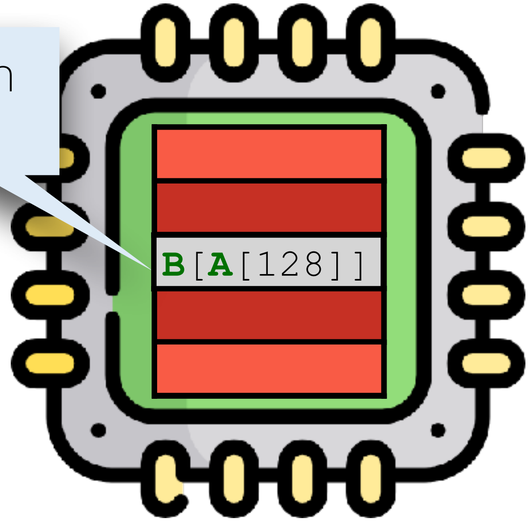

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Training  `f(0);f(1);f(2); …`

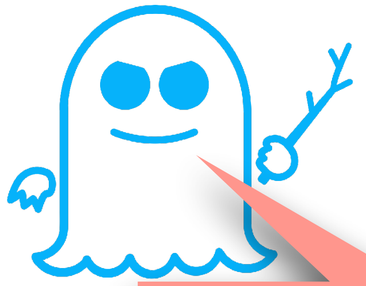
2) Prepare cache

3) Run with `x = 128`

Depends on `A[128]`



Spectre V1

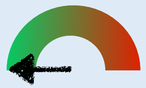


What is in A[128]?

A_size=16

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

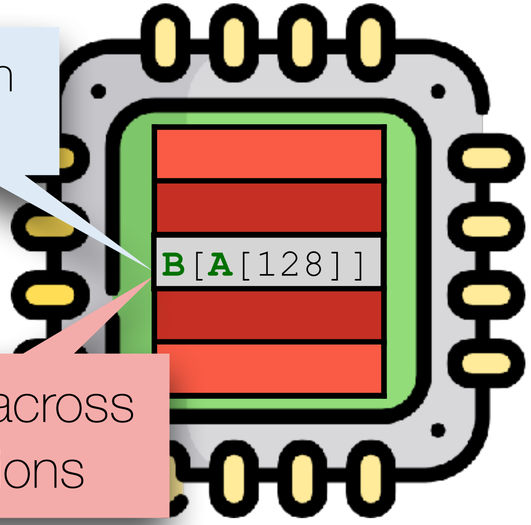
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

1) Training  f(0);f(1);f(2); …

2) Prepare cache

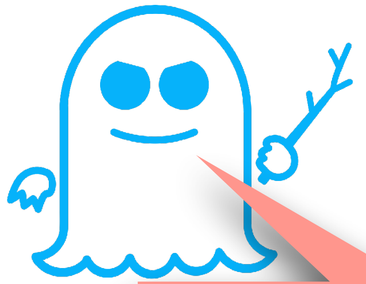
3) Run with x = 128

Depends on A[128]



Persistent across speculations

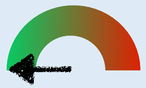
Spectre V1



What is in `A[128]`?

`A_size`=16


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

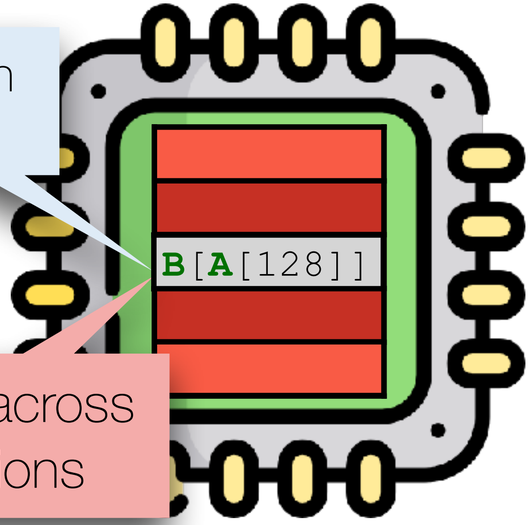
1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with `x = 128`

4) Extract from cache

Depends on `A[128]`



Persistent across speculations

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?

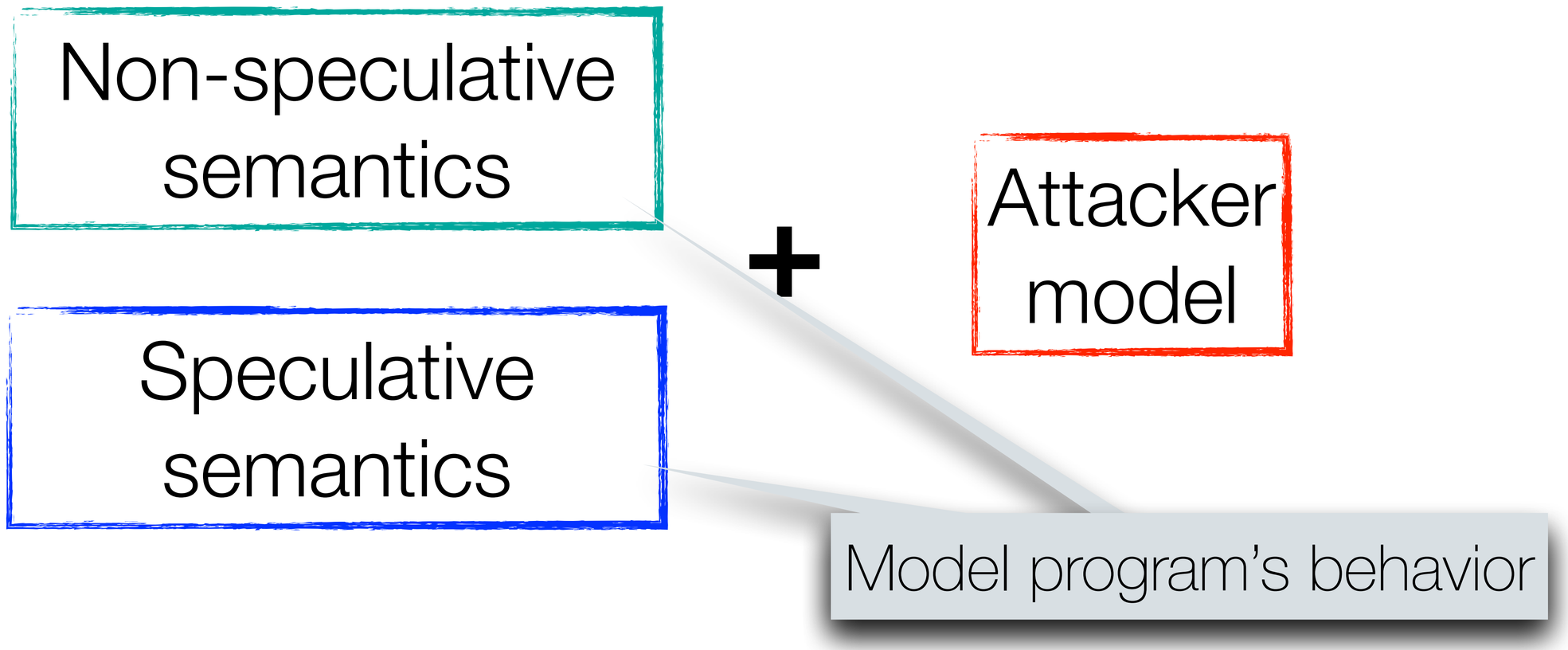
Non-speculative semantics

Speculative semantics

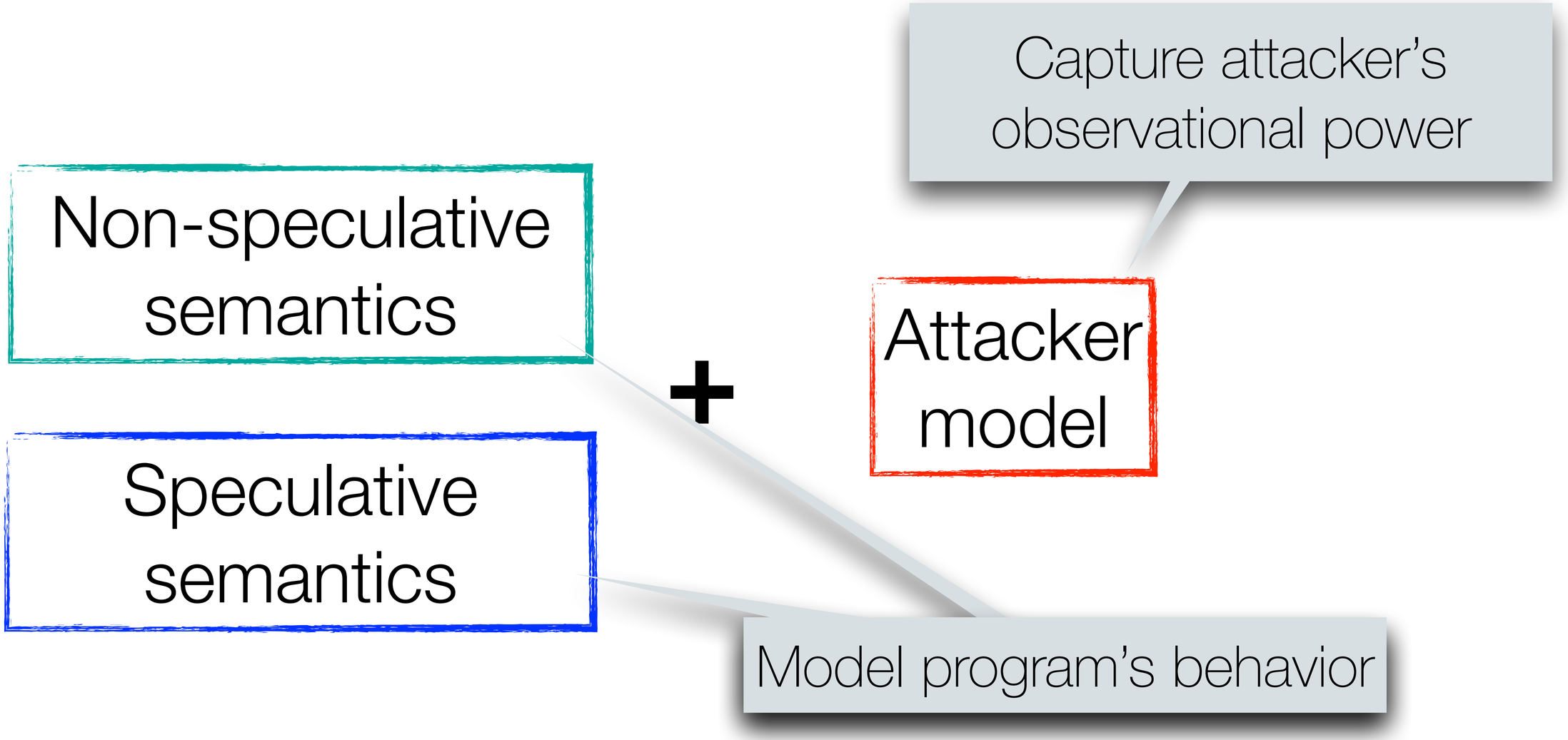
+

Attacker model

How to capture leakage?

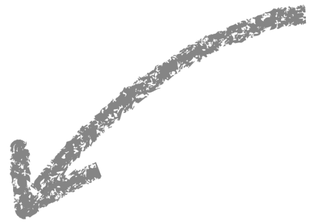


How to capture leakage?



μAssembly + non-speculative semantics

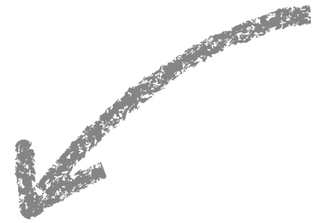
```
if (x < A_size)
    y = B[A[x]]
```



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

μAssembly + non-speculative semantics

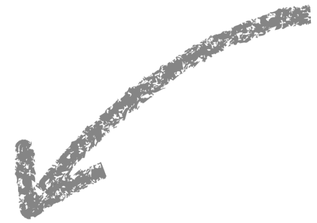
```
if (x < A_size)
    y = B[A[x]]
```



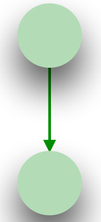
```
      rax <- A_size
      rcx <- x
      jmp rcx≥rax, END
L1:   load rax, A + rcx
      load rax, B + rax
END:
```

μAssembly + non-speculative semantics

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

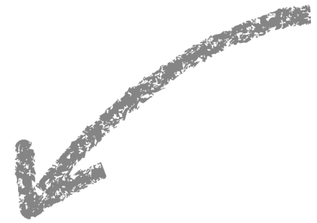


```
        rax <- A_size
        rcx <- x
        jmp rcx≥rax, END
L1:     load rax, A + rcx
        load rax, B + rax
END:
```

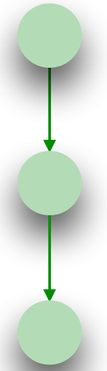


μAssembly + non-speculative semantics

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

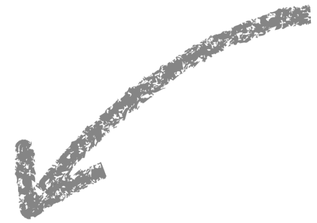


```
        rax <- A_size
        rcx <- x
        jmp rcx≥rax, END
L1:     load rax, A + rcx
        load rax, B + rax
END:
```

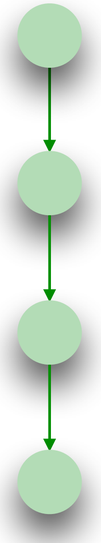


μAssembly + non-speculative semantics

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

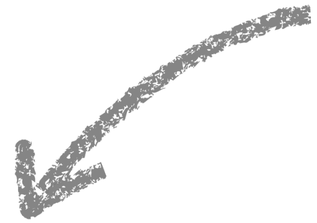


```
        rax <- A_size
        rcx <- x
        jmp rcx≥rax, END
L1:     load rax, A + rcx
        load rax, B + rax
END:
```

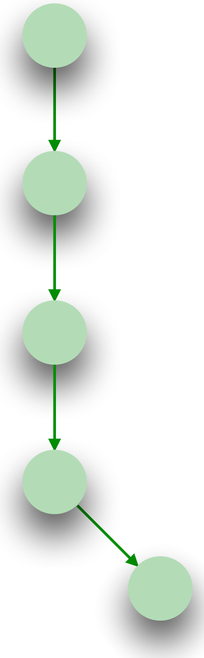


μAssembly + non-speculative semantics

```
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

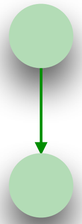
Committed upon correct speculation

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

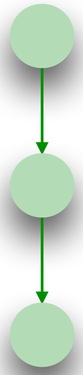
Committed upon correct speculation

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

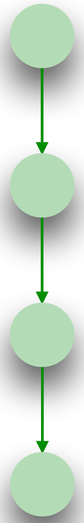
Committed upon correct speculation

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

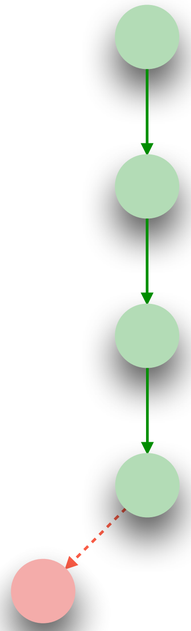
Committed upon correct speculation

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

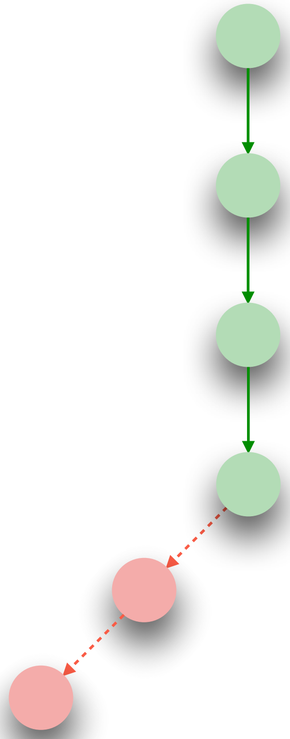
Committed upon correct speculation

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

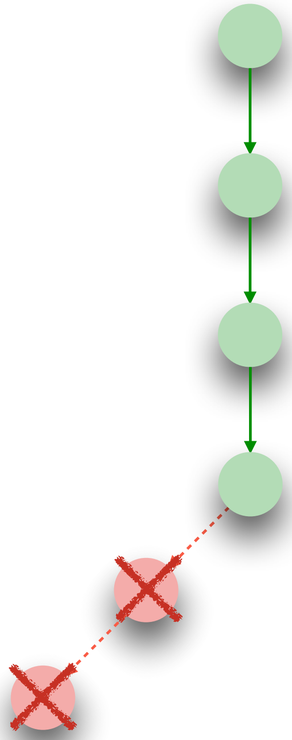
Committed upon correct speculation

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

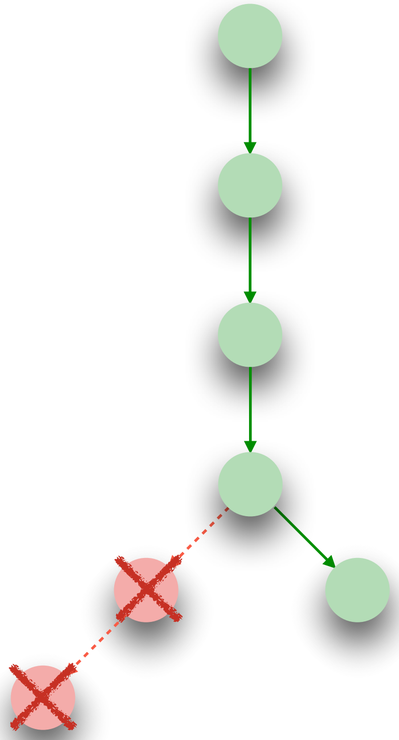
Committed upon correct speculation

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

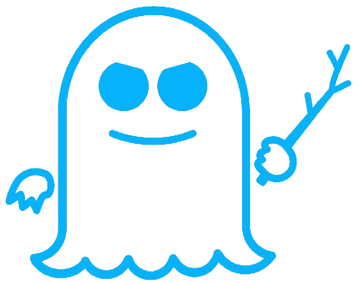
Committed upon correct speculation

Rolled back upon misspeculation

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

Leakage into μarchitecture

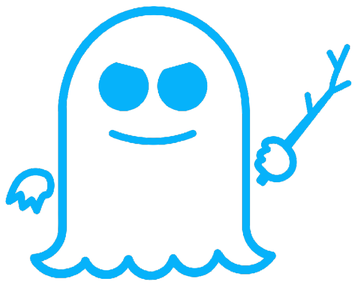
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```



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

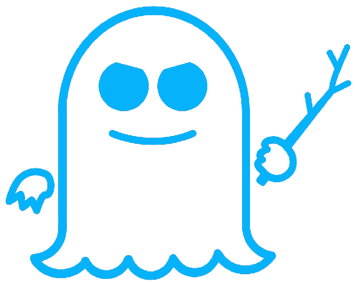


Leakage into μarchitecture

```
     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

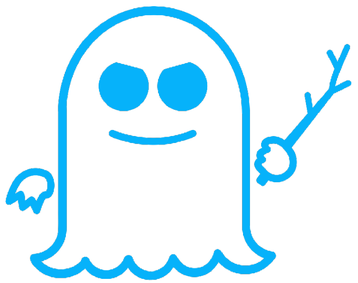


Leakage into μarchitecture

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

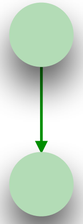
Attacker can observe:
- locations of **memory accesses**
- **branch/jump** targets
- **start/end** speculative execution

Inspired by “constant-time” rqmts



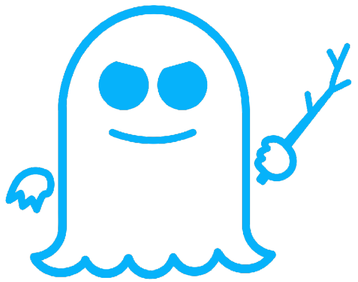
Leakage into μarchitecture

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



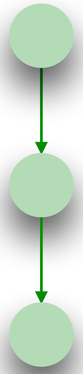
Attacker can observe:
- locations of **memory accesses**
- **branch/jump** targets
- **start/end** speculative execution

Inspired by “constant-time” rqmts



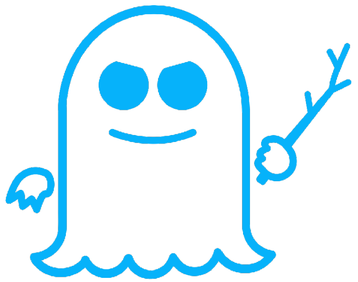
Leakage into μarchitecture

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



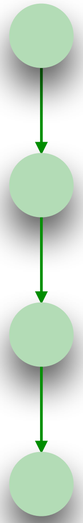
Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by “constant-time” rqmts



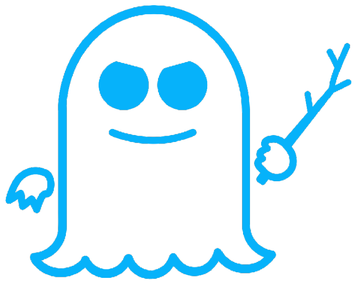
Leakage into μarchitecture

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



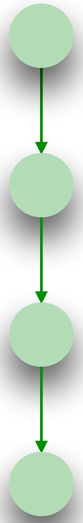
Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by “constant-time” rqmts



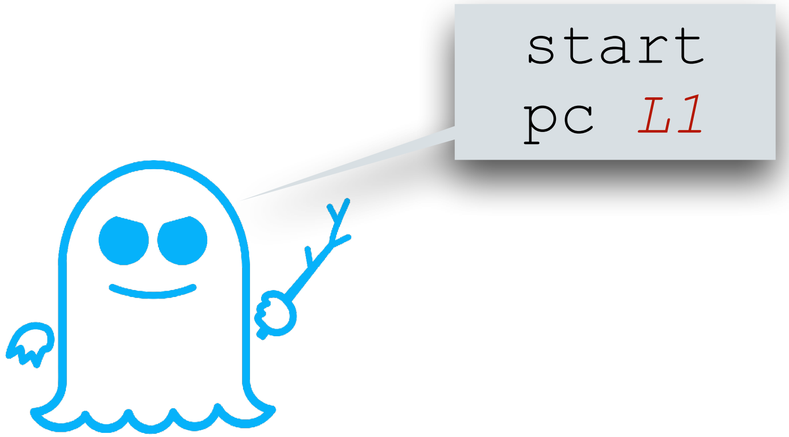
Leakage into μarchitecture

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



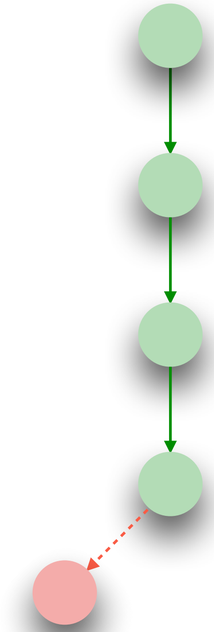
Attacker can observe:
- locations of **memory accesses**
- **branch/jump** targets
- **start/end** speculative execution

Inspired by “constant-time” rqmts



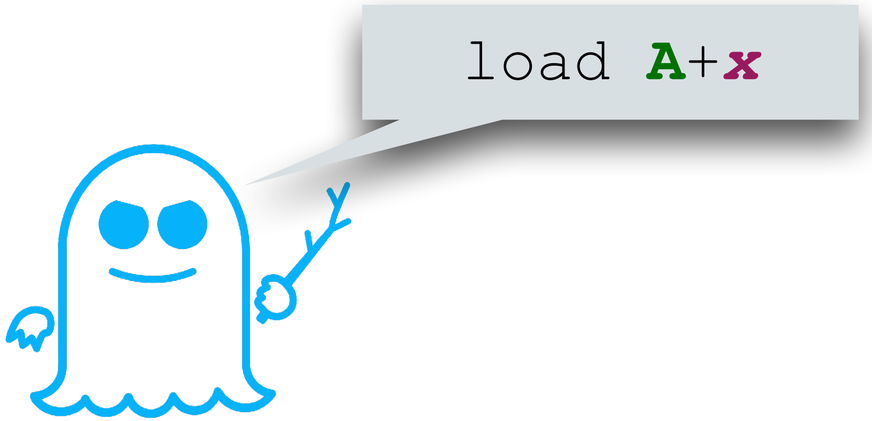
Leakage into μarchitecture

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



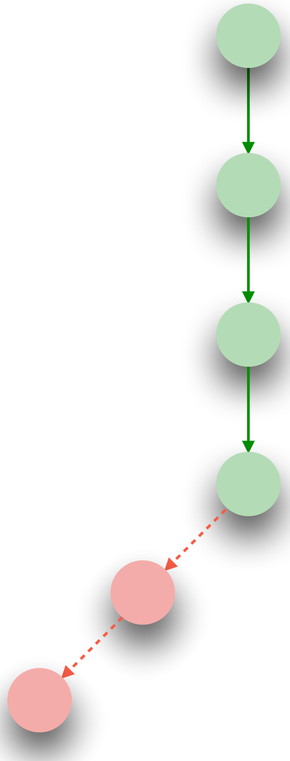
Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

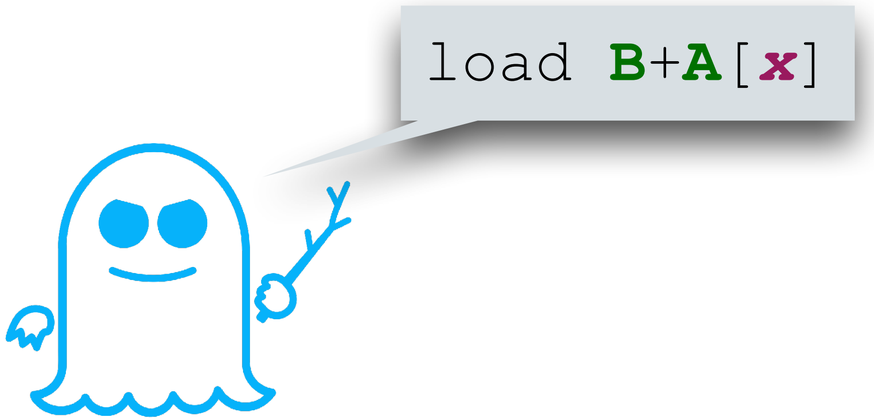
Inspired by “constant-time” rqmts



Leakage into μarchitecture

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



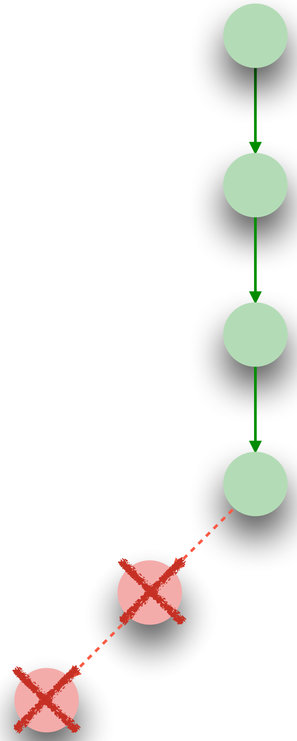


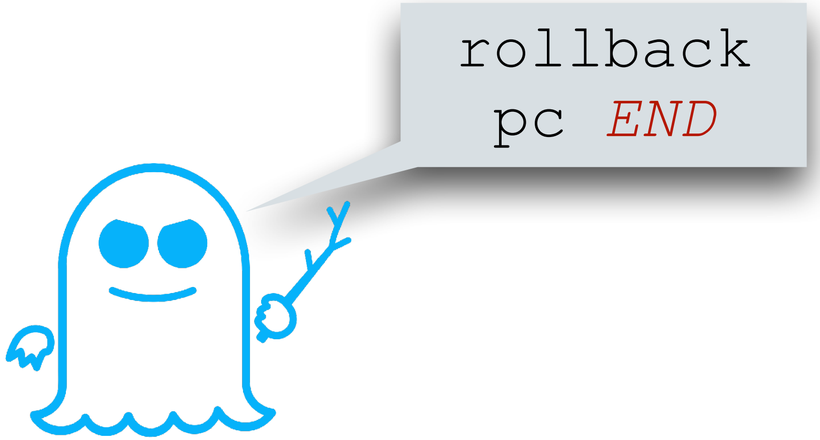
Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

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:
```



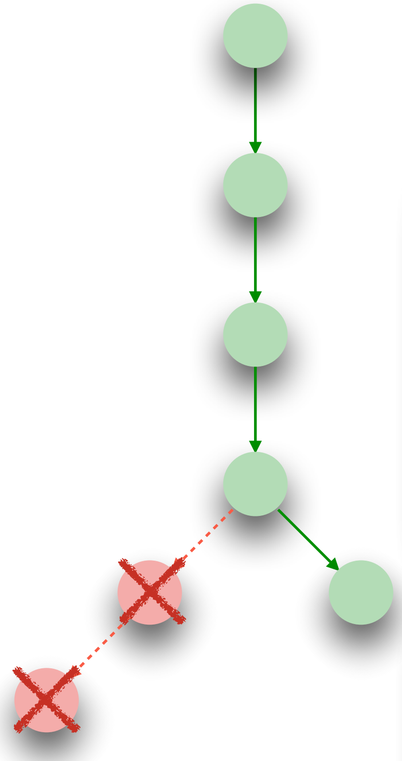


Attacker can observe:
- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Inspired by “constant-time” rqmts

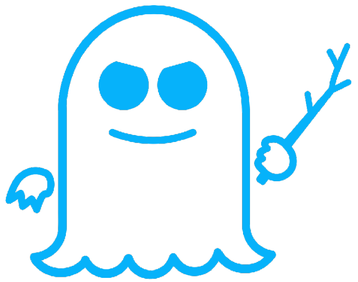
Leakage into μarchitecture

```
      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′***:

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

Speculative non-interference

Formally!

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

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

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

Speculative non-interference

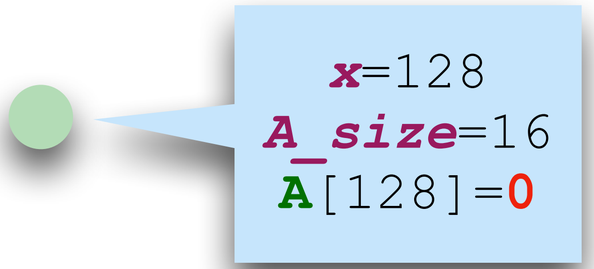
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```

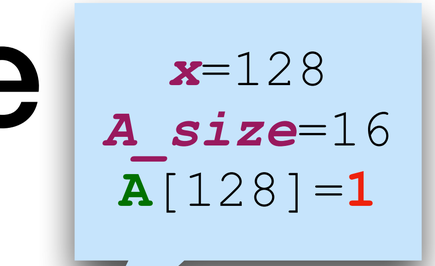
Speculative non-interference

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

Speculative non-interference

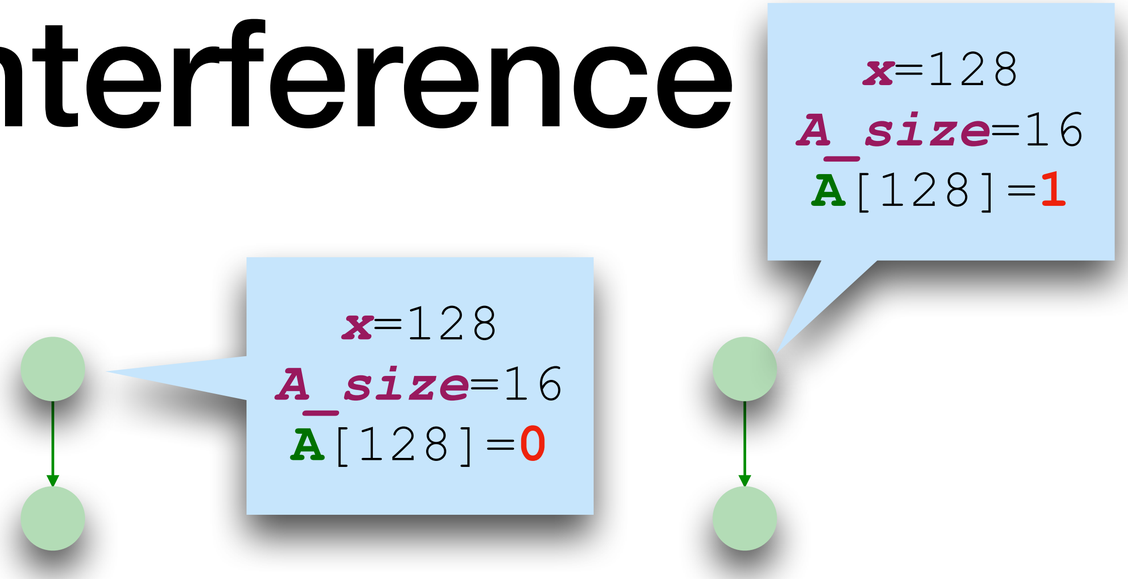
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```





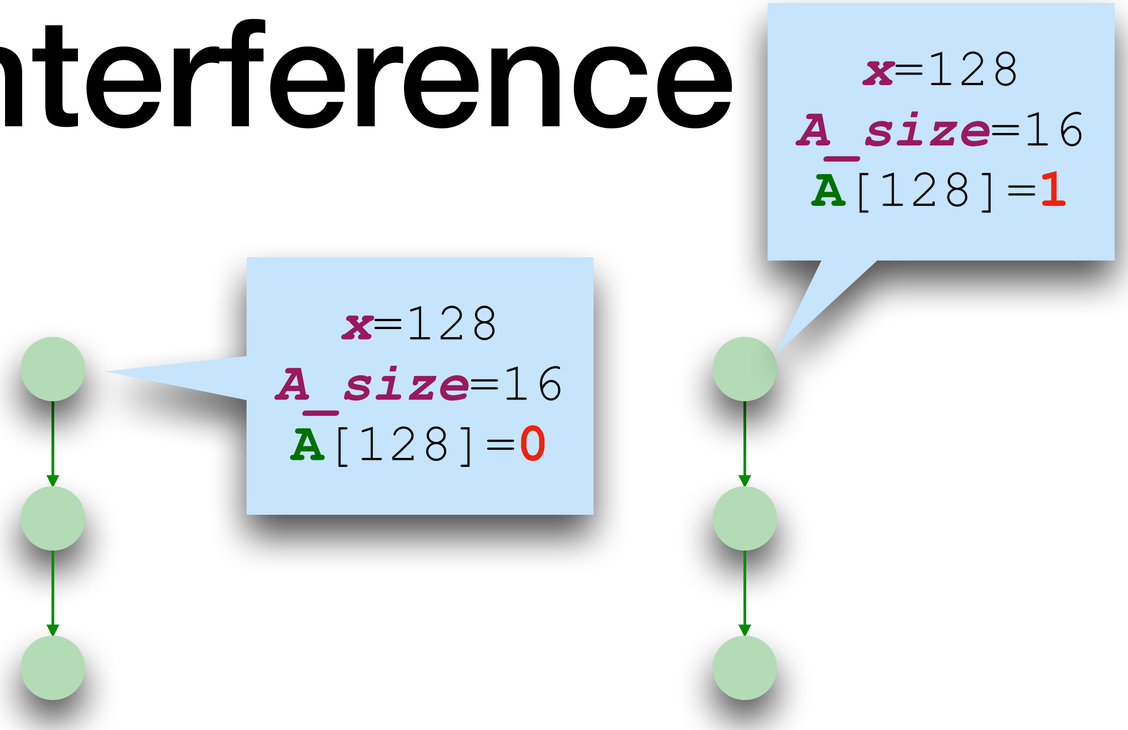
Speculative non-interference

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



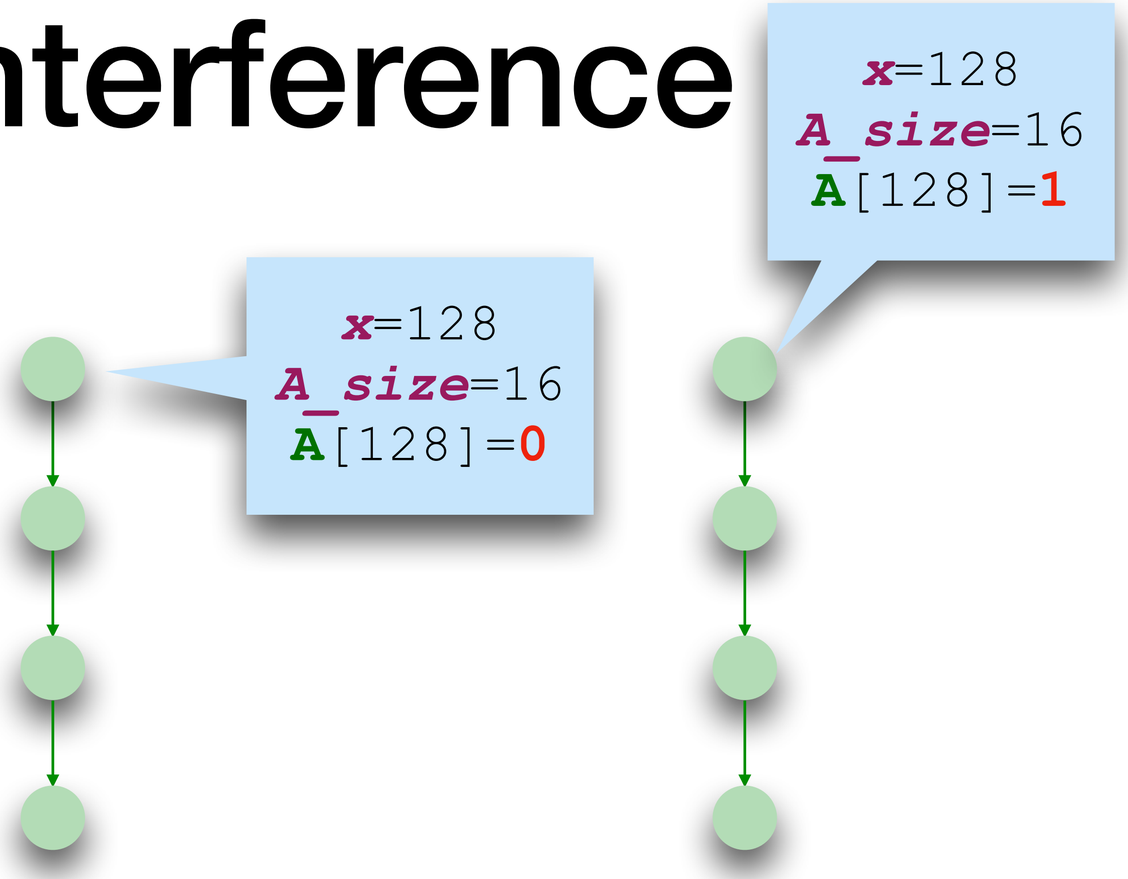
Speculative non-interference

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



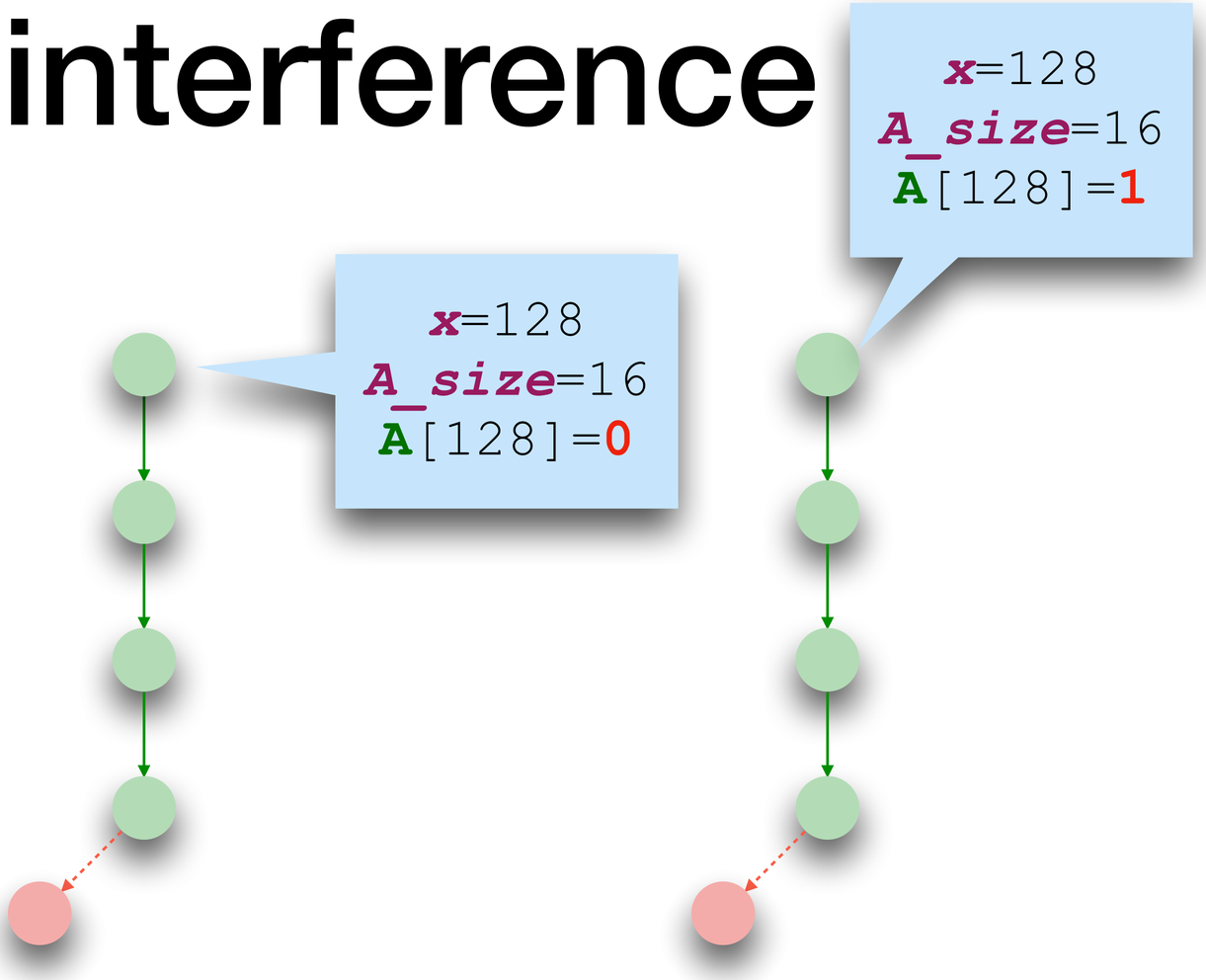
Speculative non-interference

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



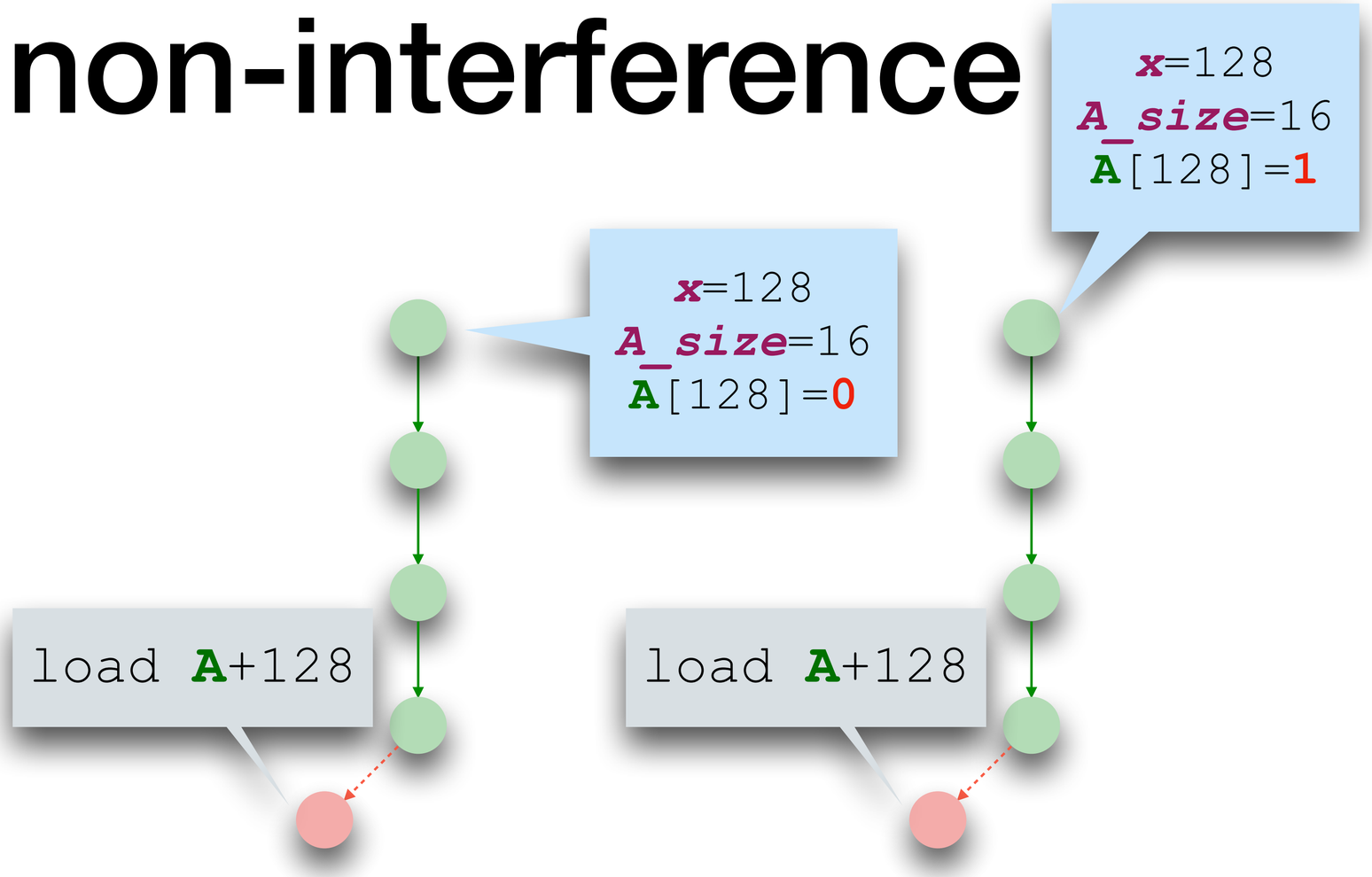
Speculative non-interference

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



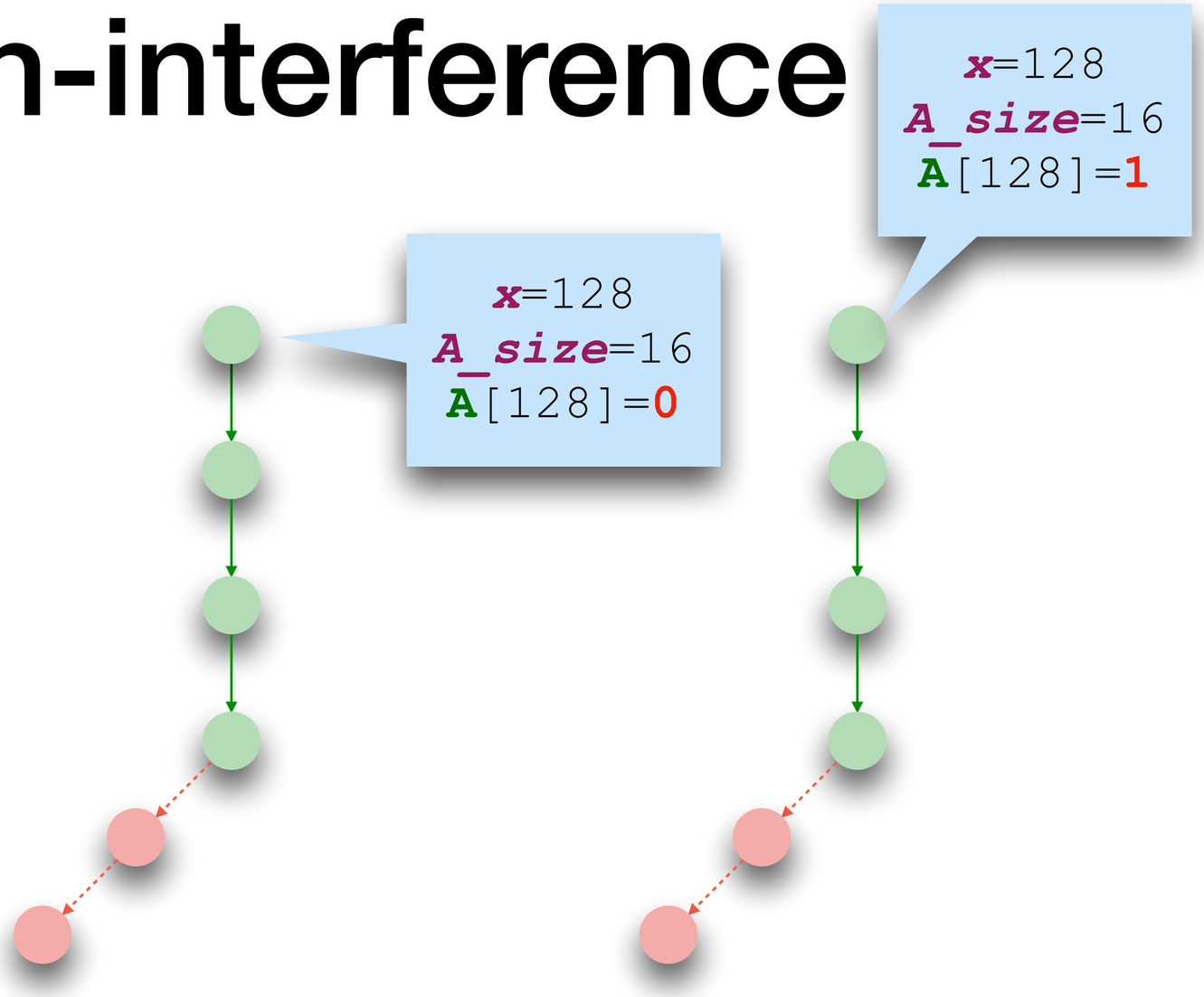
Speculative non-interference

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



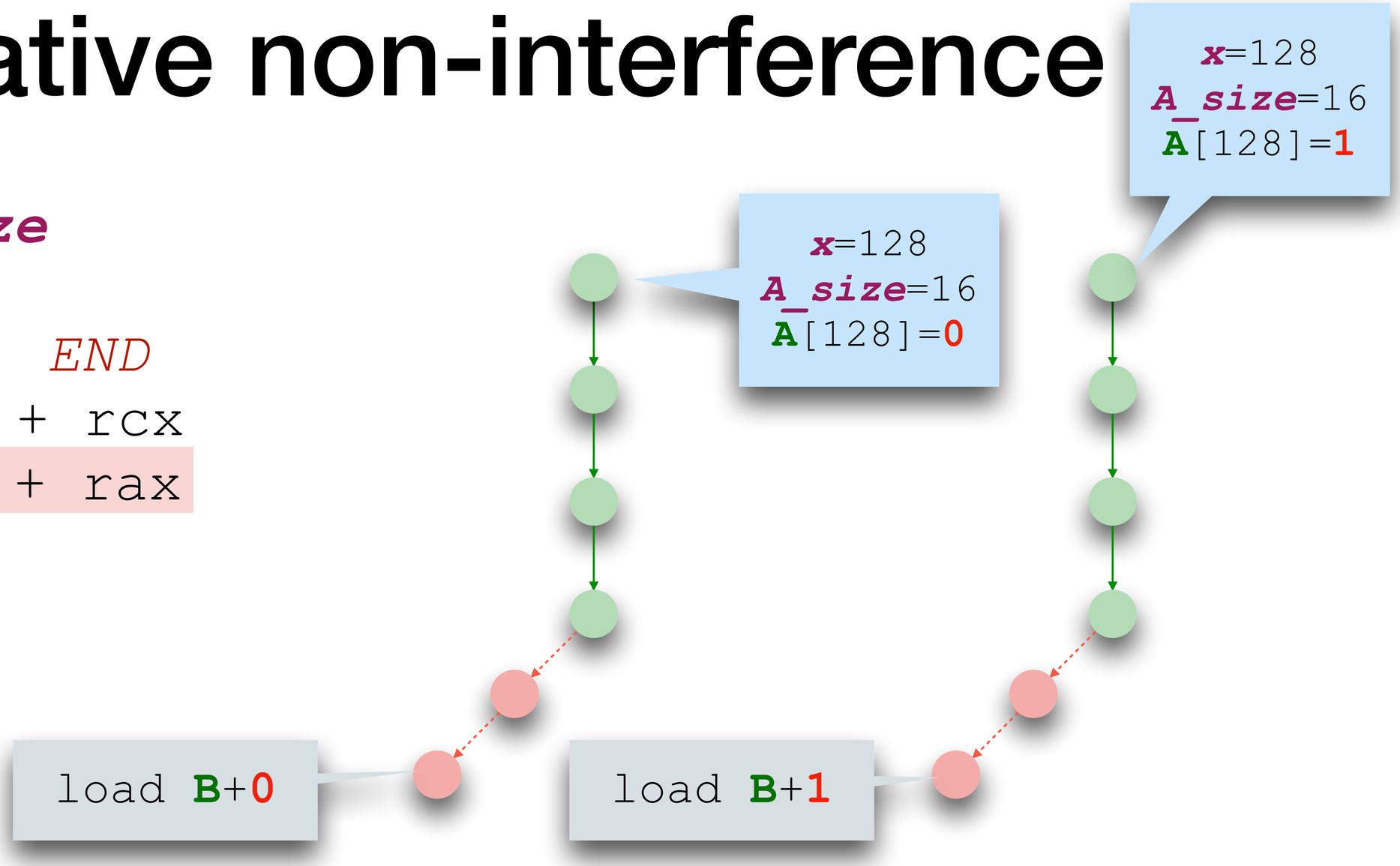
Speculative non-interference

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



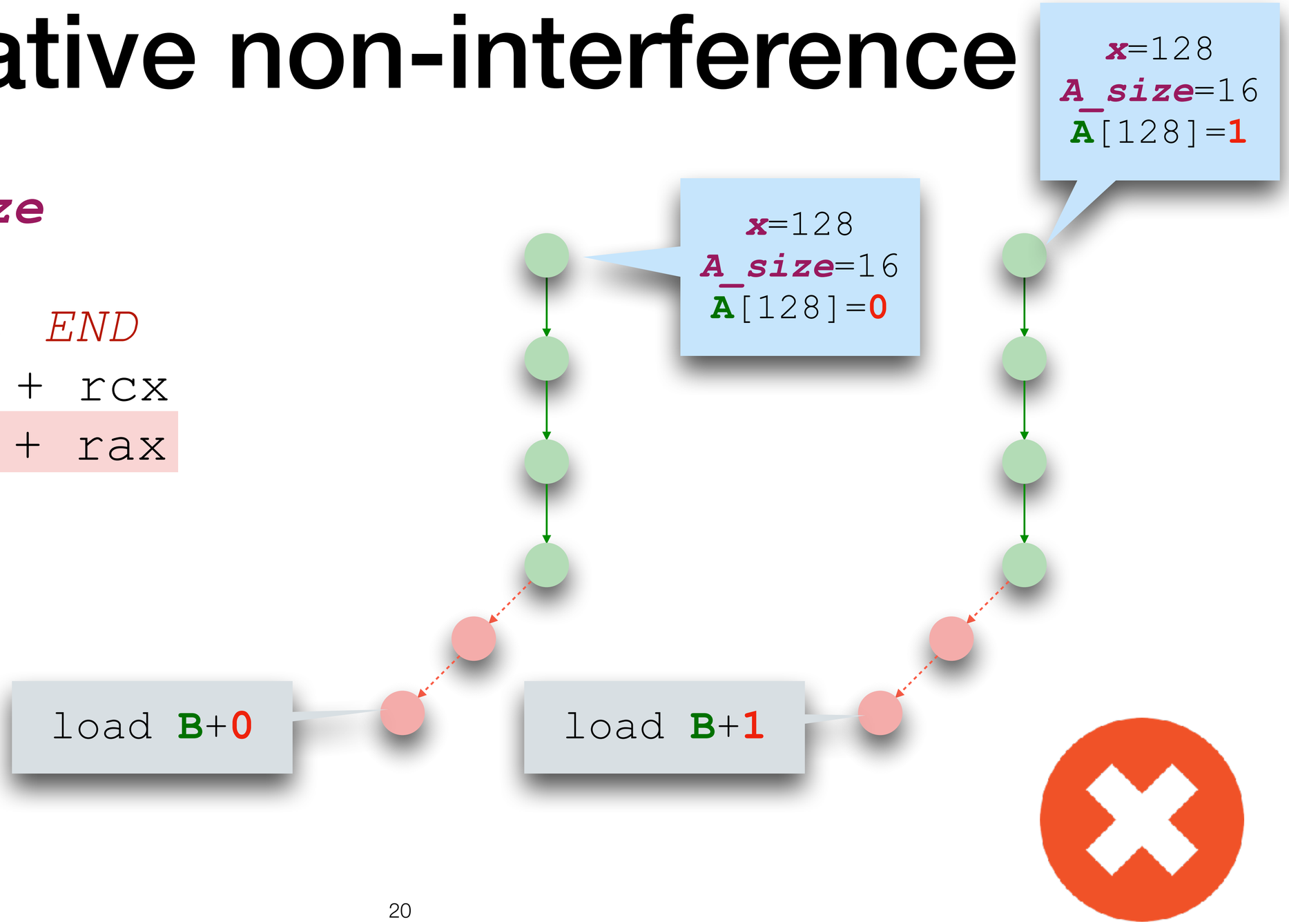
Speculative non-interference

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



Speculative non-interference

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



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***

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

Reasoning about arbitrary oracles

Always-mispredict speculative semantics

- Mispredict ***all*** branch instructions
- Fixed speculative window
- Rollback of every transaction

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

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

If program $\mathbf{P}$ satisfies

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

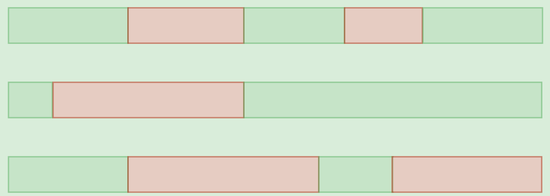
then $\mathbf{P}$ satisfies ***SNI*** w.r.t. all O

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:
```

Symbolic execution

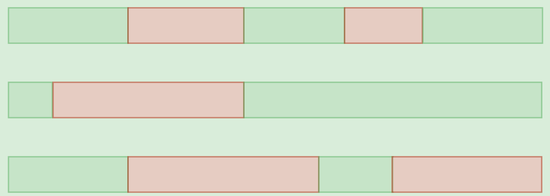


Detect leaks

Detecting speculative leaks

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

Symbolic execution



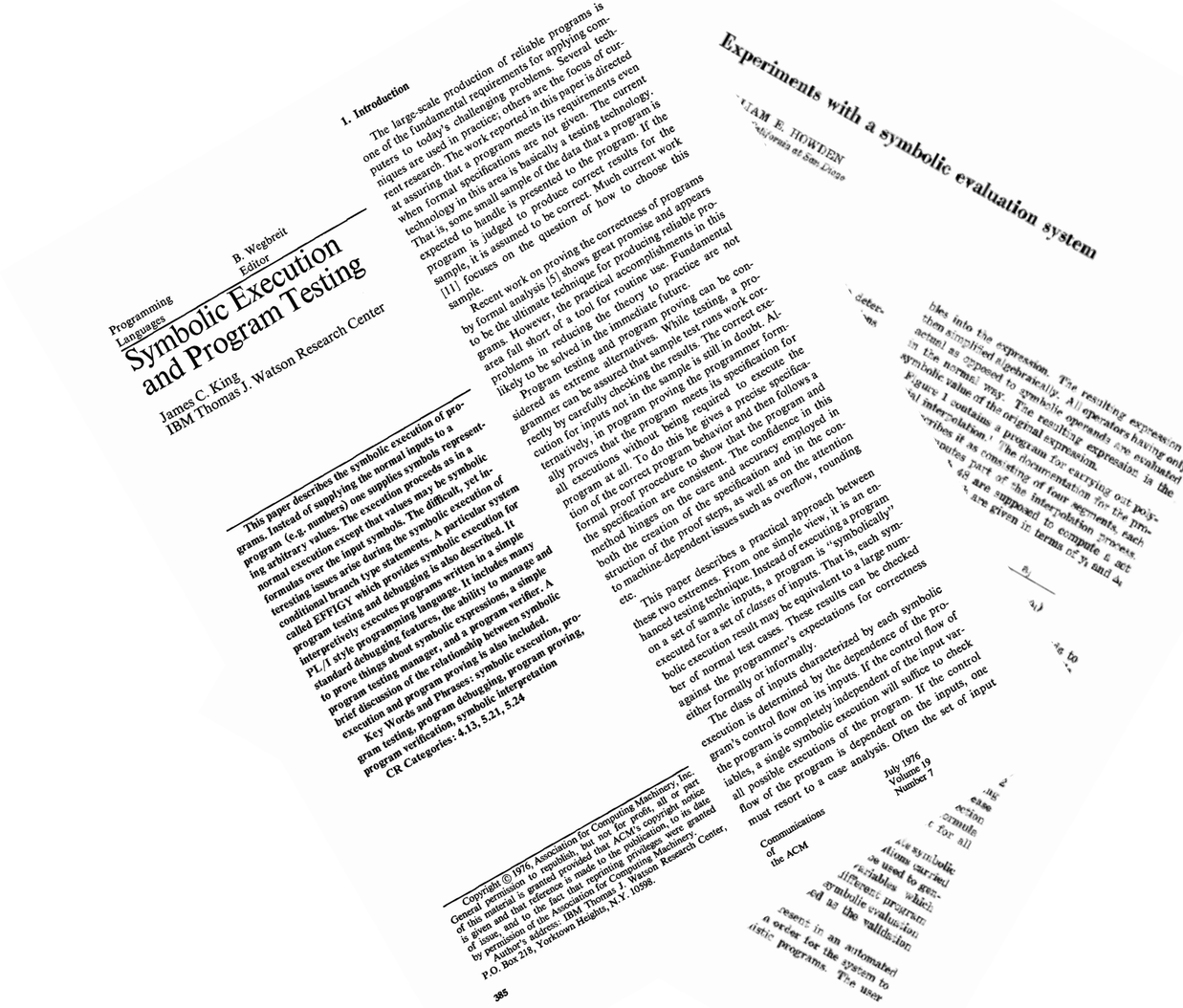
Detect leaks

Symbolic trace: path condition + observations along the symbolic path

Symbolic execution

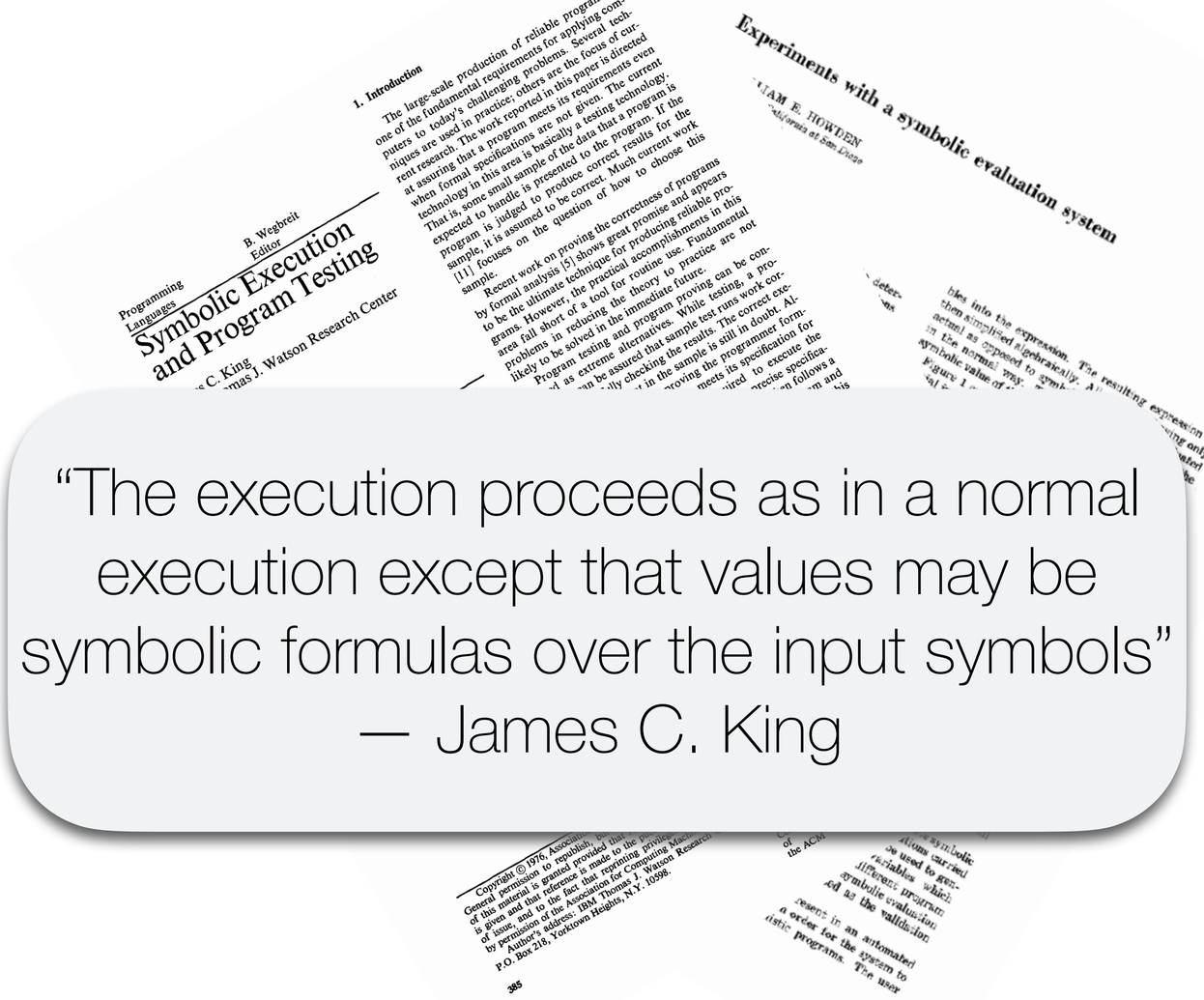
Symbolic execution

- Program analysis technique



Symbolic execution

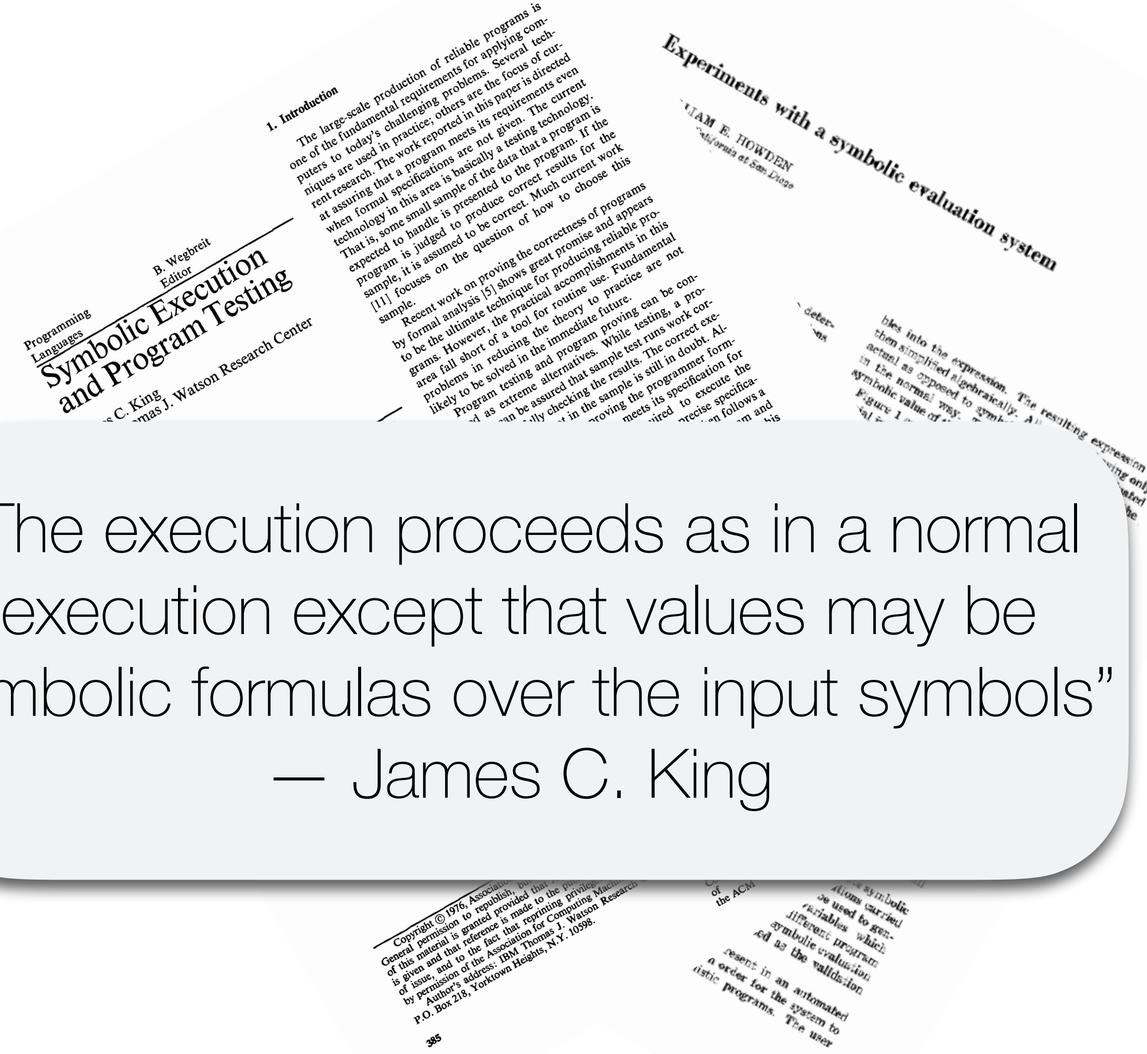
- Program analysis technique
- Execute programs over symbolic values



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

Symbolic execution

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



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

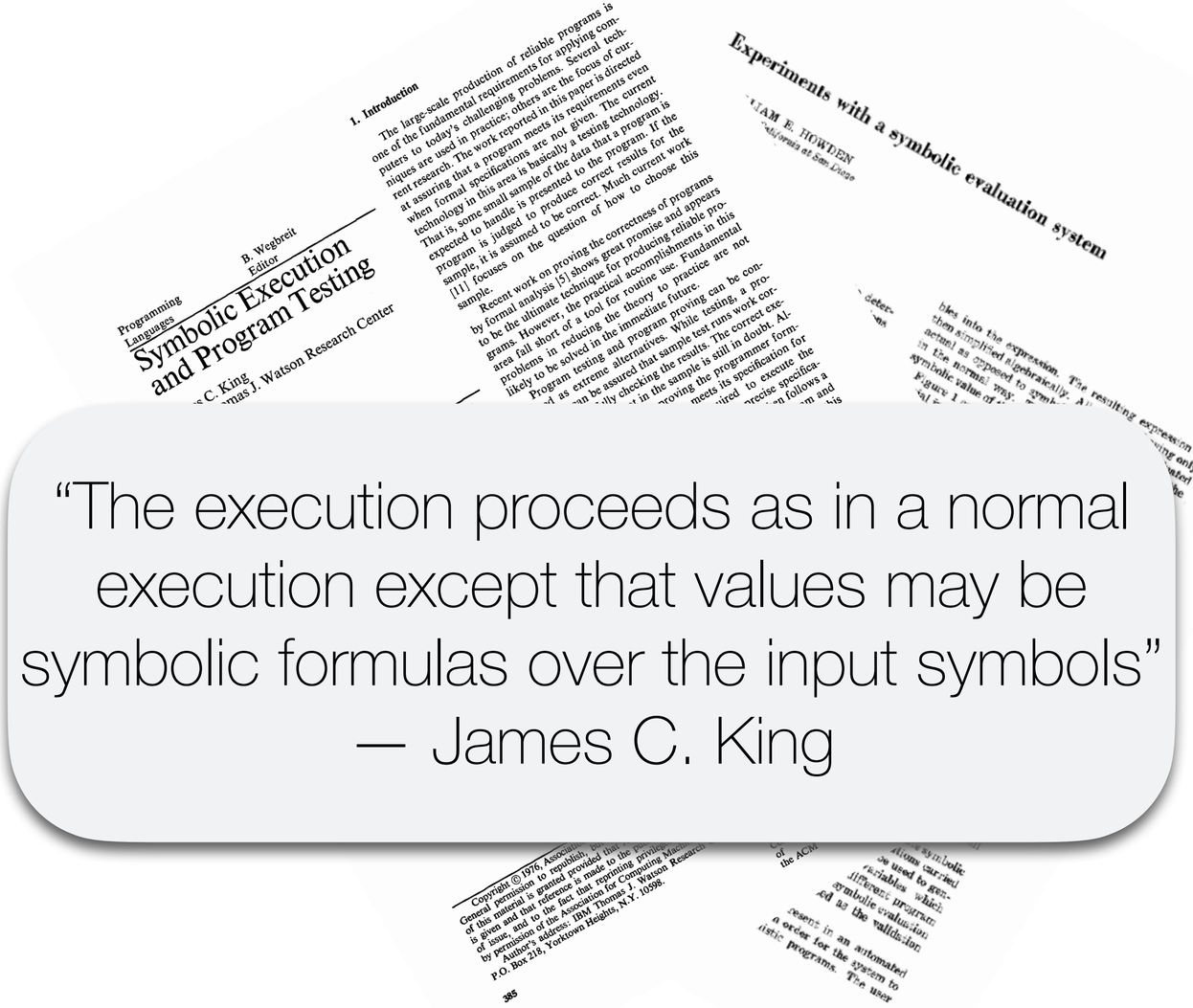
Symbolic execution

- Program analysis technique

- Execute programs over symbolic values

  - Explore all paths, each with its own path constraint

  - Each path represents all concrete executions satisfying the constraint



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

Symbolic execution

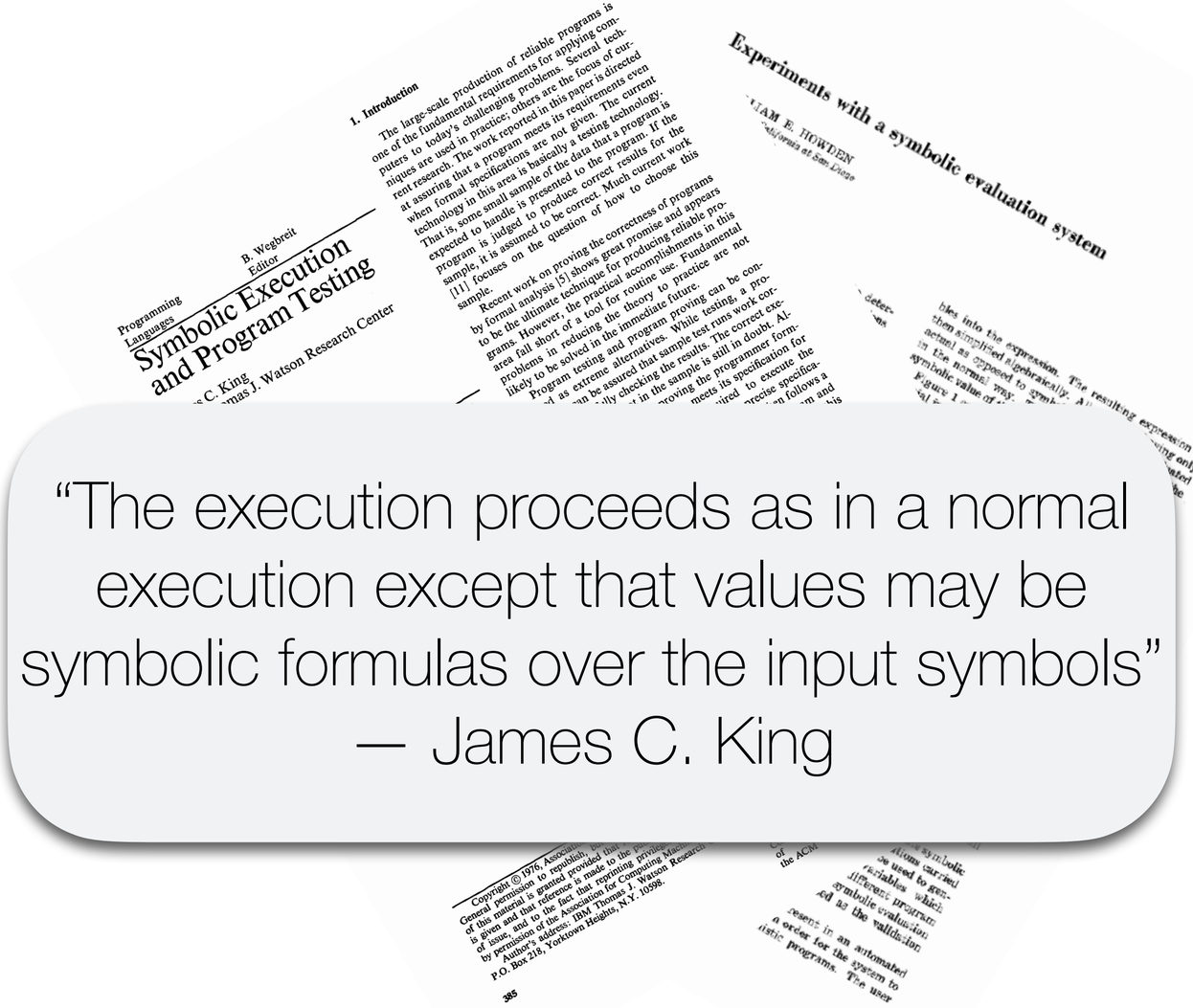
- Program analysis technique

- Execute programs over symbolic values

  - Explore all paths, each with its own path constraint

  - Each path represents all concrete executions satisfying the constraint

  - Branch and jump instructions: fork paths and update path constraint



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

Symbolic execution

```
     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:
```

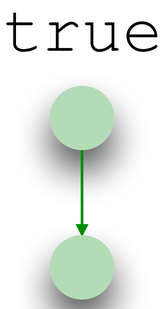
true

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:
```

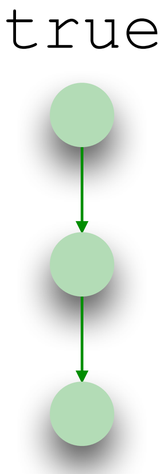
Always mispredict branch instructions



Symbolic execution

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

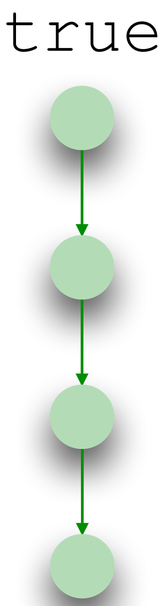
Always mispredict
branch instructions



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

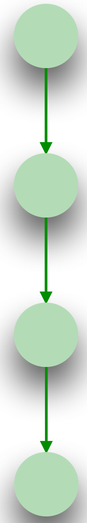


Symbolic execution

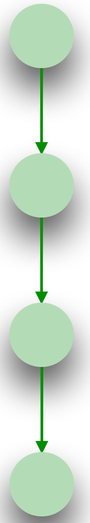
```
     rax <- A_size
     rcx <- x
     jmp rcx≥rax, END
L1:  load rax, A + rcx
     load rax, B + rax
END:
```

Always mispredict branch instructions

$x \geq$ *A_size*



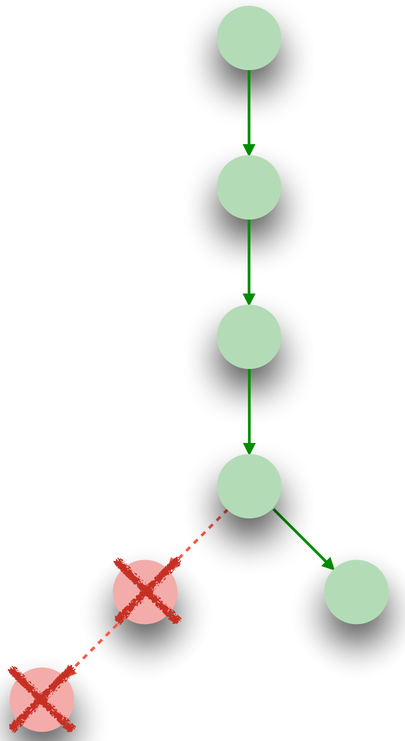
$x <$ *A_size*

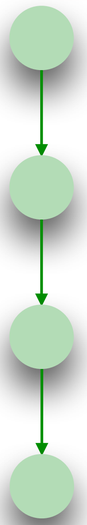


Symbolic execution

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

Always mispredict branch instructions

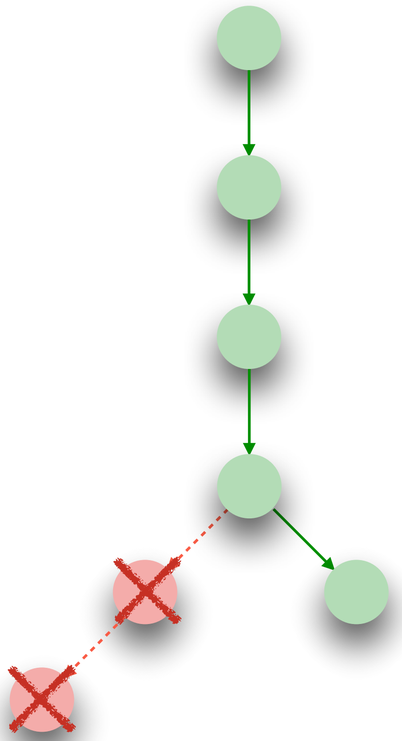


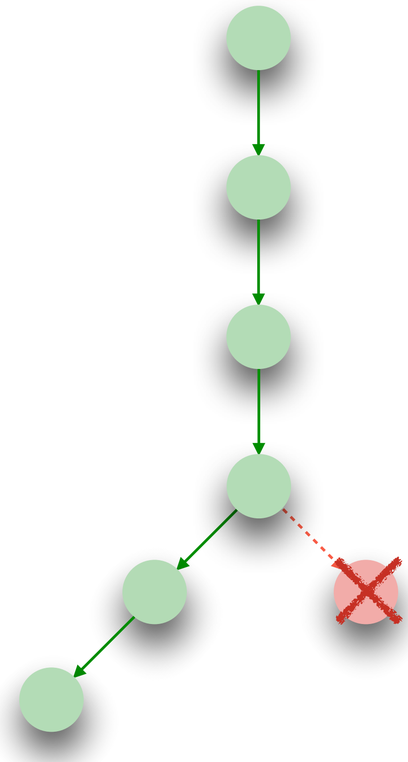


Symbolic execution

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

Always mispredict branch instructions

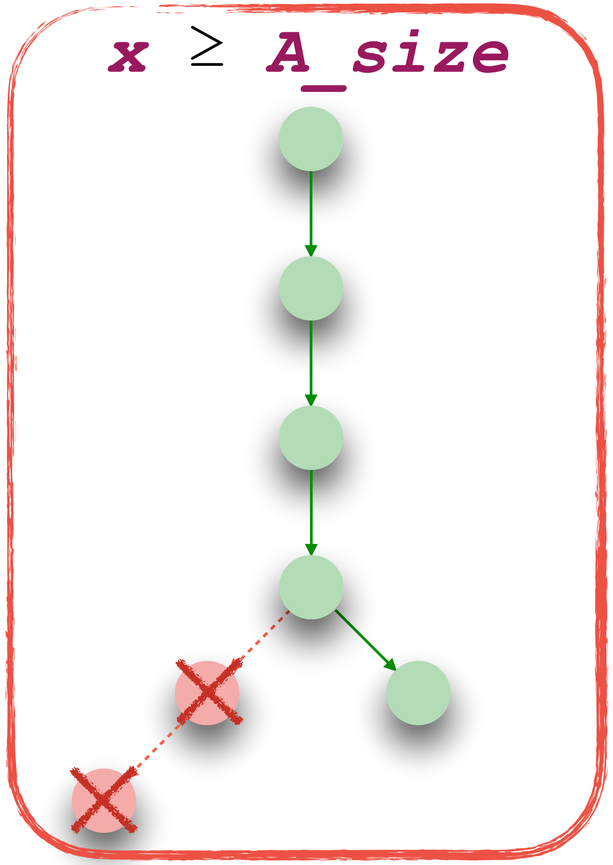


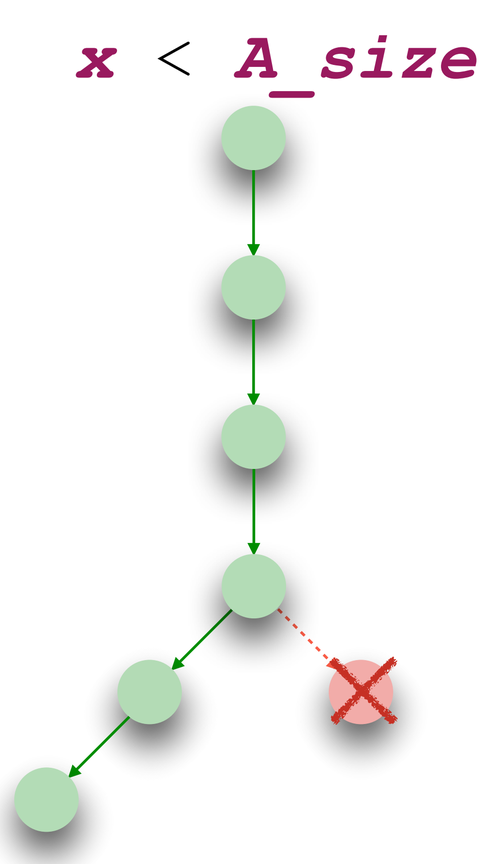


Symbolic execution

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

Always mispredict branch instructions

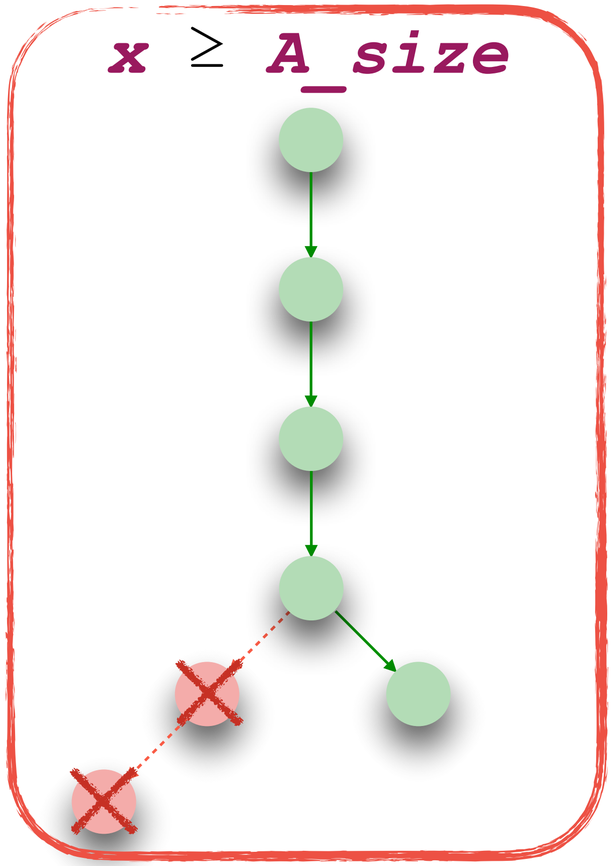


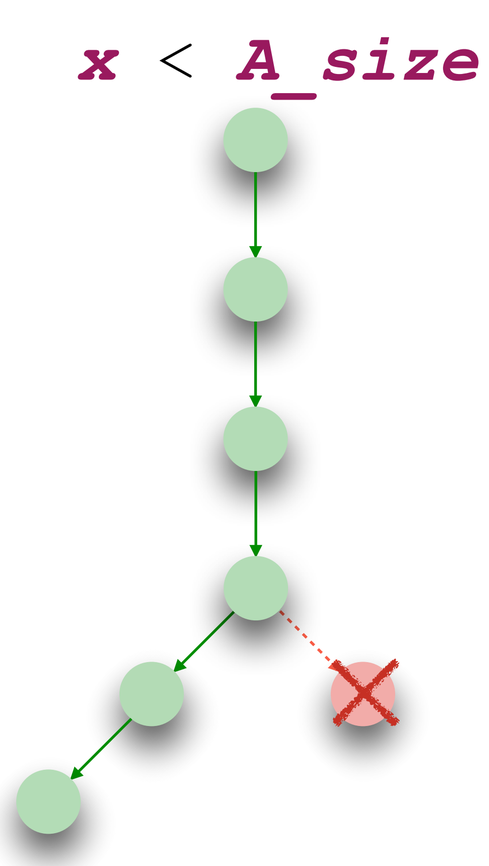


Symbolic execution

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

Always mispredict branch instructions



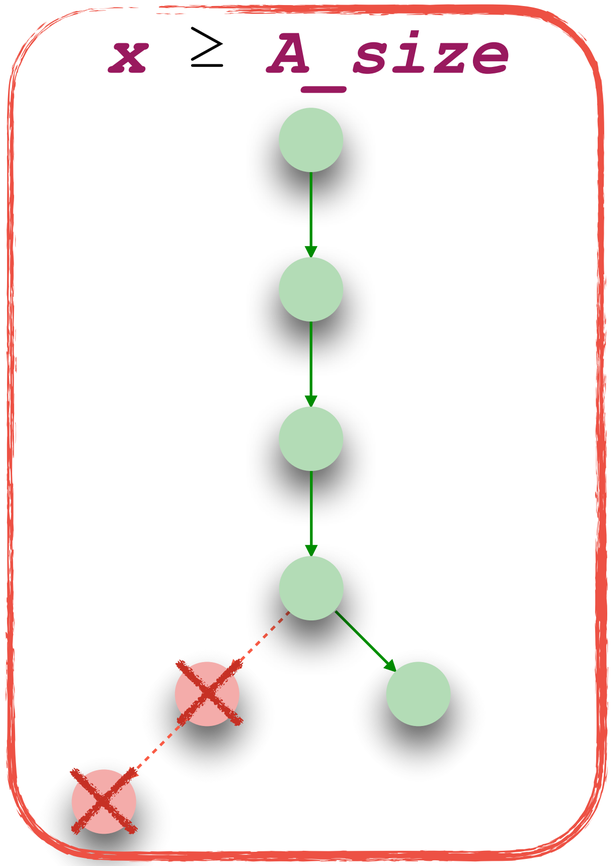


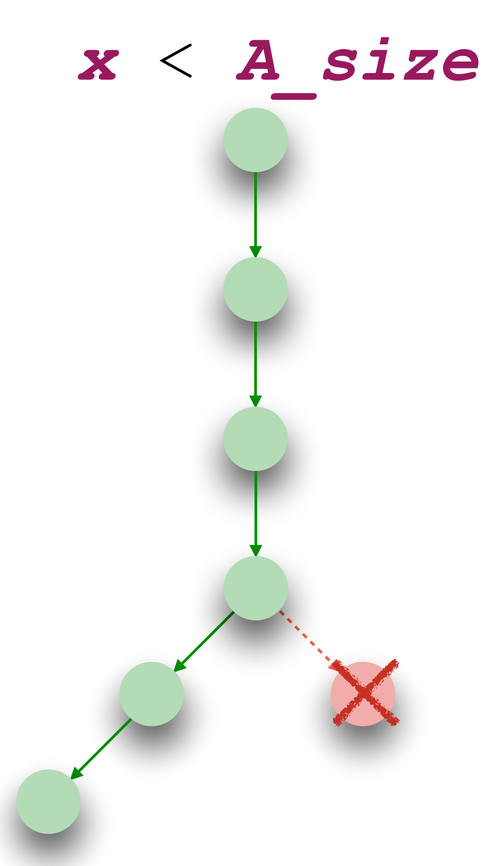

```
start pc L1 load A+x load B+A[x] rollback pc END
```

Symbolic execution

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

Always mispredict branch instructions



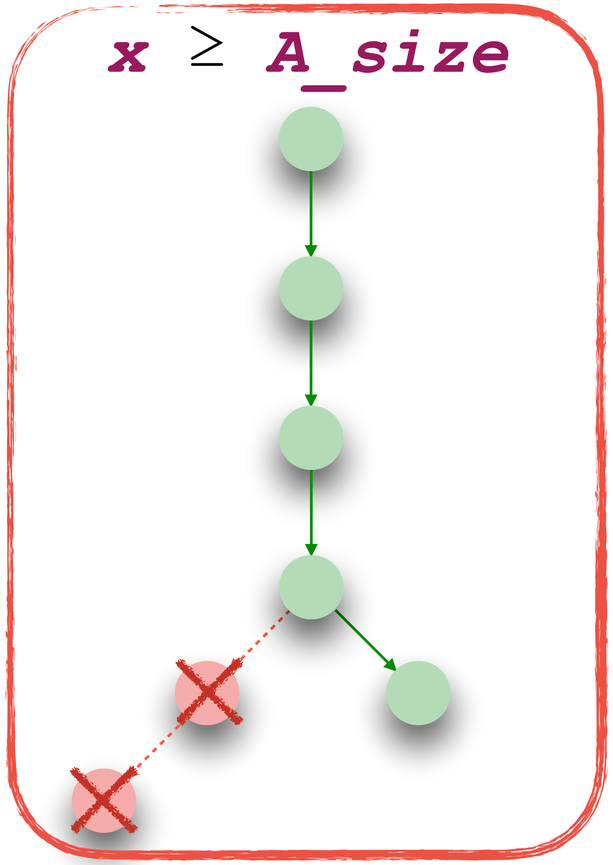


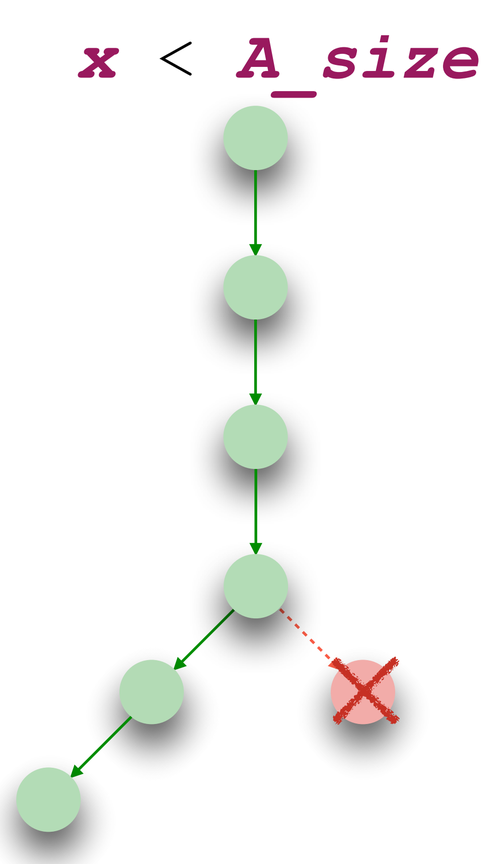

```
start pc L1 load A+x load B+A[x] rollback pc END
```

Symbolic execution

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

Always mispredict branch instructions

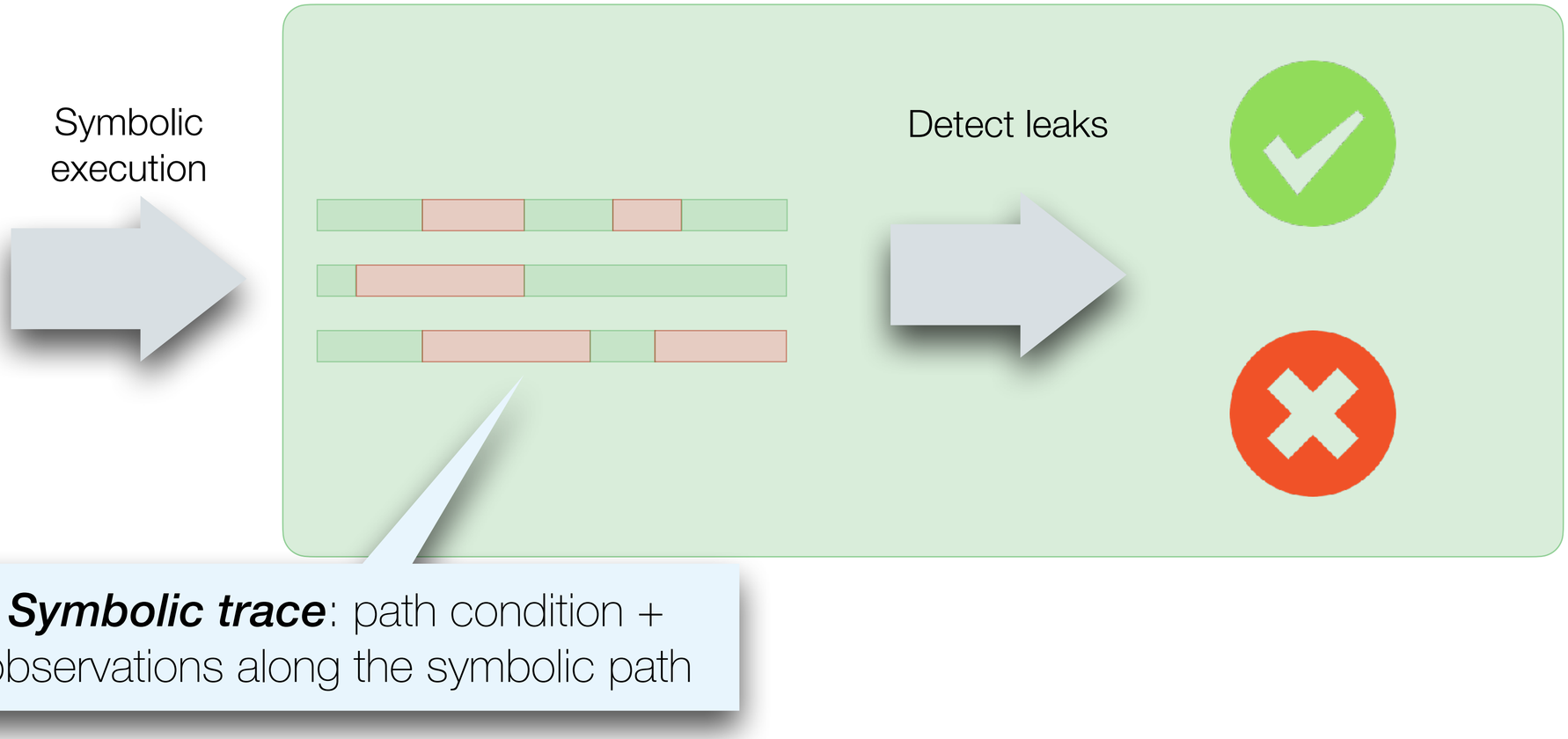




`start pc L1 load A+x load B+A[x] rollback pc END`

Detecting speculative leaks

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



Detecting speculative leaks

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

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

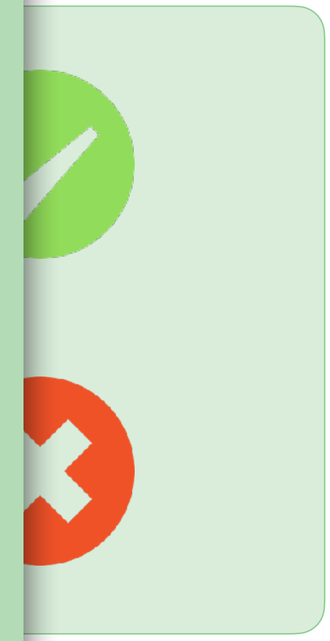
if $MemLeak(\tau)$ **then**

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

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

Memory leaks

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

- Non-sensitive information
- Non-speculative observations

Check with self-composition

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

s_1

s_2

Memory leaks

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

- Non-sensitive information
- Non-speculative observations

Check with self-composition

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

Equivalent wrt ***policy***

s_1

s_2

Memory leaks

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

- Non-sensitive information
- Non-speculative observations

Check with self-composition

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

Equivalent wrt ***policy***

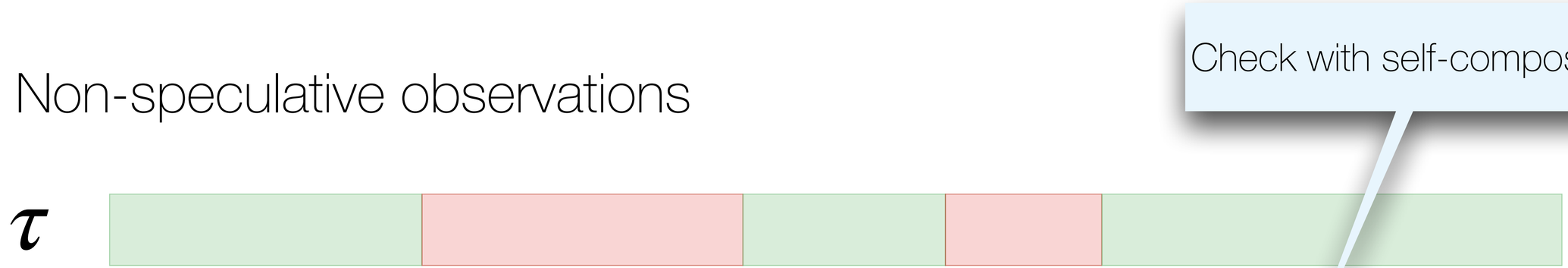
$$s_1 \models \varphi$$

$$s_2 \models \varphi$$

Memory leaks

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

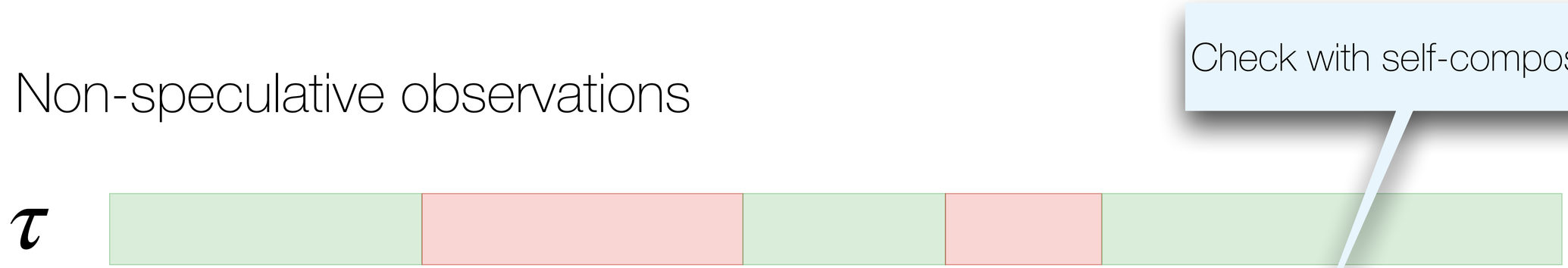
- Non-sensitive information
- Non-speculative observations



Memory leaks

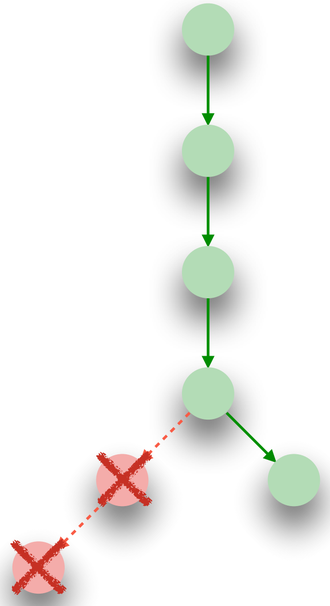
Speculative memory accesses ***must*** depend only on

- Non-sensitive information
- Non-speculative observations



Memory leaks

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



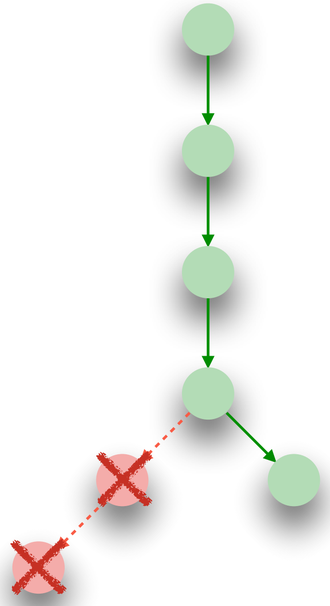
Policy

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

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

Memory leaks

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



Policy

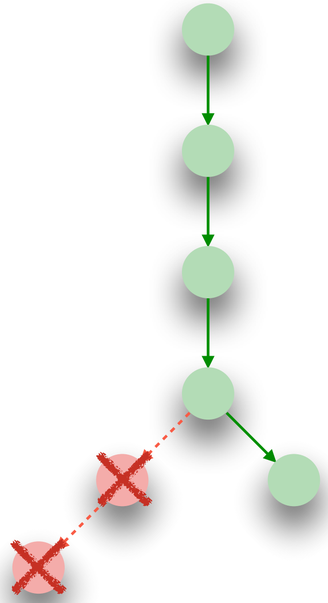
x, **A_size**, **A**, **B** are public

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

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

Memory leaks

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



Policy

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

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

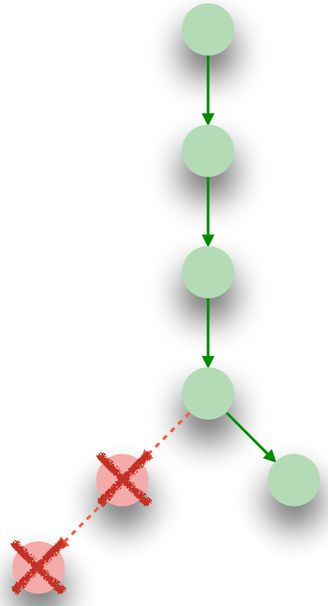
$$pathCnd(\tau) \wedge obsEqv(\tau|_{non-spec}) \wedge \neg obsEqv(\tau|_{spec})$$

s_1

s_2

Memory leaks

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



Policy

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

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

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

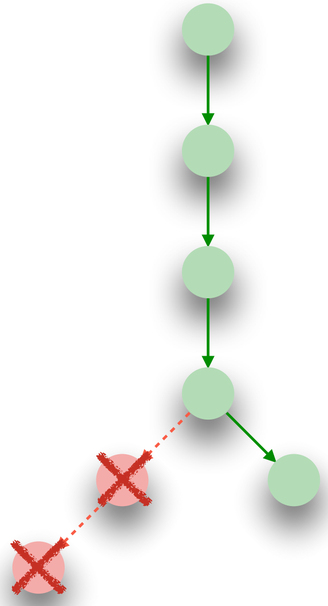
s_1

s_2

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

Memory leaks

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



Policy

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

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

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

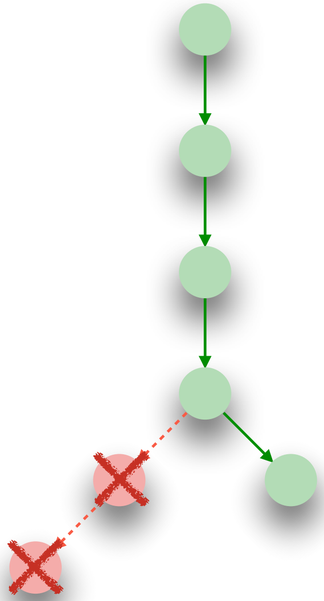
$s_1 \models$ $x_1 \geq A\_size_1$

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

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

Memory leaks

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



Policy

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

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

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

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

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

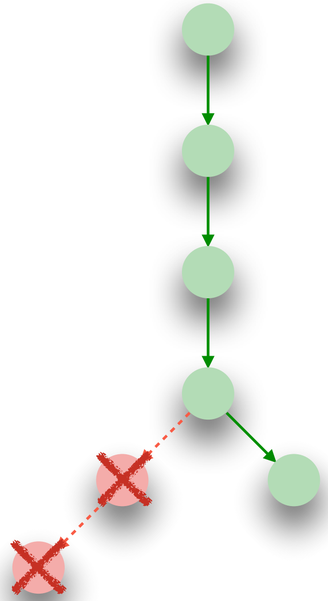
=

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

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

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

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

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

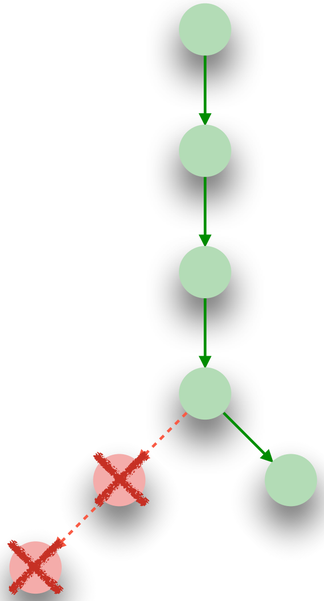
$\Vert$

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

Always true!

Memory leaks

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



Policy

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

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

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

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

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

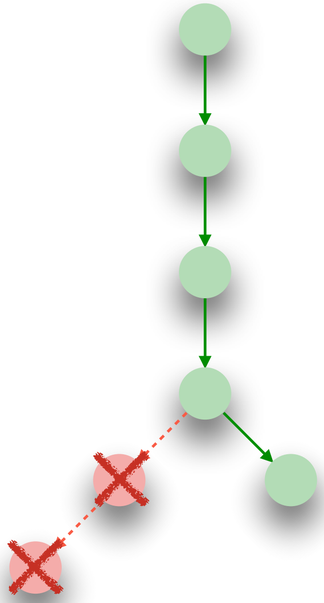
$A_1 + x_1$ ≠ $A_2 + x_2$

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

Always true!

Memory leaks

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



Policy

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

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

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

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

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

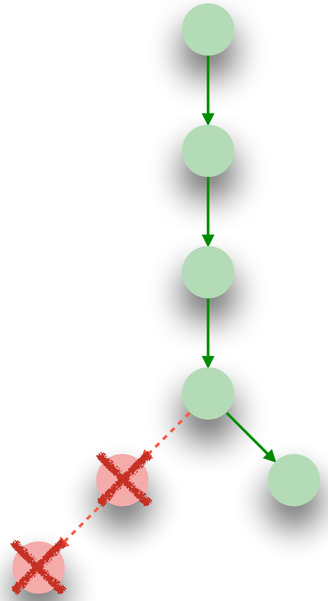
$x_1 = x_2 \wedge A\_size_1 = A\_size_2 \wedge A_1 = A_2 \wedge B_1 = B_2$

$A_1 + x_1$ ∦ $A_2 + x_2$

Always true!

Memory leaks

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



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

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

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

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

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

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

$\not\parallel$

$A_1 + x_1$ $\not\parallel$ $A_2 + x_2$

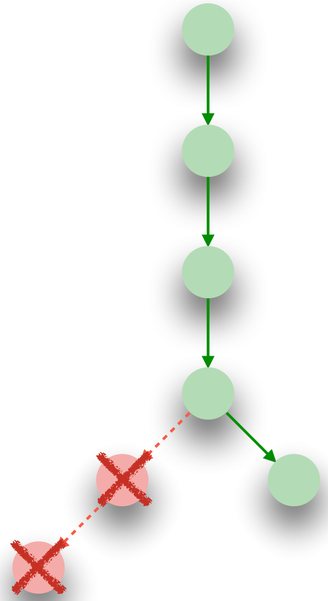
$\vee$

$B_1 + A_1[x_1]$ $\not\parallel$ $B_2 + A_2[x_2]$

Always true!

Memory leaks

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



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

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

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

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

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

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

$\not\parallel$

$A_1 + x_1$ $\not\parallel$ $A_2 + x_2$ $\vee$ $B_1 + A_1[x_1]$ $\not\parallel$ $B_2 + A_2[x_2]$

Always true!

Spectector + Case studies

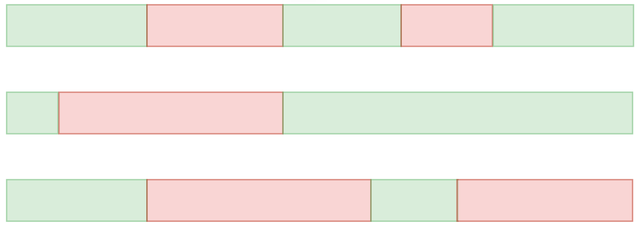
Spectector

```
    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[
```

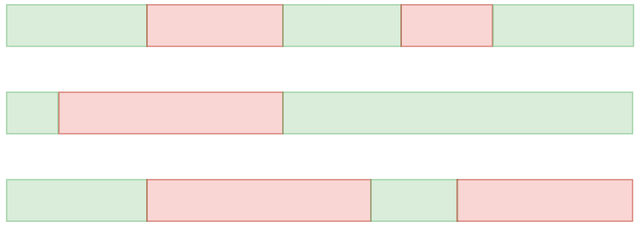
x64 to μASM

More details

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

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

Symbolic execution



Check for speculative leaks

Case study: compiler mitigations

Target:

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

How:

- Use Spectector to prove security or detect leaks

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

Results

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

Results

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

Results

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

Results

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

Results

No countermeasures

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

Results

Automated insertion of fences

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

Results

Speculative load hardening

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

Results

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

Results

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

Results

Ex.	VCC						ICC				CLANG					
	UNP		FEN 19.15		FEN 19.20		UNP		FEN		UNP		FEN		SLH	
	-O0	-O2	-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	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Summary

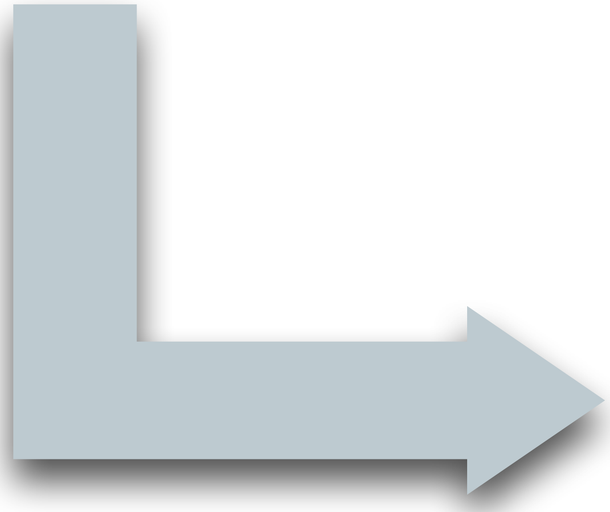
- Leaks in all unprotected programs (except example #08 with optimizations)
- Confirm all vulnerabilities in VCC pointed out by Paul Kocher
- Programs with fences (ICC and Clang) are secure
  - Unnecessary fences
- Programs with SLH are secure except #10 and #15

Example #01 - SLH

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

Example #01 - SLH

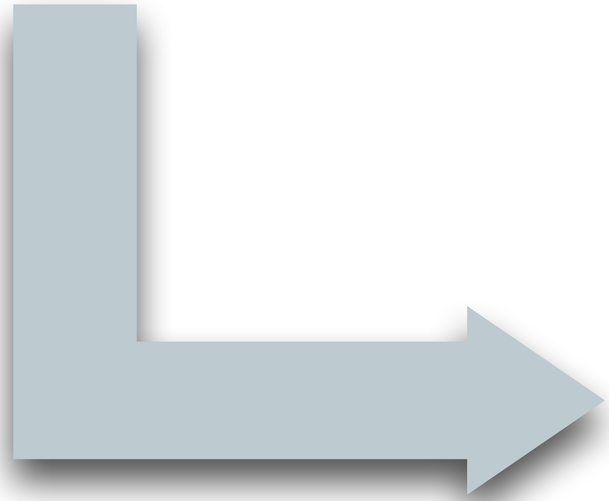
```
if (x < A_size)
   y = B[A[x]*512]
```



```
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]
```



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

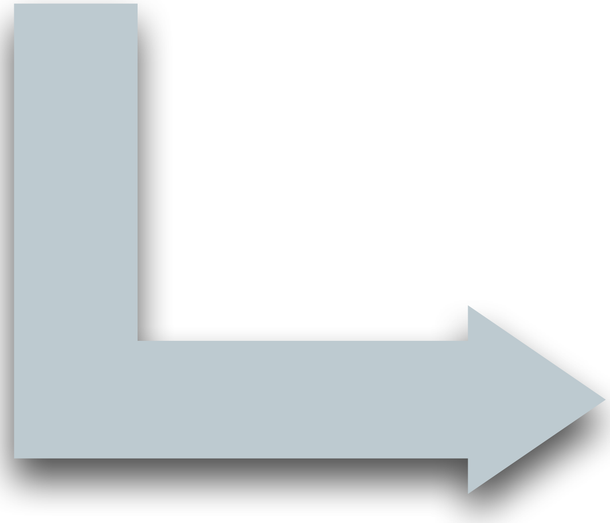
```
mov     rax, A_size
mov     rcx, x
mov     rdx, 0
cmp     rcx, rax
jae     END
cmovae  -1, rdx
mov     rax, A[rcx]
shl     rax, 9
or      rax, rdx
mov     rax, B[rax]
```

Example #10 - SLH

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

Example #10 - SLH

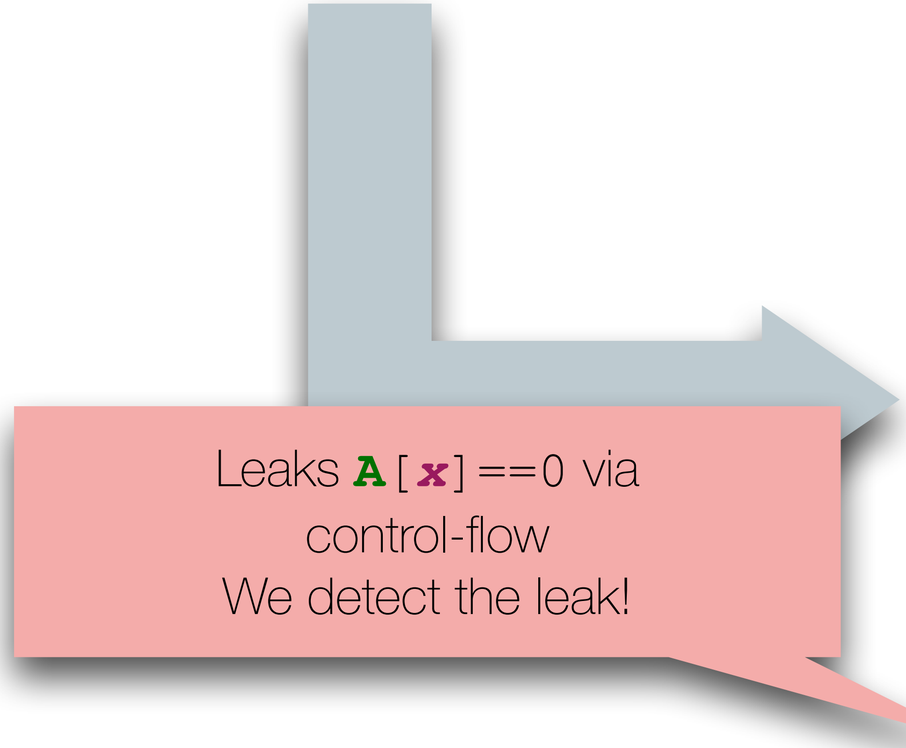
```
if (x < A_size)
   if (A[x]==k)
      y = B[0]
```



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

Example #10 - SLH

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



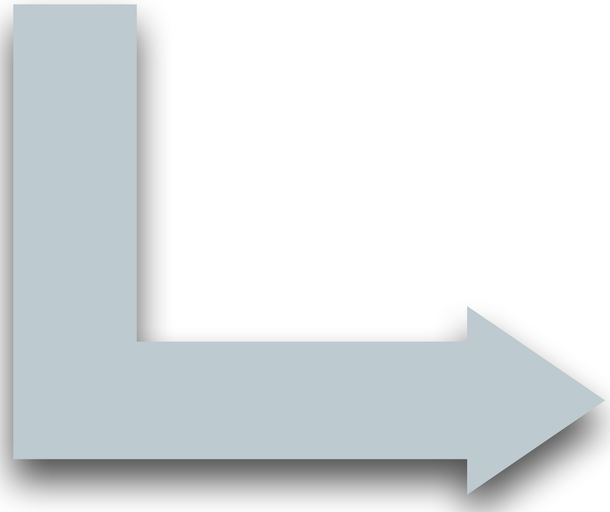

```
mov     rax, A_size
mov     rcx, x
mov     rdx, 0
cmp     rcx, rax
jae     END
cmovae  -1, rdx
mov     rax, A[rcx]
jne     rax, END
cmovne  -1, rdx
mov     rax, [B]
```

Example #08 - FEN

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

Example #08 - FEN

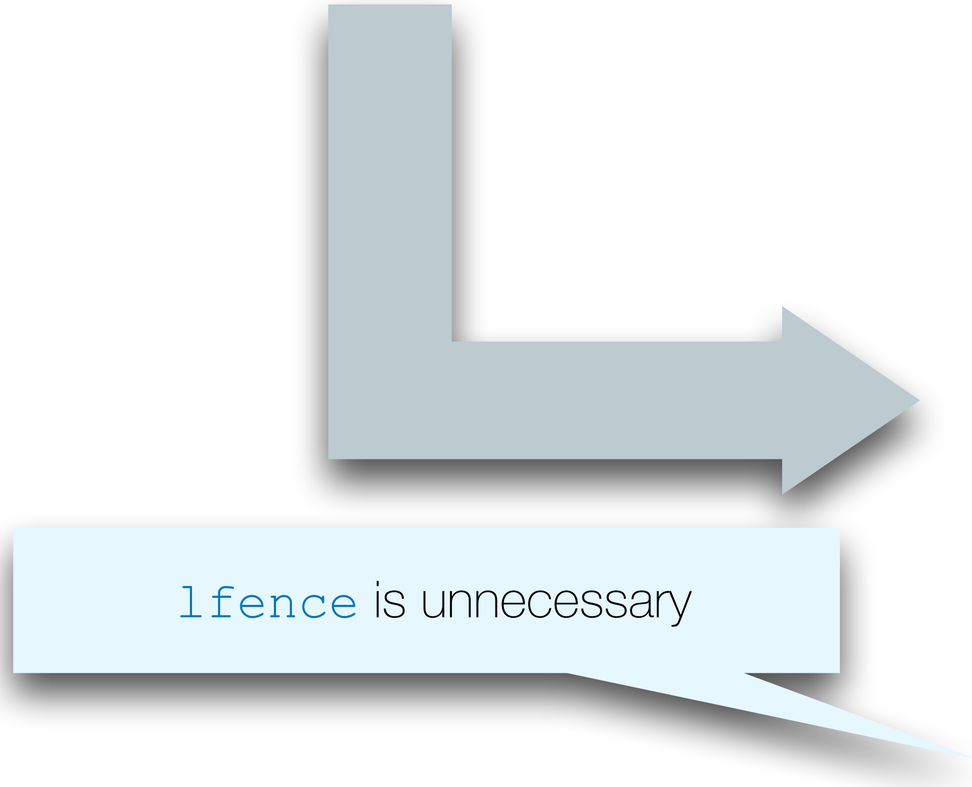
```
y = B[A[x<A_size?(x+1):0]*512]
```



```
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]
```

Conclusion

Speculative non-interference

Formally!

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

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

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

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

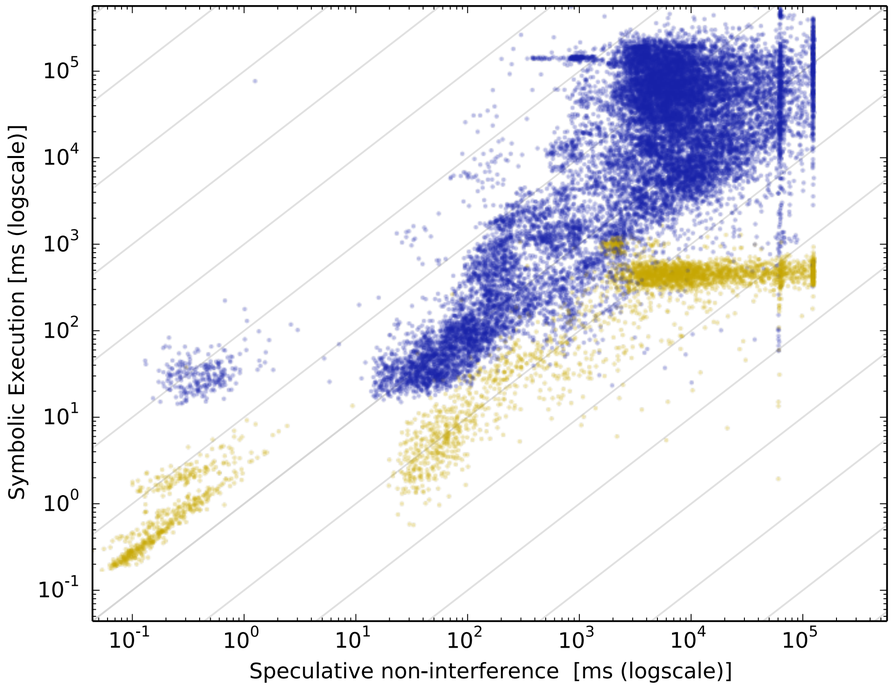
Check for speculative leaks

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
Pnon

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