GadgetMeter: Quantitatively and Accurately Gauging the Exploitability of Speculative Gadgets

Qi Ling, Yujun Liang, Yi Ren, Baris Kasikci, Shuwen Deng

Purdue University, University of Washington, Tsinghua University

Spectre Attacks

01-04-18

“Spectre” And “Meltdown” Chip Flaws Touch “Almost Every System,” Say Researchers

The critical vulnerabilities in Intel, AMD, and ARM processors will “haunt us for some time.”

```
e01d8340: 65 3a 20 68 74 74 70 3a  2f 2f 6d 65 73 61 33 64 |e: http://mesa3
e01d8350: 2e 6f 72 67 2f 0a 4f 72  69 67 69 6e 61 6c 2d 4d |.org/.Original
e01d8360: 61 69 6e 74 61 69 6e 65  72 3a 20 44 65 62 69 61 |aintainer: De
```

HOME IT/ENTERPRISE

New Spectre Chip Security Vulnerability Found That Leaves Billions Of PCs Still Defenseless

by Nathan Ord — Saturday, May 01, 2021, 10:04 AM EDT

Spectre Attacks

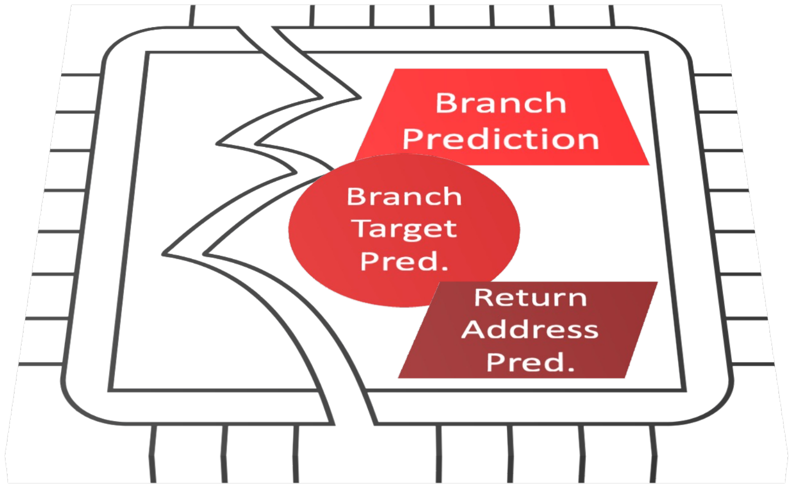
01-04-18

“Spectre” And “Meltdown” Chip Flaws Touch “Almost Every System,” Say Researchers

The critical vulnerabilities in Intel, AMD, and ARM processors will “haunt us for some time.”

New Spectre Chip Security Vulnerability Found That Leaves Billions Of PCs Still Defenseless

by Nathan Ord — Saturday, May 01, 2021, 10:04 AM EDT



Spectre Attacks

01-04-18

“Spectre” And “Meltdown” Chip Flaws Touch “Almost Every System,” Say Researchers

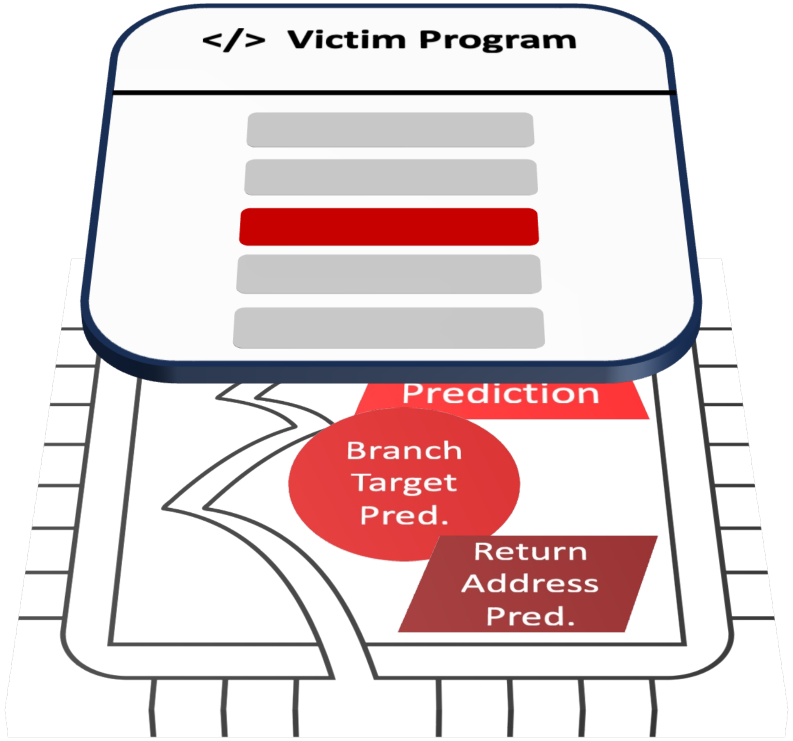
The critical vulnerabilities in Intel, AMD, and ARM processors will “haunt us for some time.”

```
e01d8340: 65 3a 20 68 74 74 70 3a  2f 2f 6d 65 73 61 33 64 |e: http://mesa3
e01d8350: 2e 6f 72 67 2f 0a 4f 72  69 67 69 6e 61 6c 2d 4d |.org/.Original
e01d8360: 61 69 6e 74 61 69 6e 65  72 3a 20 44 65 62 69 61 |aintainer: De
```

HOME IT/ENTERPRISE

New Spectre Chip Security Vulnerability Found That Leaves Billions Of PCs Still Defenseless

by Nathan Ord — Saturday, May 01, 2021, 10:04 AM EDT



Spectre Attacks

01-04-18

“Spectre” And “Meltdown” Chip Flaws Touch “Almost Every System,” Say Researchers

The critical vulnerabilities in Intel, AMD, and ARM processors will “haunt us for some time.”

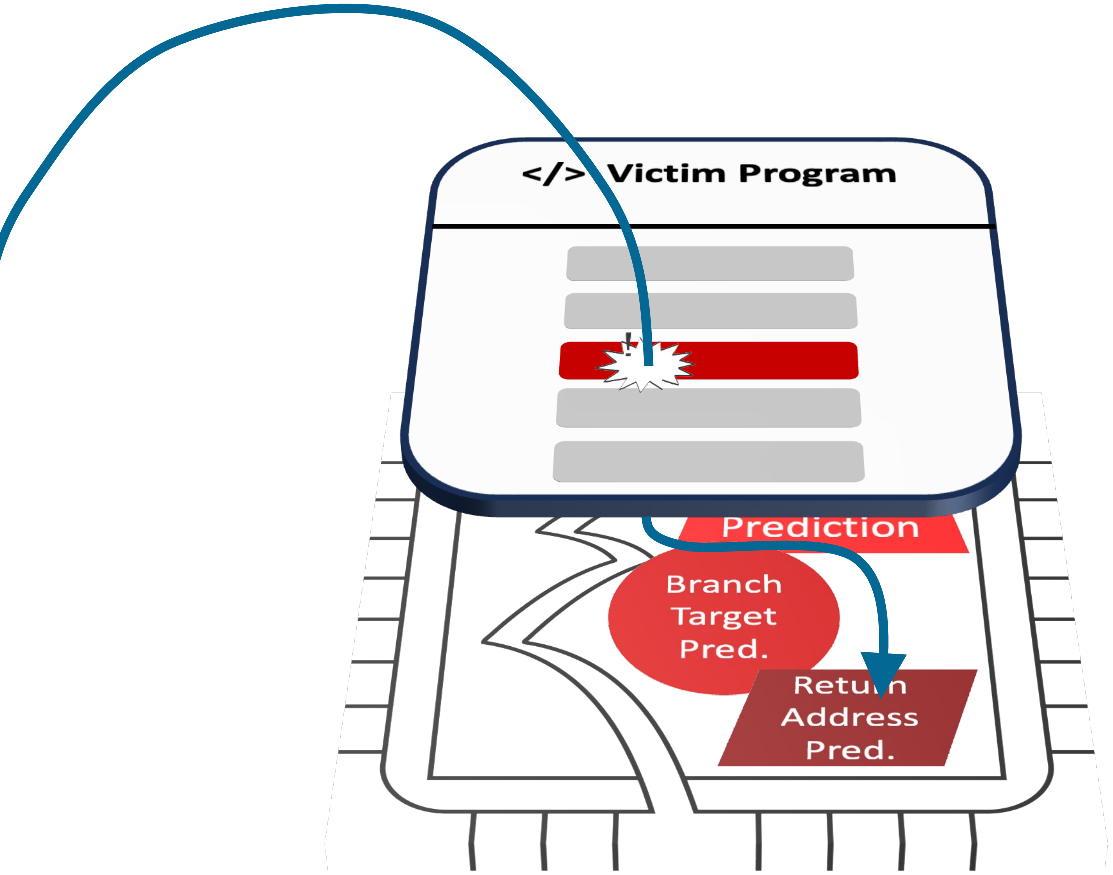
```
e01d8340: 65 3a 20 68 74 74 70 3a  2f 2f 6d 65 73 61 33 64 |e: http://mesa3
e01d8350: 2e 6f 72 67 2f 0a 4f 72  69 67 69 6e 61 6c 2d 4d |.org/.Original
e01d8360: 61 69 6e 74 61 69 6e 65  72 3a 20 44 65 62 69 61 |aintainer: De
```

HOME IT/ENTERPRISE

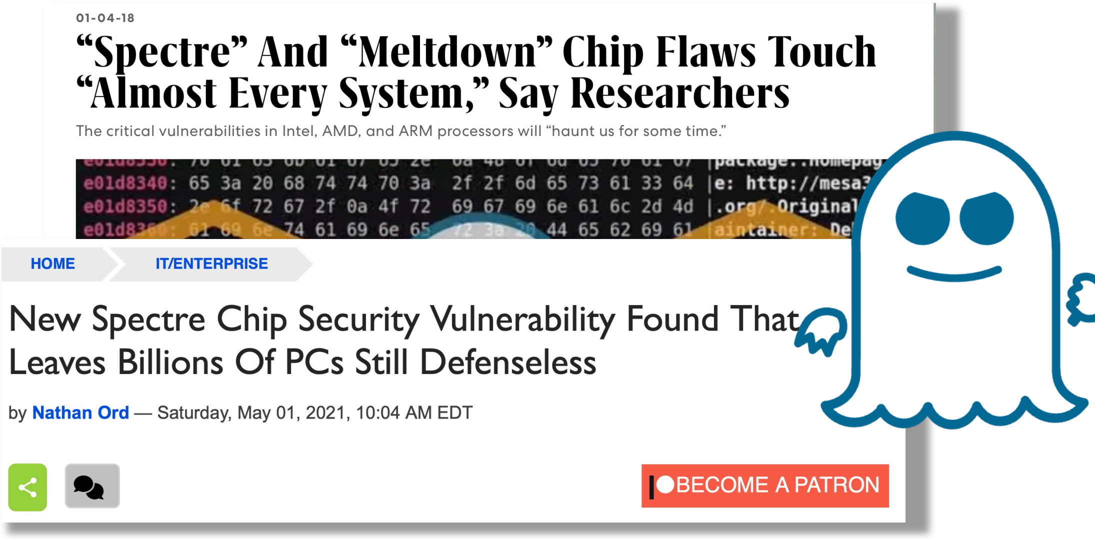
New Spectre Chip Security Vulnerability Found That Leaves Billions Of PCs Still Defenseless

by Nathan Ord — Saturday, May 01, 2021, 10:04 AM EDT

BECOME A PATRON



Spectre Attacks



01-04-18

“Spectre” And “Meltdown” Chip Flaws Touch “Almost Every System,” Say Researchers

The critical vulnerabilities in Intel, AMD, and ARM processors will “haunt us for some time.”

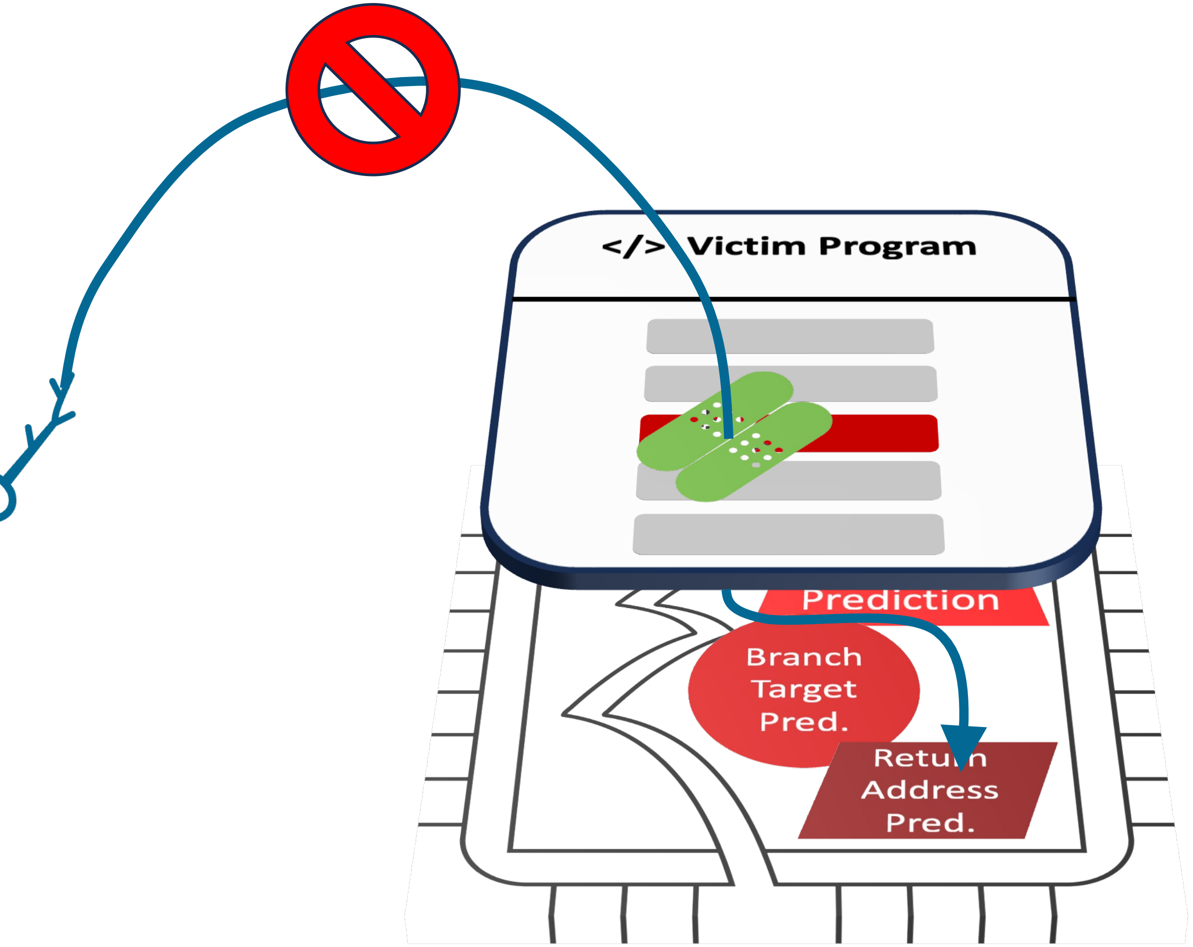
```
e01d8340: 65 3a 20 68 74 74 70 3a  2f 2f 6d 65 73 61 33 64 |e: http://mesa3
e01d8350: 2e 6f 72 67 2f 0a 4f 72  69 67 69 6e 61 6c 2d 4d |.org/.Original
e01d8360: 61 69 6e 74 61 69 6e 65  72 3a 20 44 65 62 69 61 |aintainer: De
```

HOME | IT/ENTERPRISE

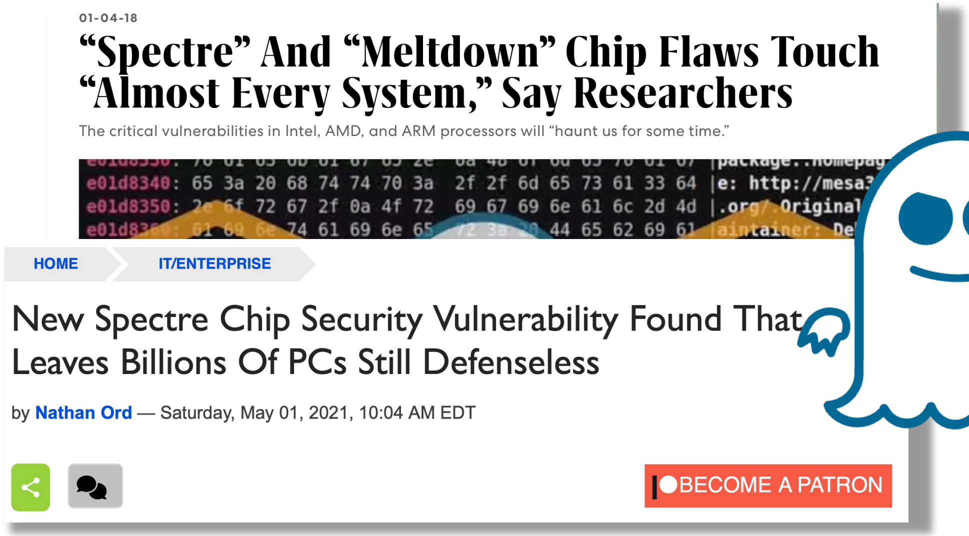
New Spectre Chip Security Vulnerability Found That Leaves Billions Of PCs Still Defenseless

by Nathan Ord — Saturday, May 01, 2021, 10:04 AM EDT

BECOME A PATRON



Spectre Attacks



01-04-18

“Spectre” And “Meltdown” Chip Flaws Touch “Almost Every System,” Say Researchers

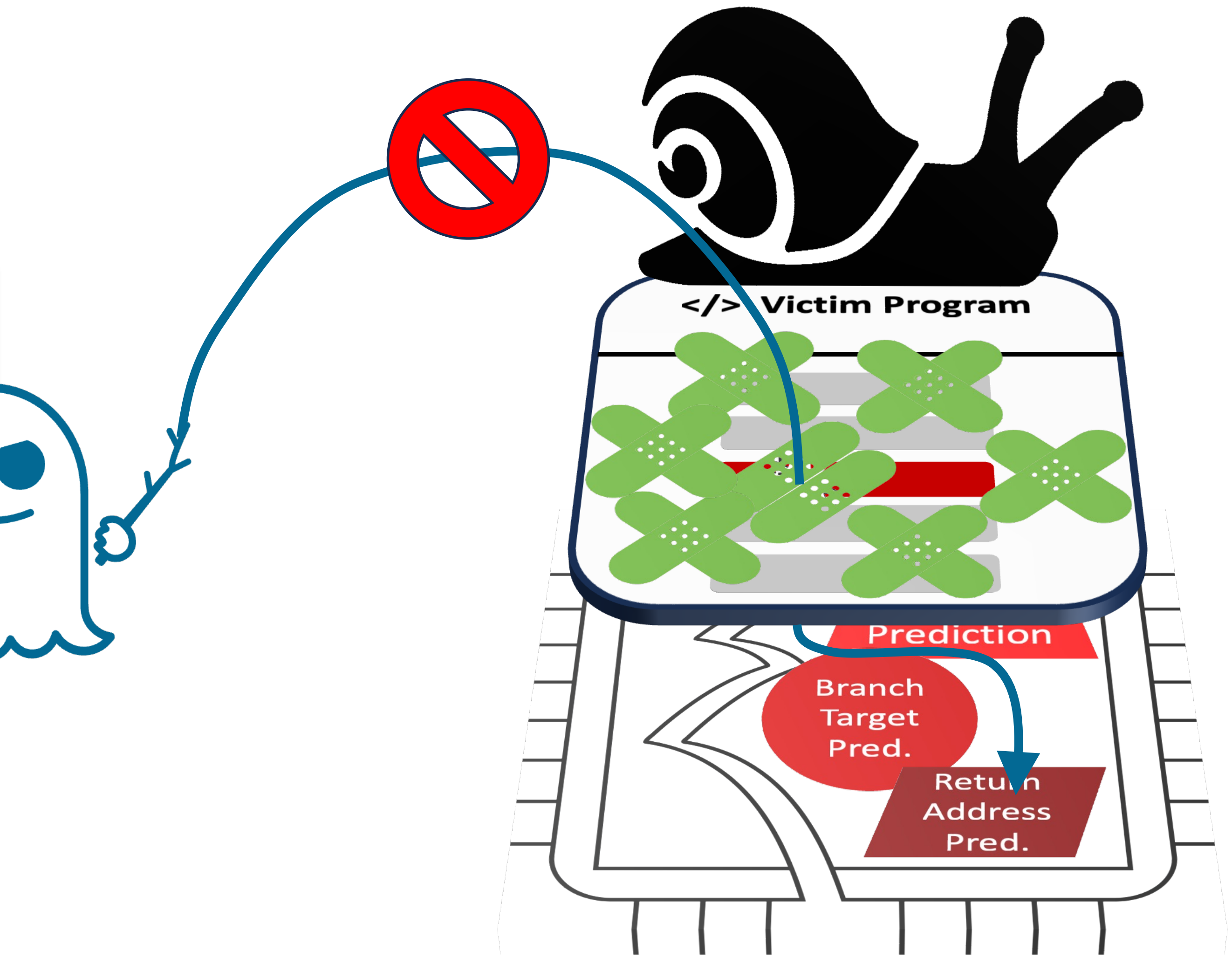
The critical vulnerabilities in Intel, AMD, and ARM processors will “haunt us for some time.”

```
e01d8340: 65 3a 20 68 74 74 70 3a  2f 2f 6d 65 73 61 33 64 |e: http://mesa3
e01d8350: 2e 6f 72 67 2f 0a 4f 72  69 67 69 6e 61 6c 2d 4d |.org/.Original
e01d8360: 61 69 6e 74 61 69 6e 65  72 3a 20 44 65 62 69 61 |aintainer: De
```

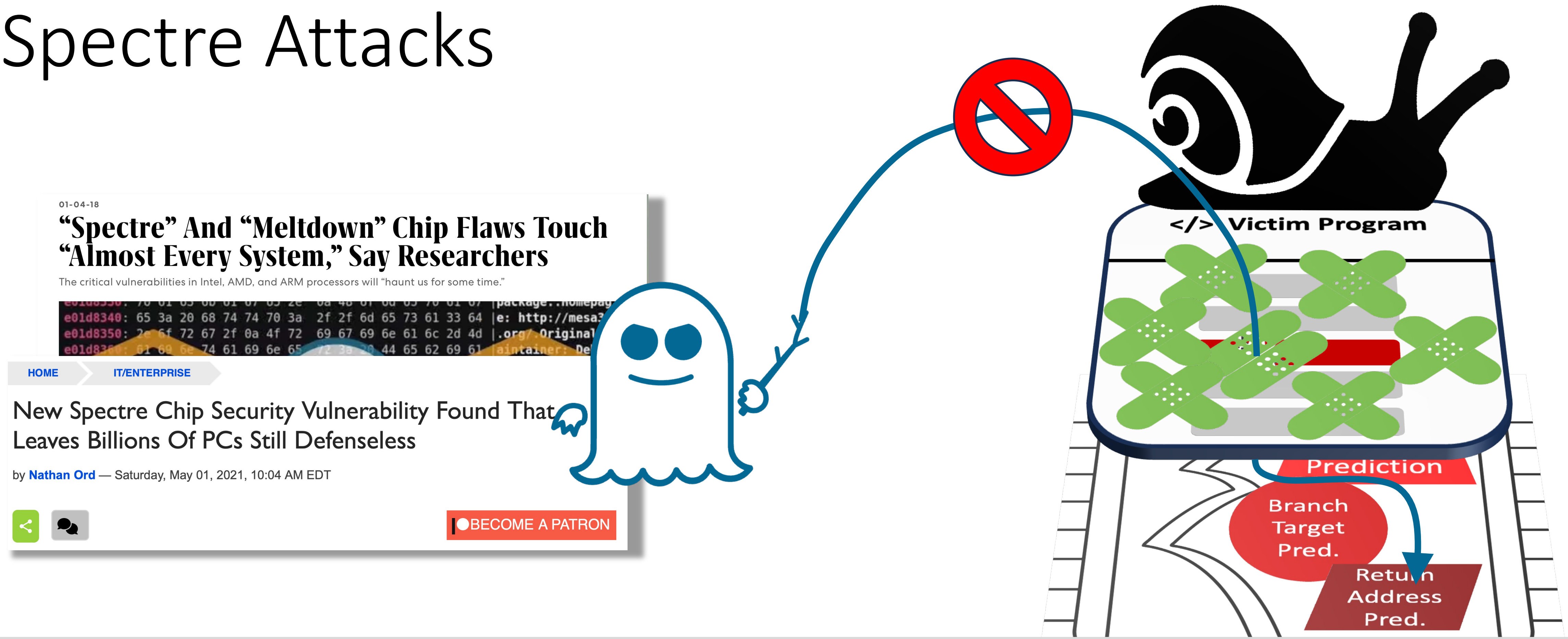
HOME | IT/ENTERPRISE

New Spectre Chip Security Vulnerability Found That Leaves Billions Of PCs Still Defenseless

by Nathan Ord — Saturday, May 01, 2021, 10:04 AM EDT



Spectre Attacks



01-04-18

“Spectre” And “Meltdown” Chip Flaws Touch “Almost Every System,” Say Researchers

The critical vulnerabilities in Intel, AMD, and ARM processors will “haunt us for some time.”

```
e01d8340: 65 3a 20 68 74 74 70 3a  2f 2f 6d 65 73 61 33 64 |e: http://mesa3
e01d8350: 2e 6f 72 67 2f 0a 4f 72  69 67 69 6e 61 6c 2d 4d |.org/.Original
e01d8360: 61 69 6e 74 61 69 6e 65  72 3a 20 44 65 62 69 61 |aintainer: De
```

HOME | IT/ENTERPRISE

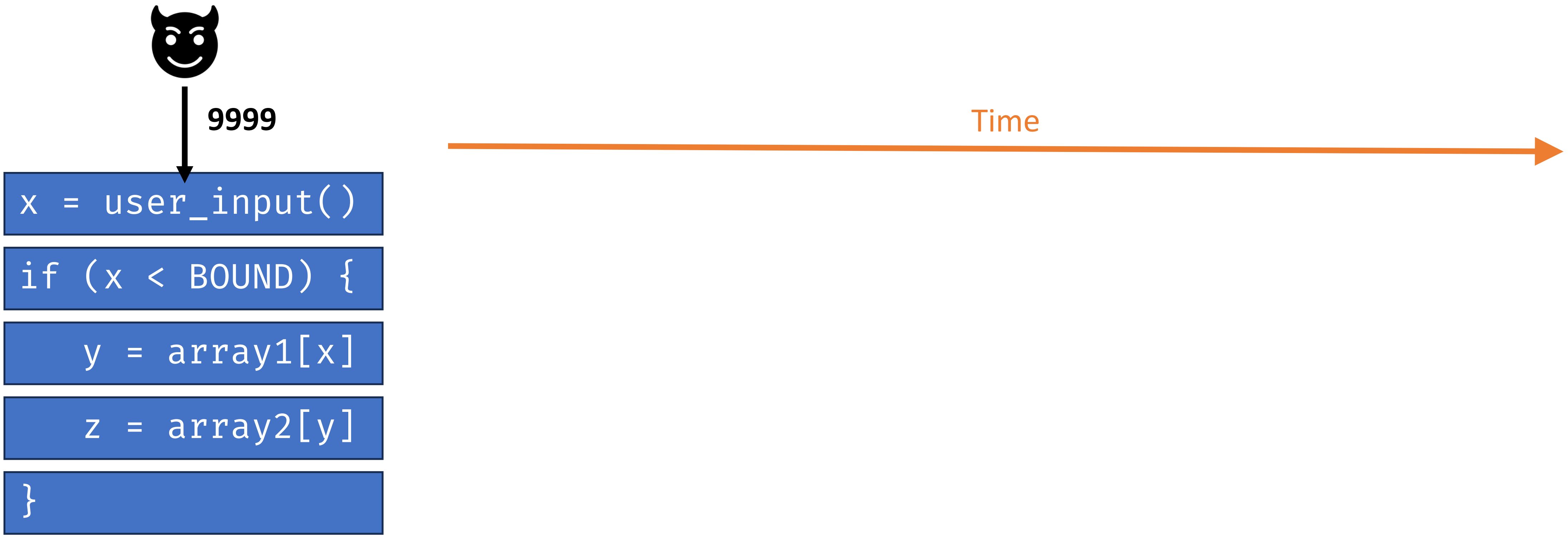
New Spectre Chip Security Vulnerability Found That Leaves Billions Of PCs Still Defenseless

by Nathan Ord — Saturday, May 01, 2021, 10:04 AM EDT

BECOME A PATRON

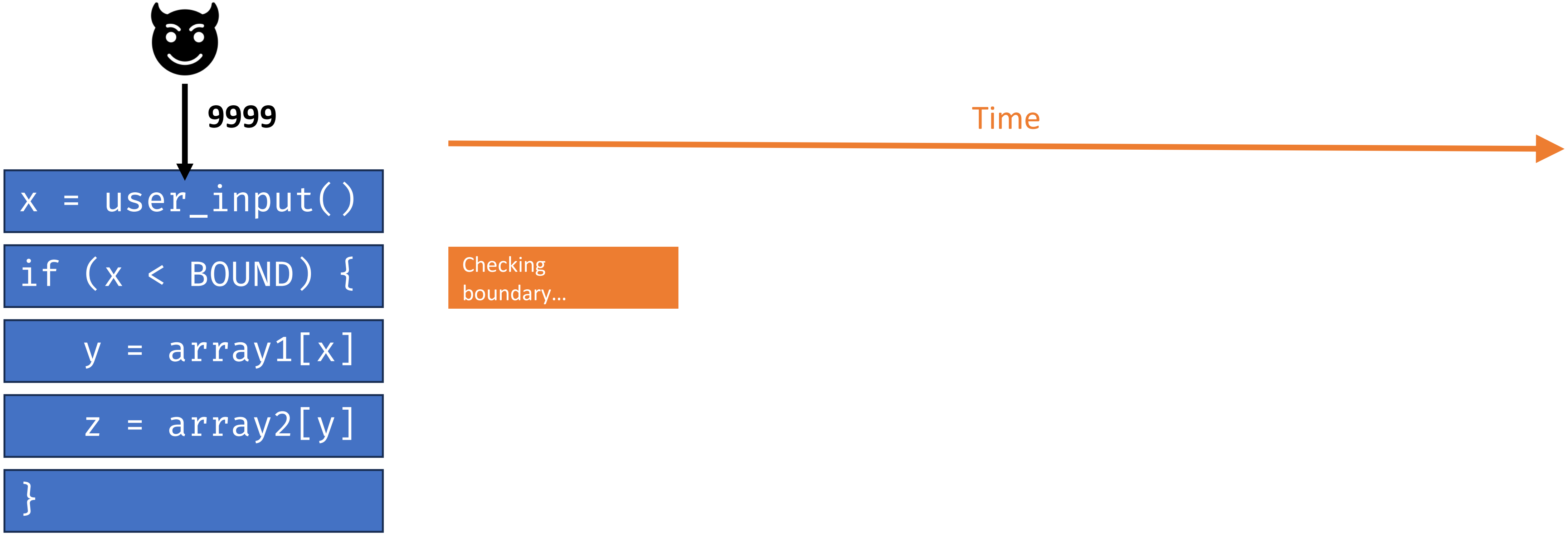
Accurate detection of Spectre gadgets.

What is a Spectre gadget?



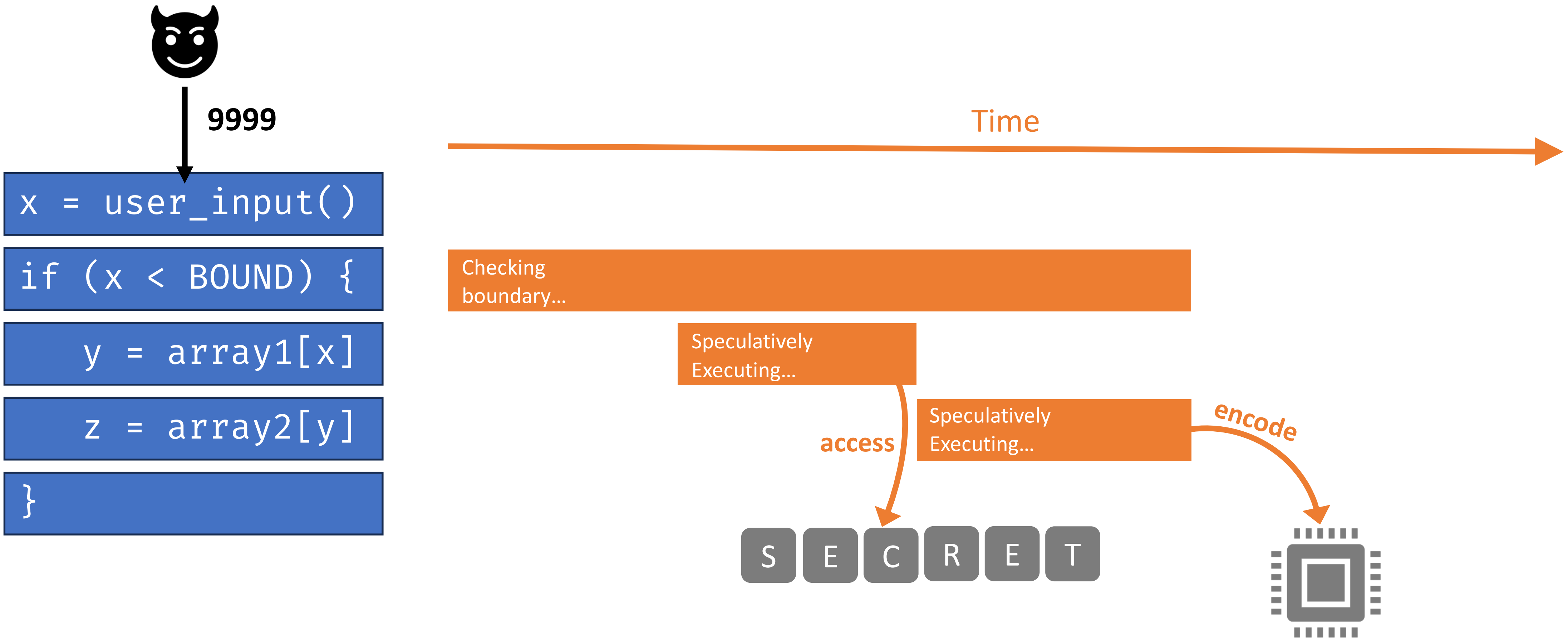
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

What is a Spectre gadget?



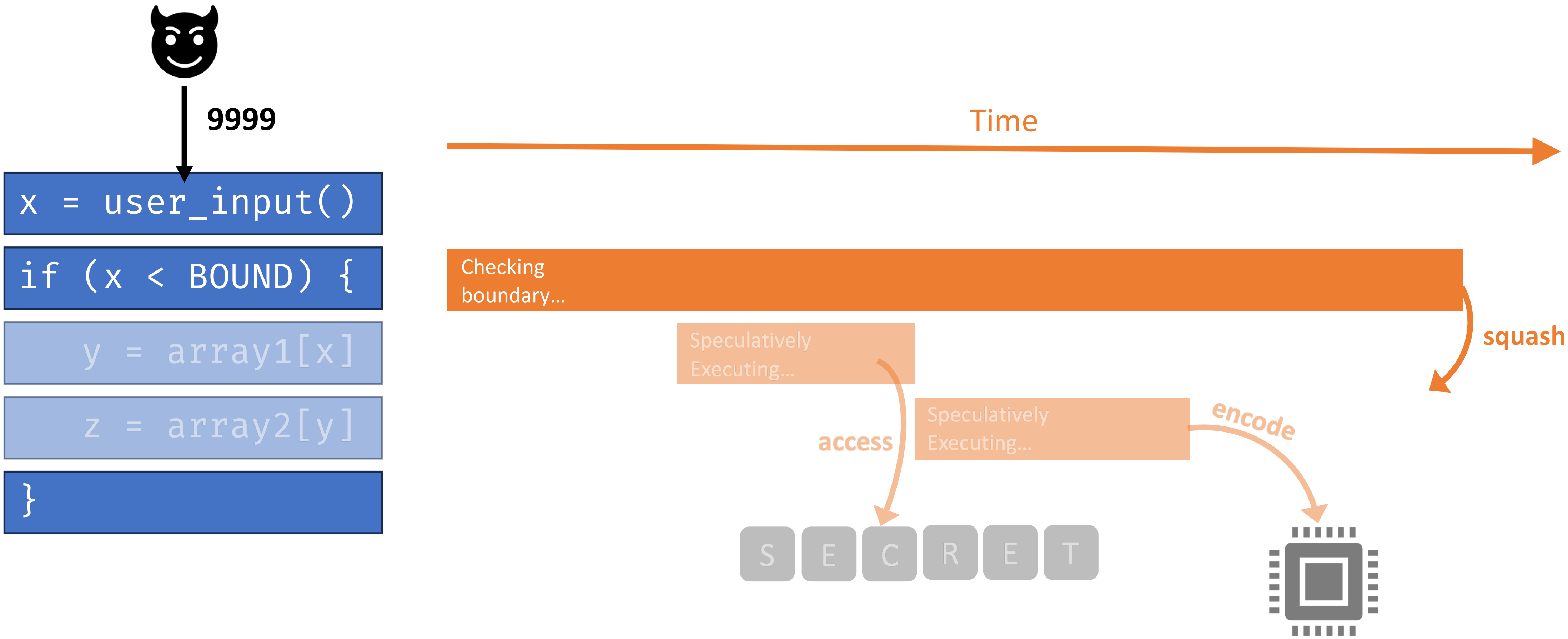
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

What is a Spectre gadget?



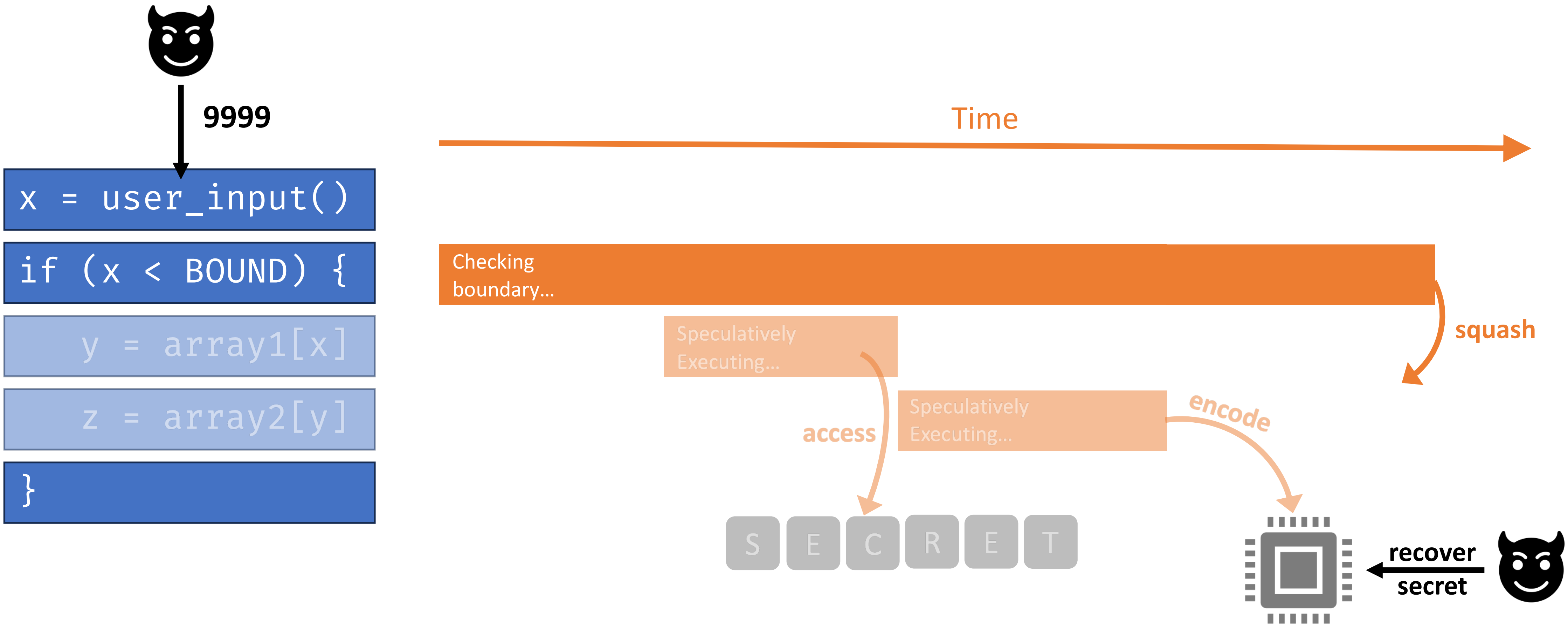
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

What is a Spectre gadget?



```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

What is a Spectre gadget?

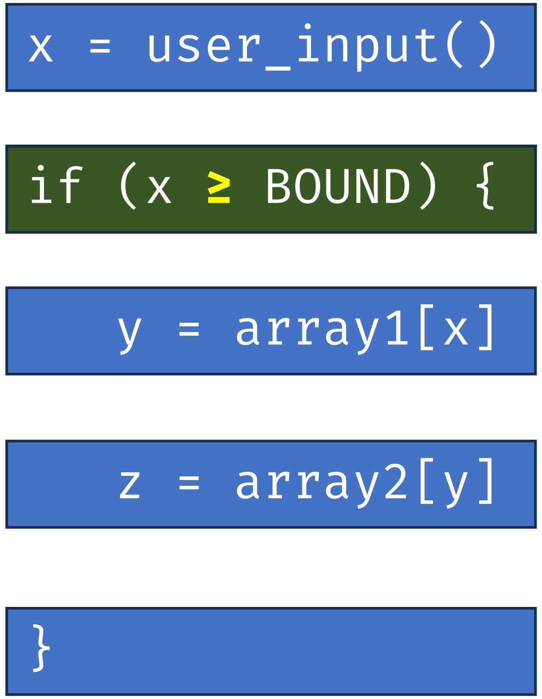


```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

How do existing gadget scanners work?

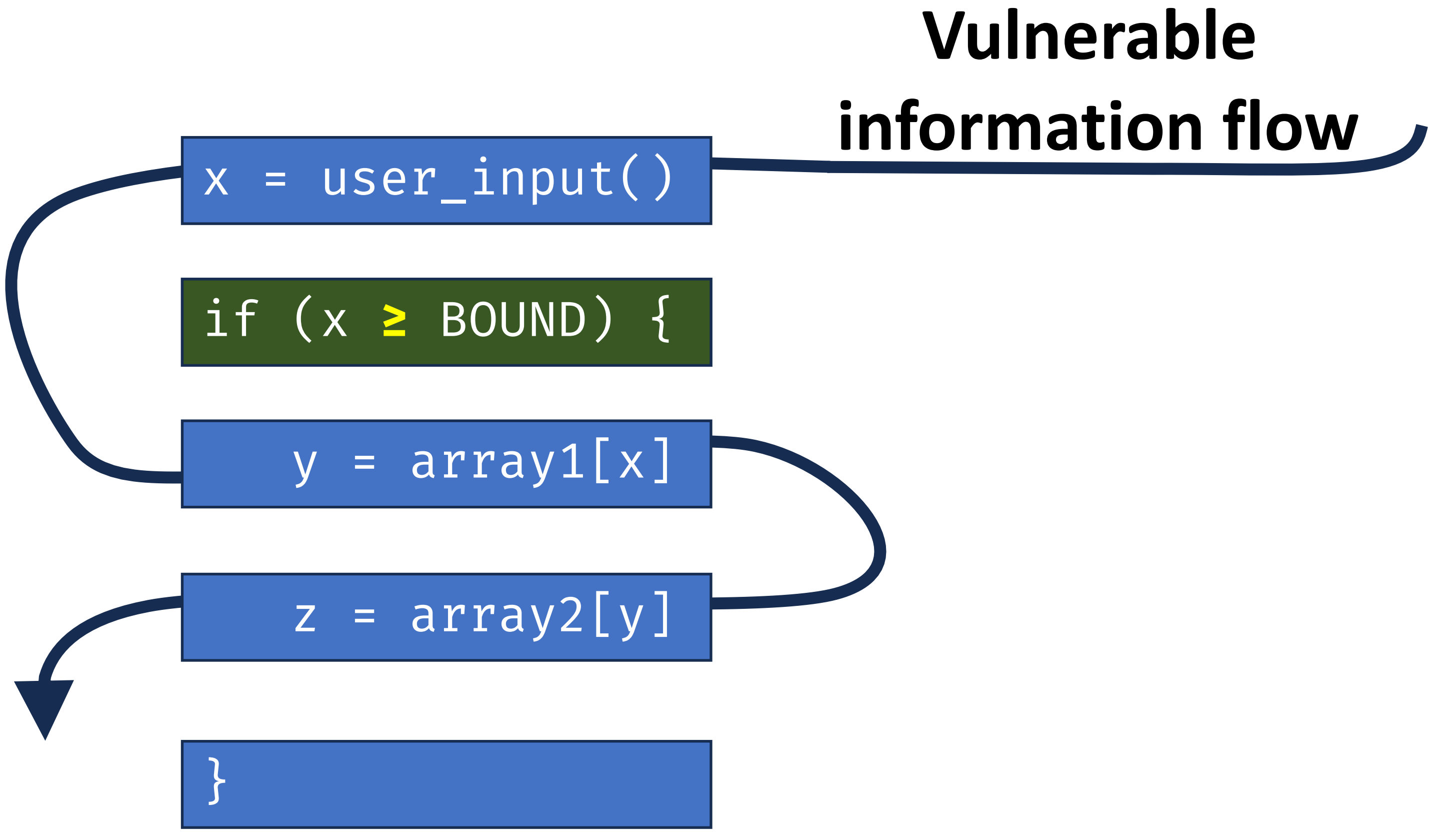
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

How do existing gadget scanners work?



```
x = user_input()
if (x ≥ BOUND) {
   y = array1[x]
   z = array2[y]
}
```

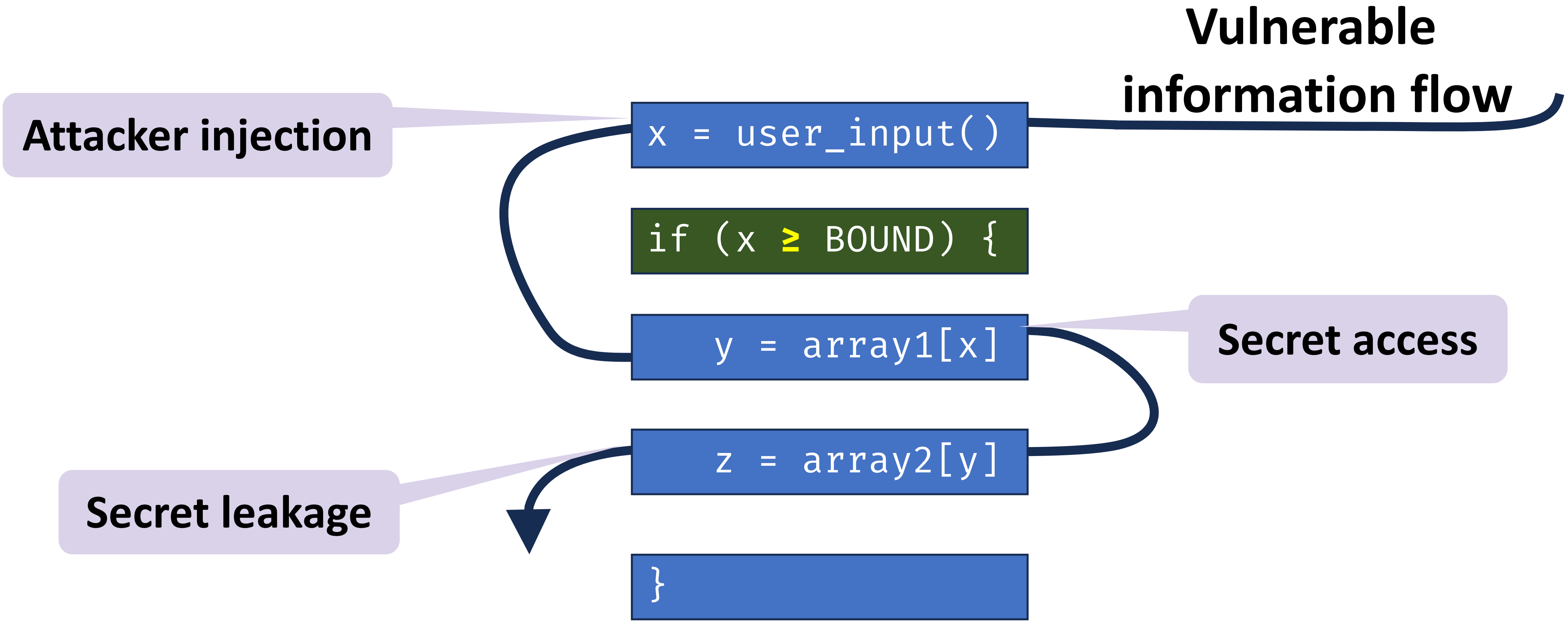
How do existing gadget scanners work?



Vulnerable information flow

```
x = user_input()
if (x ≥ BOUND) {
    y = array1[x]
    z = array2[y]
}
```

How do existing gadget scanners work?



```
x = user_input()
if (x ≥ BOUND) {
    y = array1[x]
    z = array2[y]
}
```

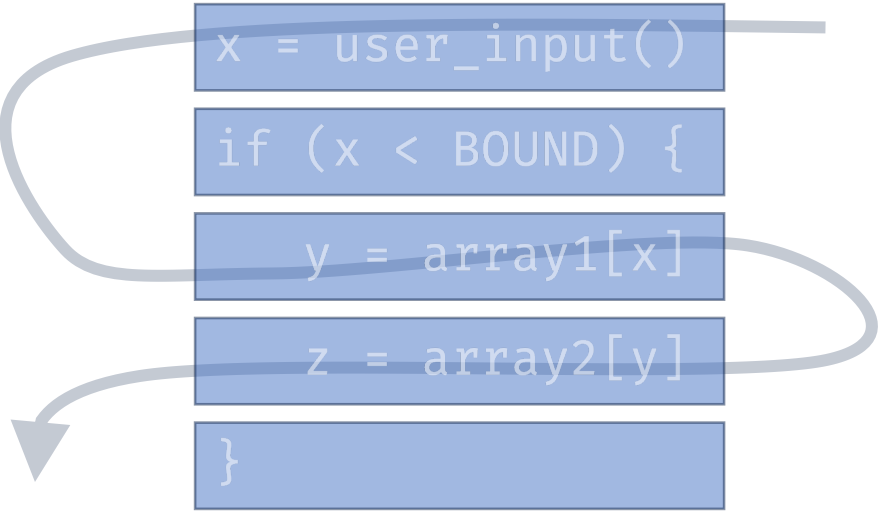
Vulnerable information flow

Attacker injection

Secret access

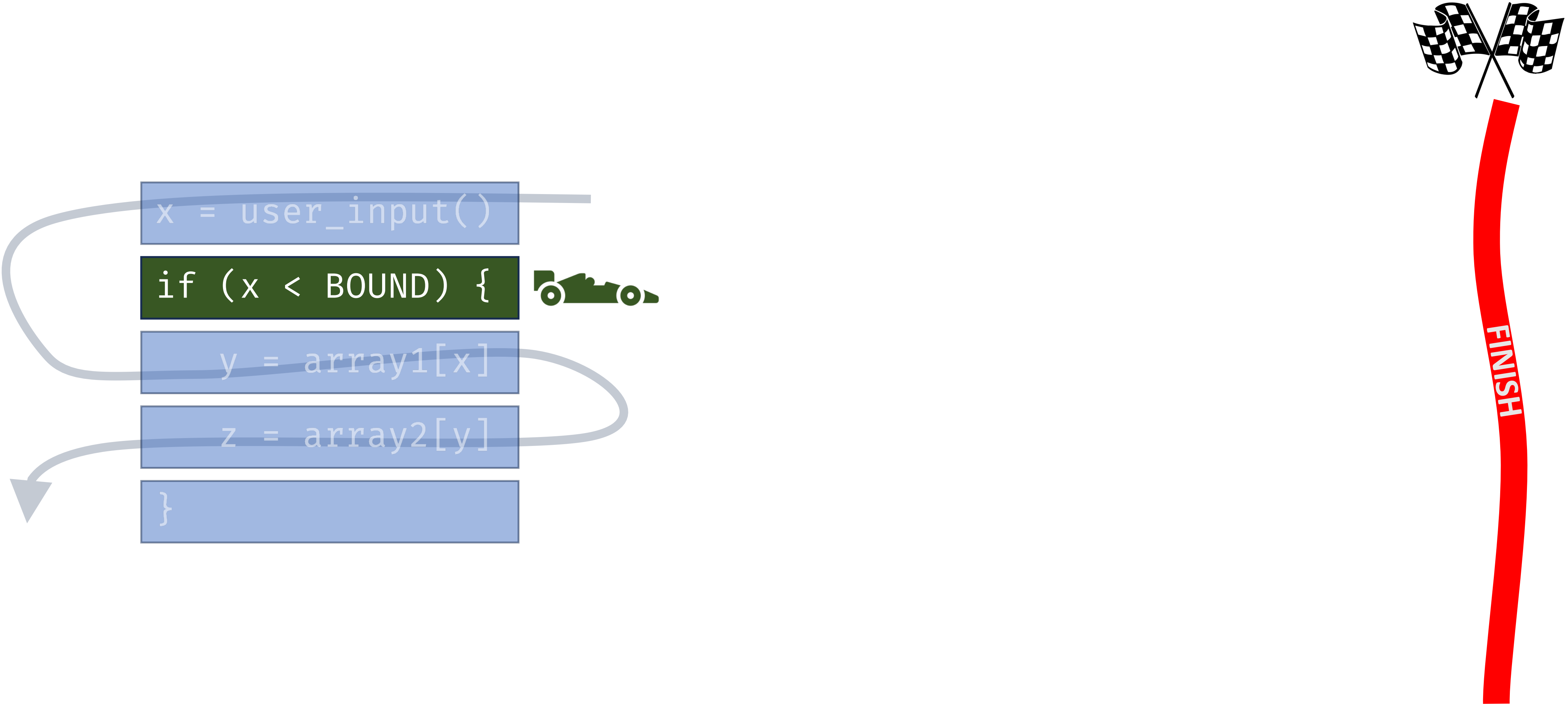
Secret leakage

Missing piece: Timing condition



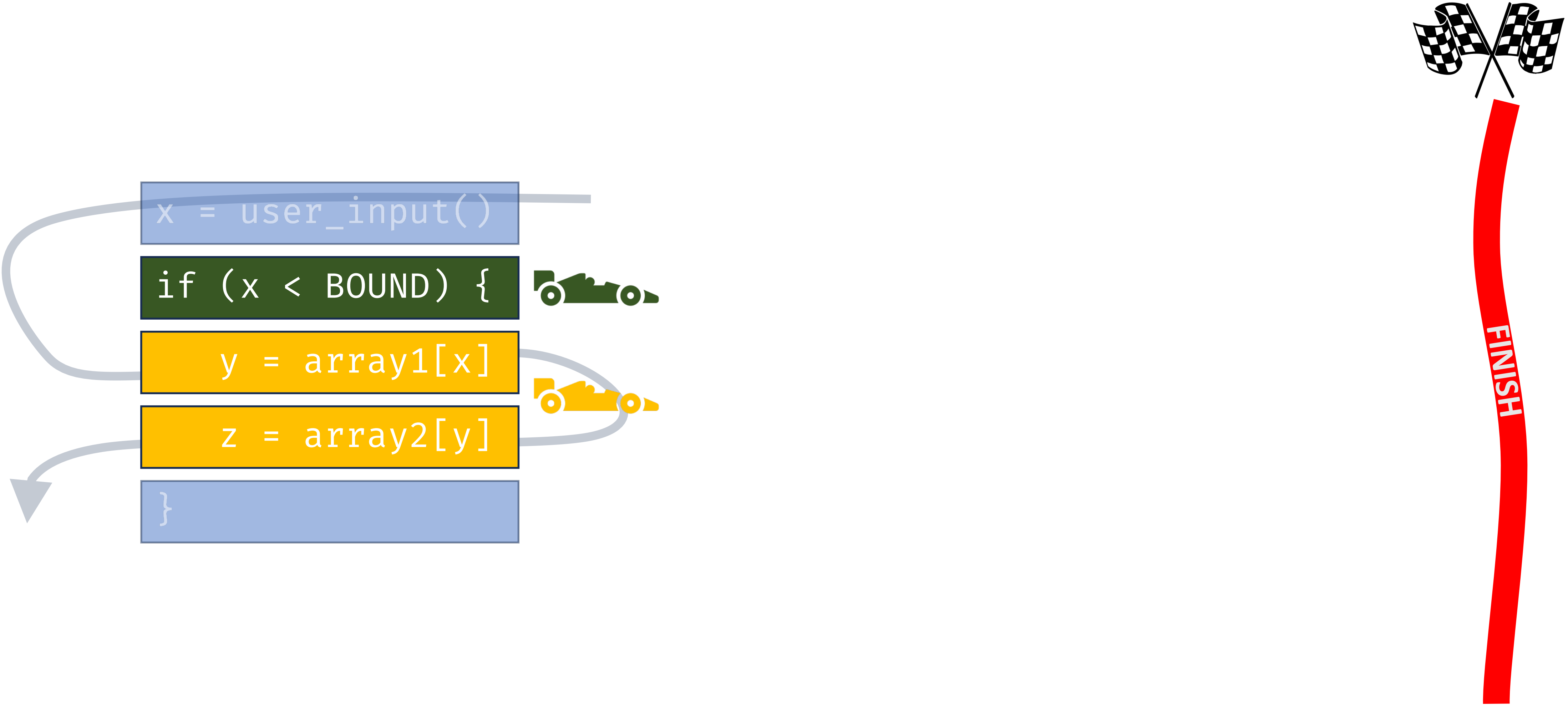
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Missing piece: Timing condition



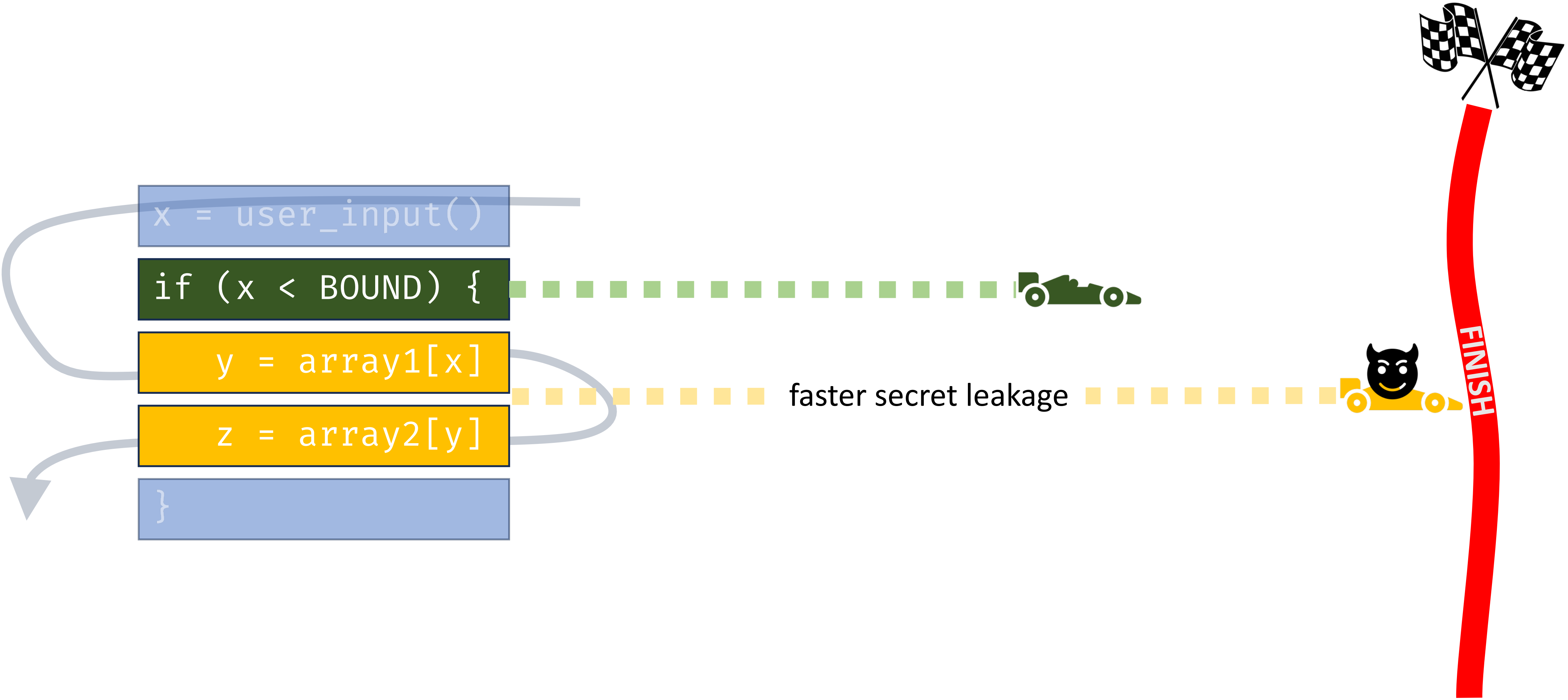
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Missing piece: Timing condition



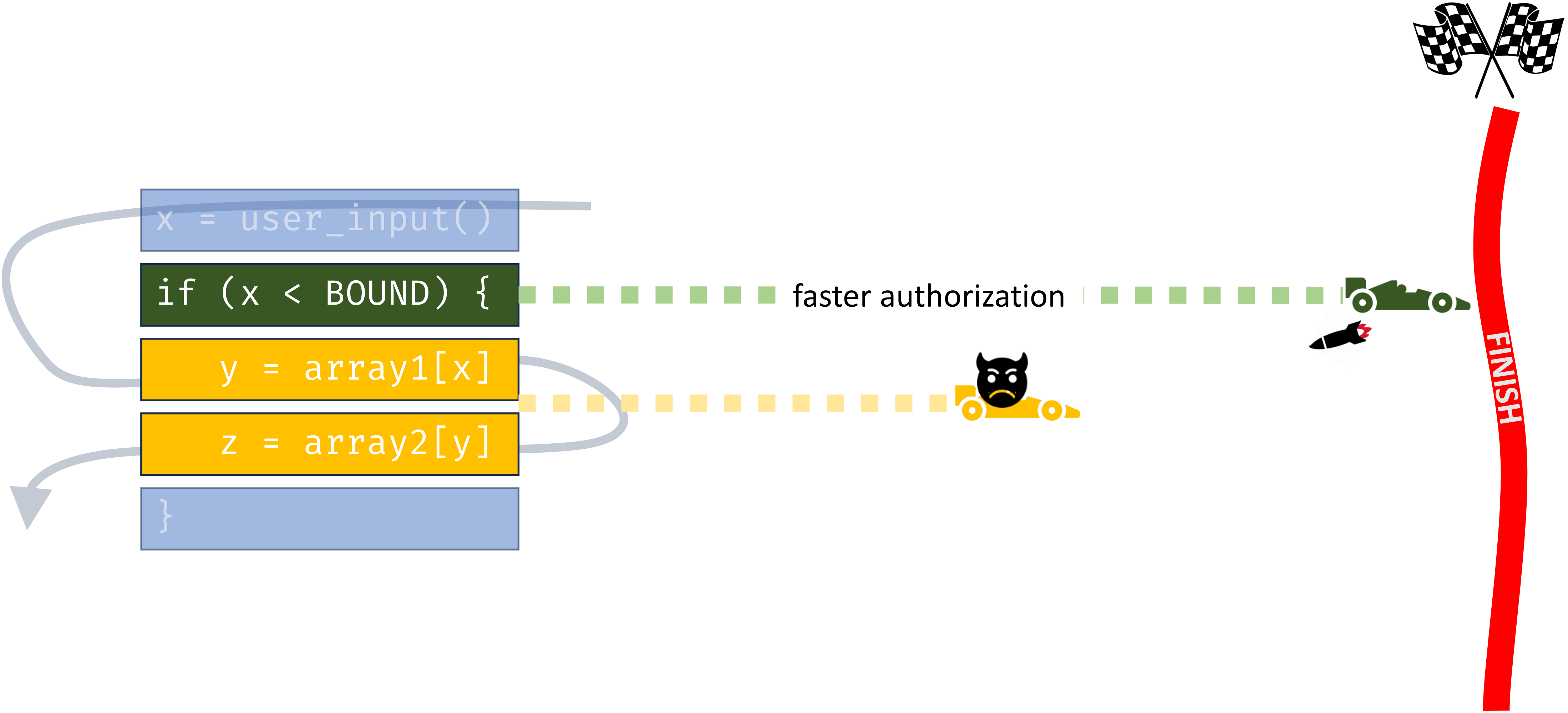
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Missing piece: Timing condition



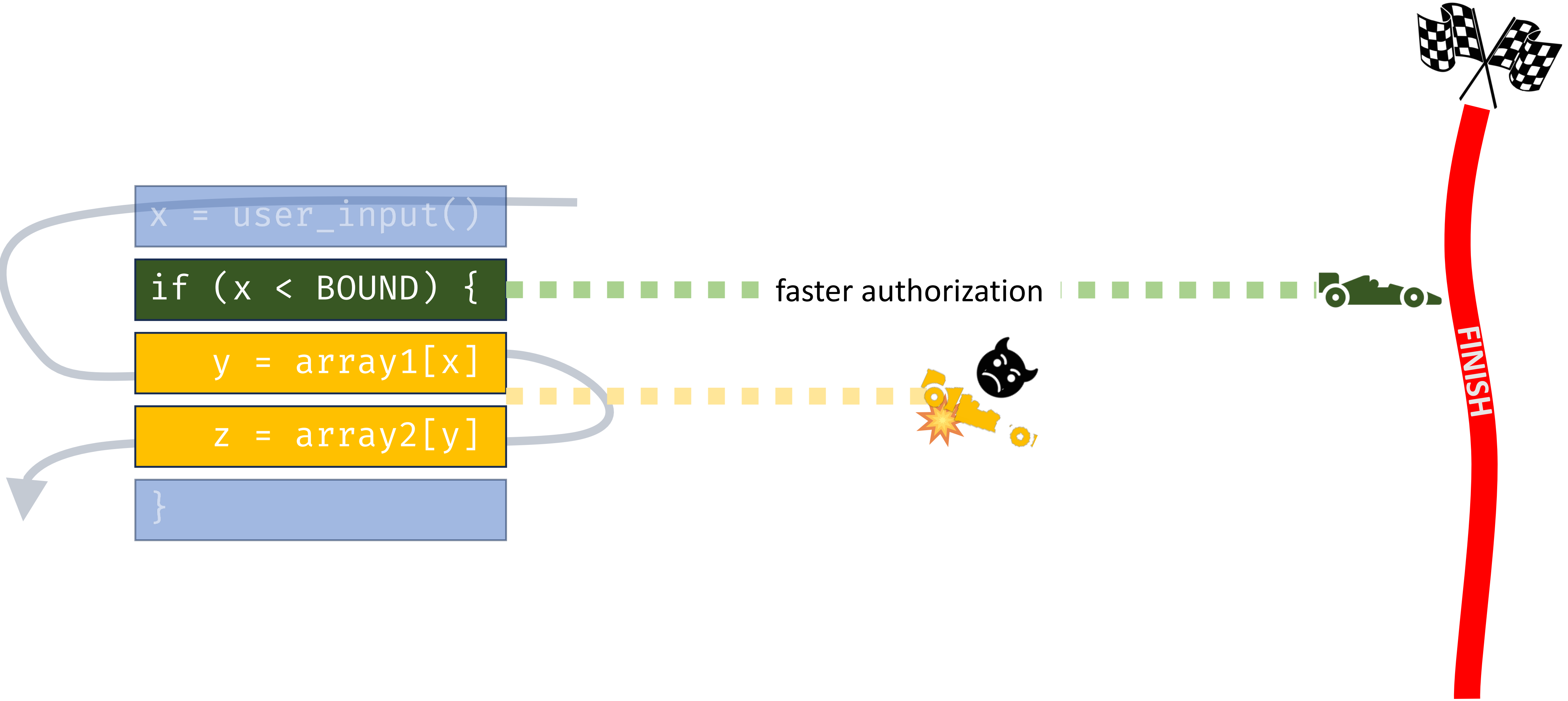
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Missing piece: Timing condition



```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Missing piece: Timing condition



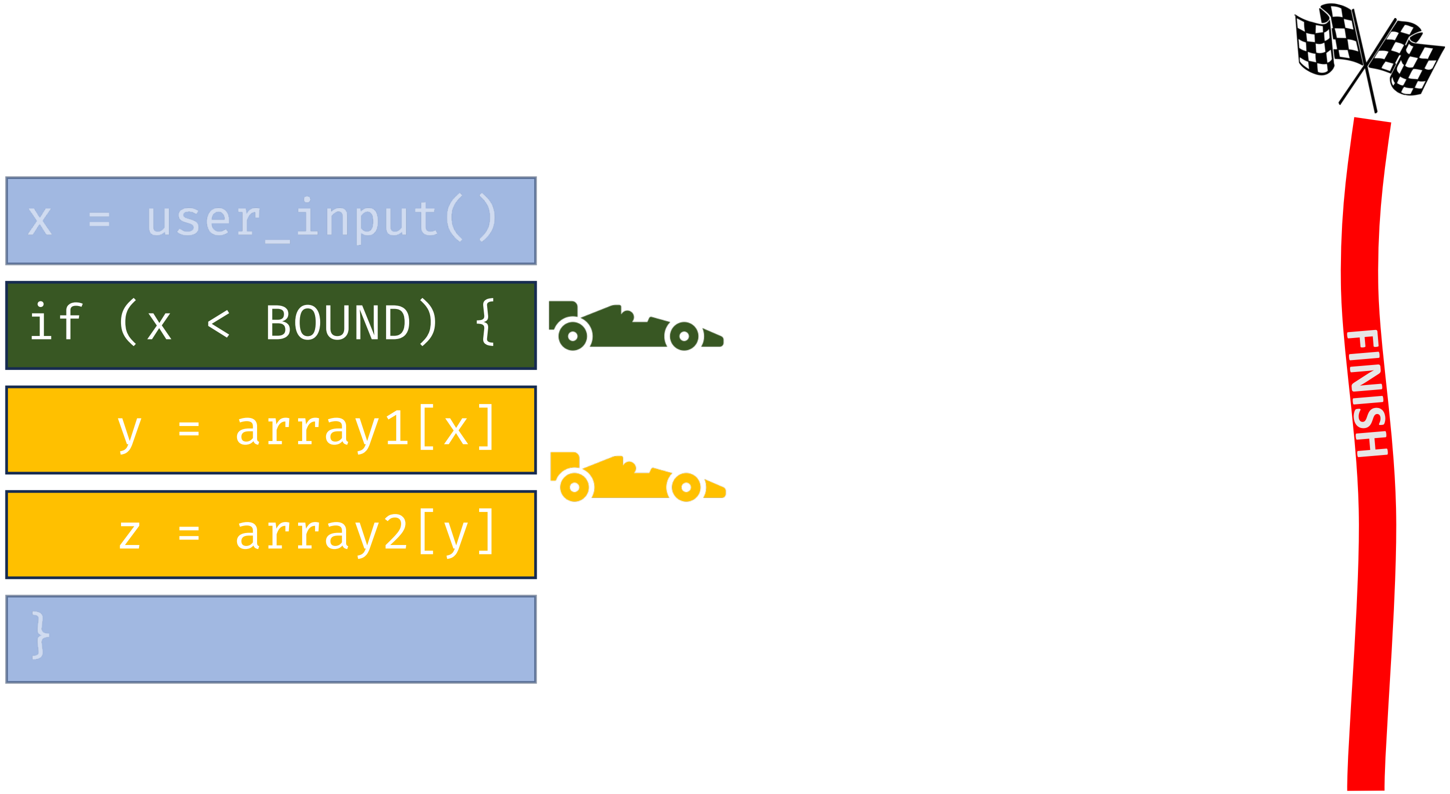
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Satisfying the **timing condition** is necessary for a gadget to be **exploitable**.

Satisfying the **timing condition** is necessary for a gadget to be **exploitable**.

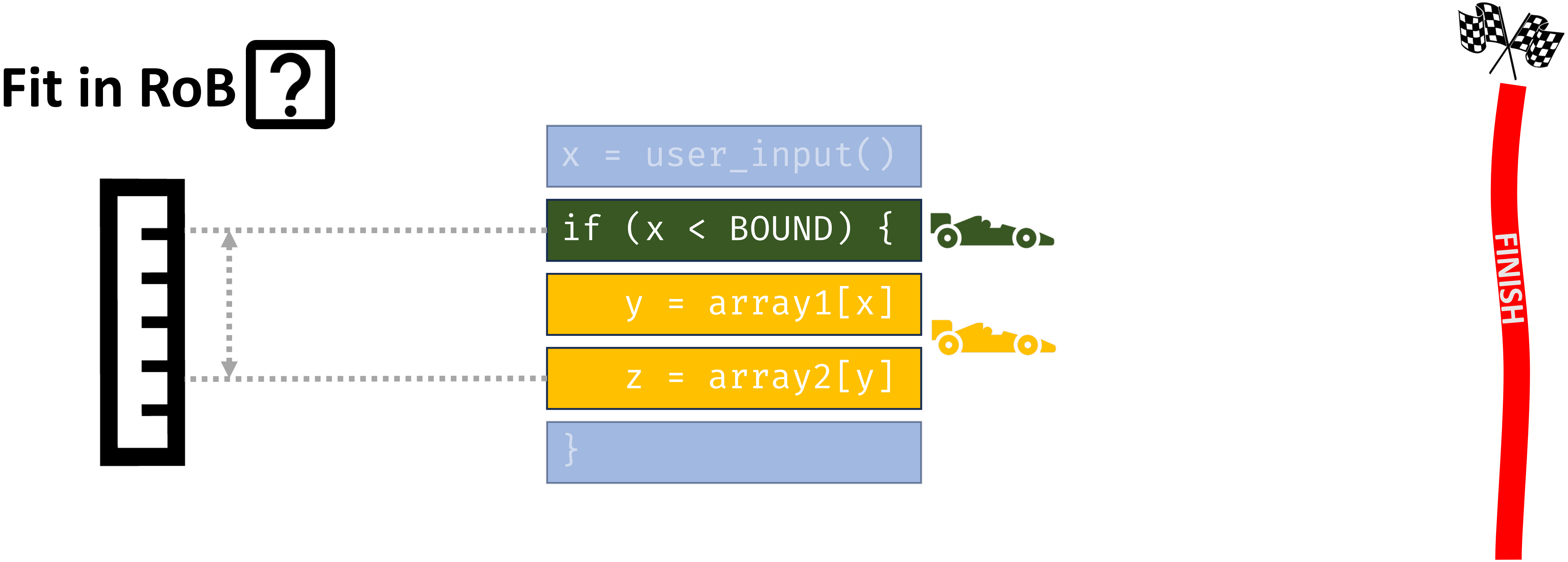
How do **existing works** model the timing condition?

Most scanners: Approximating with RoB size



```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Most scanners: Approximating with RoB size

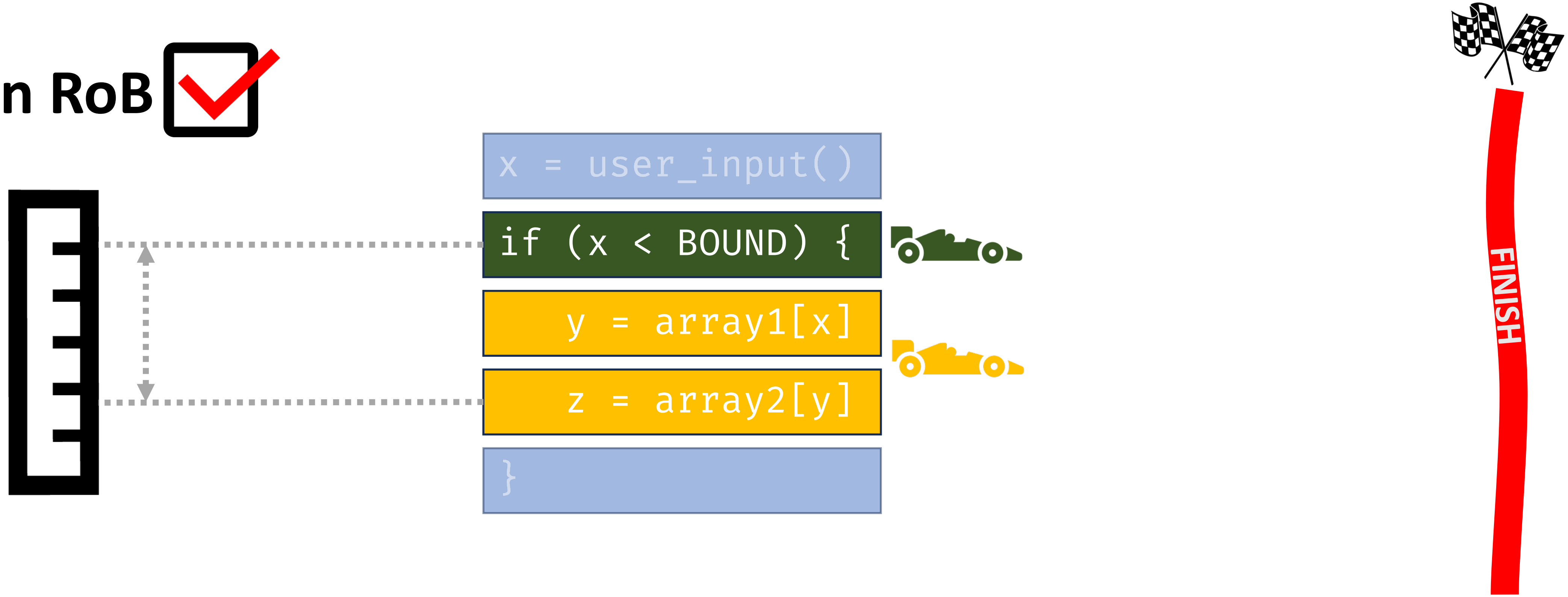


Fit in RoB ?

```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Most scanners: Approximating with RoB size

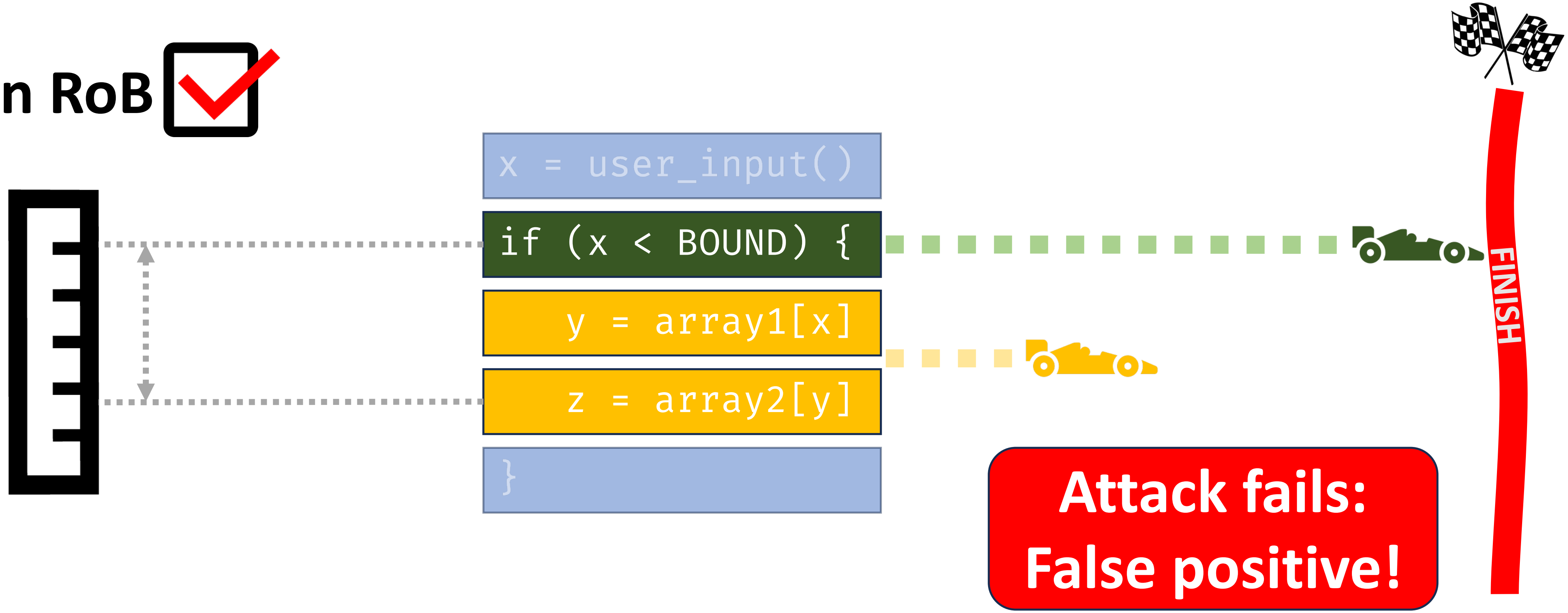
Fit in RoB ☑



```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Most scanners: Approximating with RoB size

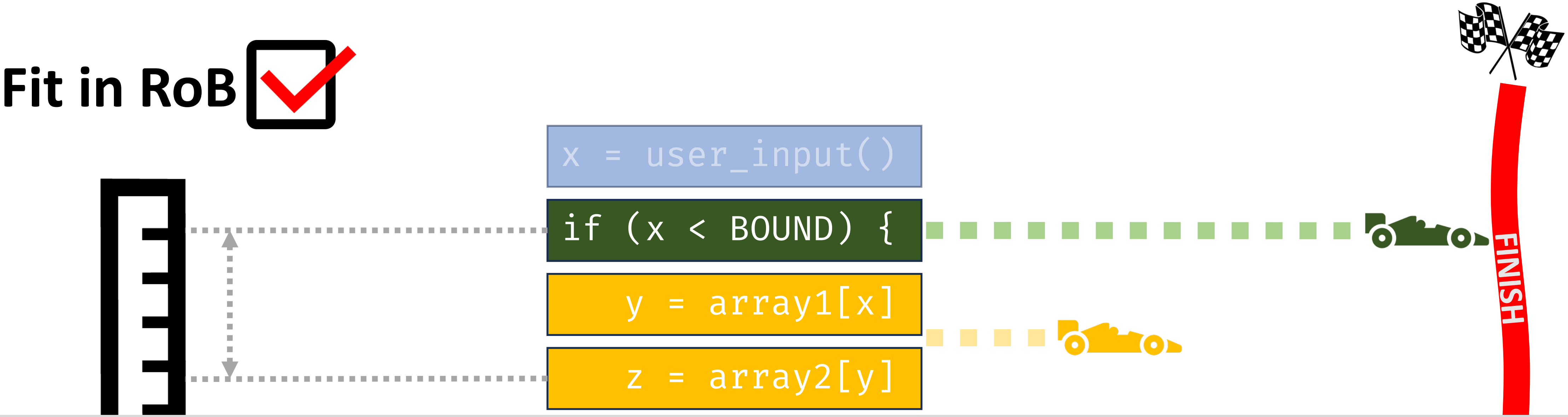
Fit in RoB ☑



```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Attack fails:
False positive!

Most scanners: Approximating with RoB size



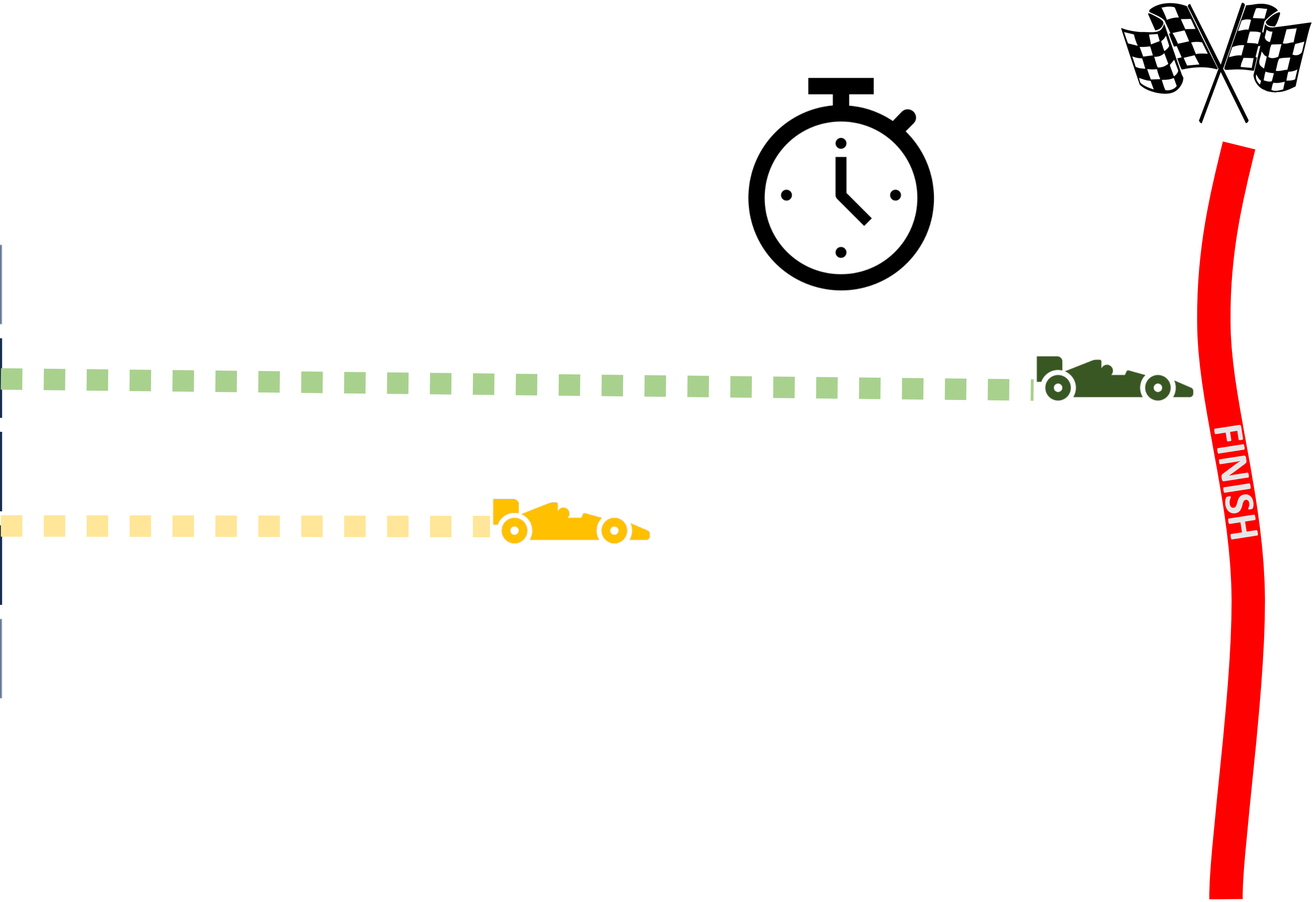
Fit in RoB ☑

```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
```

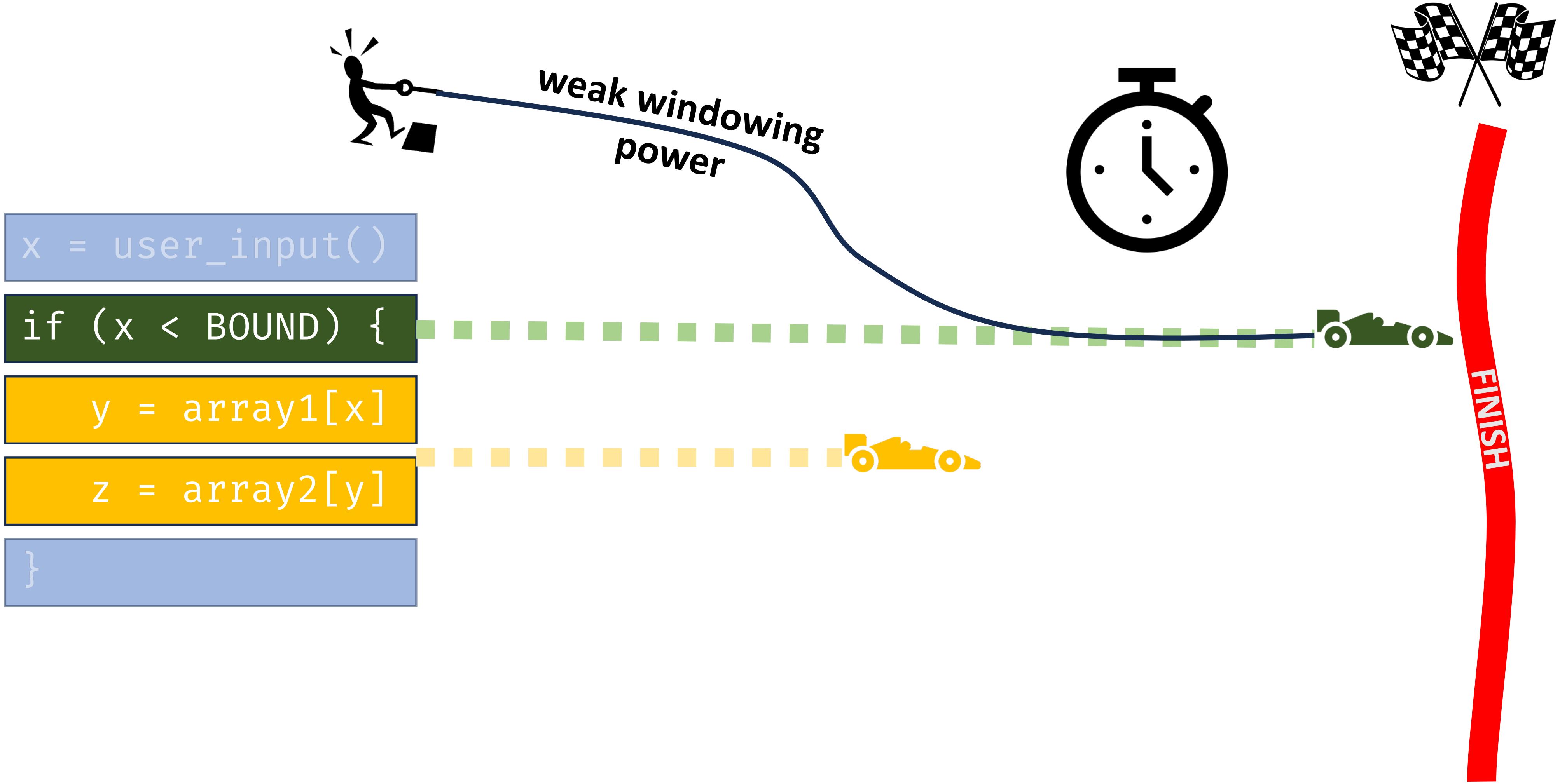
Measure the timing condition accurately!

Some others: Timing modelling

```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

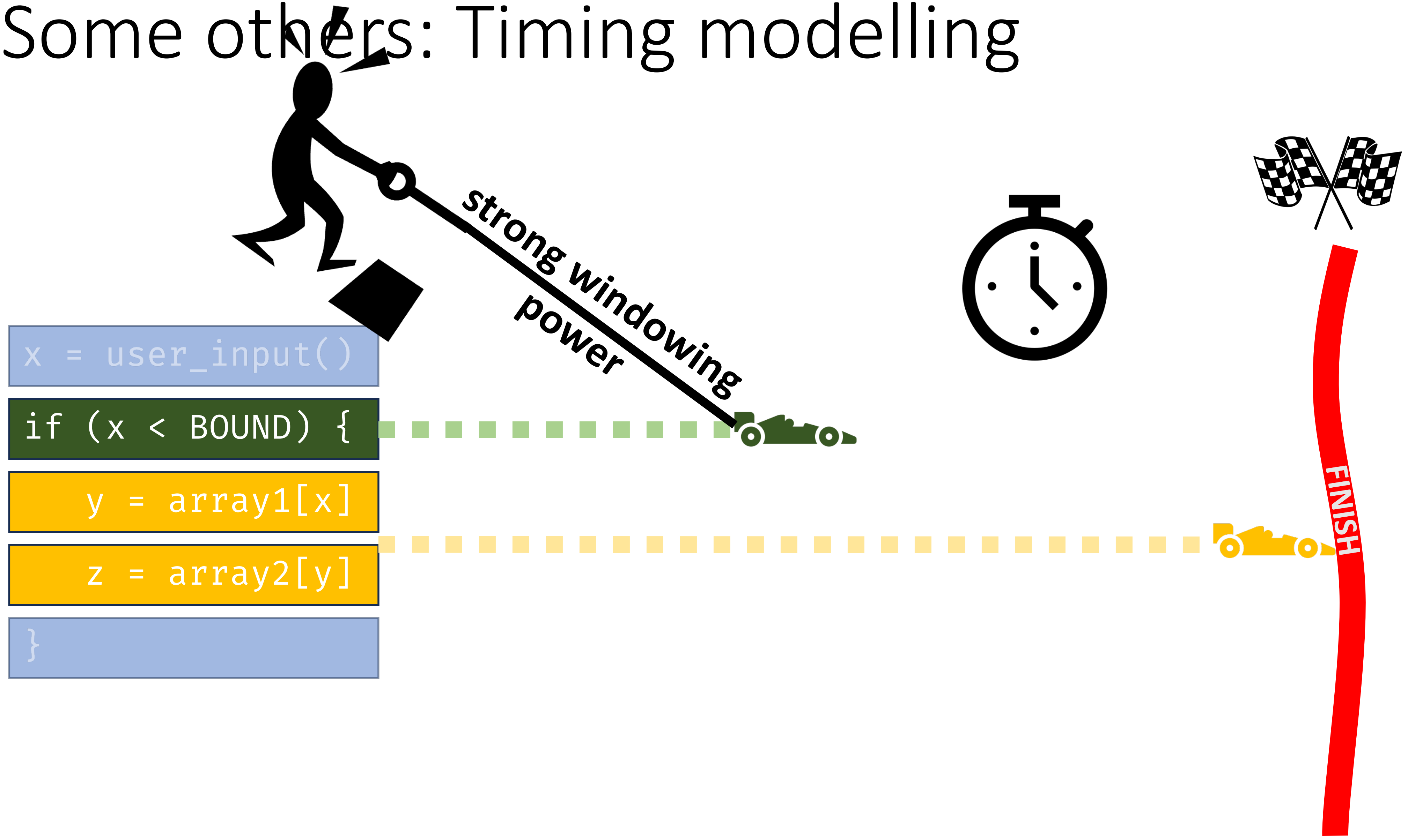


Some others: Timing modelling



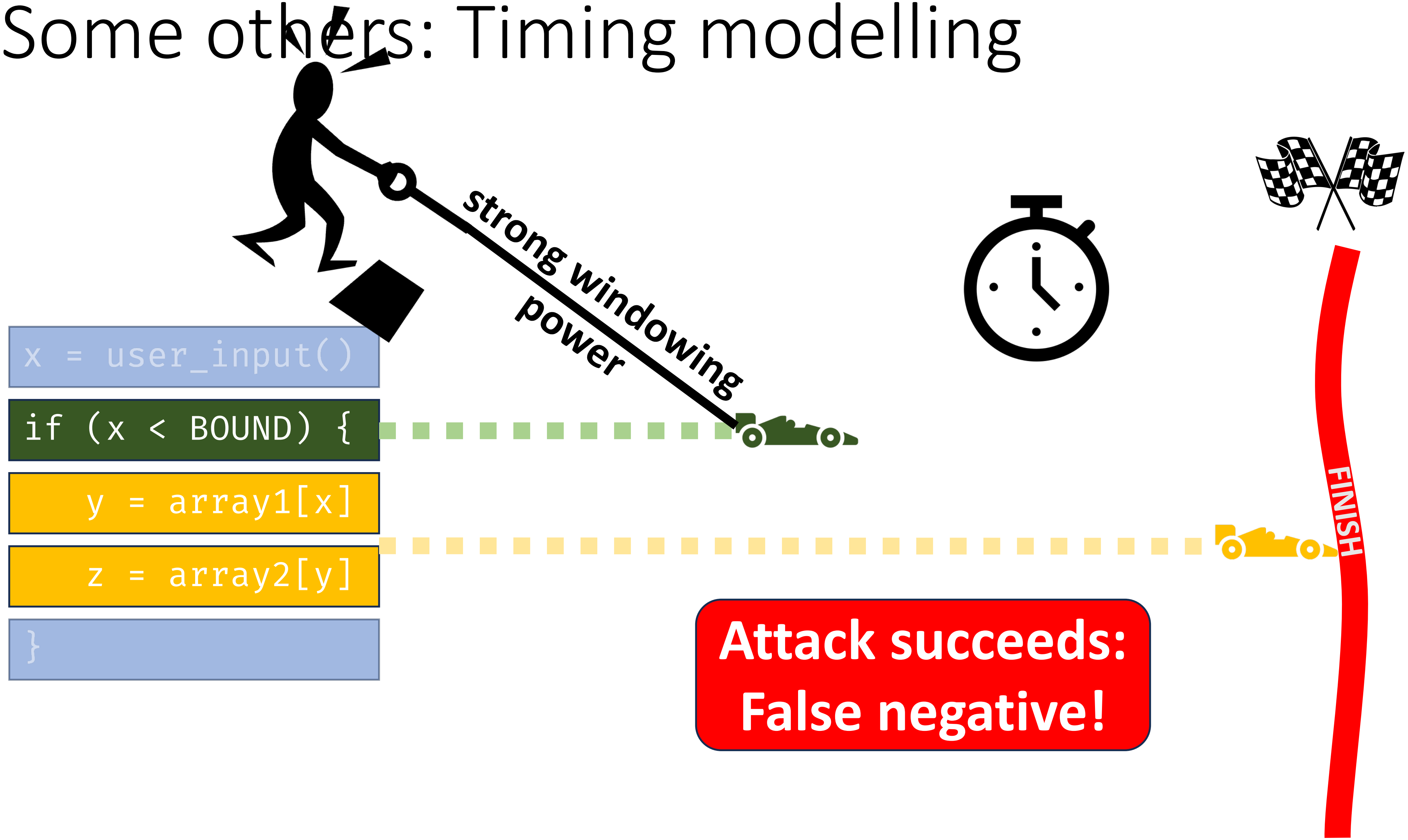
```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Some others: Timing modelling



```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

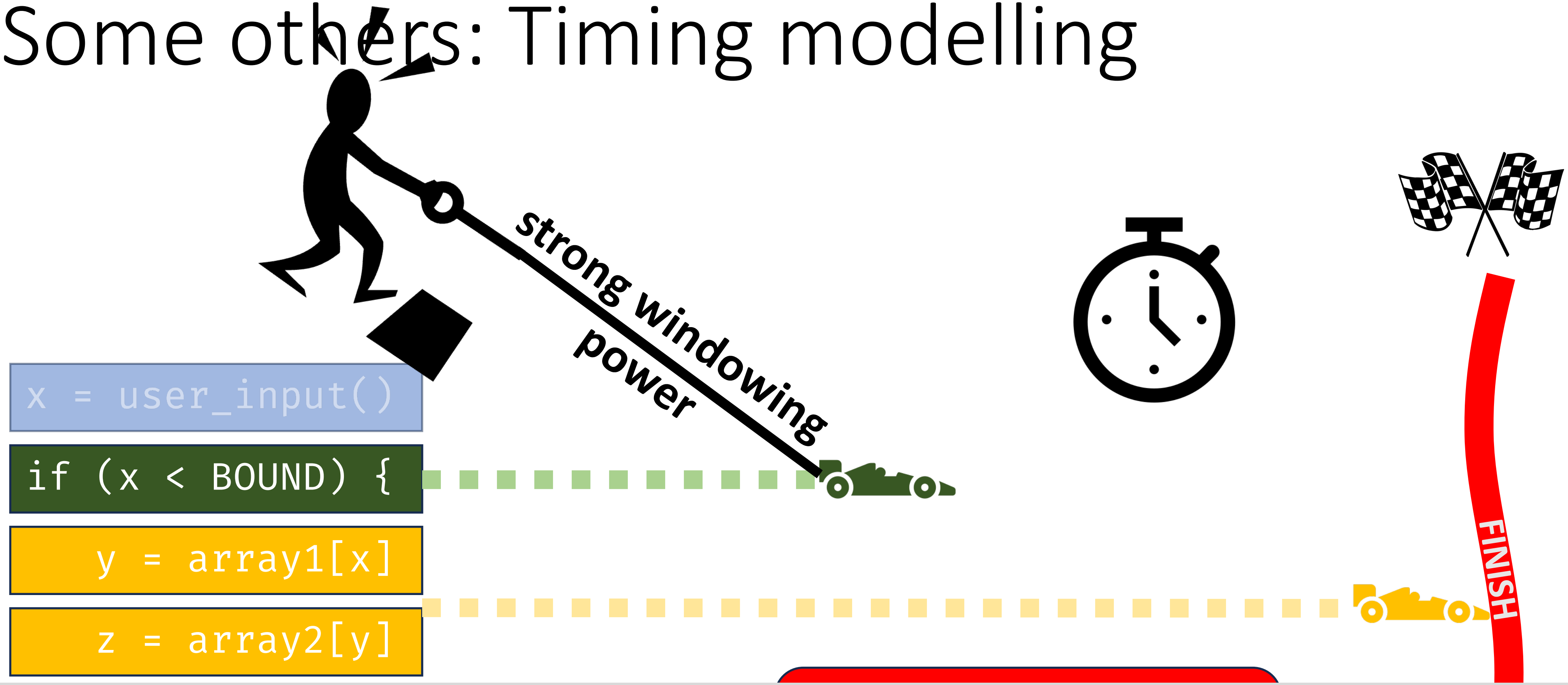
Some others: Timing modelling



```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
}
```

Attack succeeds:
False negative!

Some others: Timing modelling

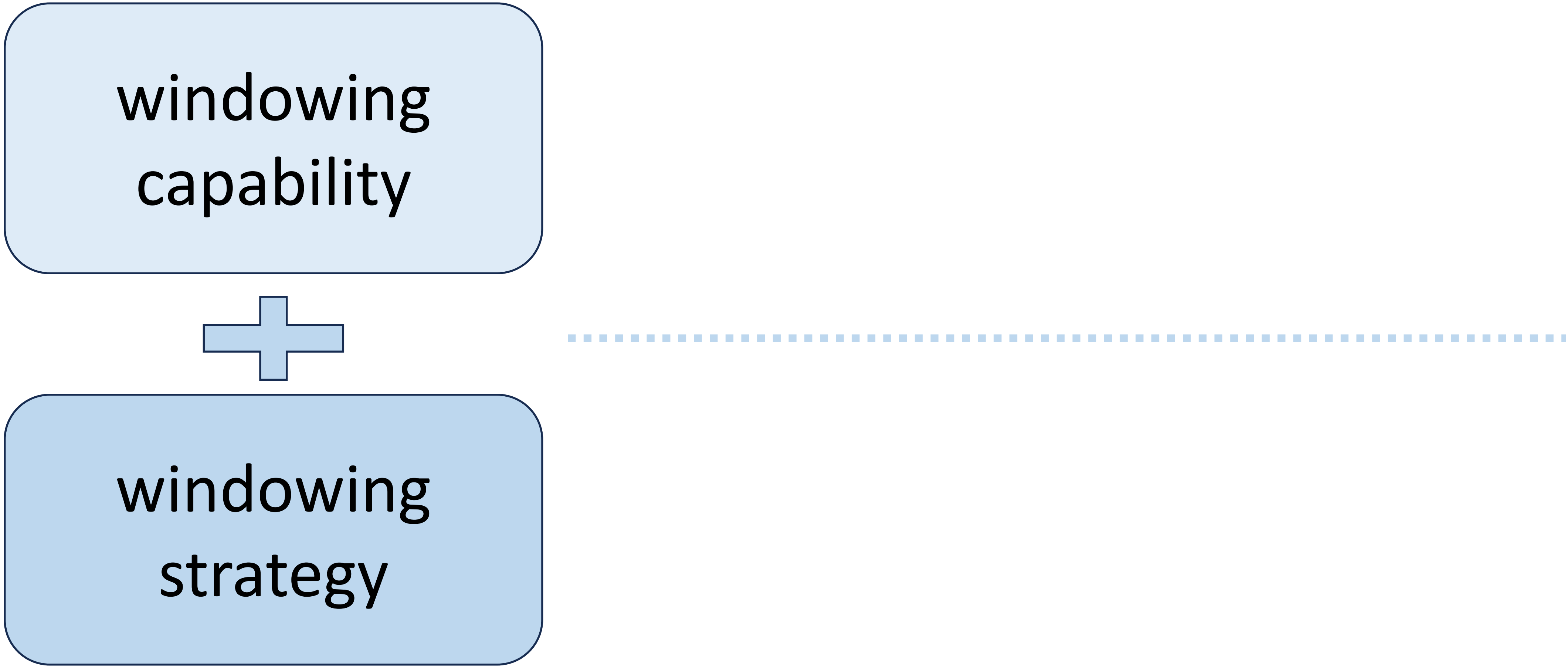


```
x = user_input()
if (x < BOUND) {
    y = array1[x]
    z = array2[y]
```

Measure the timing condition accurately,

under strong windowing power!

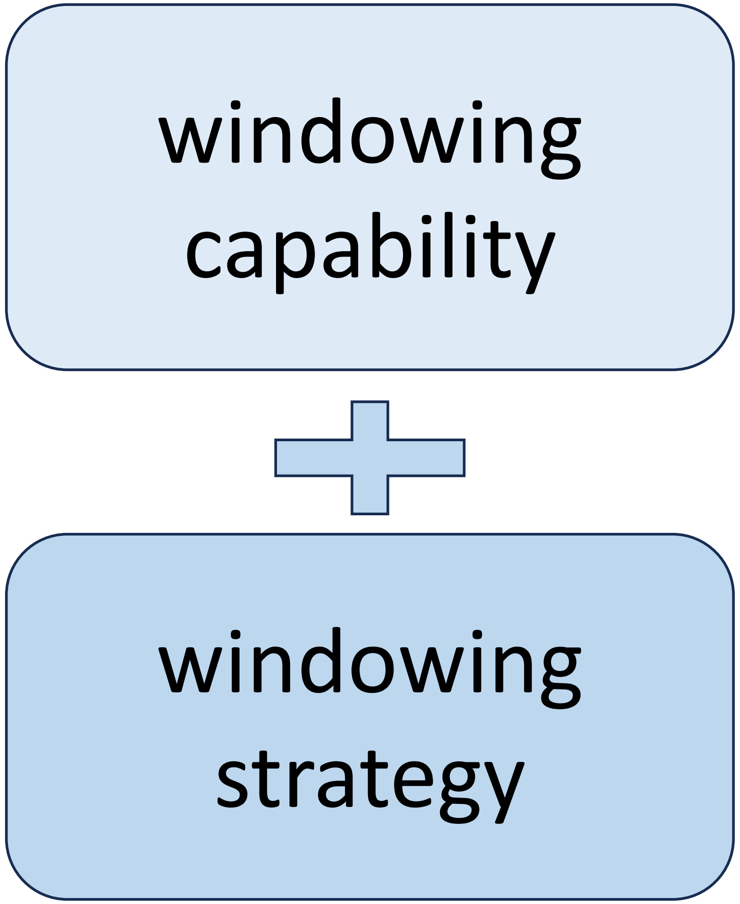
Our approach: Modelling windowing power



windowing capability

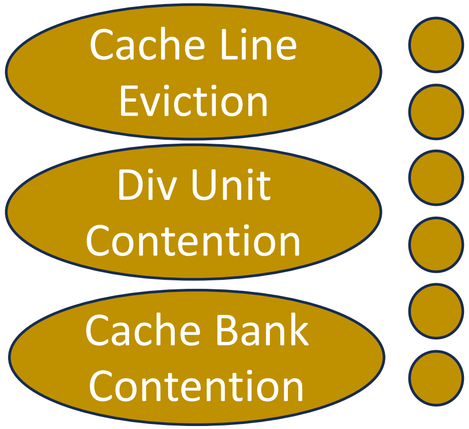
windowing strategy

Our approach: Modelling windowing power



windowing capability

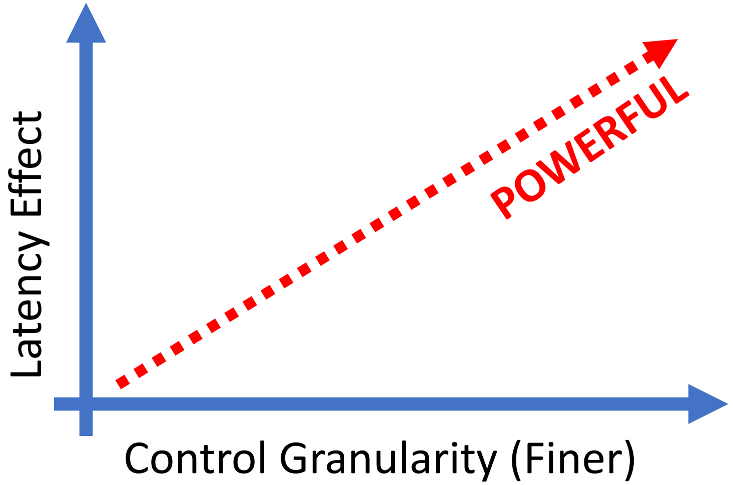
windowing strategy



Cache Line Eviction

Div Unit Contention

Cache Bank Contention

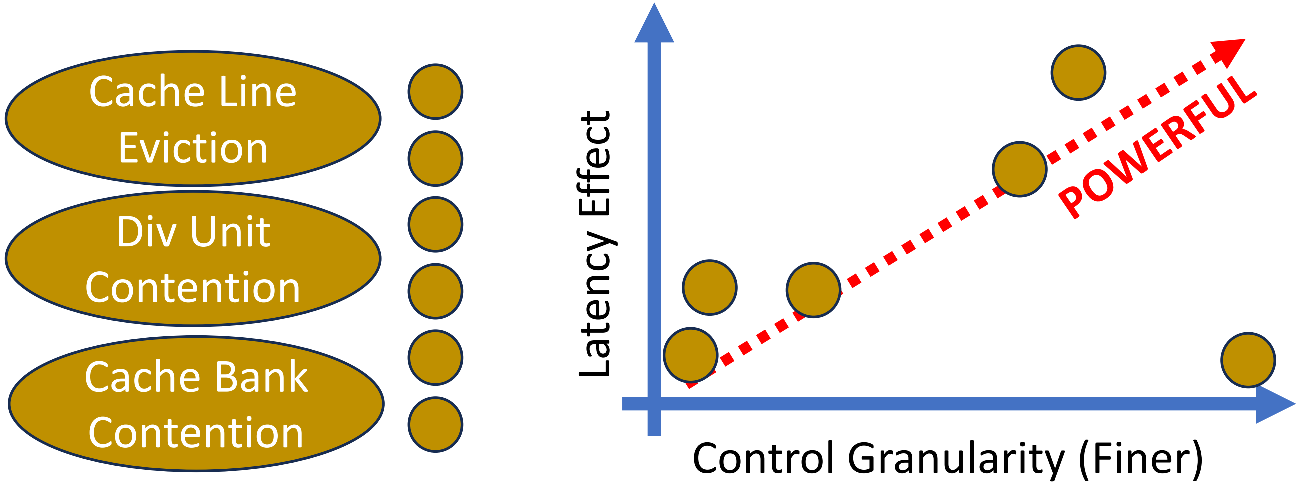


Our approach: Modelling windowing power

windowing capability

+

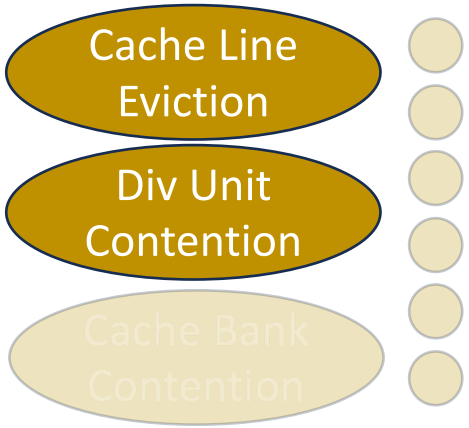
windowing strategy



Our approach: Modelling windowing power

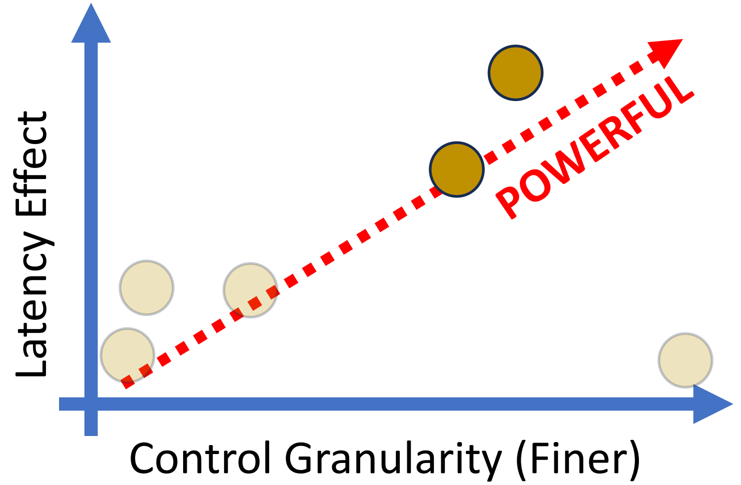
windowing capability

windowing strategy


Cache Line Eviction

Div Unit Contention

Cache Bank Contention



Our approach: Modelling windowing power

windowing capability

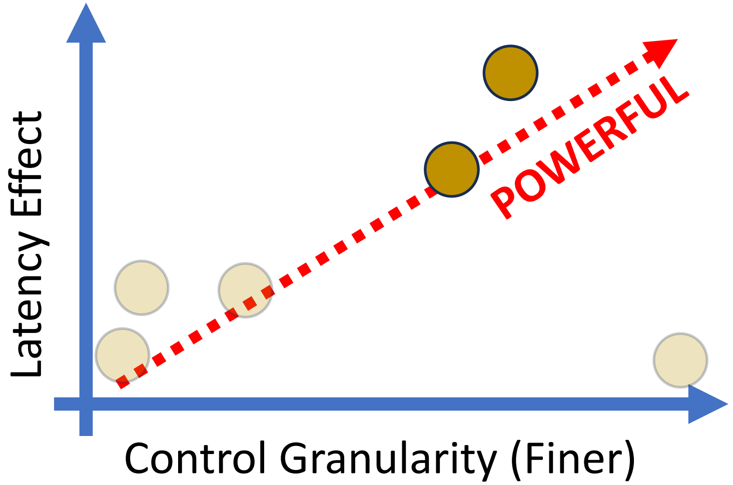
+

windowing strategy

Cache Line Eviction

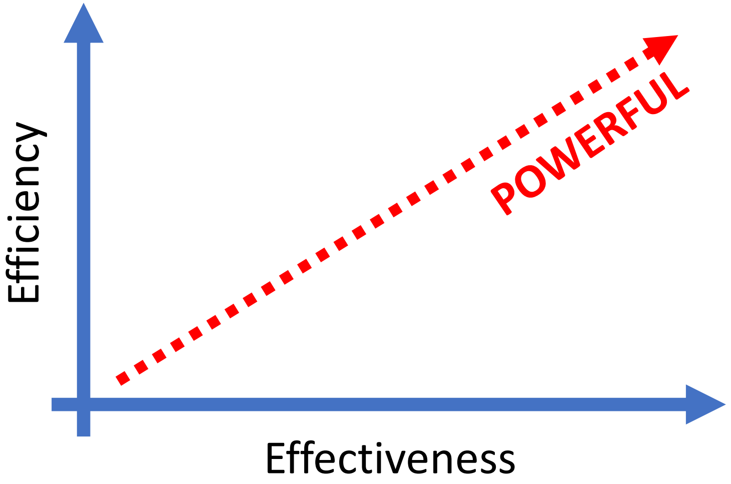
Div Unit Contention

Cache Bank Contention

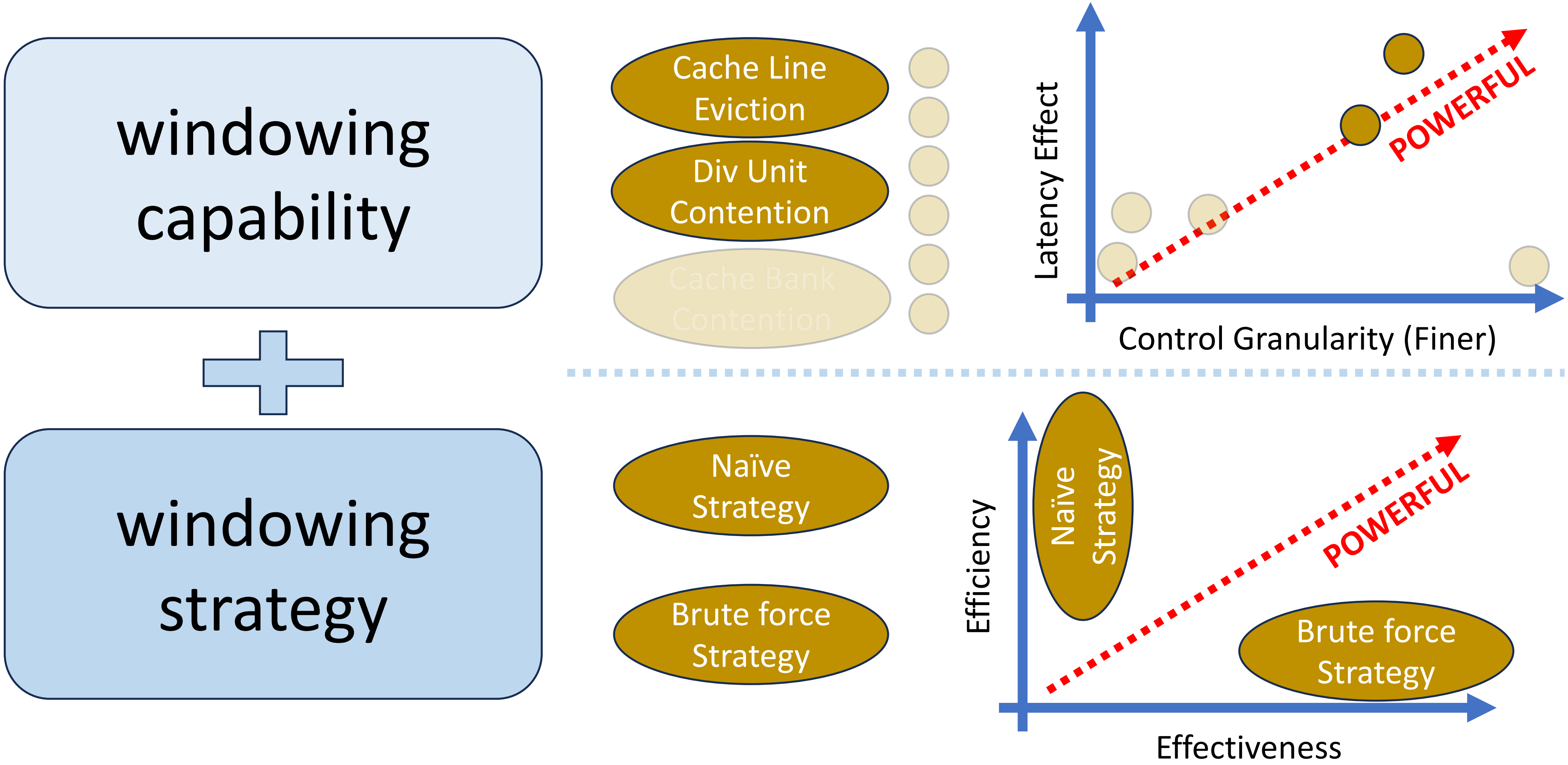


Naïve Strategy

Brute force Strategy



Our approach: Modelling windowing power

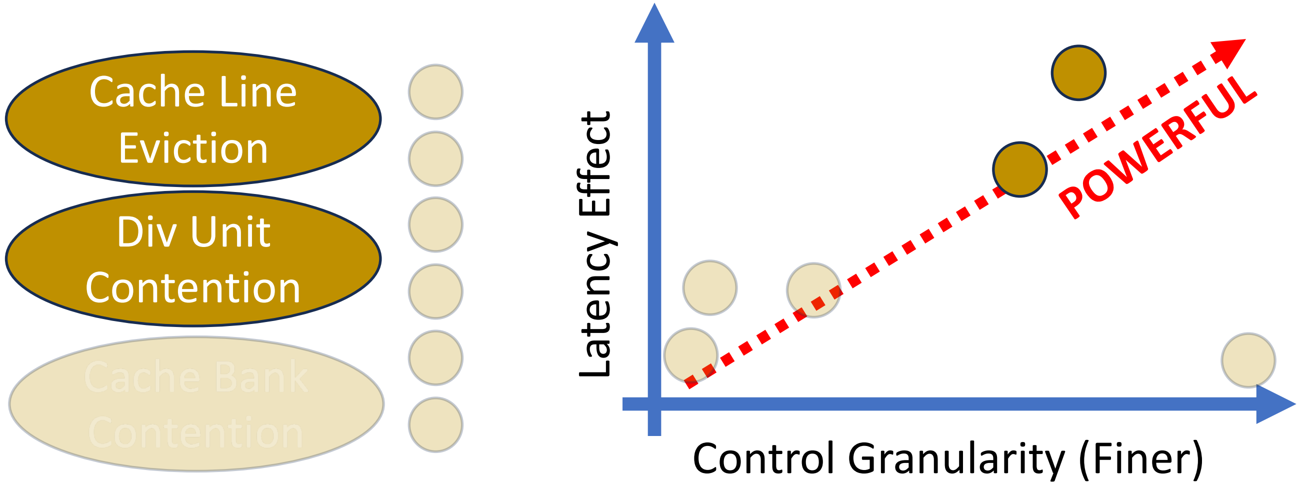


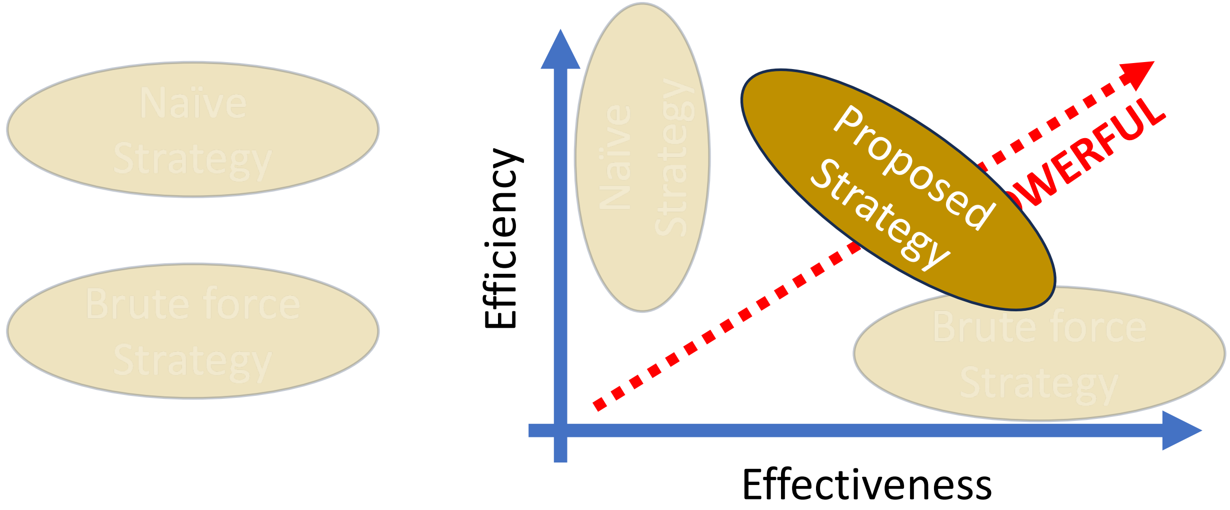
Our approach: Modelling windowing power

windowing capability

+

windowing strategy





Well, how do we evaluate gadgets?

Well, how do we evaluate gadgets?

Step A: **Modelling** timing condition

Well, how do we evaluate gadgets?

Step A: **Modelling** timing condition

Step B: **Simulating** windowing power

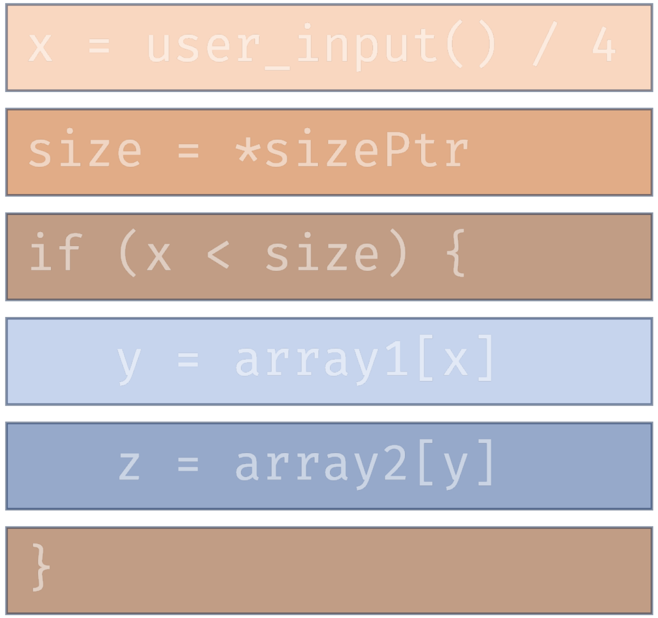
Well, how do we evaluate gadgets?

Step A: **Modelling** timing condition

Step B: **Simulating** windowing power

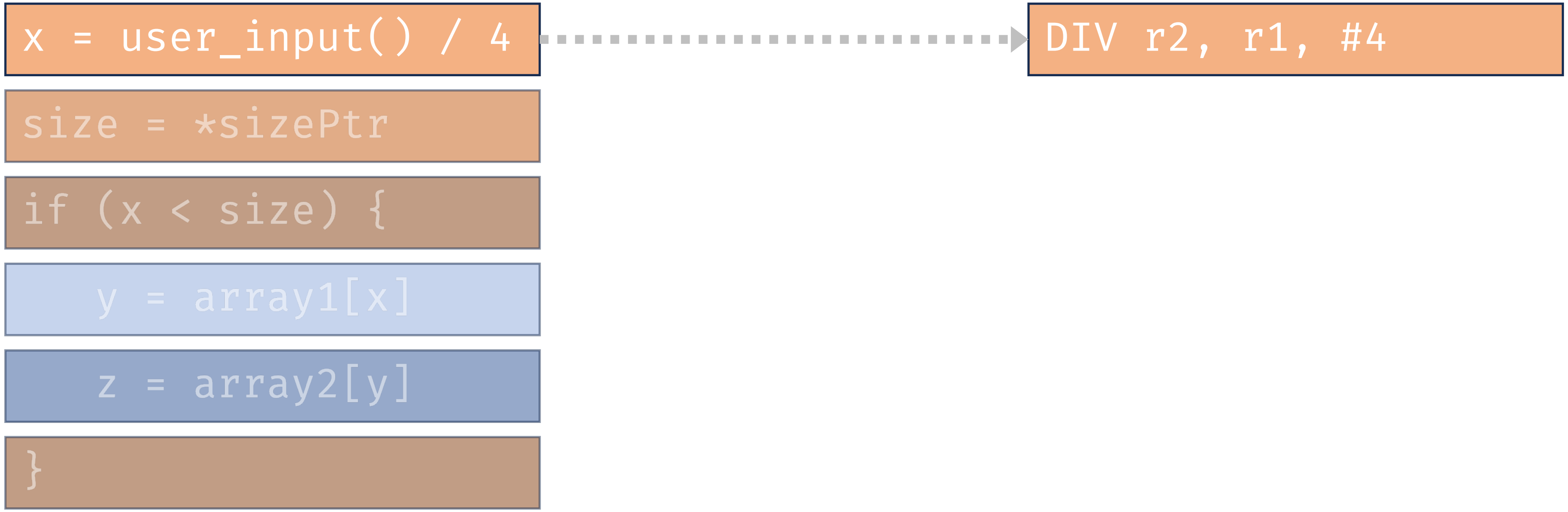
Step C: **Quantifying** exploitability

Our approach: An example



```
x = user_input() / 4
size = *sizePtr
if (x < size) {
   y = array1[x]
   z = array2[y]
}
```

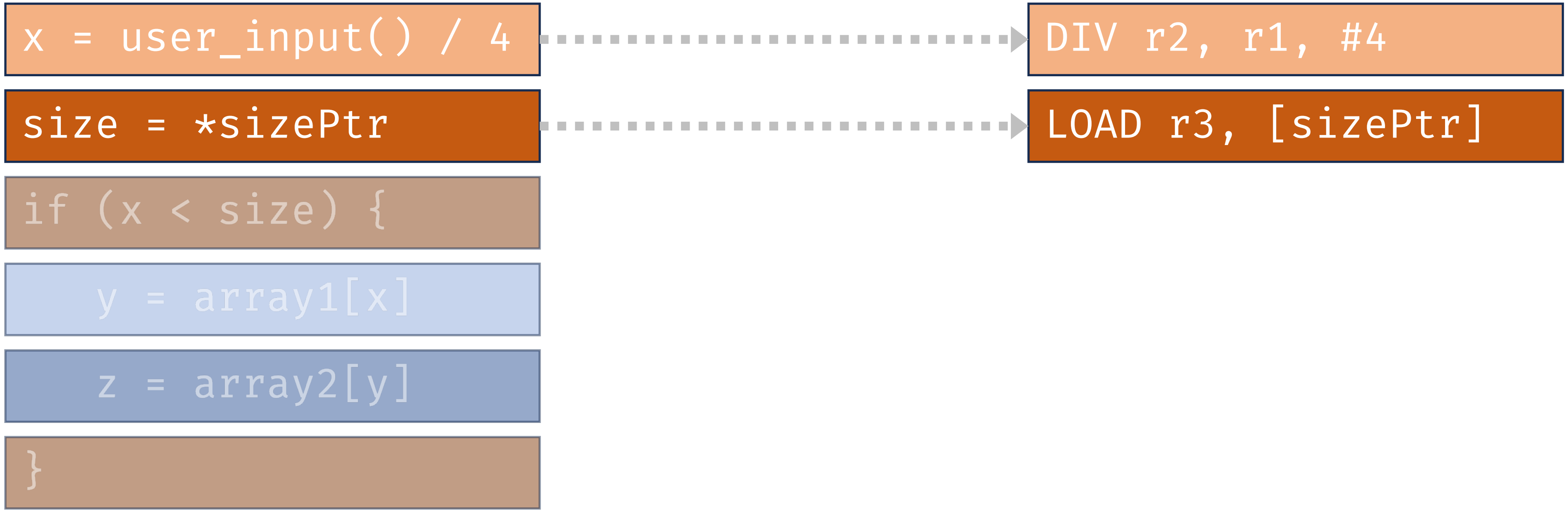
Our approach: An example



```
x = user_input() / 4
size = *sizePtr
if (x < size) {
    y = array1[x]
    z = array2[y]
}
```

```
DIV r2, r1, #4
```

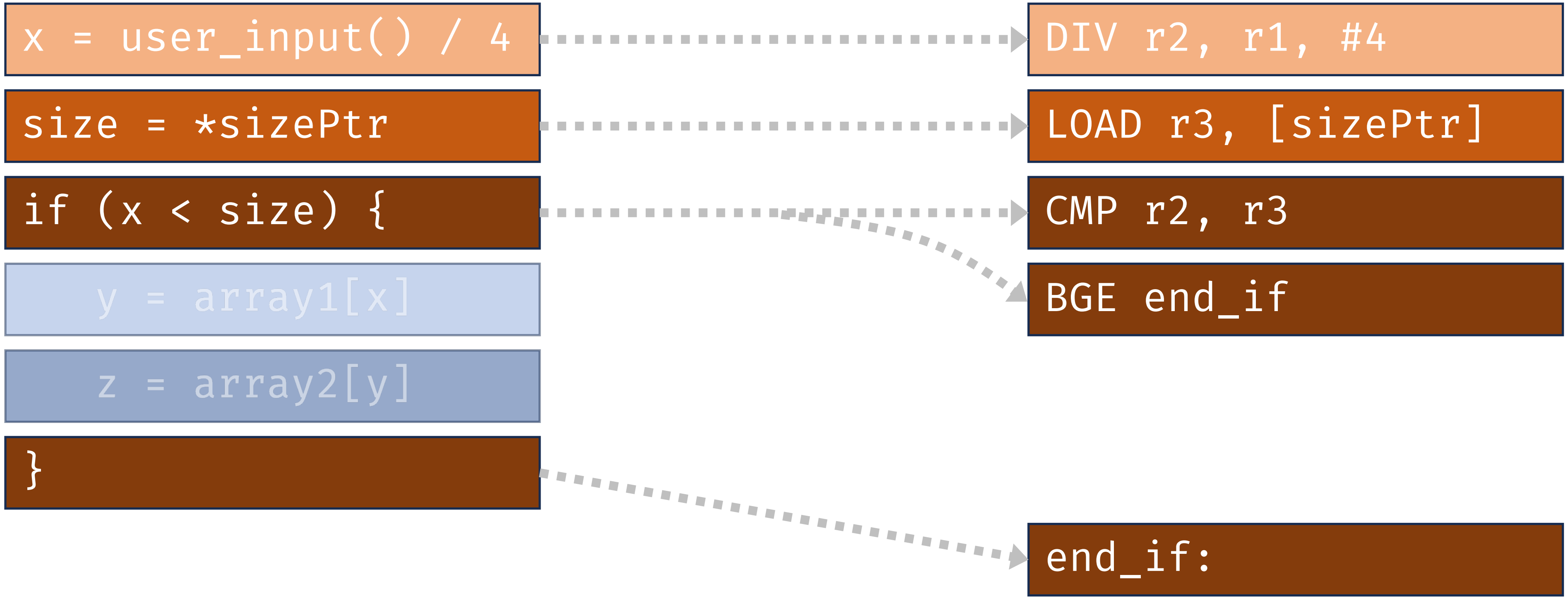
Our approach: An example



```
x = user_input() / 4
size = *sizePtr
if (x < size) {
    y = array1[x]
    z = array2[y]
}
```

```
DIV r2, r1, #4
LOAD r3, [sizePtr]
```

Our approach: An example

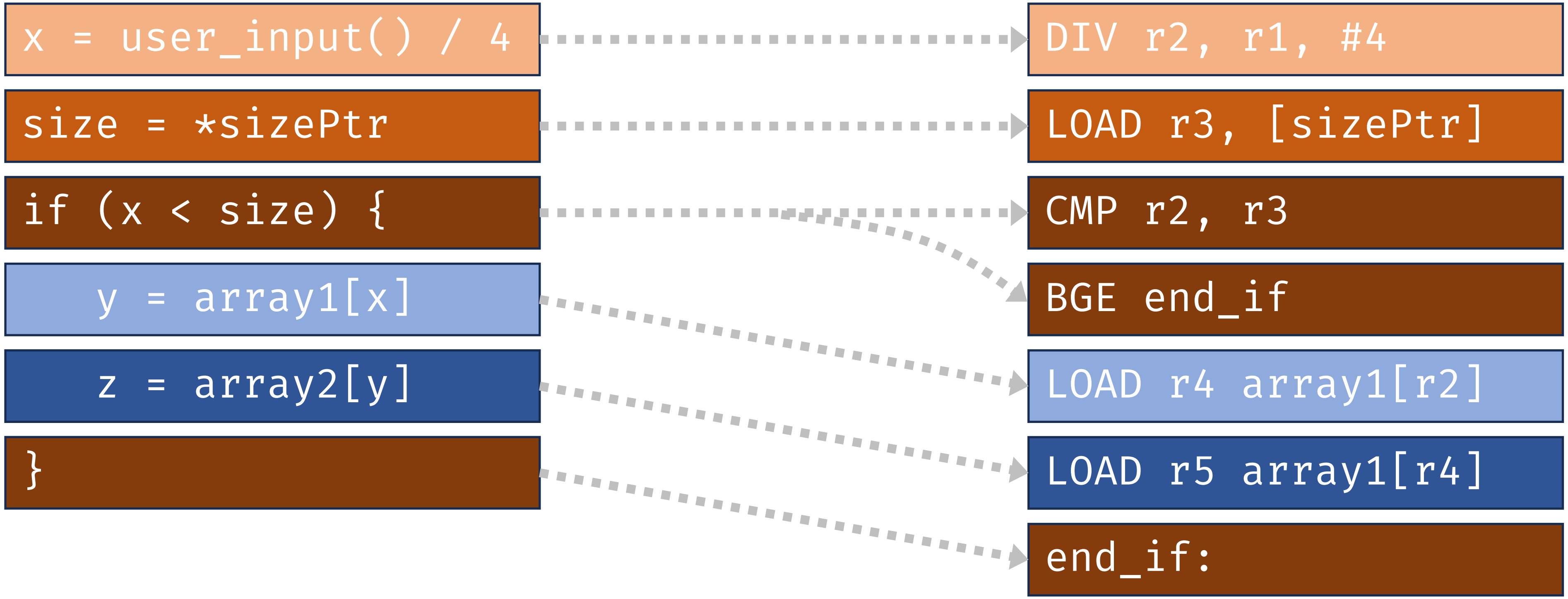


```
x = user_input() / 4
size = *sizePtr
if (x < size) {
    y = array1[x]
    z = array2[y]
}
```

```
DIV r2, r1, #4
LOAD r3, [sizePtr]
CMP r2, r3
BGE end_if

end_if:
```

Our approach: An example

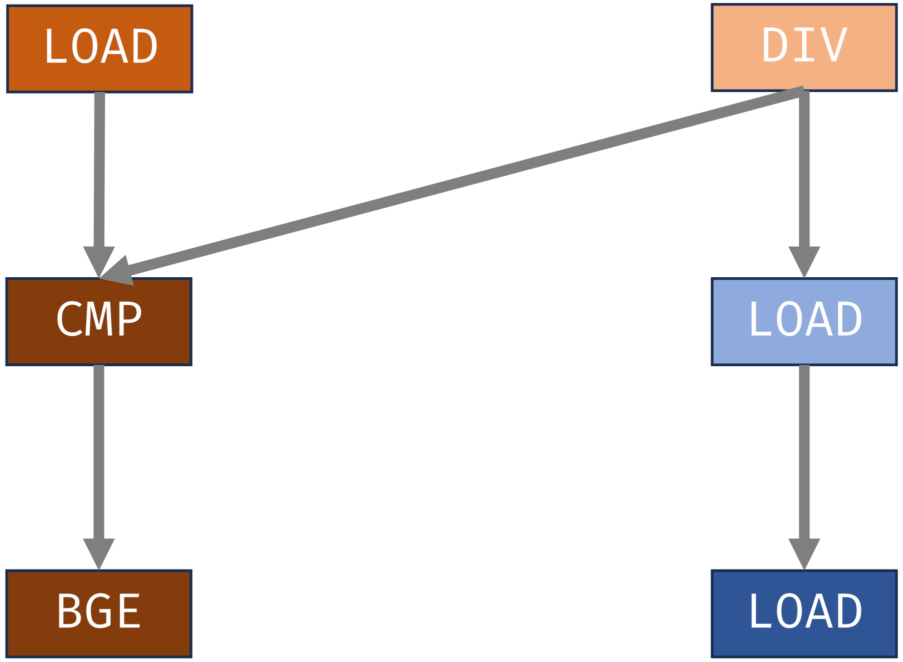


```
x = user_input() / 4
size = *sizePtr
if (x < size) {
    y = array1[x]
    z = array2[y]
}
```

```
DIV r2, r1, #4
LOAD r3, [sizePtr]
CMP r2, r3
BGE end_if
LOAD r4 array1[r2]
LOAD r5 array1[r4]
end_if:
```

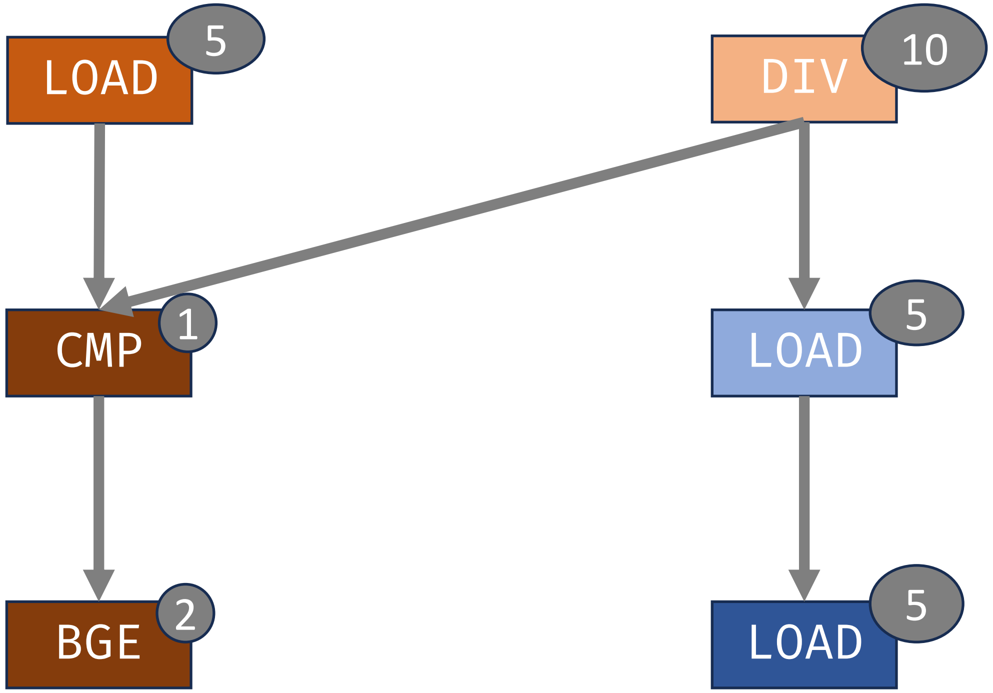
Step A: Modelling timing condition

```
DIV r2, r1, #4
LOAD r3, [sizePtr]
CMP r2, r3
BGE end_if
LOAD r4 array1[r2]
LOAD r5 array1[r4]
end_if:
```



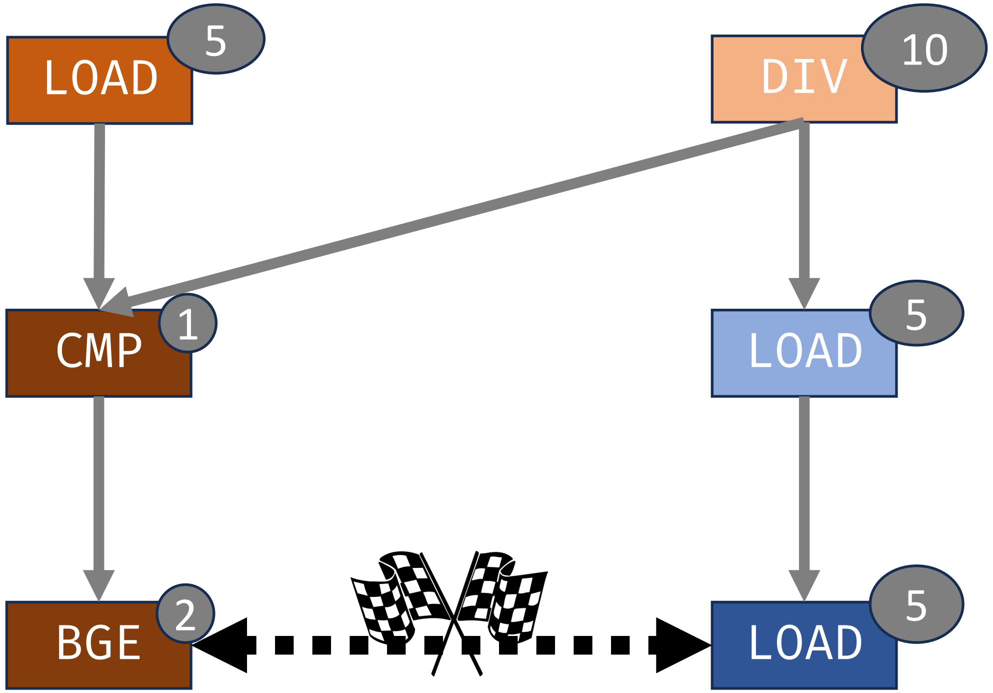
Step A: Modelling timing condition

```
DIV r2, r1, #4
LOAD r3, [sizePtr]
CMP r2, r3
BGE end_if
LOAD r4 array1[r2]
LOAD r5 array1[r4]
end_if:
```



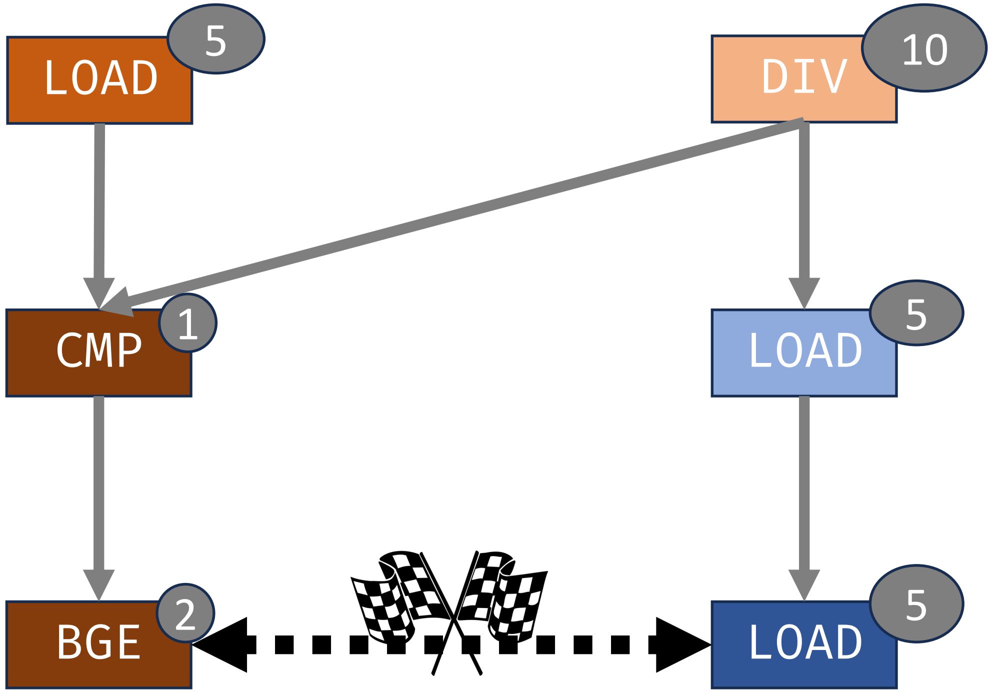
Step A: Modelling timing condition

```
DIV r2, r1, #4
LOAD r3, [sizePtr]
CMP r2, r3
BGE end_if
LOAD r4 array1[r2]
LOAD r5 array1[r4]
end_if:
```



Step A: Modelling timing condition

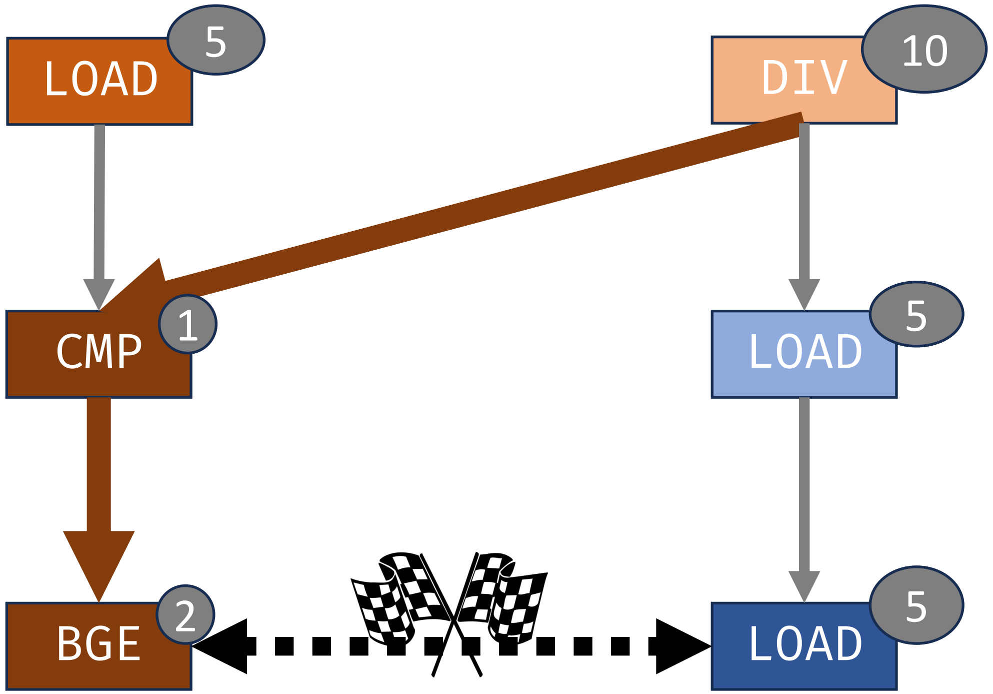
```
DIV r2, r1, #4
LOAD r3, [sizePtr]
CMP r2, r3
BGE end_if
LOAD r4 array1[r2]
LOAD r5 array1[r4]
end_if:
```



MaxPathWeight(BGE) **−** ***MaxPathWeight***(LOAD)

Step A: Modelling timing condition

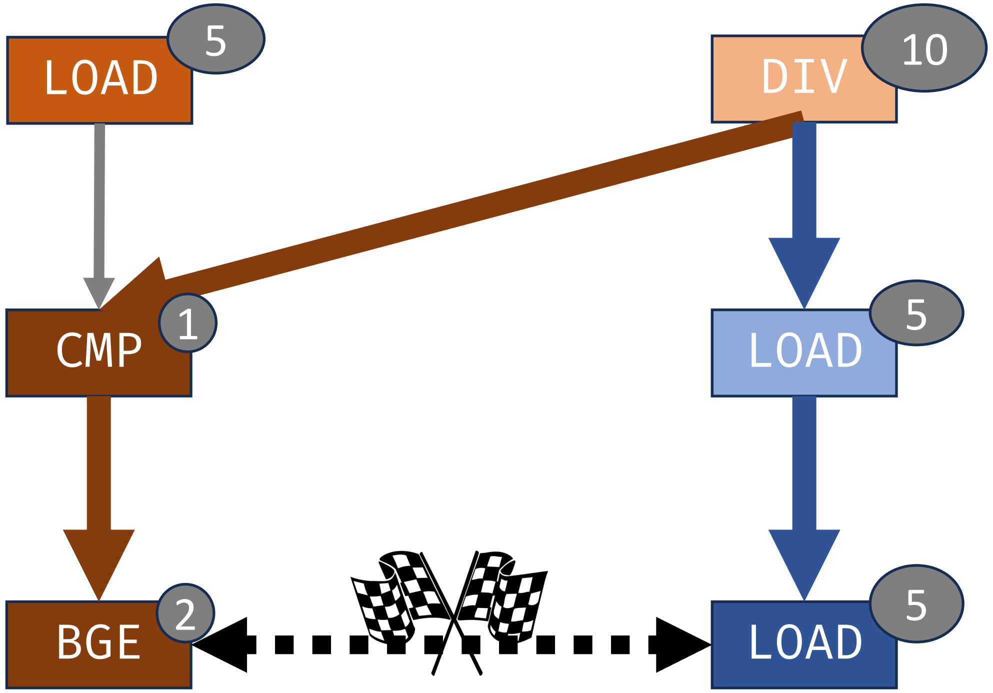
```
DIV r2, r1, #4
LOAD r3, [sizePtr]
CMP r2, r3
BGE end_if
LOAD r4 array1[r2]
LOAD r5 array1[r4]
end_if:
```



MaxPathWeight(BGE) − ***MaxPathWeight***(LOAD)

Step A: Modelling timing condition

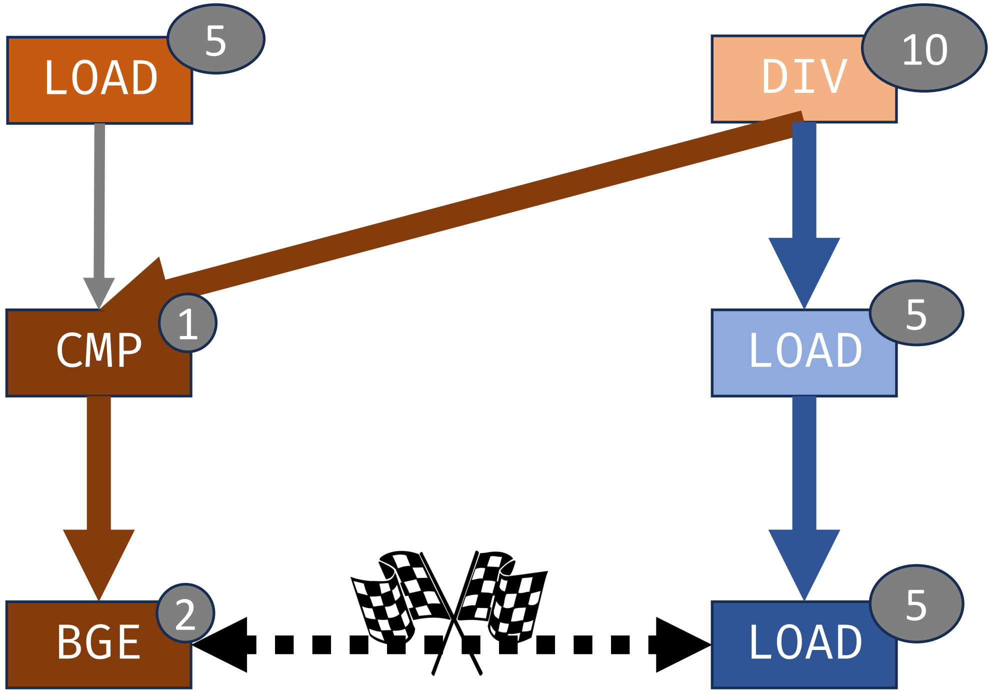
```
DIV r2, r1, #4
LOAD r3, [sizePtr]
CMP r2, r3
BGE end_if
LOAD r4 array1[r2]
LOAD r5 array1[r4]
end_if:
```



MaxPathWeight(BGE) **−** ***MaxPathWeight***(LOAD)

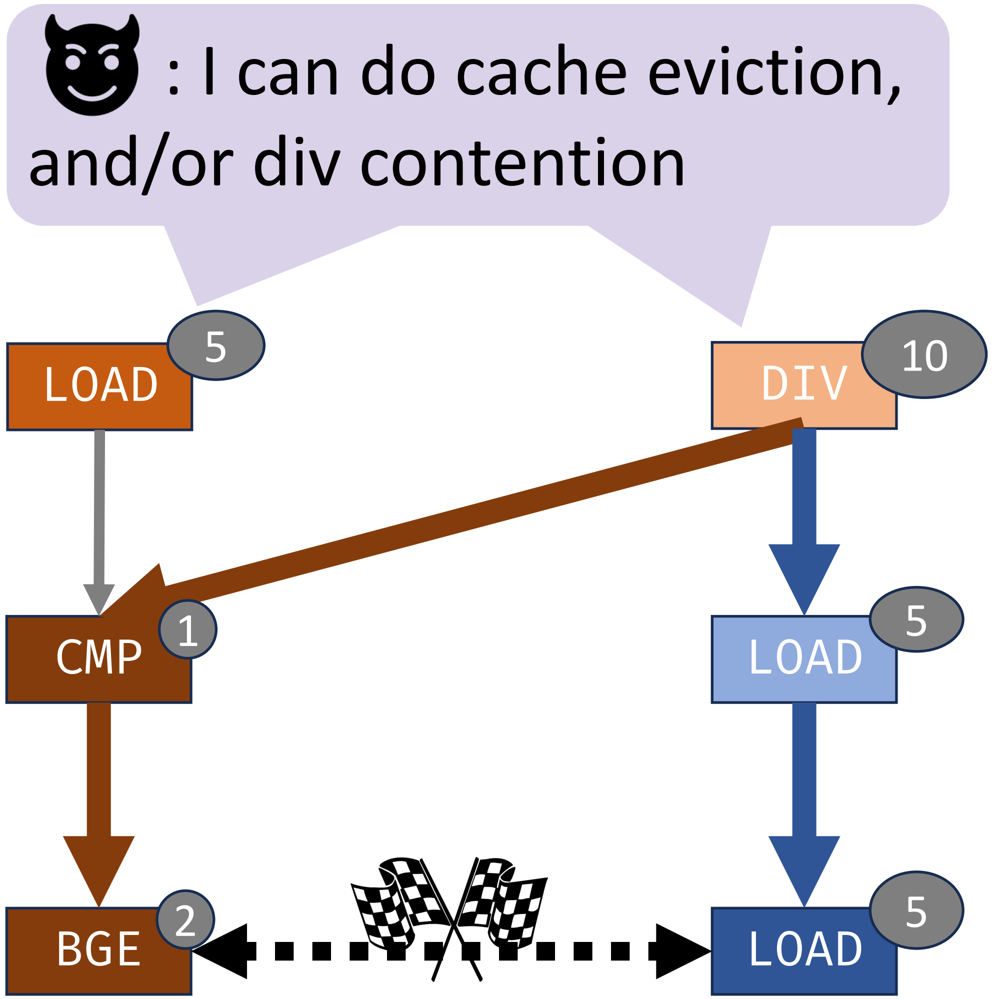
Step A: Modelling timing condition

```
DIV r2, r1, #4
LOAD r3, [sizePtr]
CMP r2, r3
BGE end_if
LOAD r4 array1[r2]
LOAD r5 array1[r4]
end_if:
```



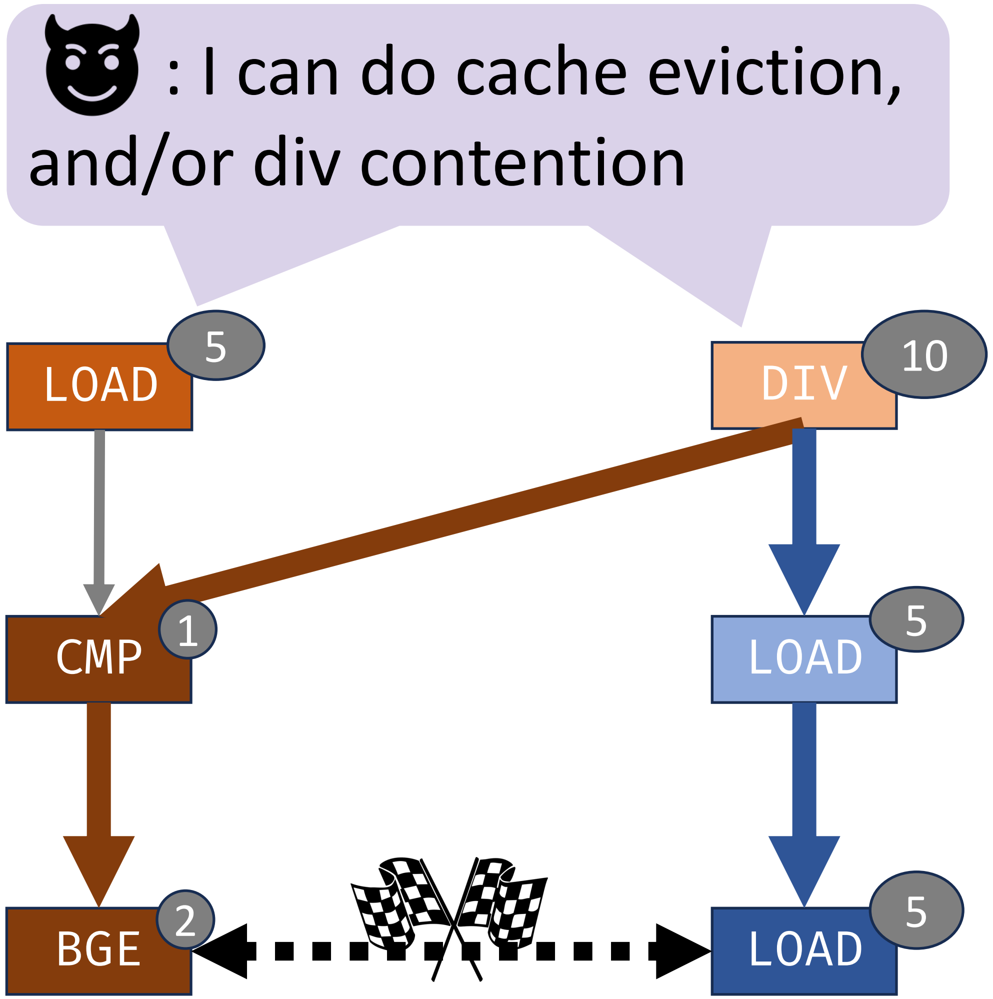
Timing Condition Index **=** ***MaxPathWeight***(BGE) **–** ***MaxPathWeight***(LOAD)

Step B: Simulating windowing power



😈 : I can do cache eviction, and/or div contention

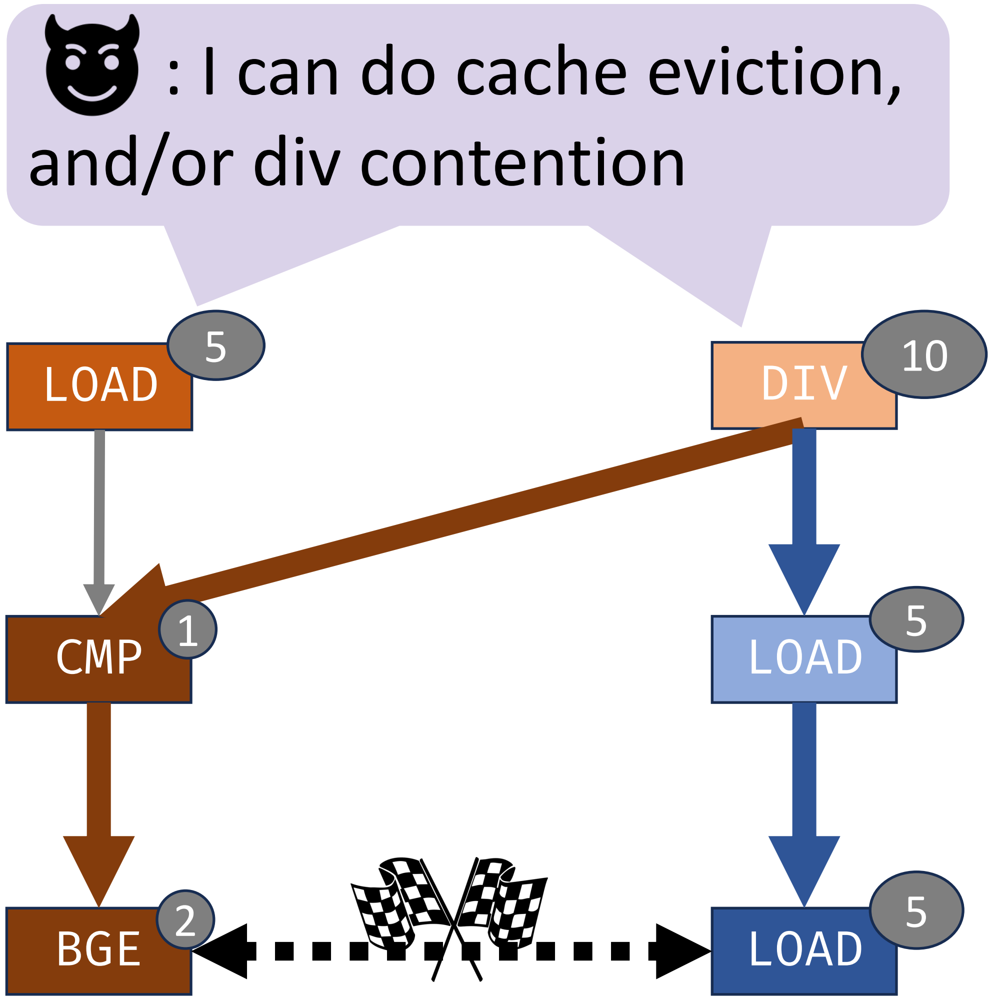
Step B: Simulating windowing power



: I can do cache eviction, and/or div contention

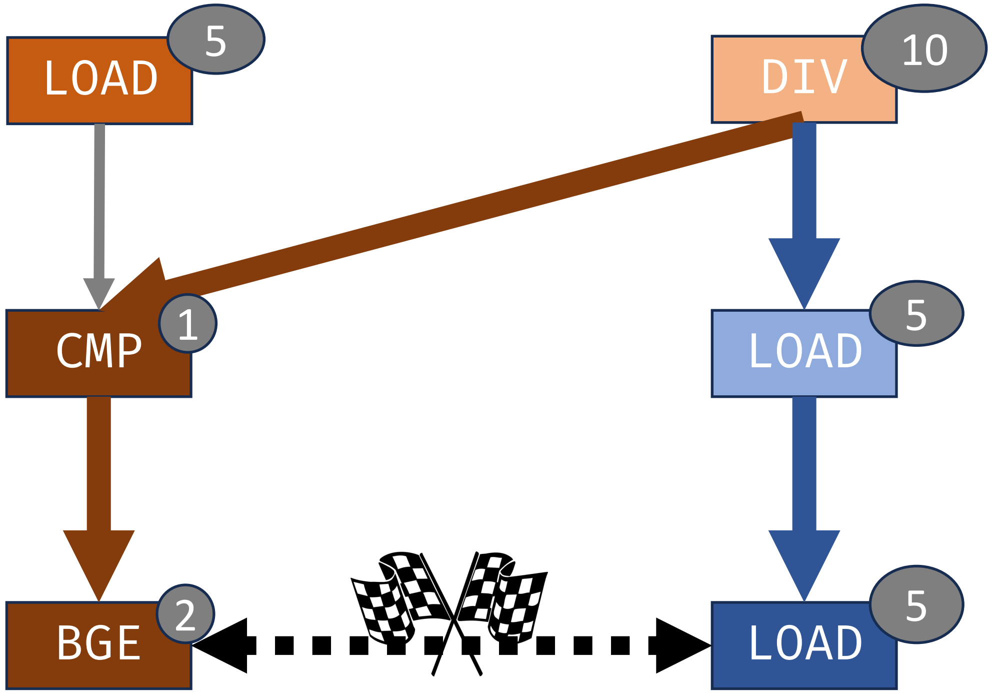
Attack Pattern
Do nothing
Cache eviction
Div contention
Cache eviction + Div contention

Step B: Simulating windowing power

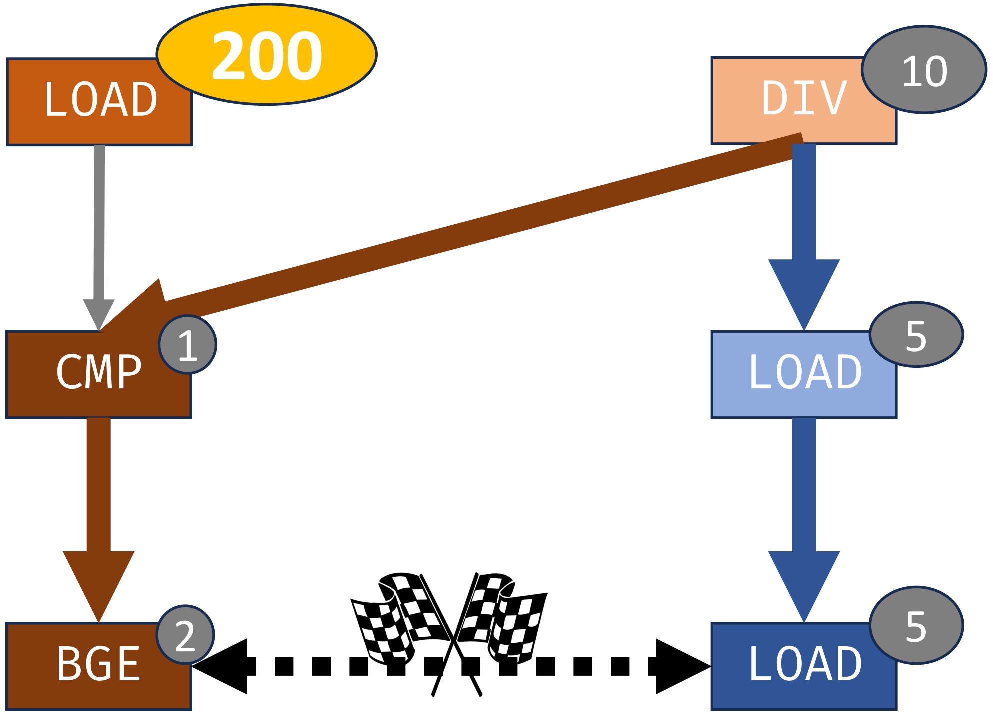


Attack Pattern	**Increase** in Timing Condition Index
Do nothing	?
Cache eviction	?
Div contention	?
Cache eviction + Div contention	?

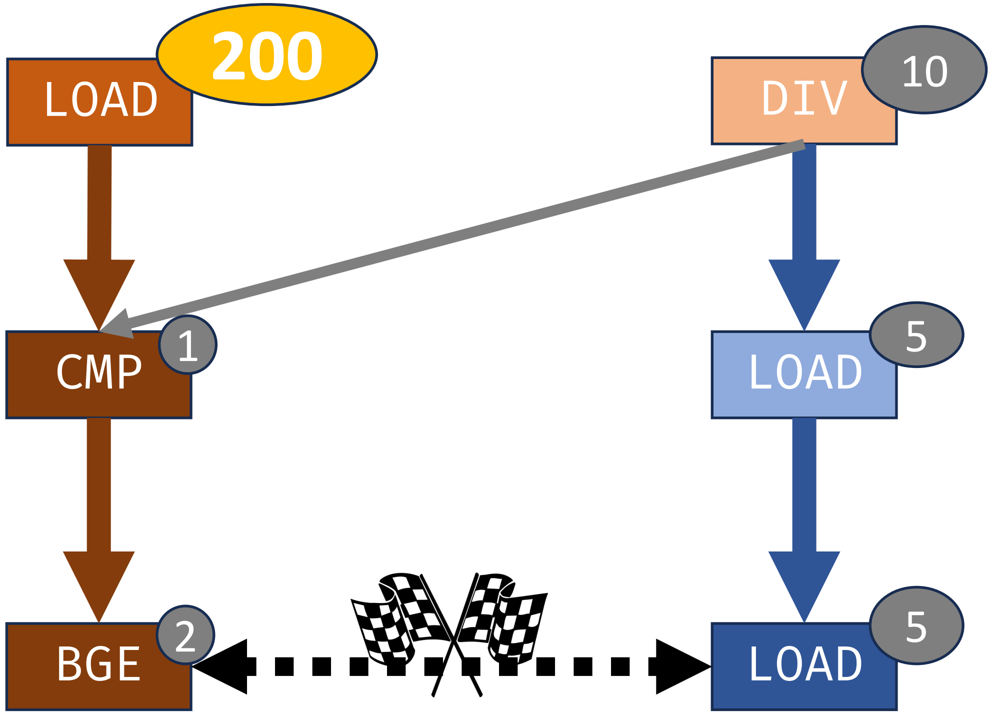
What if I do cache eviction?



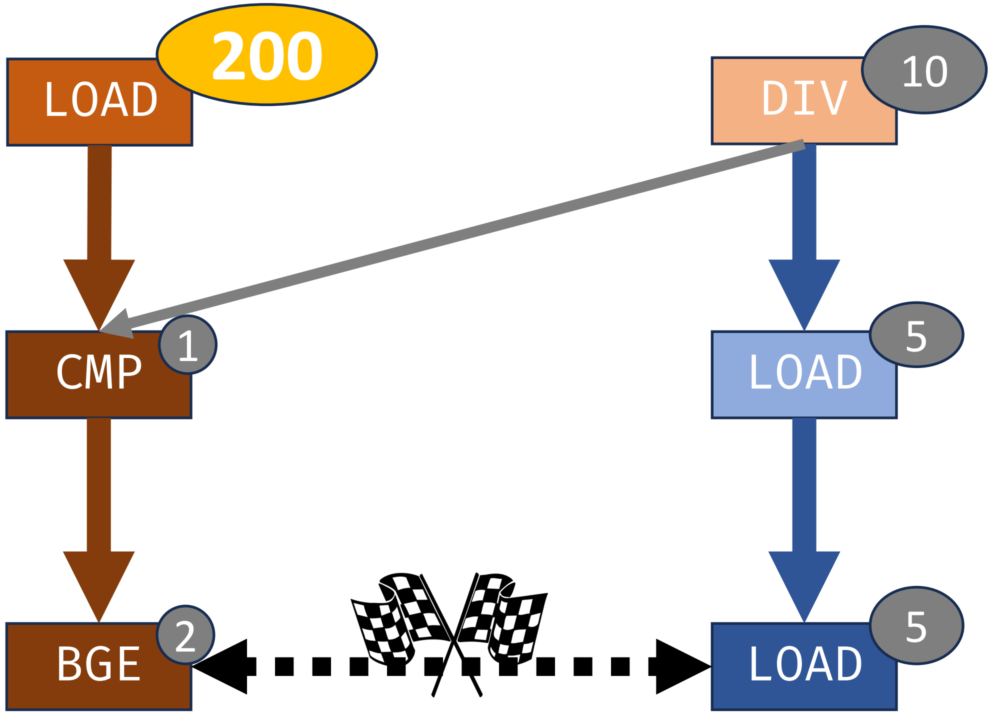
What if I do cache eviction?



What if I do cache eviction?



What if I do cache eviction?

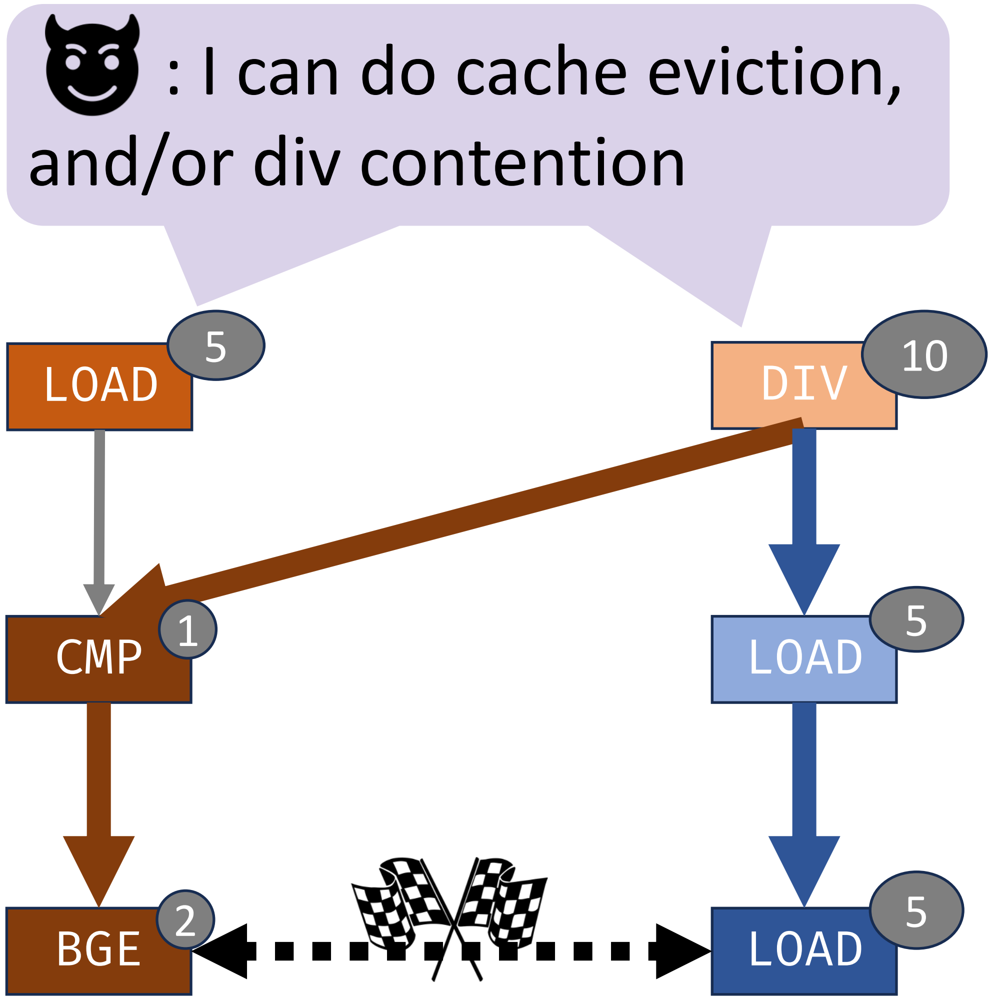


Timing Condition Index was -7.

Now, it increases to **183**!

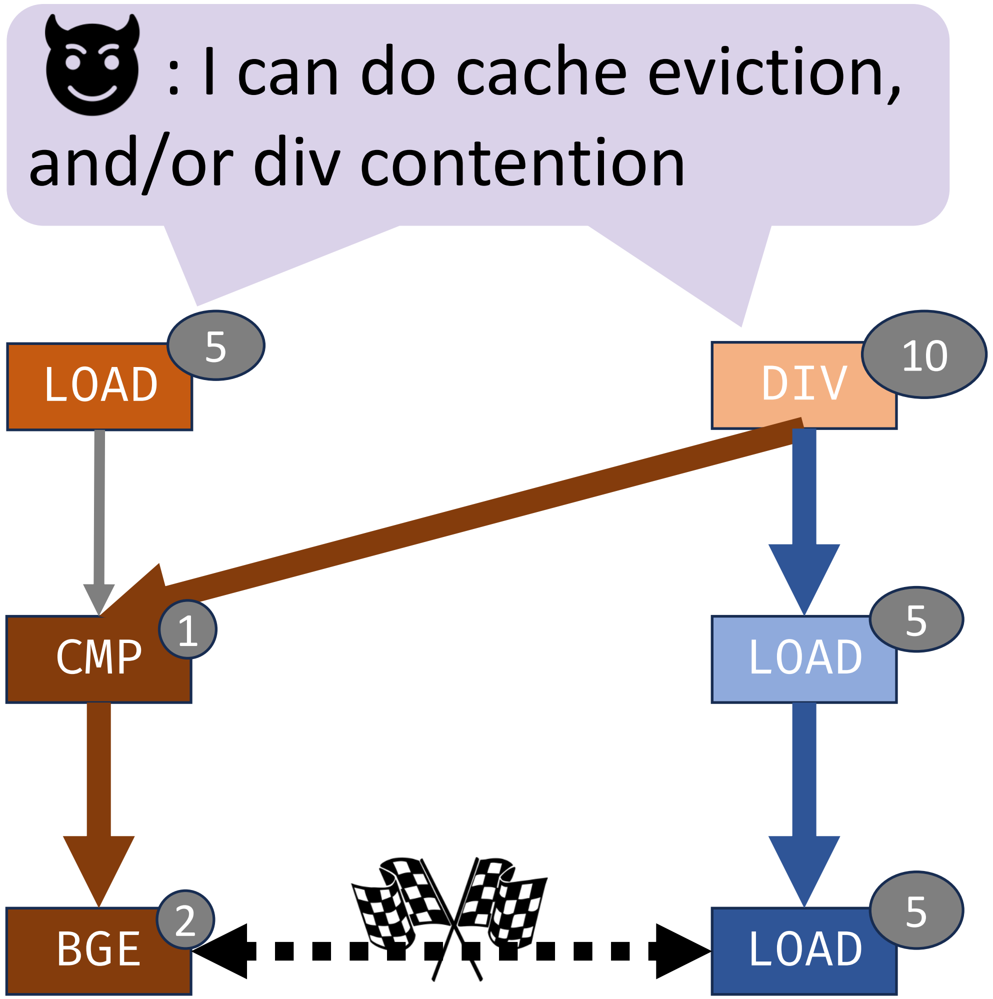
The attack pattern is effective :-)

Step B: Simulating windowing power



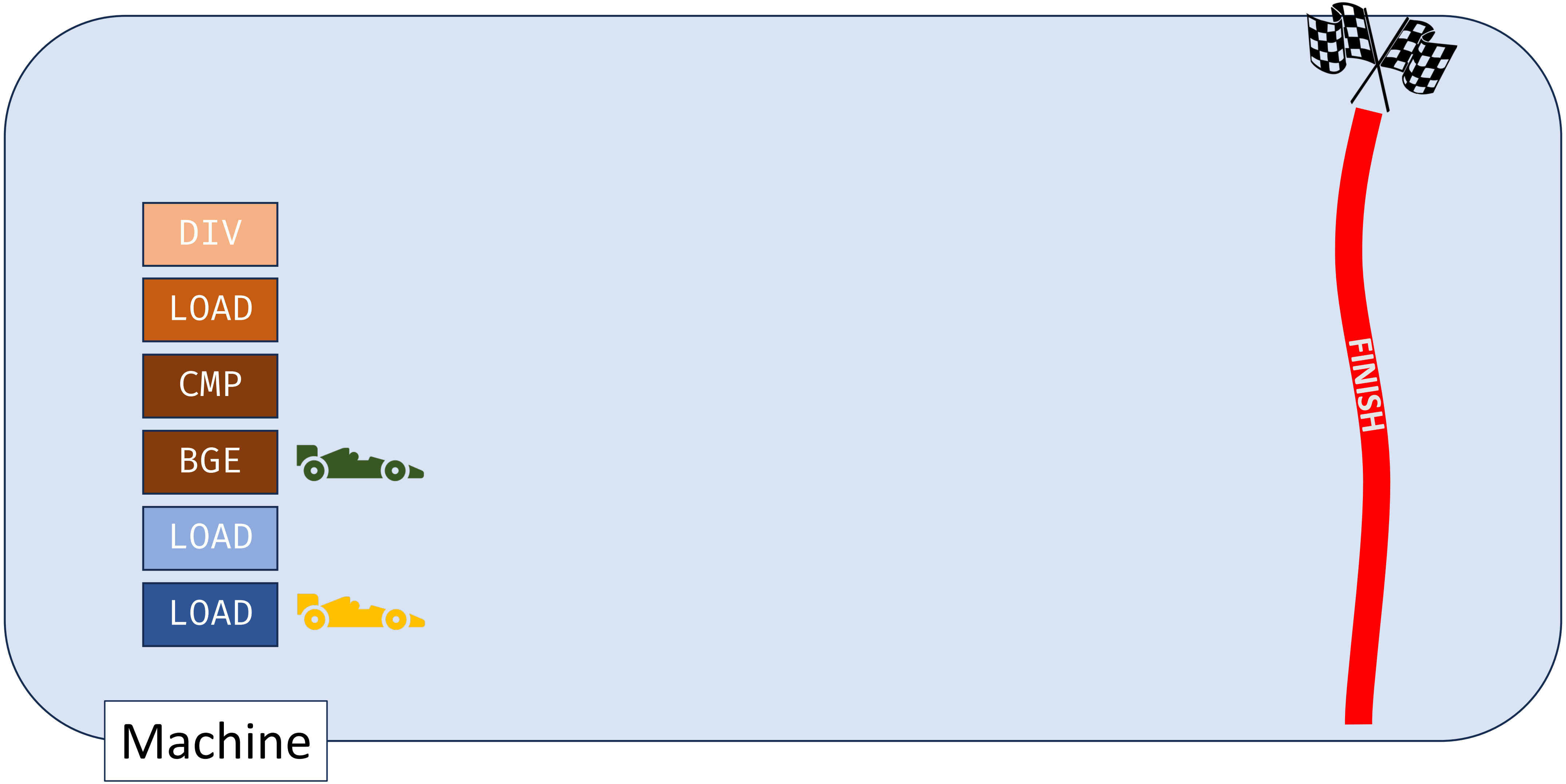
Attack Pattern	**Increase** in Timing Condition Index
Do nothing	0
Cache eviction	+190
Div contention	-140
Cache eviction + Div contention	+50

Step B: Simulating windowing power

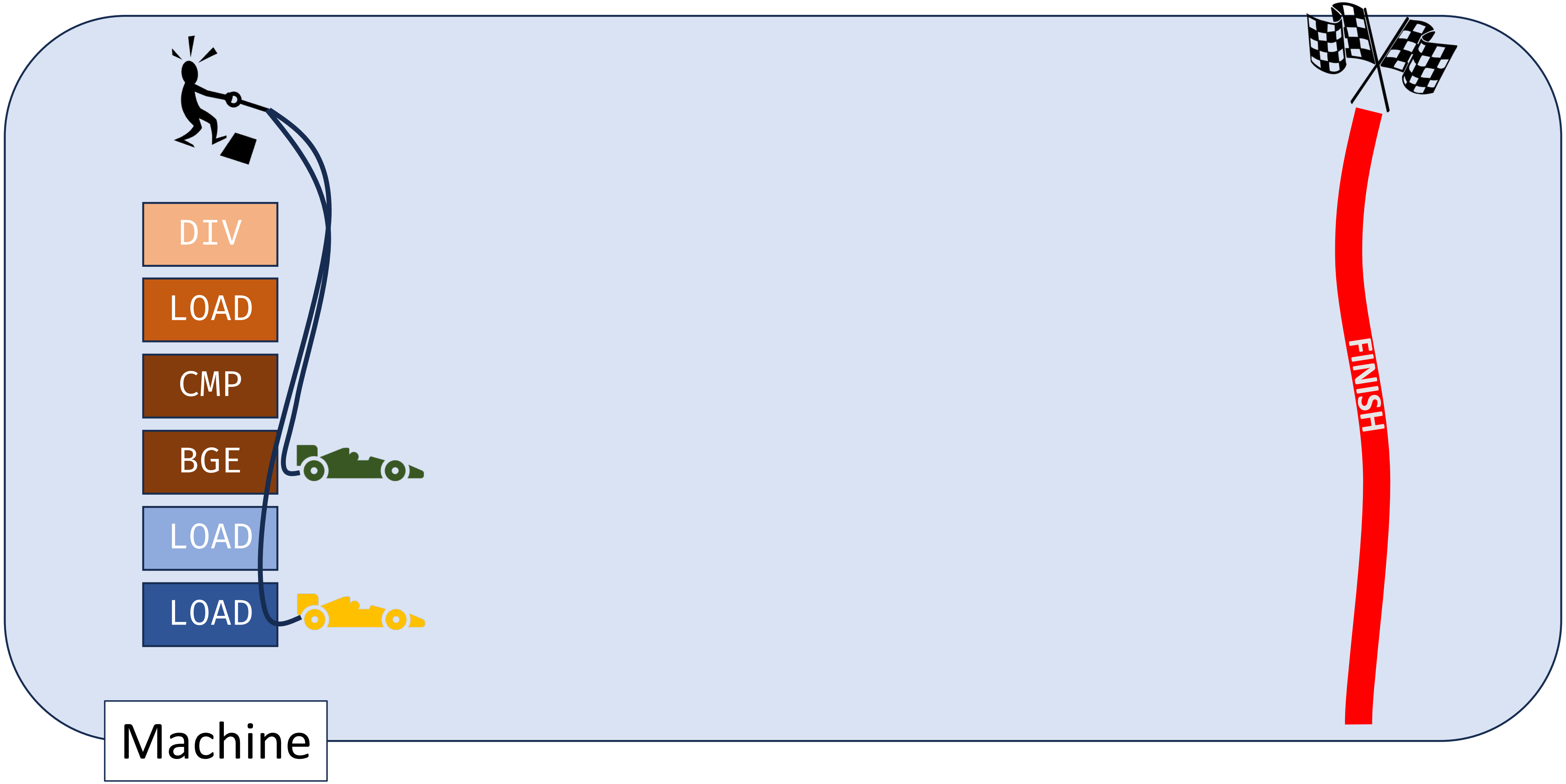


Attack Pattern	**Increase** in Timing Condition Index
Do nothing	0
Cache eviction	+190
Div contention	-140
Cache eviction + Div contention	+50

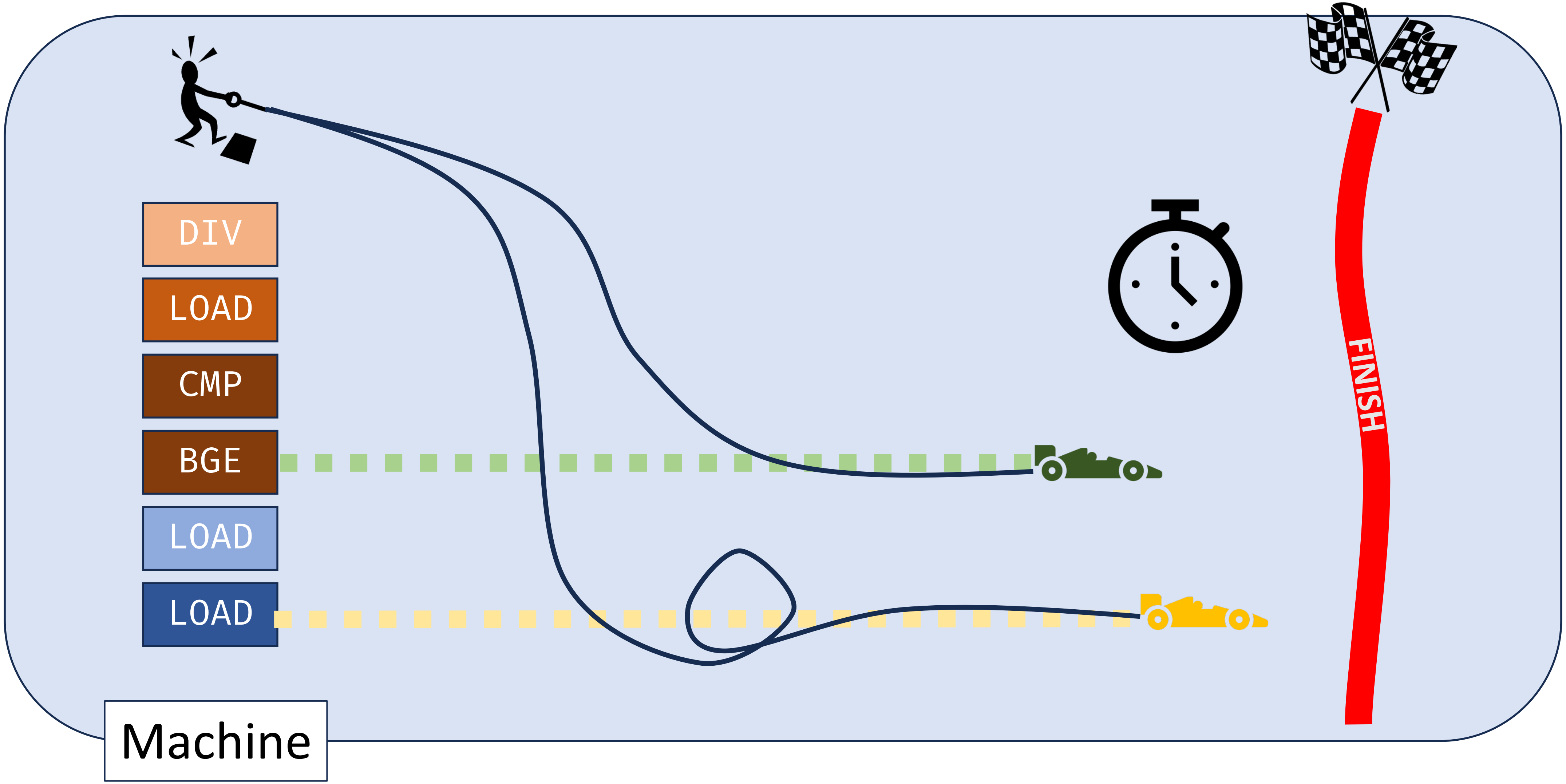
Step C: Measuring exploitability



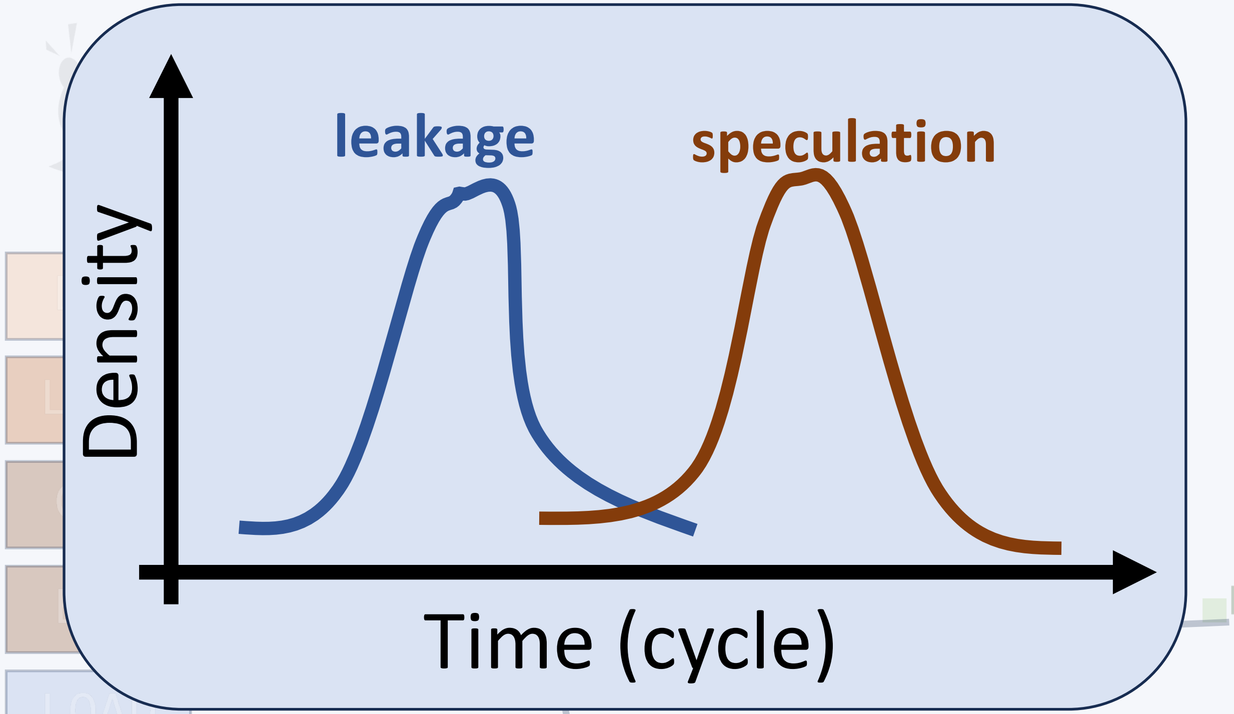
Step C: Measuring exploitability

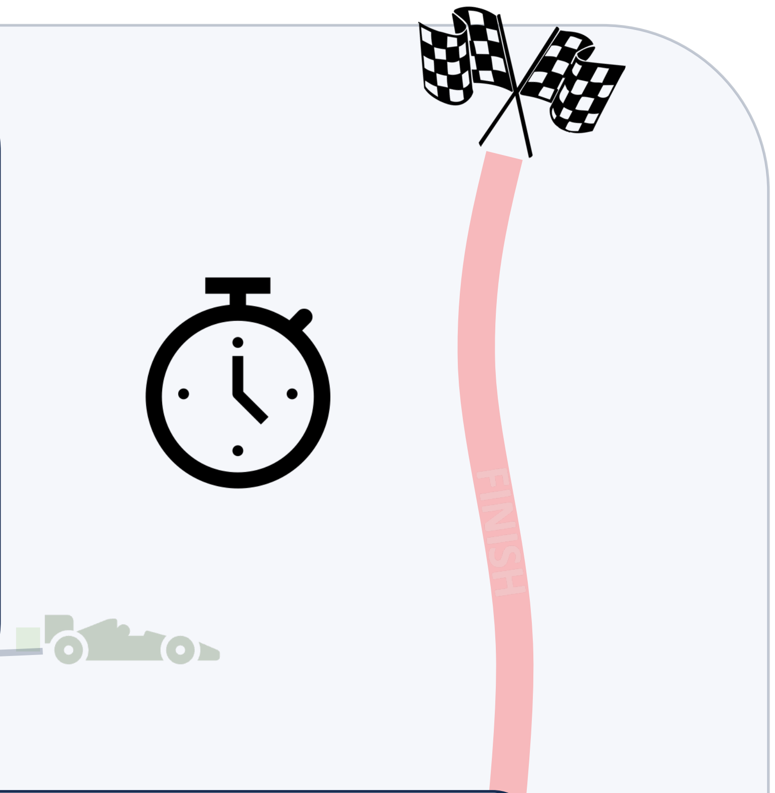


Step C: Measuring exploitability



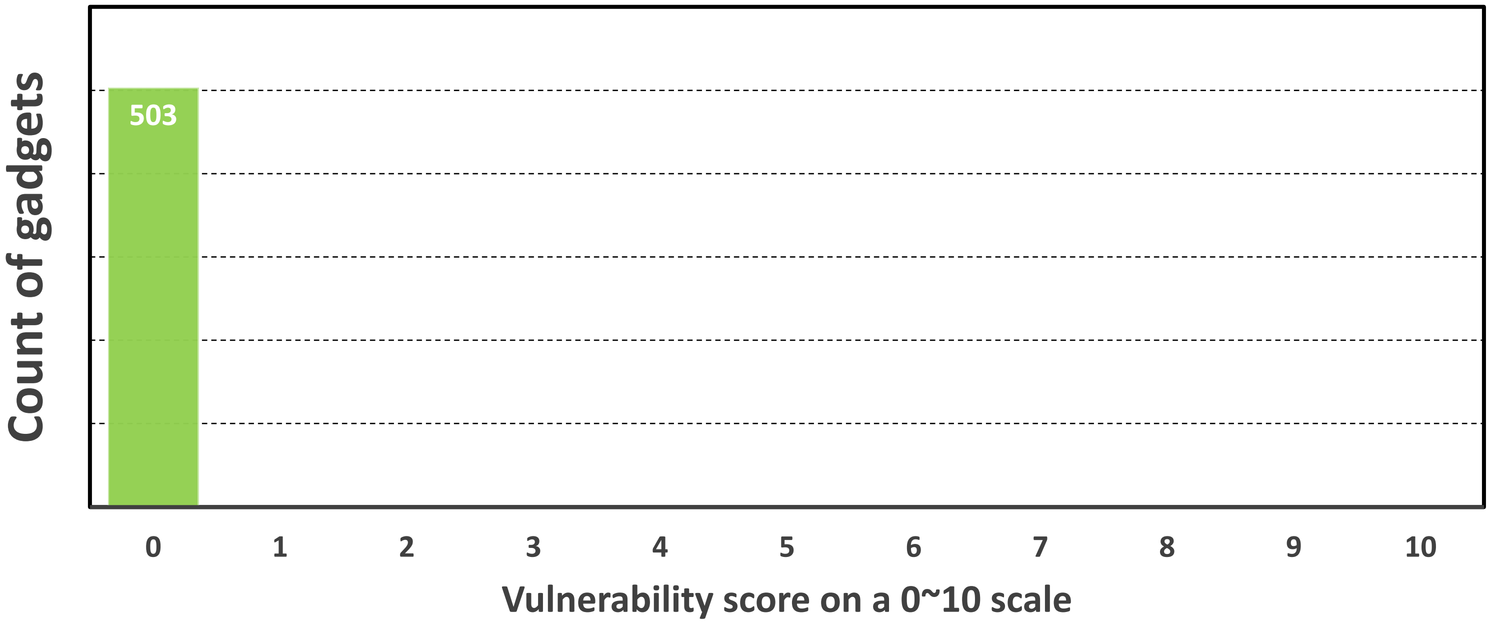
Step C: Measuring exploitability





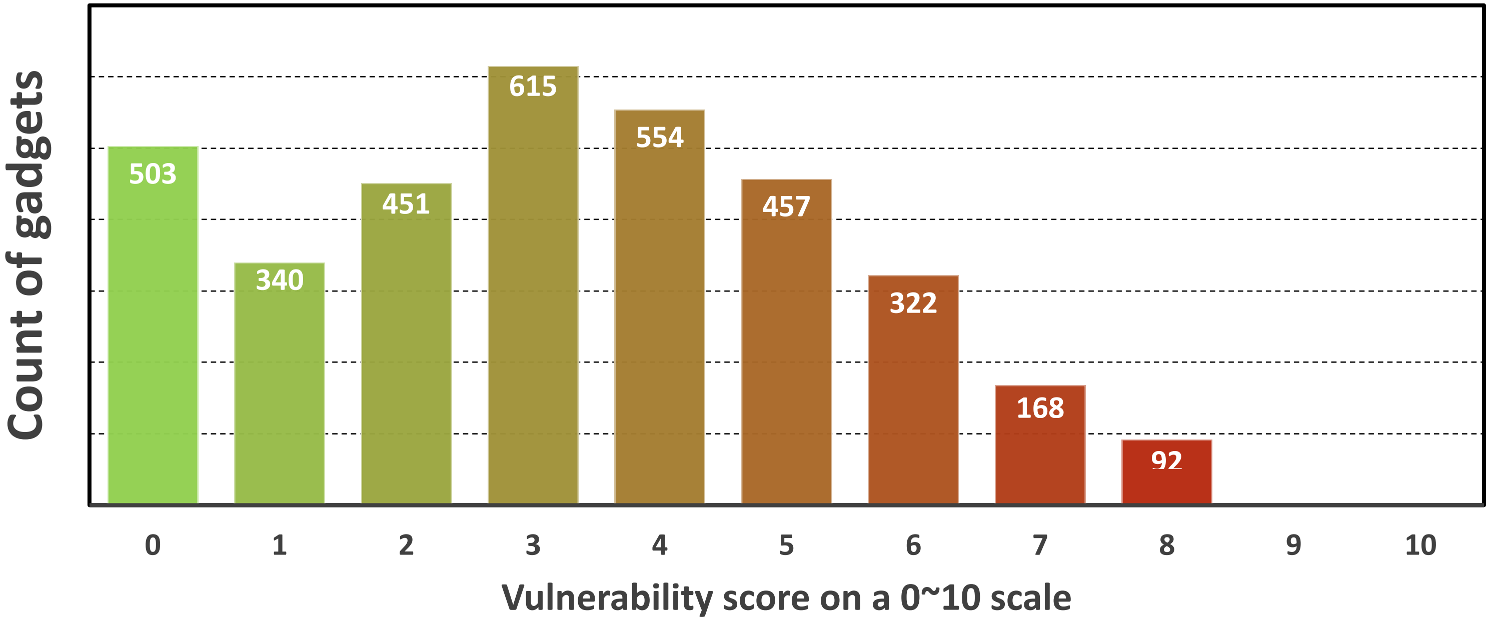
score = 10 * P[speculation > leakage] = **10**/10

Evaluation: Gauging exploitability



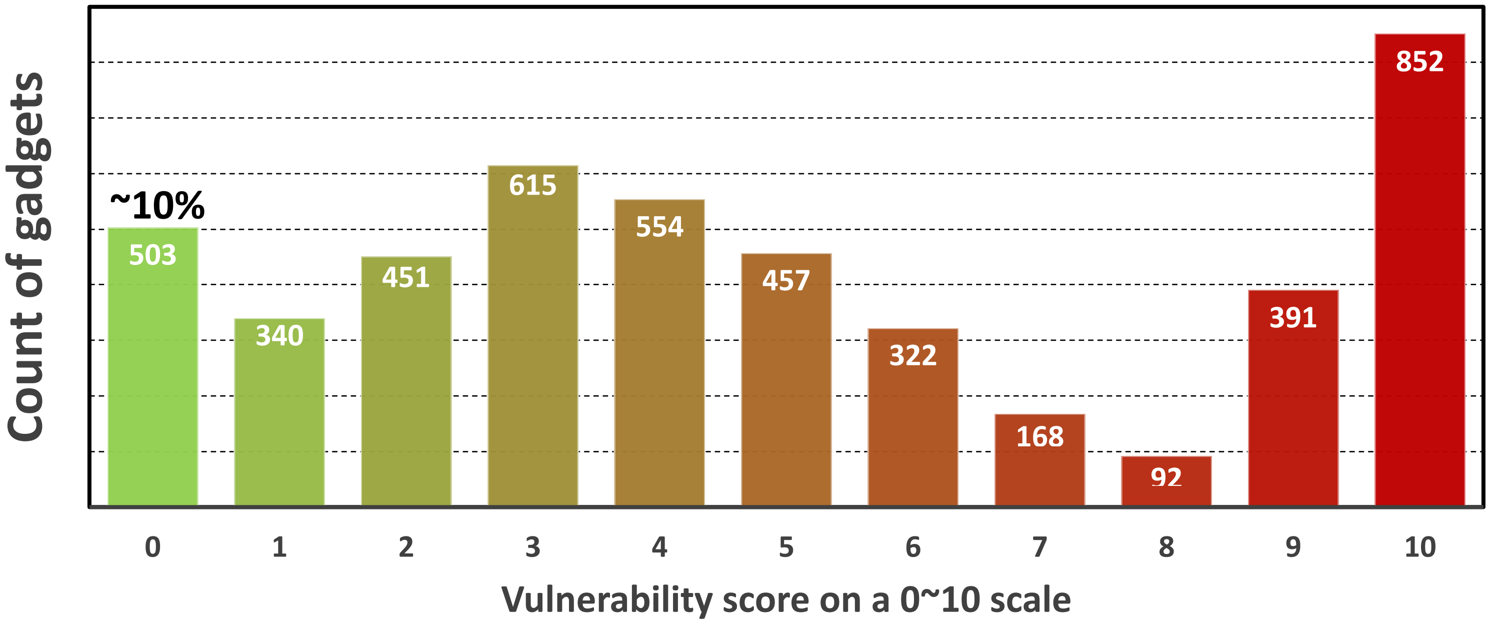
- Target: gadgets with vulnerable information flow, identified by SOTA scanners.
- Applications: 6 security-centric applications and Linux kernel

Evaluation: Gauging exploitability



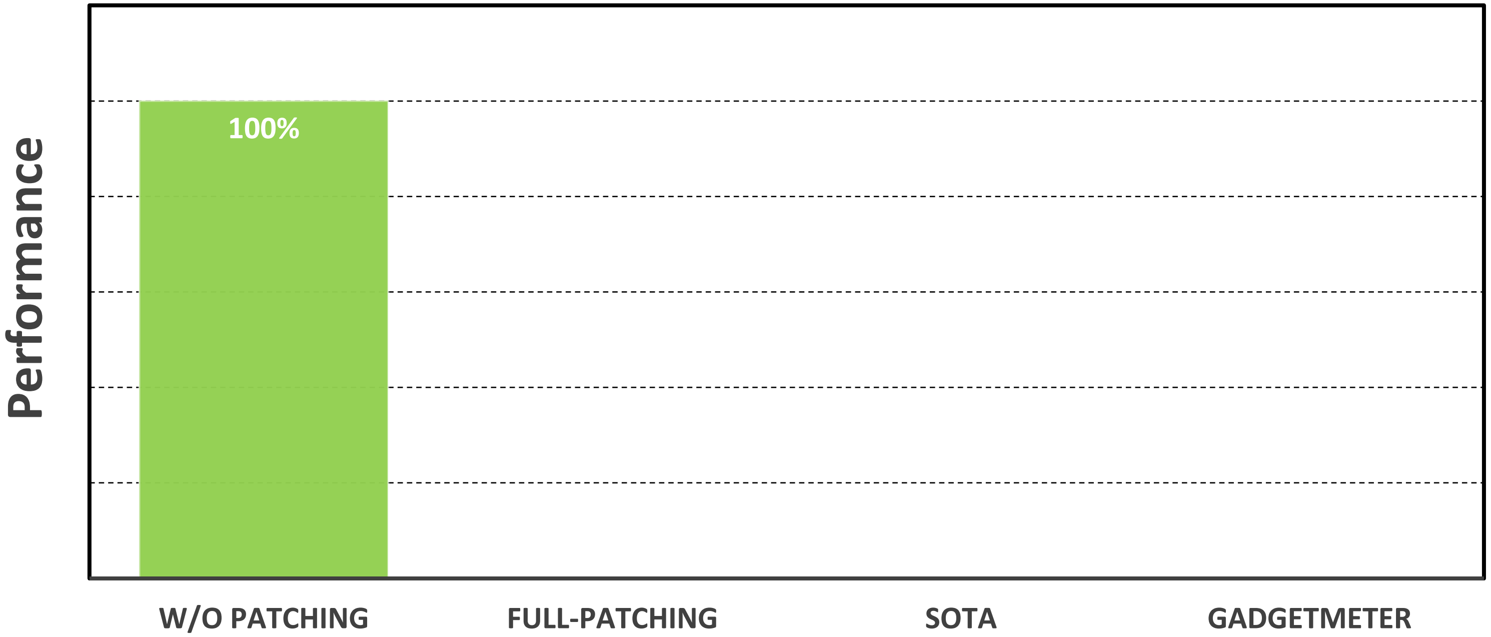
- Target: gadgets with vulnerable information flow, identified by SOTA scanners.
- Applications: 6 security-centric applications and Linux kernel

Evaluation: Gauging exploitability



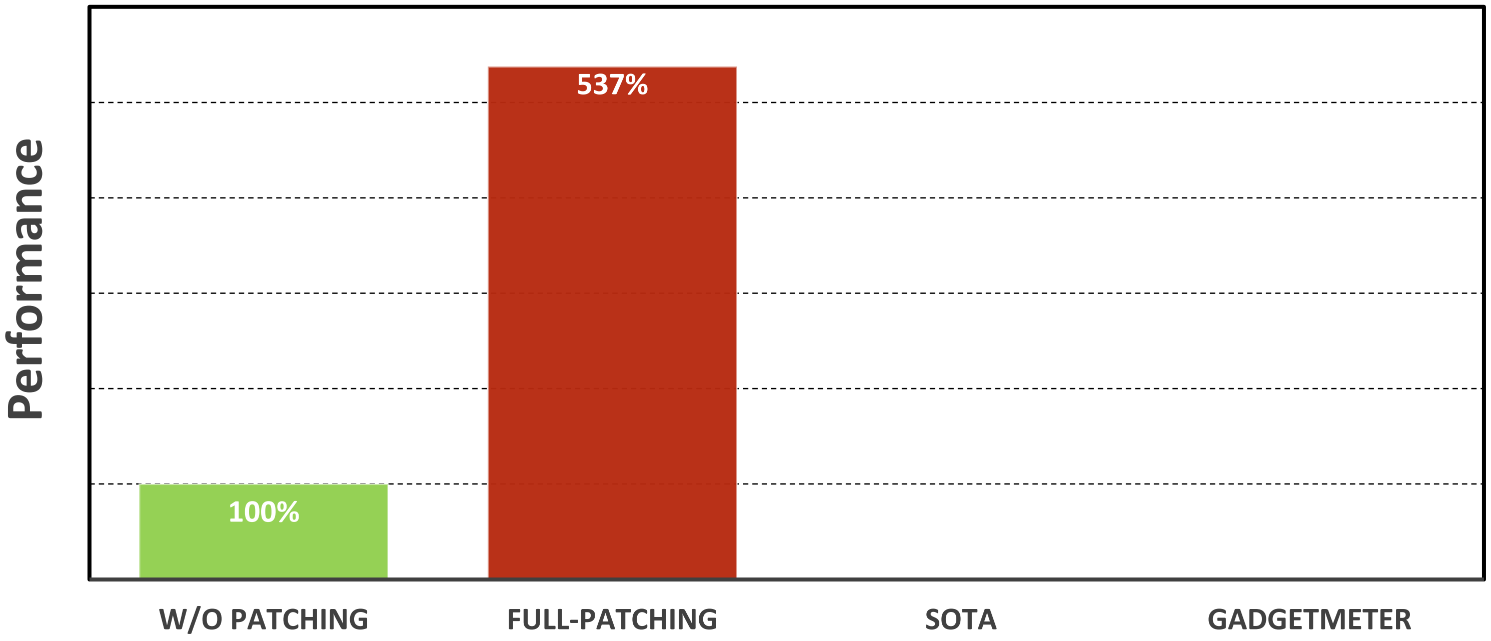
- Target: gadgets with vulnerable information flow, identified by SOTA scanners.
- Applications: 6 security-centric applications and Linux kernel

Evaluation: Performance improvement



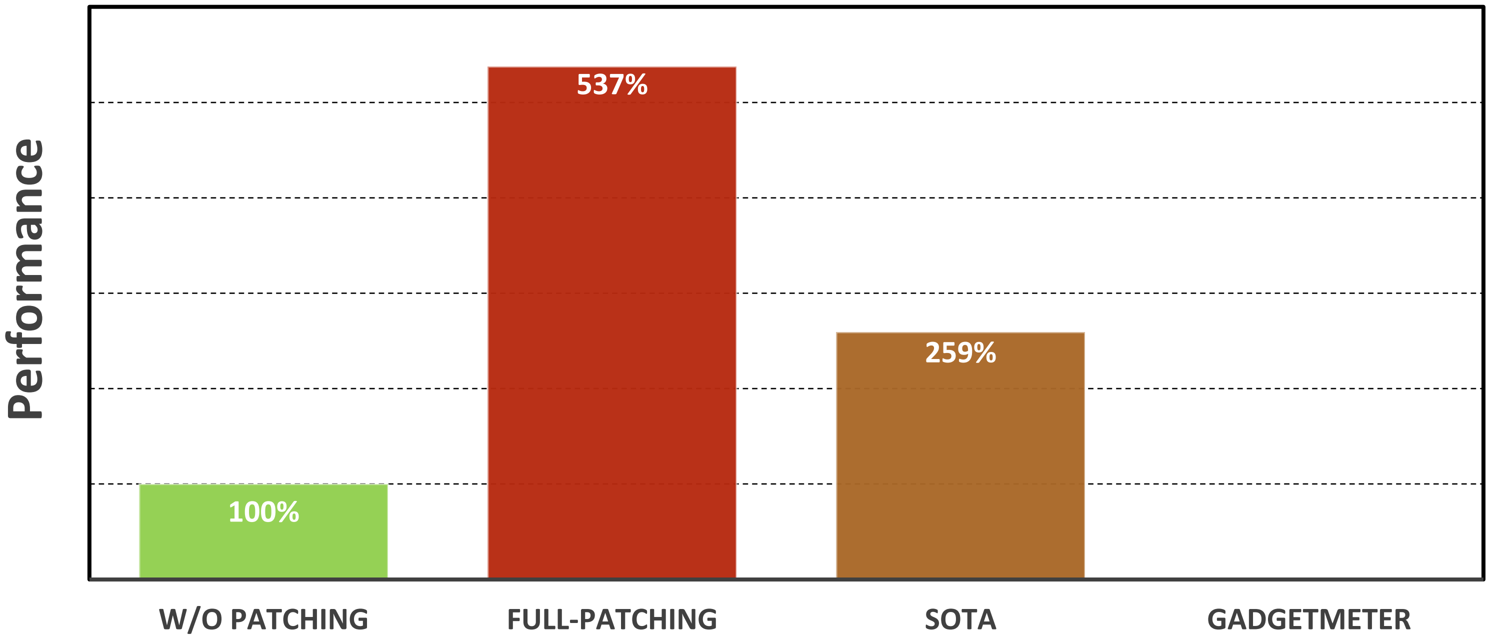
- Patching method: LFENCE serialization.

Evaluation: Performance improvement



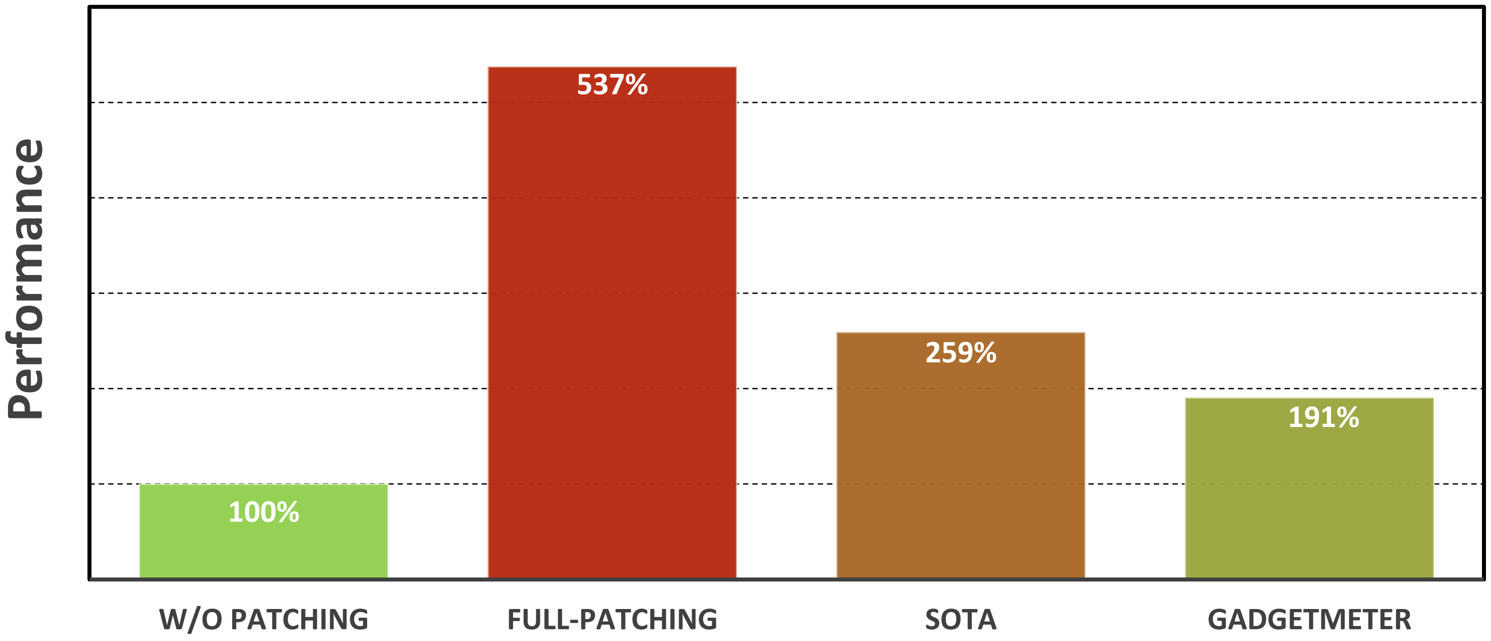
- Patching method: LFENCE serialization.

Evaluation: Performance improvement



- Patching method: LFENCE serialization.

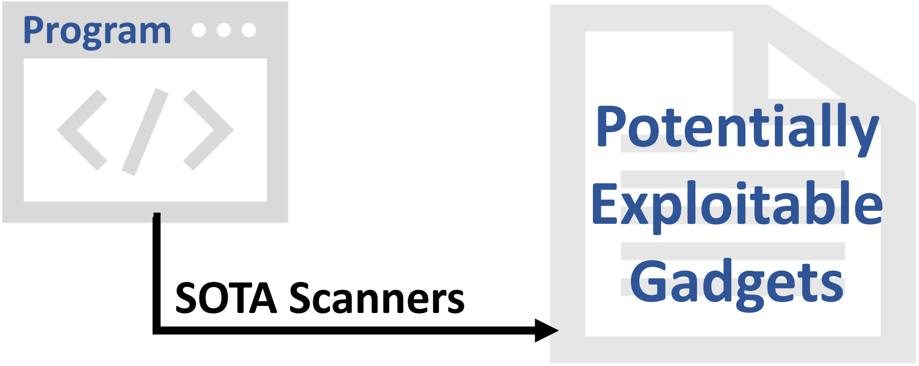
Evaluation: Performance improvement



- Patching method: LFENCE serialization.
- Reduce overhead by 20.66%, compared with SOTA.

Conclusion

qiling07.github.io



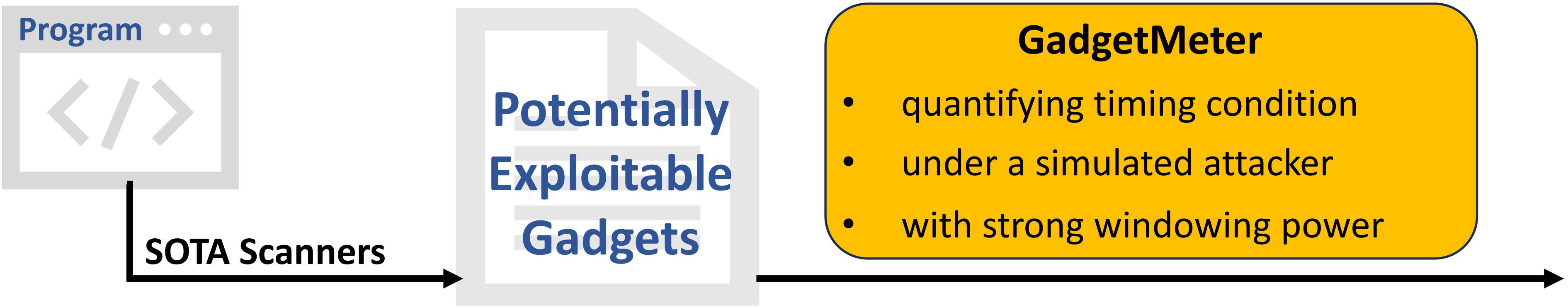
✓ **Few** false negatives

✗ **Many** false positives

✗ **Binary** detection results

Conclusion

qiling07.github.io



Program

SOTA Scanners

Potentially Exploitable Gadgets

GadgetMeter

- quantifying timing condition
- under a simulated attacker
- with strong windowing power

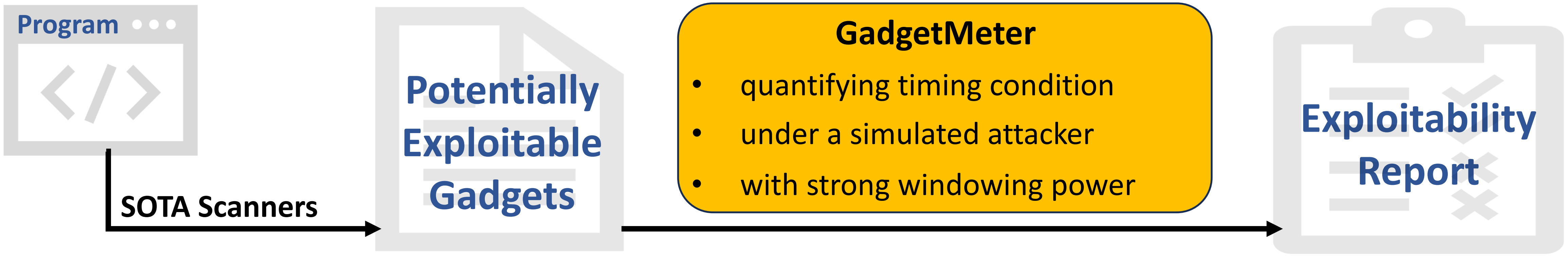
✓ **Few** false negatives

✗ **Many** false positives

✗ **Binary** detection results

Conclusion

qiling07.github.io



Program

SOTA Scanners

Potentially Exploitable Gadgets

GadgetMeter

- quantifying timing condition
- under a simulated attacker
- with strong windowing power

Exploitability Report

✓ **Few** false negatives

✗ **Many** false positives

✗ **Binary** detection results

✓ **Few** false negatives

✓ **Few** false positives

✓ **Quantitative** evaluation results