Seguridad Informatica - Fall 2024

Module IV: Physical security
Lecture 3
Speculative execution attacks

Marco Guarnieri
IMDEA Software Institute

Module II

- 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 II

- 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

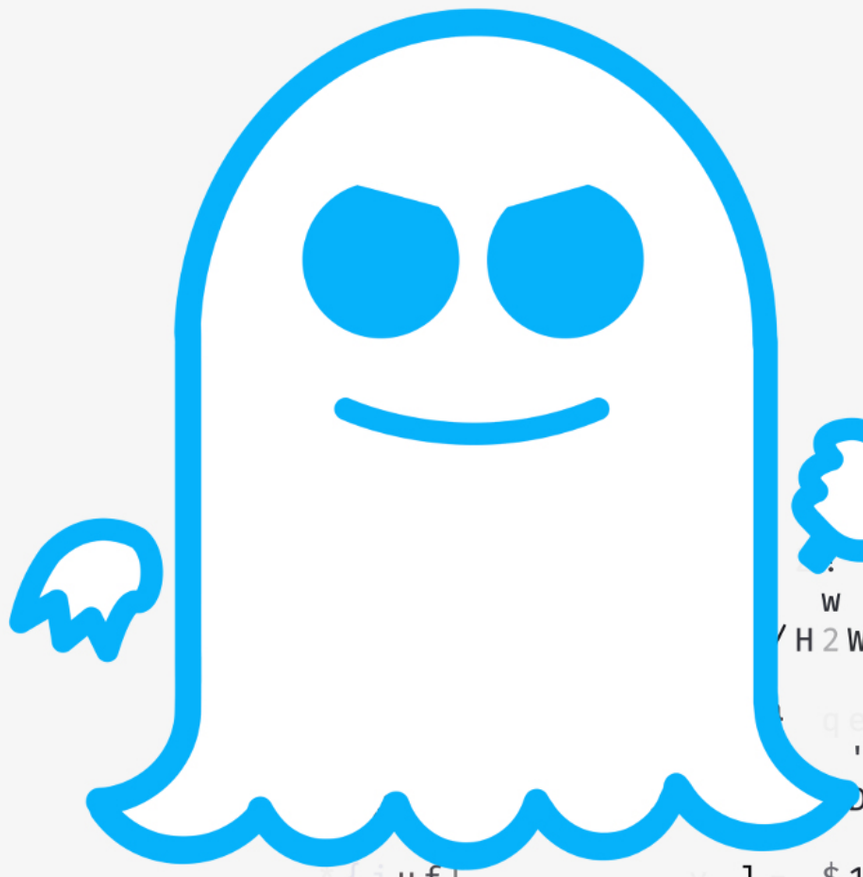
Spectre Attacks: Exploiting Speculative Execution by Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom

Available at https://spectreattack.com/spectre.pdf

Outline

- Motivation
- Recap (covert vs side channels, caches, pipelines)
- Speculative execution and branch predictors
- Attacks (SPECTRE Variant 1)
- Countermeasures (SPECTRE Variant 1)
- Attacks (SPECTRE Variant 2)

MELTDOWN



SPECTRE

MELTDOWN

SPECTRE


FIREEYE CYBER THREAT MAP
LIVE
CNN
DAX ▲164.69
DEVELOPING STORY
COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES
NEWS STREAM


FOX BUSINESS
WASHINGTON, D.C.
WASHINGTON, D.C.
NEWS ALERT
INTEL REVEALS DESIGN FLAW THAT COULD ALLOW HACKERS TO ACCESS DATA
WINTER STORM
FOX BUSINESS NETWORK
@FOXBUSINESS


DEVELOPING
COMPUTER CH
THE MELTDOWN AND SPECTRE EXPLOITS USE "SPECULATIVE EXECUTION?" WHAT'S THAT?
YOU KNOW THE TROLLEY PROBLEM? WELL, FOR A WHILE NOW, CPUs HAVE BASICALLY BEEN SENDING TROLLEYS DOWN BOTH PATHS, QUANTUM-STYLE, WHILE AWAITING YOUR CHOICE. THEN THE UNNEEDED "PHANTOM" TROLLEY DISAPPEARS.
THE PHANTOM TROLLEY ISN'T SUPPOSED TO TOUCH ANYONE. BUT IT TURNS OUT YOU CAN STILL USE IT TO DO STUFF.
AND IT CAN DRIVE THROUGH WALLS.
THAT SOUNDS BAD.
HONESTLY, I'VE BEEN ASSUMING WE WERE DOOMED EVER SINCE I LEARNED ABOUT ROWHAMMER.
WHAT'S THAT?
IF YOU TOGGLE A ROW OF MEMORY CELLS ON AND OFF REALLY FAST, YOU CAN USE ELECTRICAL INTERFERENCE TO FLIP NEARBY BITS AND—
DO WE JUST SUCK AT...COMPUTERS?
YUP. ESPECIALLY SHARED ONES.
SO YOU'RE SAYING THE CLOUD IS FULL OF PHANTOM TROLLEYS ARMED WITH HAMMERS.
...YES. THAT IS EXACTLY RIGHT.
OKAY. I'LL, UH... INSTALL UPDATES?
GOOD IDEA.
HINGTON, D.C.
BUSINESS NETWORK
INTEL REVEALS DESIGN FLAW THAT COULD ALLOW HACKERS TO ACCESS DATA
WINTER STORM
@FOXBUSINESS

Recap / background

Covert vs side channels

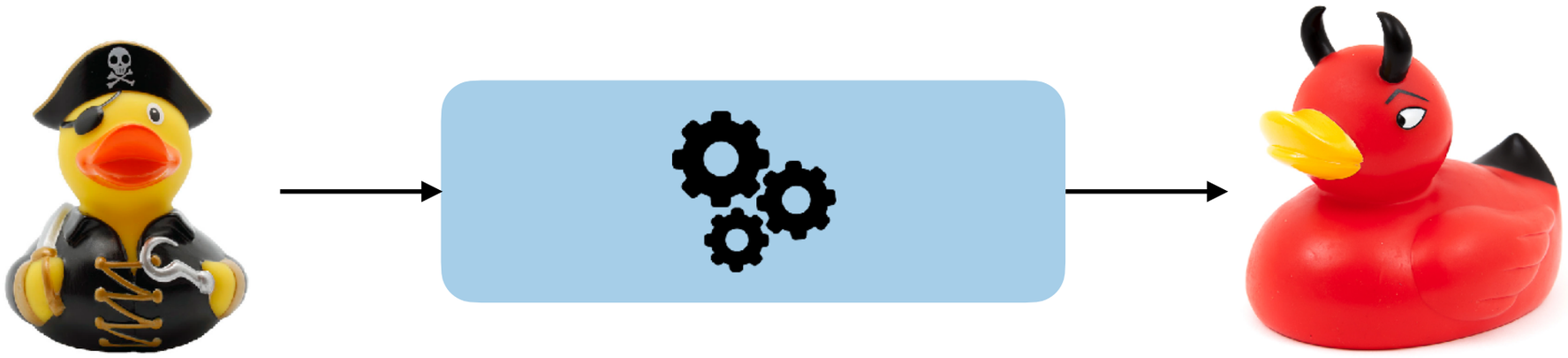
Ducks from https://shop.justducks.co.uk/

Covert vs side channels

Covert
Channel

Ducks from https://shop.justducks.co.uk/

Covert vs side channels



Covert Channel

Sender

Receiver

Ducks from https://shop.justducks.co.uk/

Covert vs side channels

Sender

Receiver

Covert Channel

Ducks from https://shop.justducks.co.uk/

Covert vs side channels

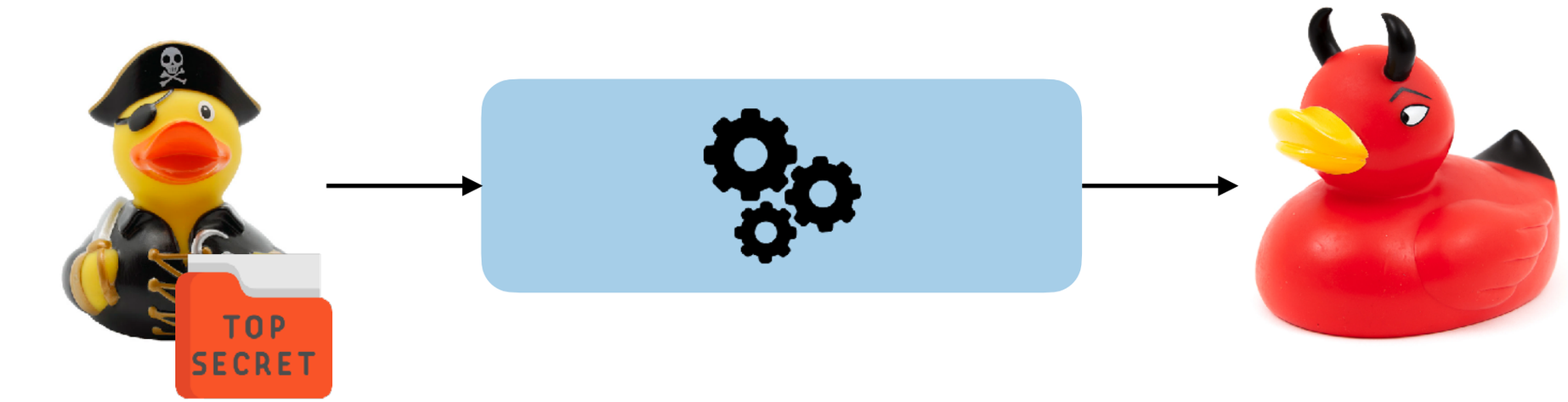
Sender

Receiver

Covert Channel

Ducks from https://shop.justducks.co.uk/

Covert vs side channels



Sender

Receiver

Covert Channel

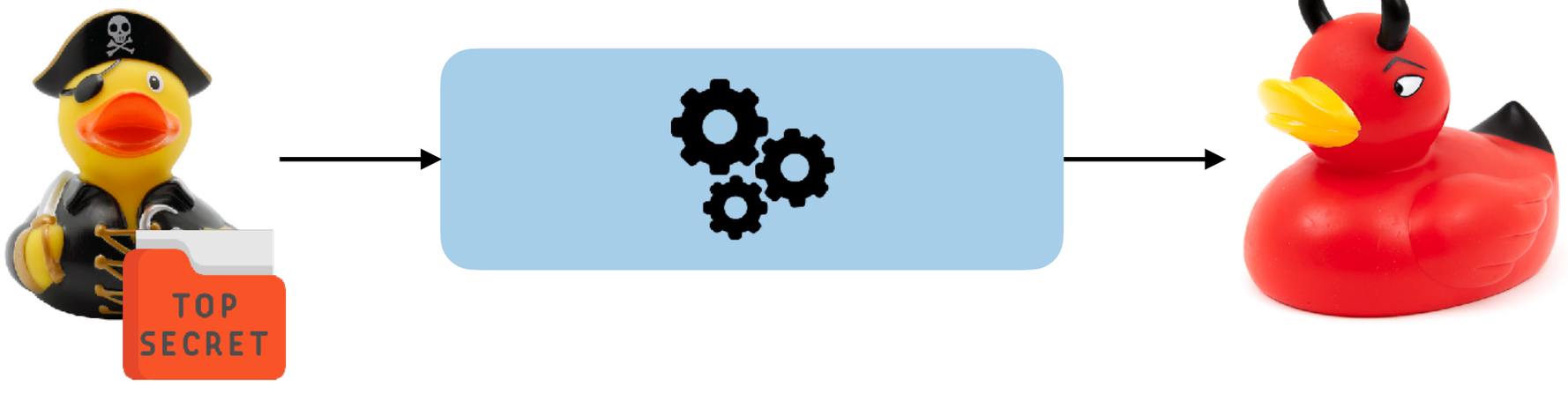
Observee

Observer

Side Channel

Ducks from https://shop.justducks.co.uk/

Covert vs side channels



Sender

Receiver

Covert Channel

Observee

Observer

Side Channel

Ducks from https://shop.justducks.co.uk/

Covert vs side channels

Sender

Receiver

Covert Channel

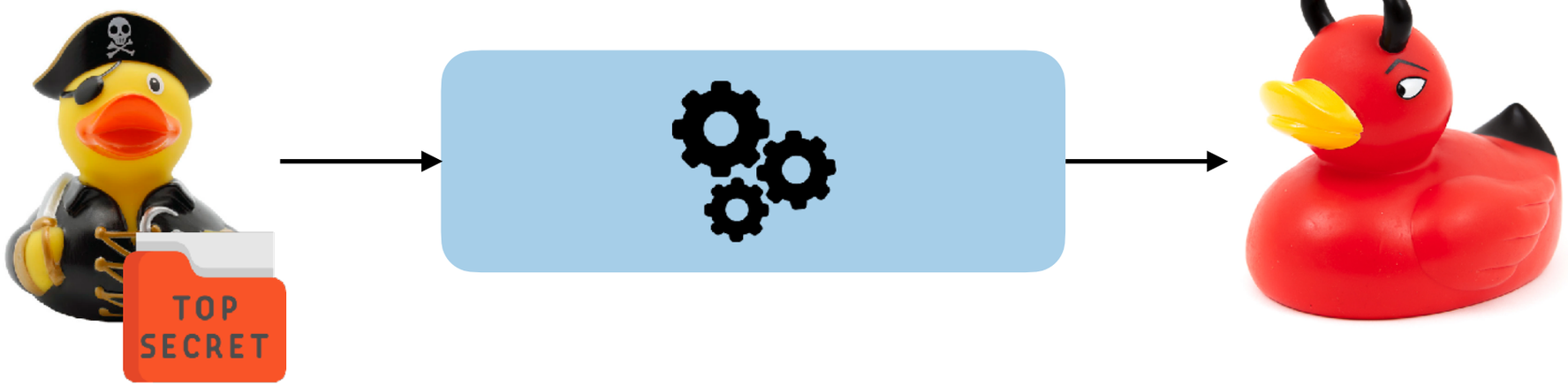
Observee

Observer

Side Channel

Ducks from https://shop.justducks.co.uk/

Covert vs side channels



Sender

Receiver

Covert Channel

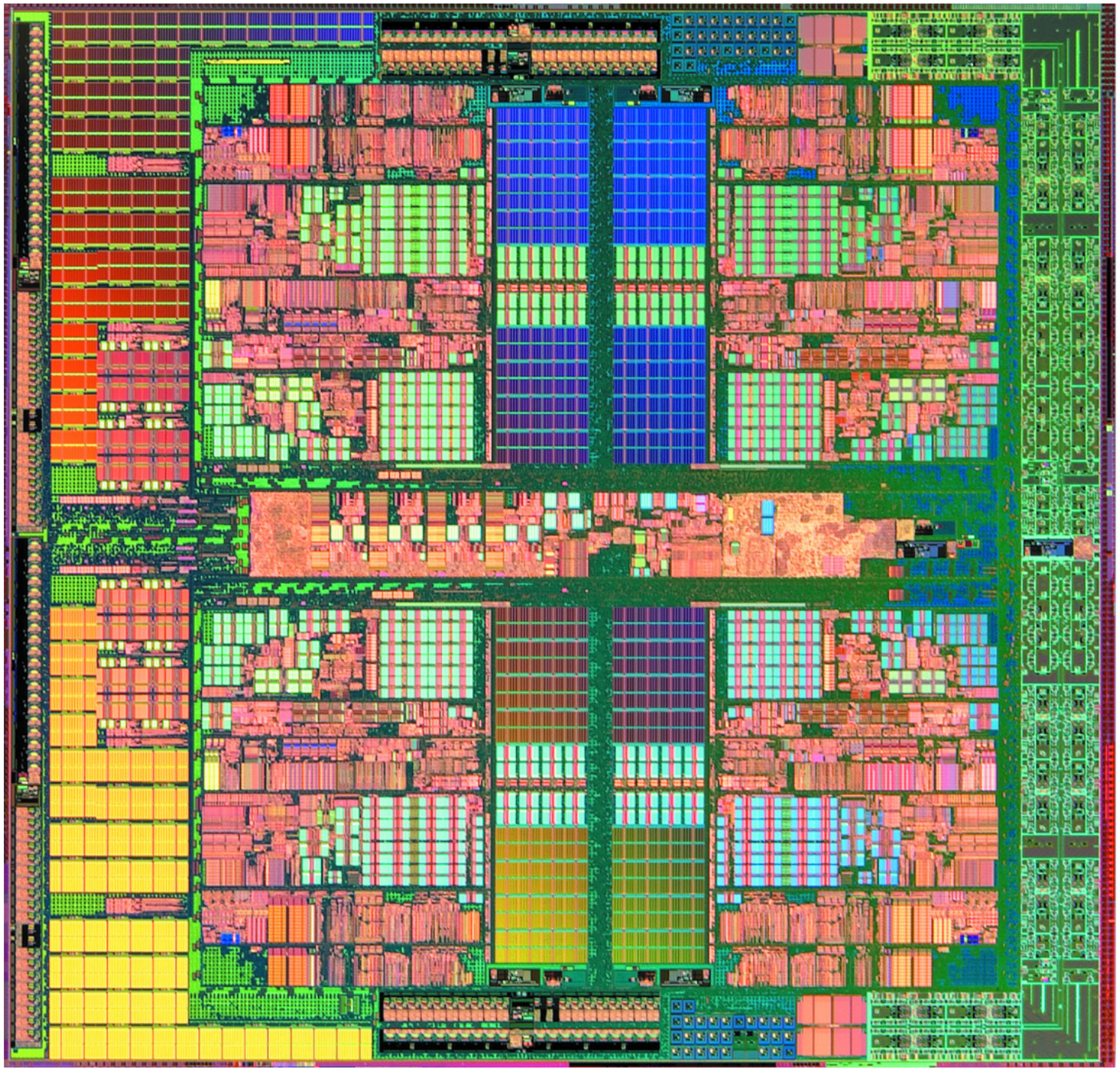
Observee

Observer

Side Channel

Ducks from https://shop.justducks.co.uk/

CPU 101

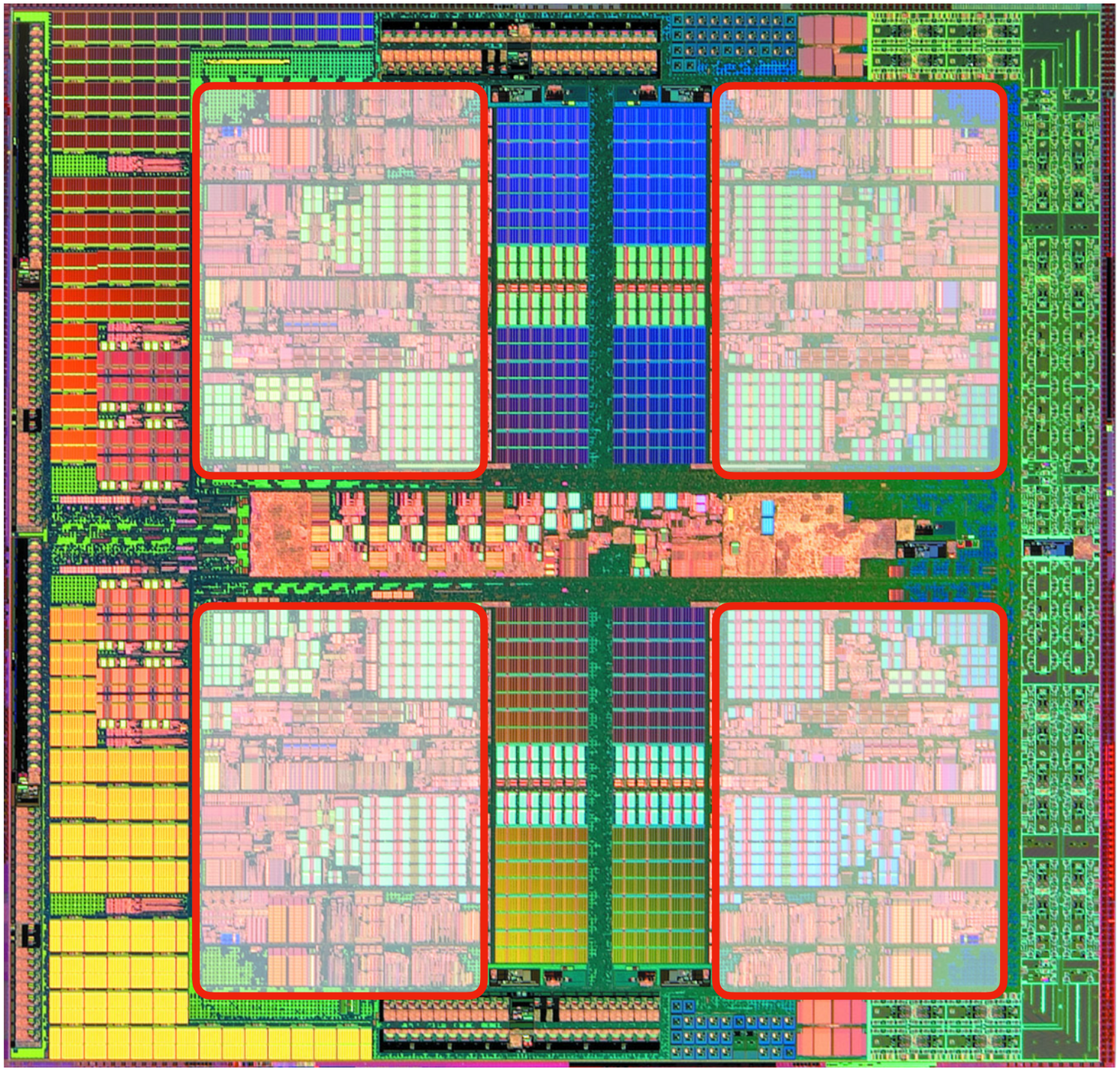


Picture of AMD “Barcelona" Quad Core CPU

CPU 101

CORE

Execute instructions

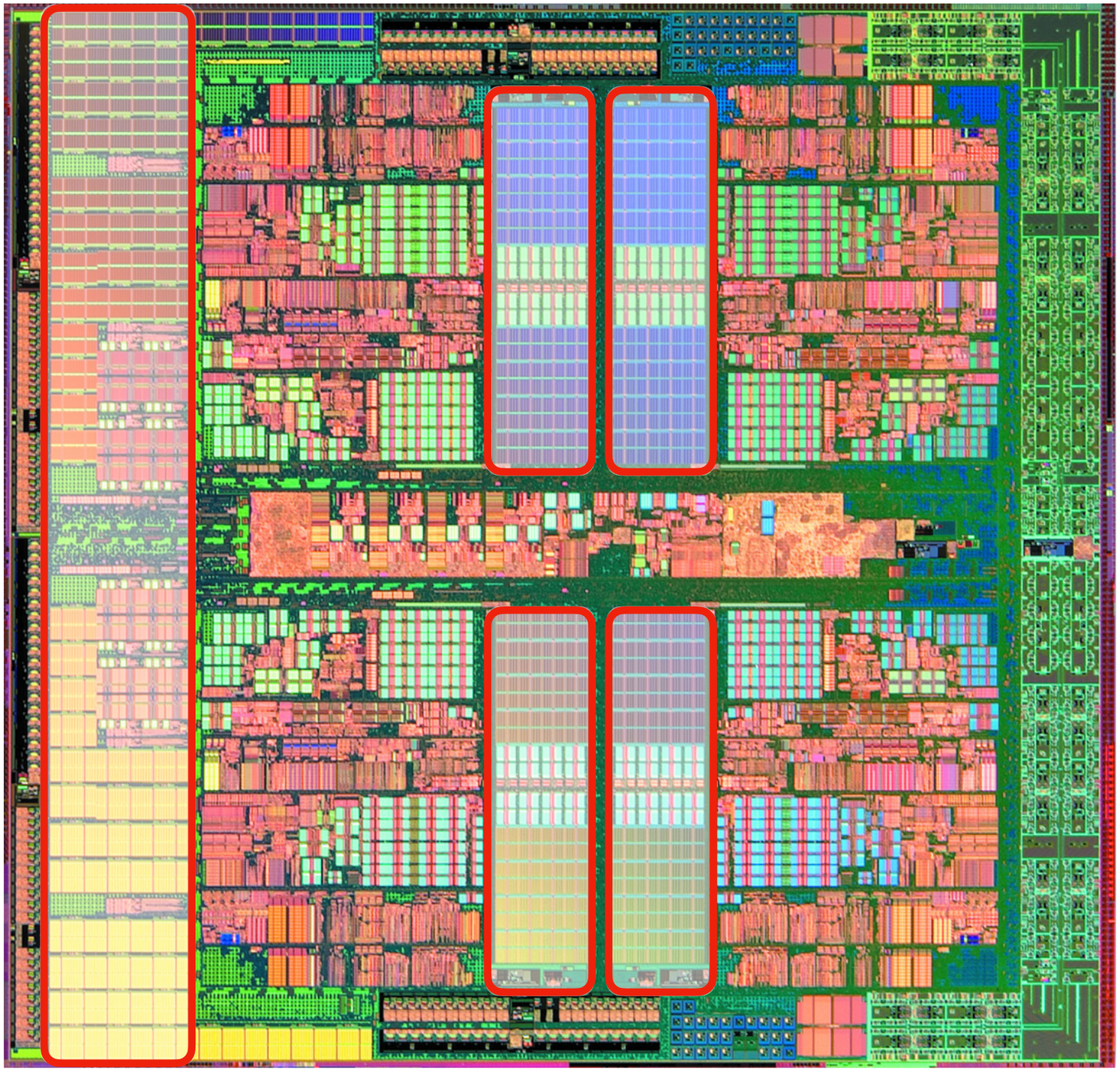


Picture of AMD “Barcelona" Quad Core CPU

CPU 101

CACHE

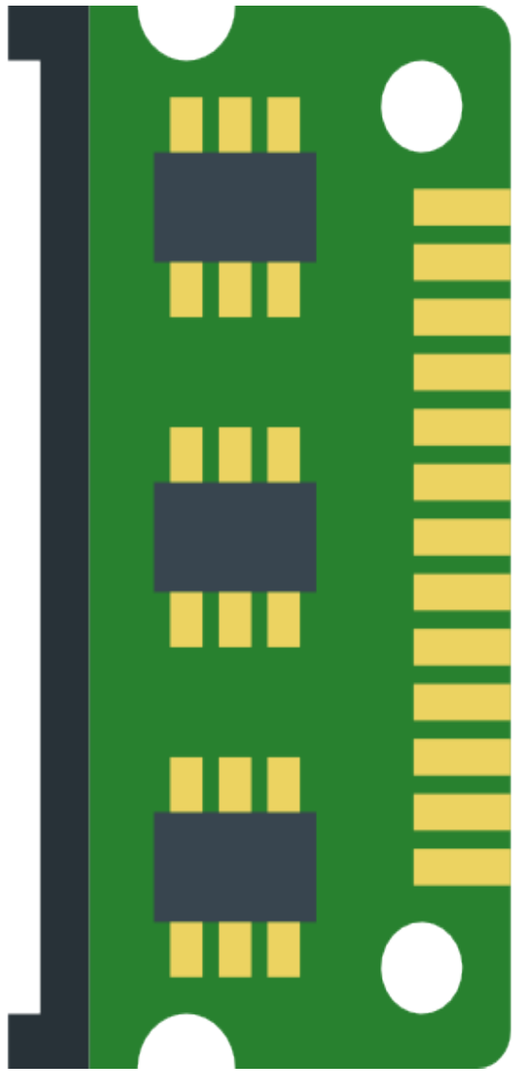
Memory storing recently accessed data



Picture of AMD “Barcelona" Quad Core CPU

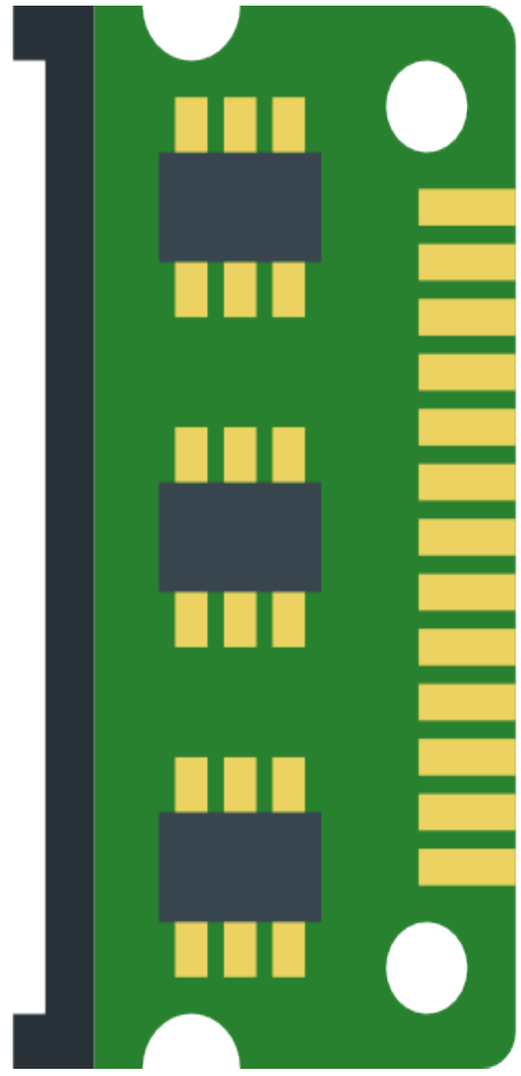
Accessing memory

printf (“%d”, i);
printf (“%d”, i);

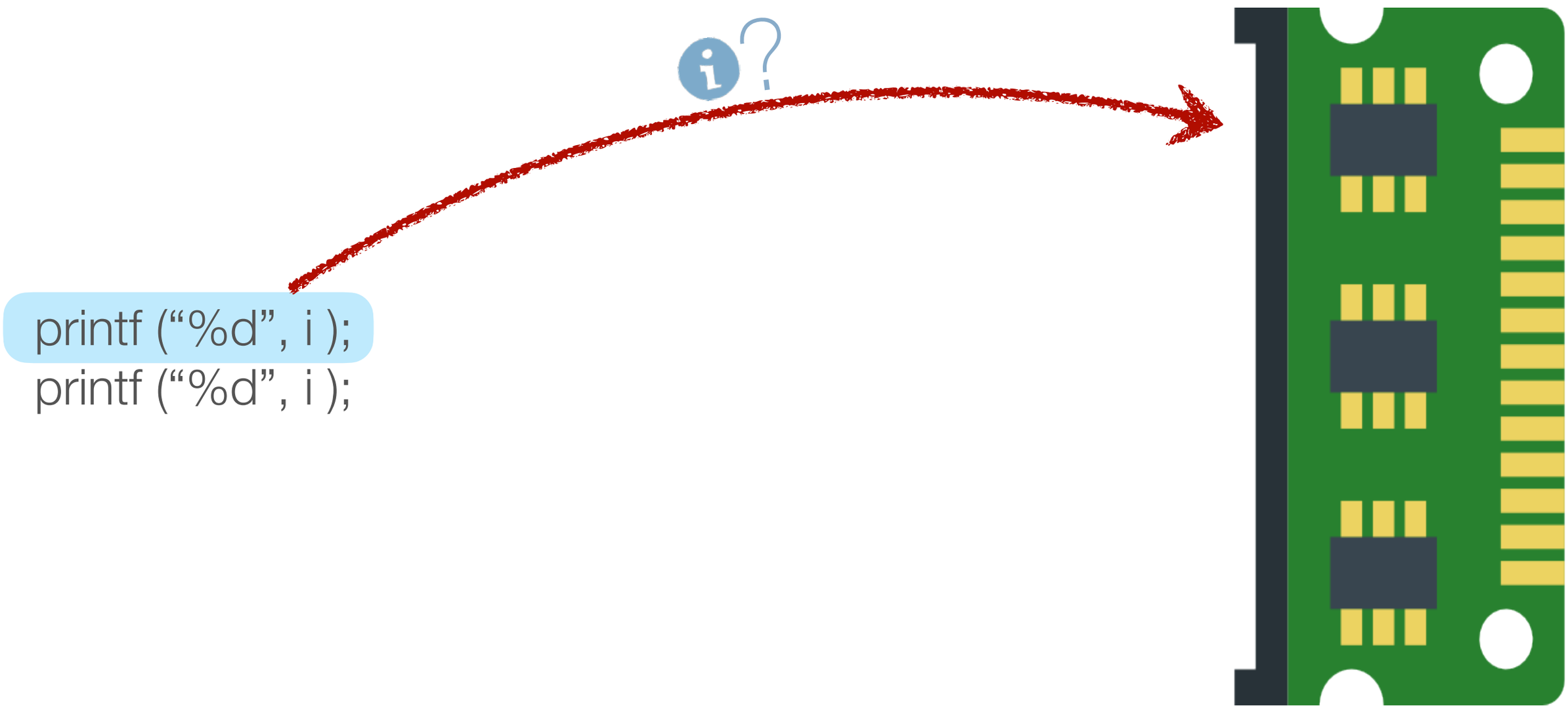


Accessing memory

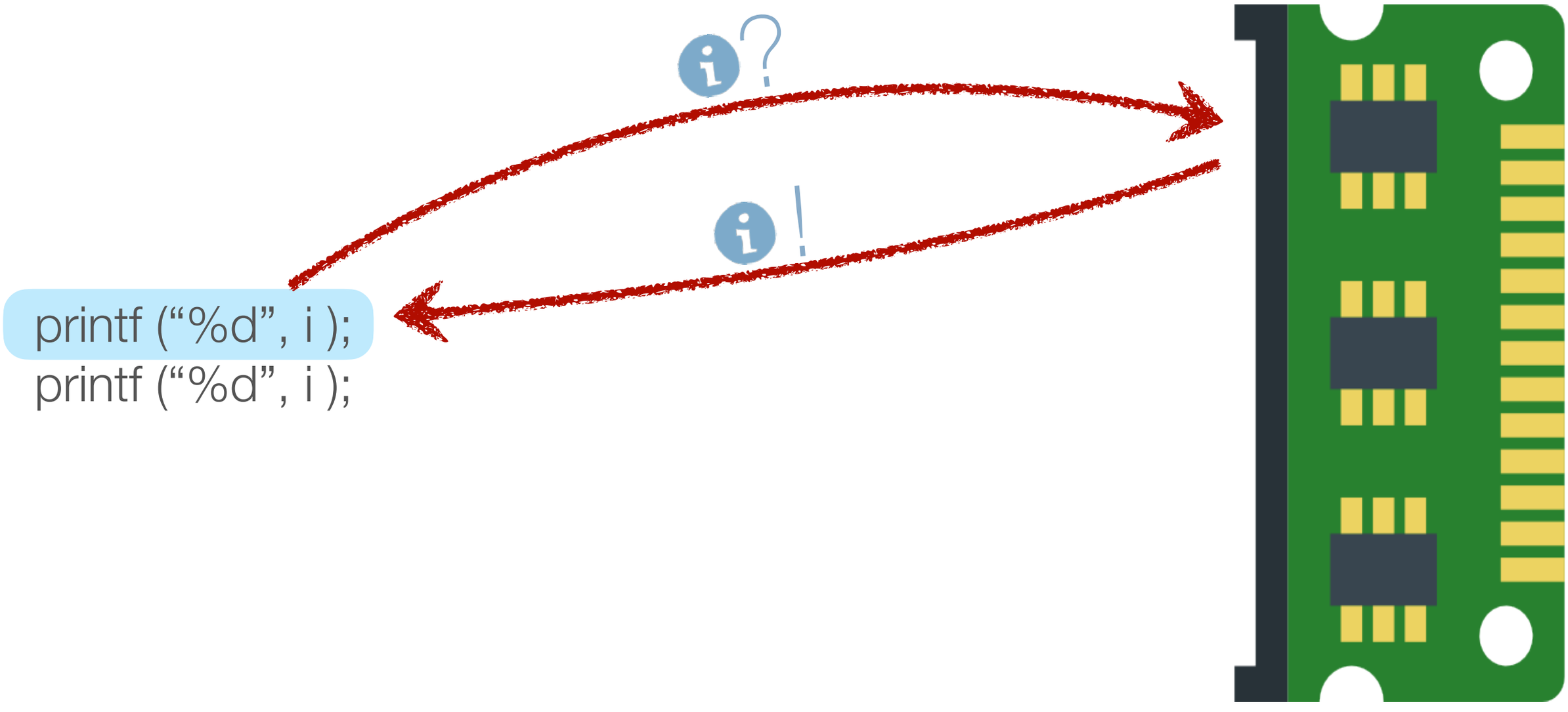
```
printf (“%d”, i );
printf (“%d”, i );
```



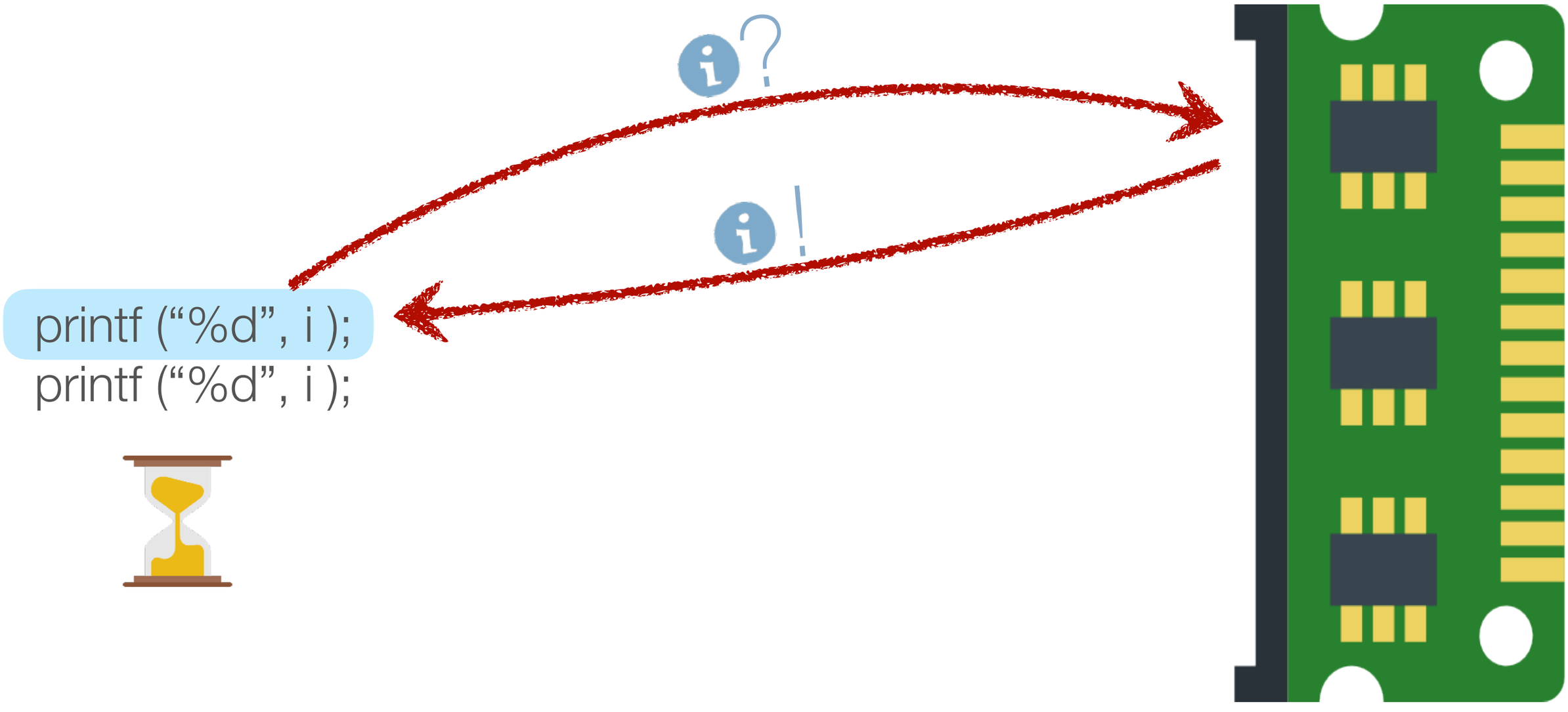
Accessing memory



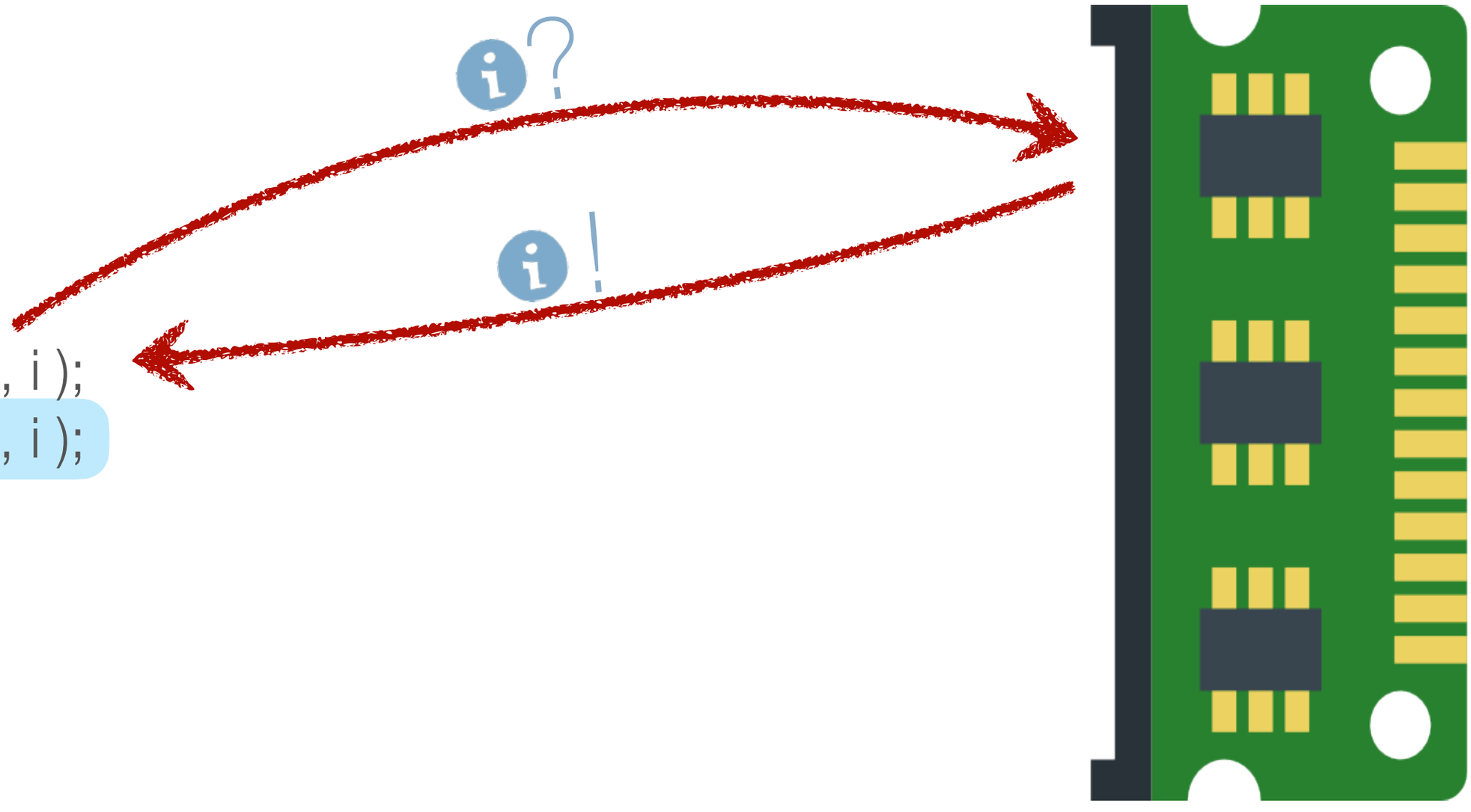
Accessing memory



Accessing memory



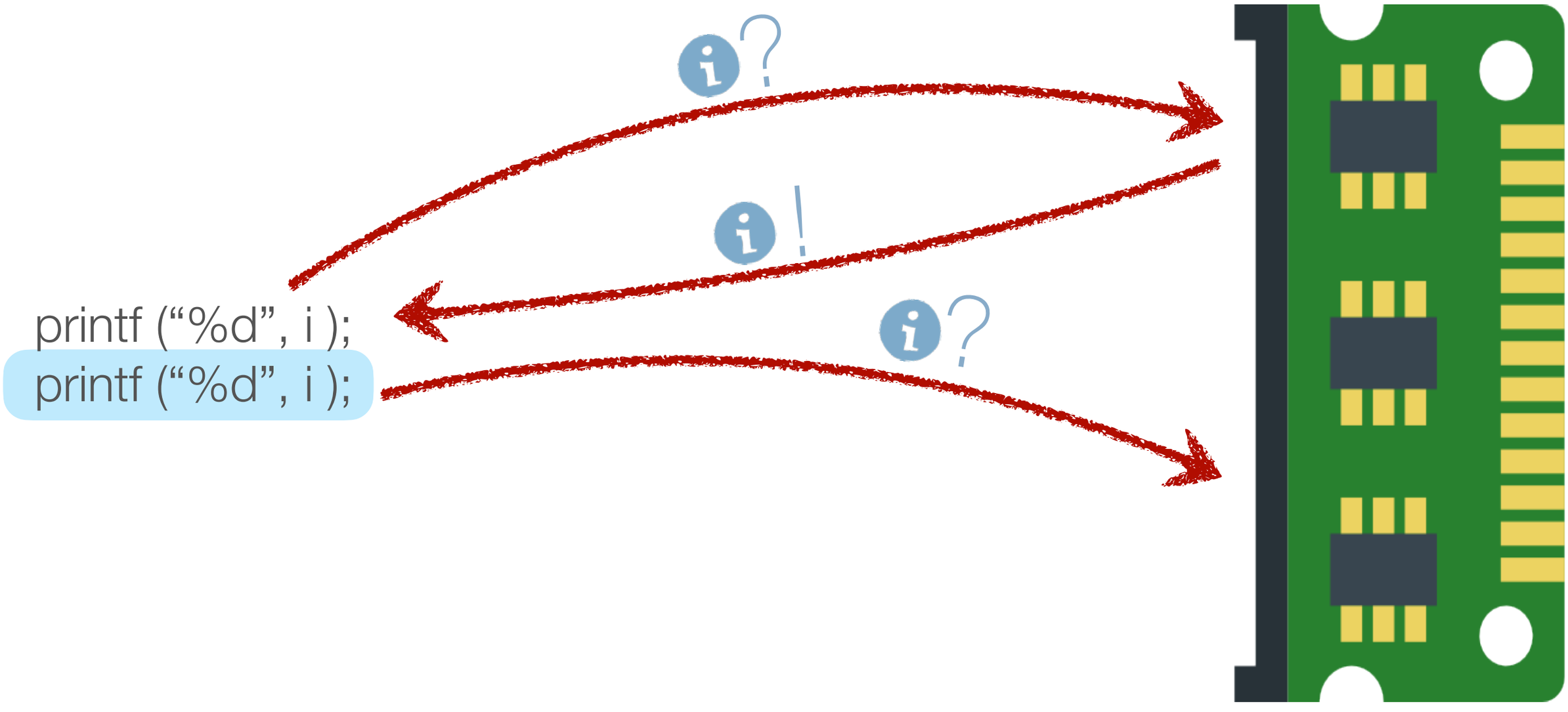
Accessing memory



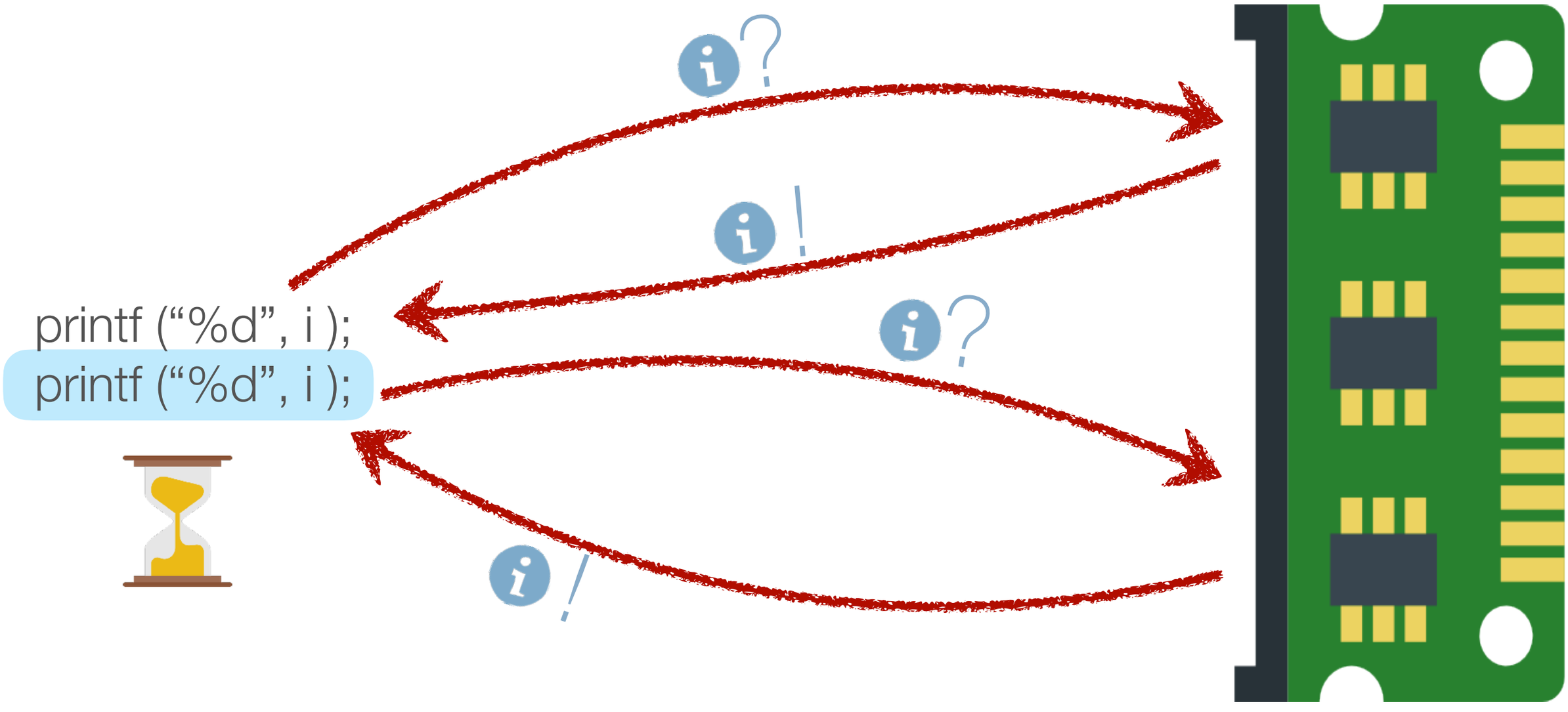
printf (“%d”, i);

printf (“%d”, i);

Accessing memory

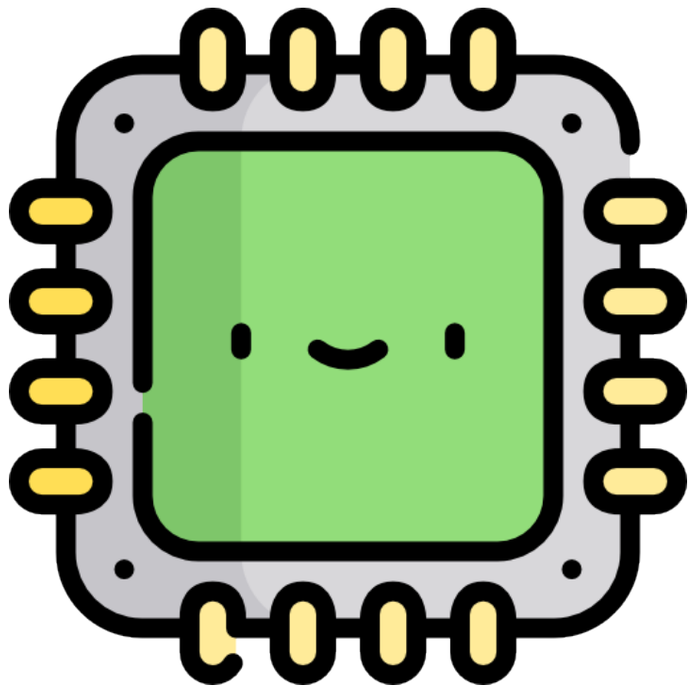


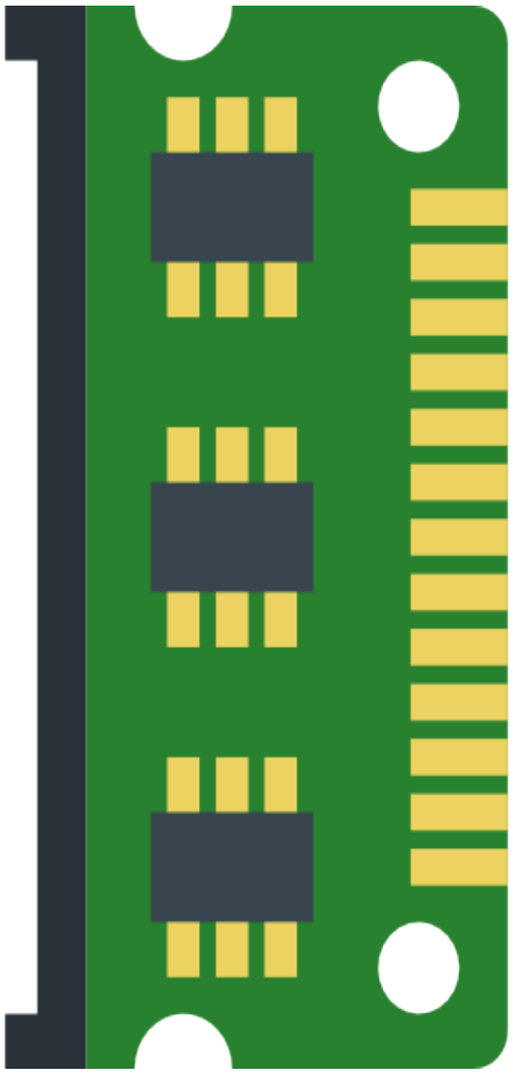
Accessing memory



Caches to the rescue!

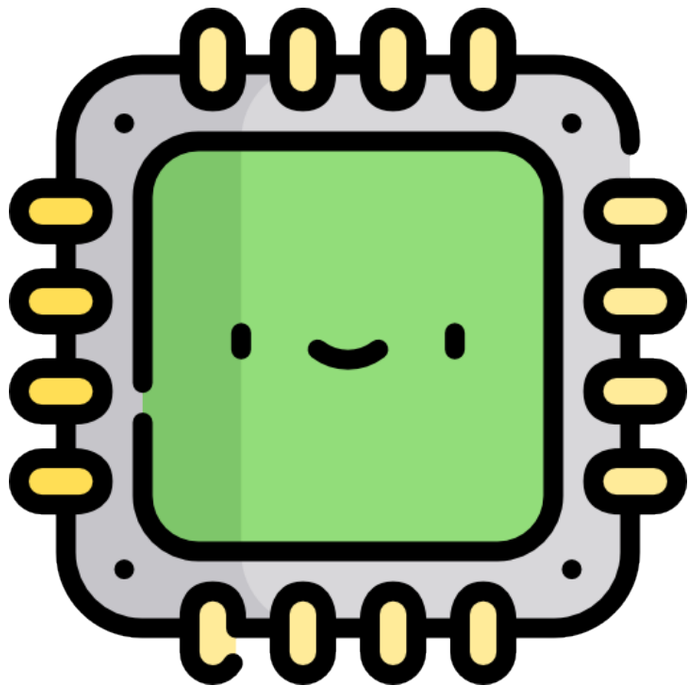
printf (“%d”, i);
printf (“%d”, i);

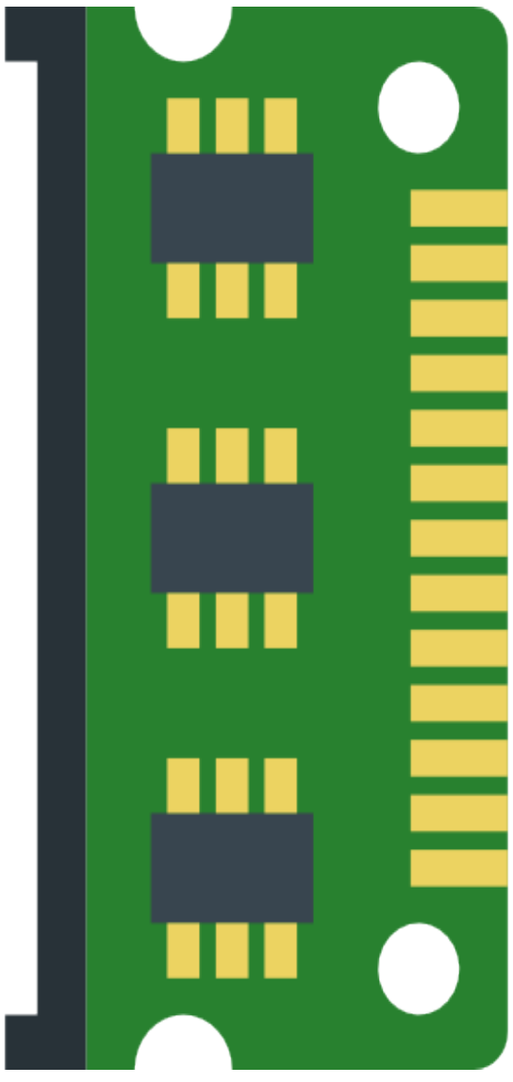




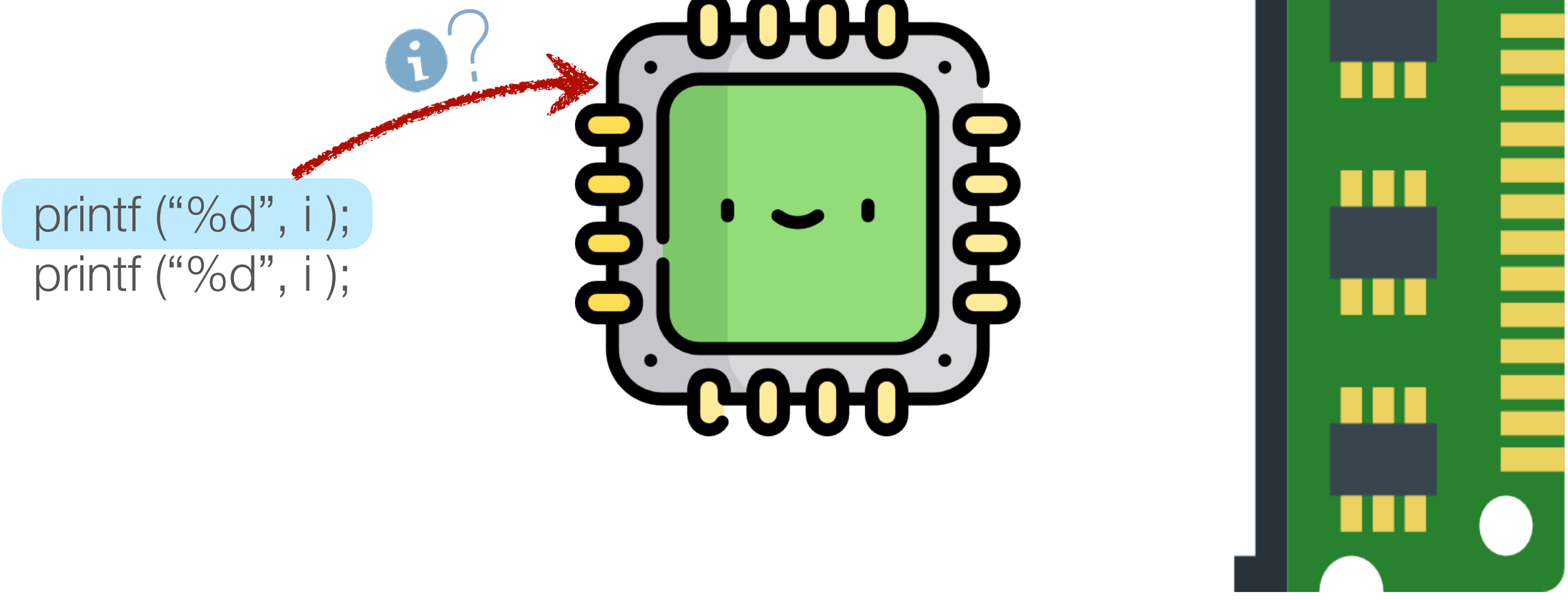
Caches to the rescue!

```
printf (“%d”, i );
printf (“%d”, i );
```



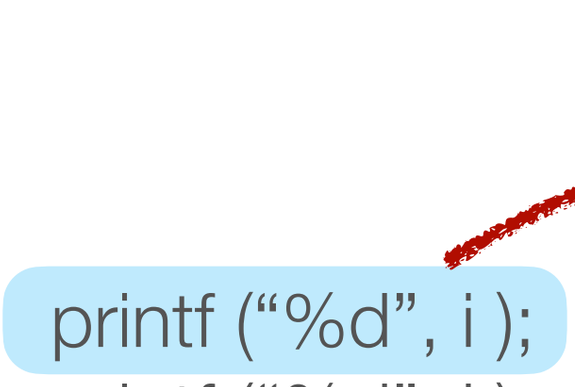


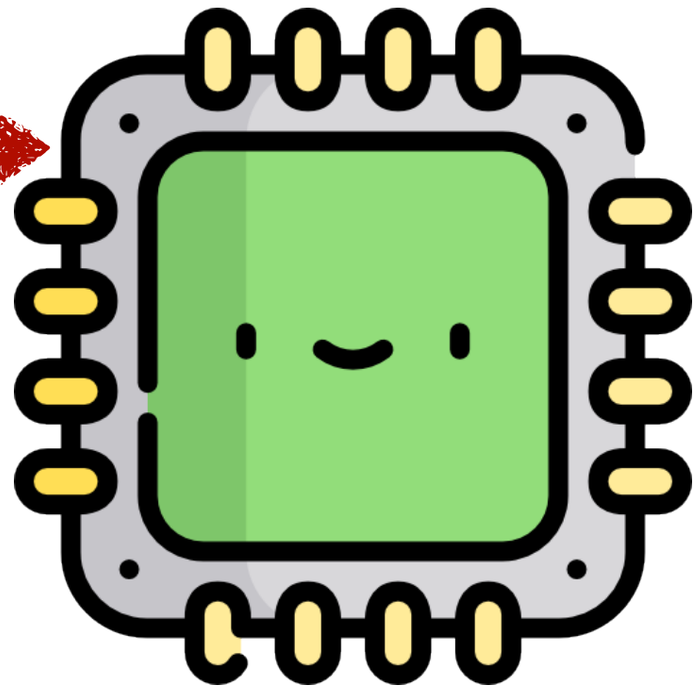
Caches to the rescue!

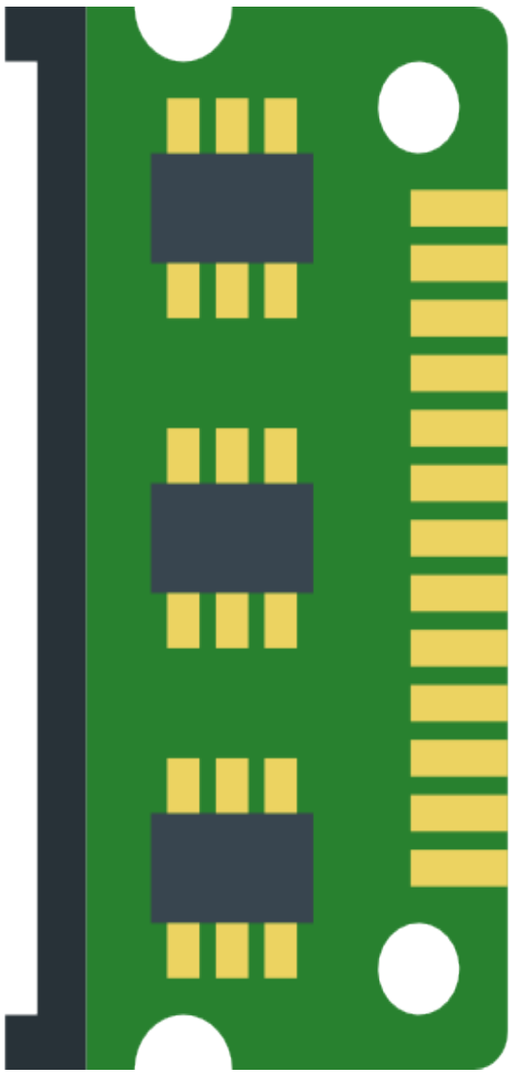


Caches to the rescue!

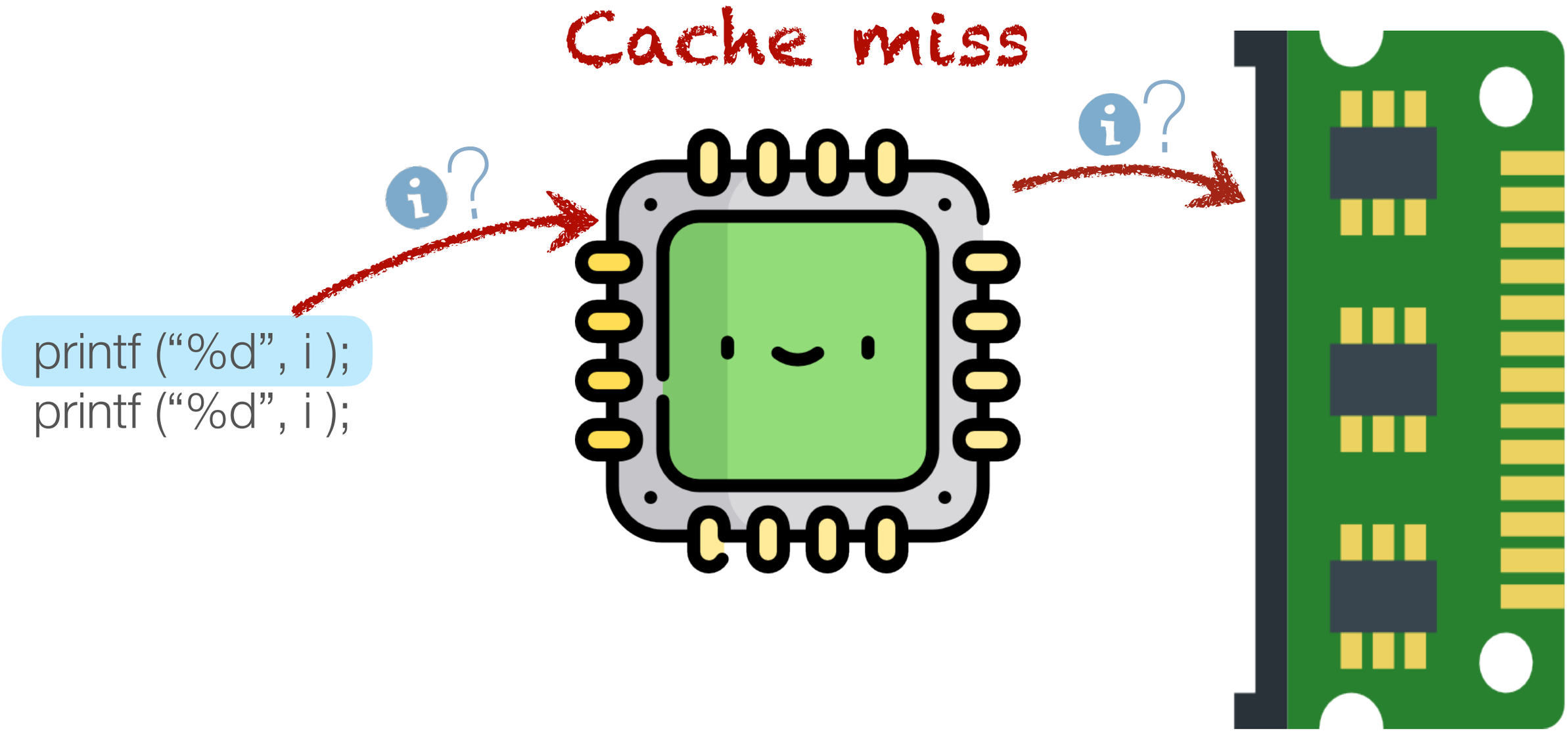
Cache miss



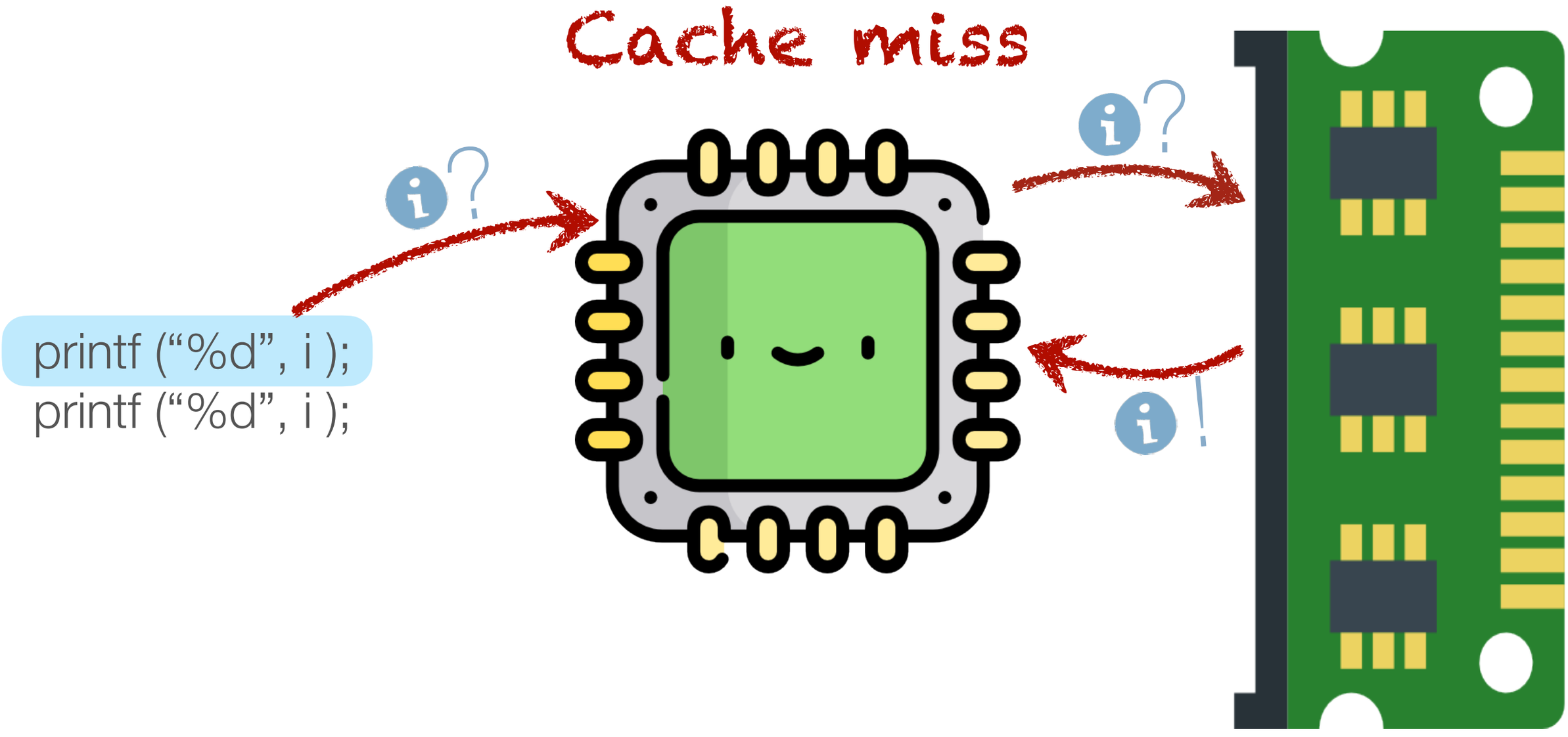




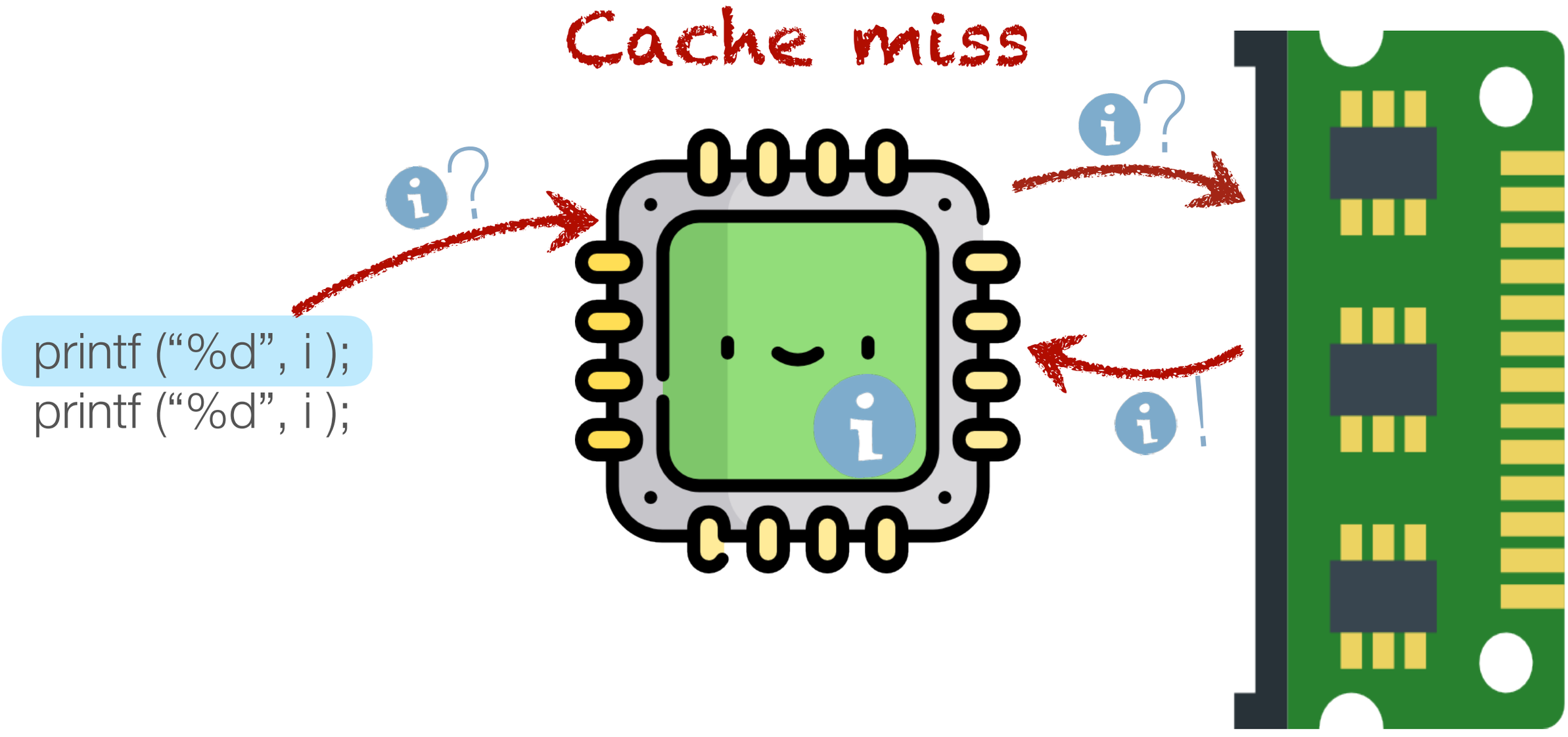
Caches to the rescue!



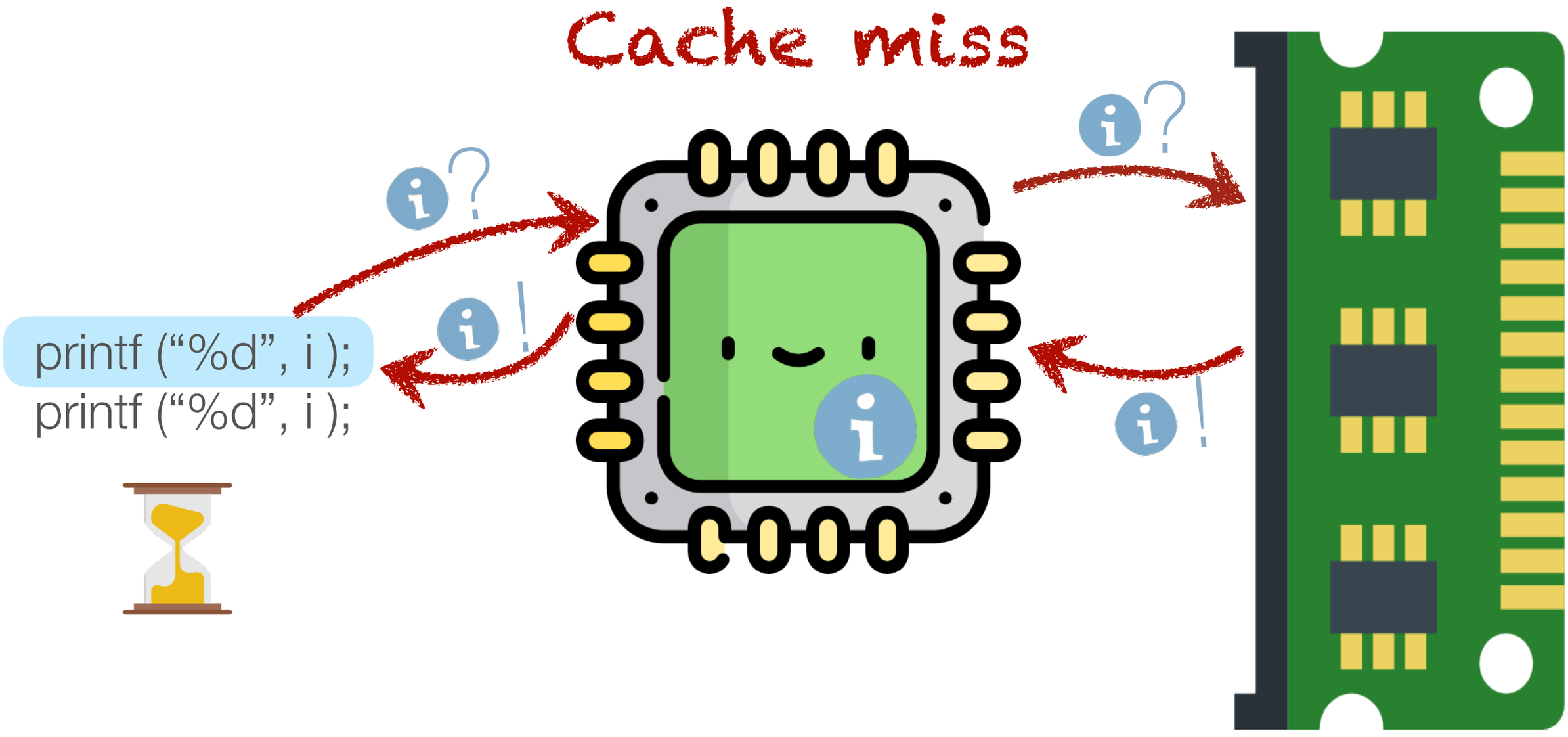
Caches to the rescue!



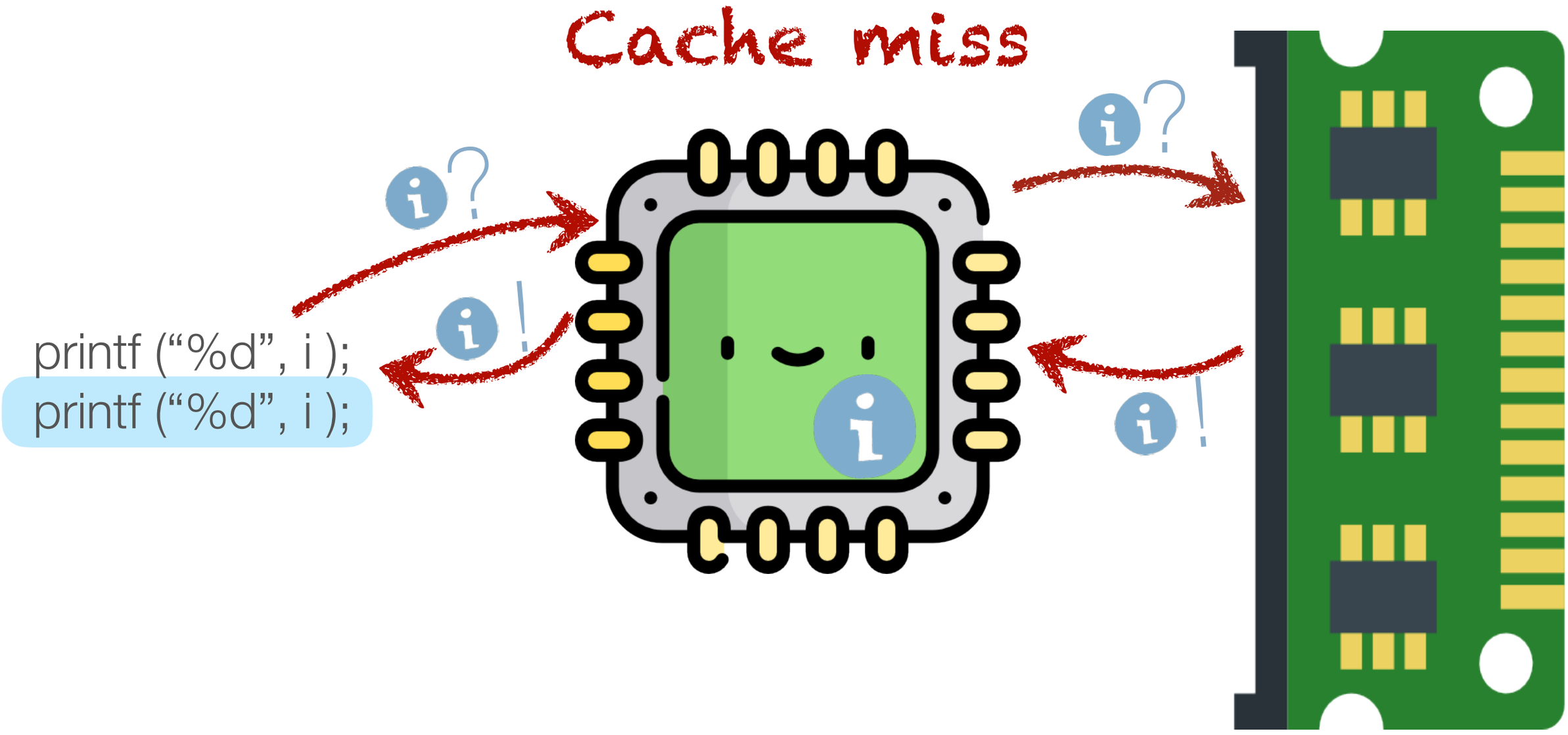
Caches to the rescue!



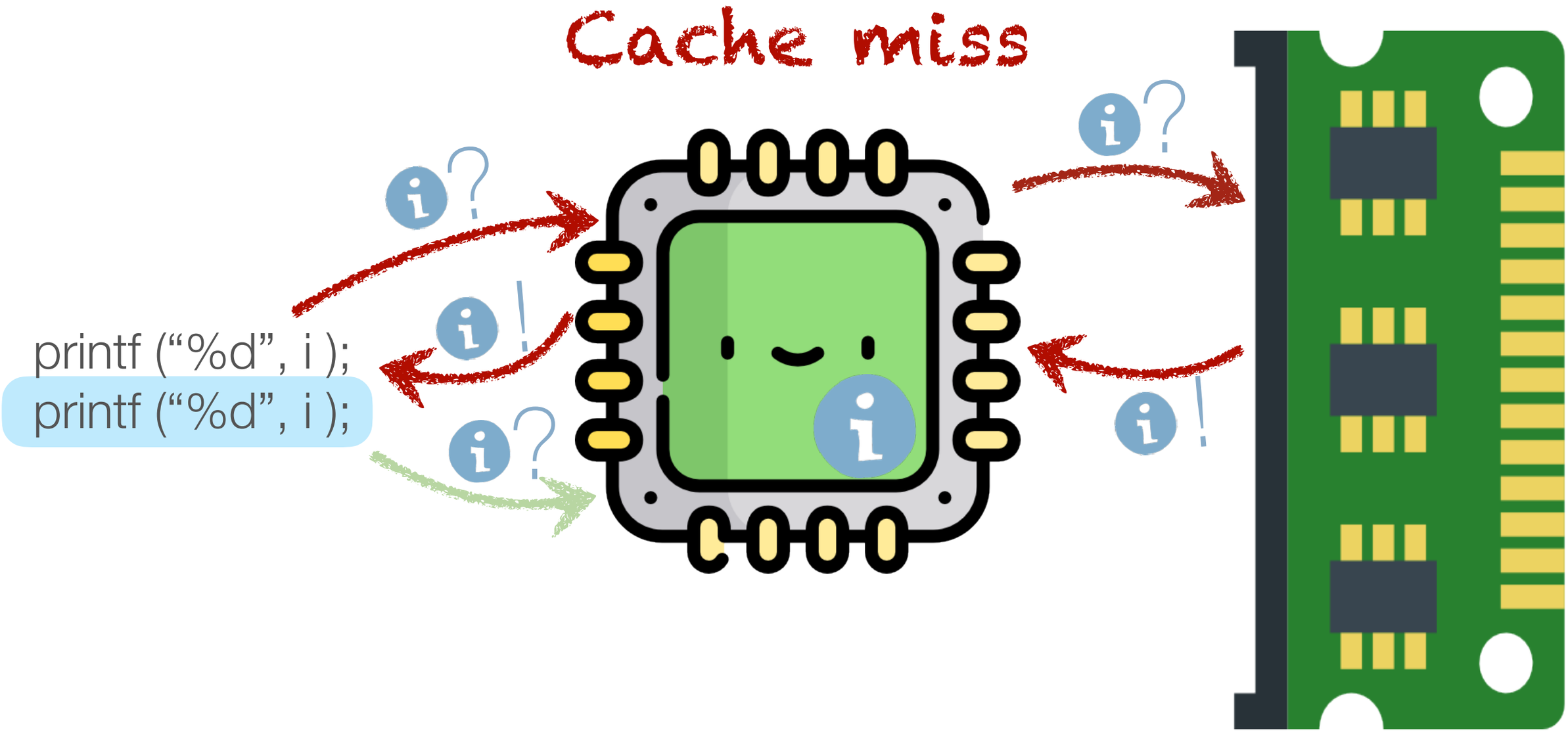
Caches to the rescue!



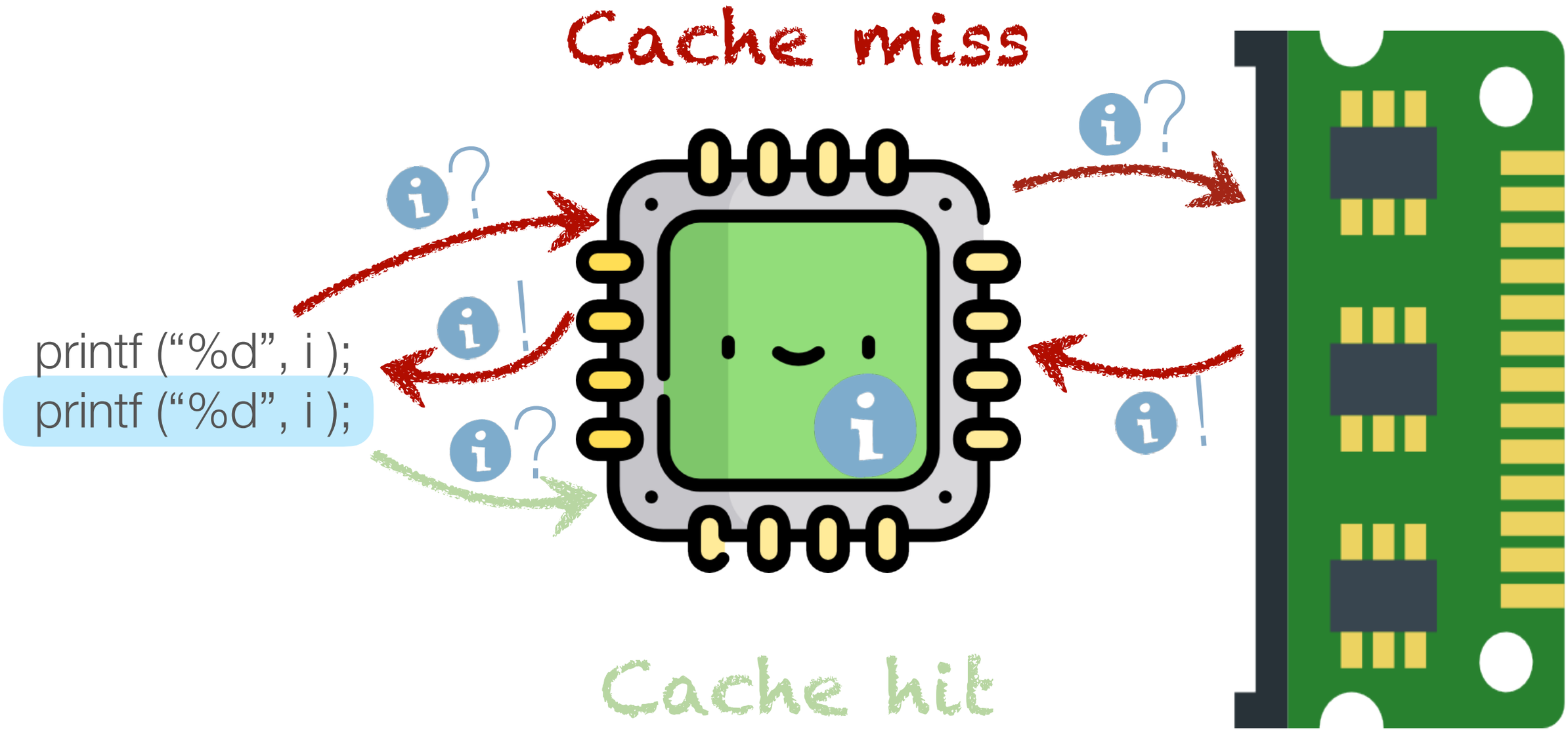
Caches to the rescue!



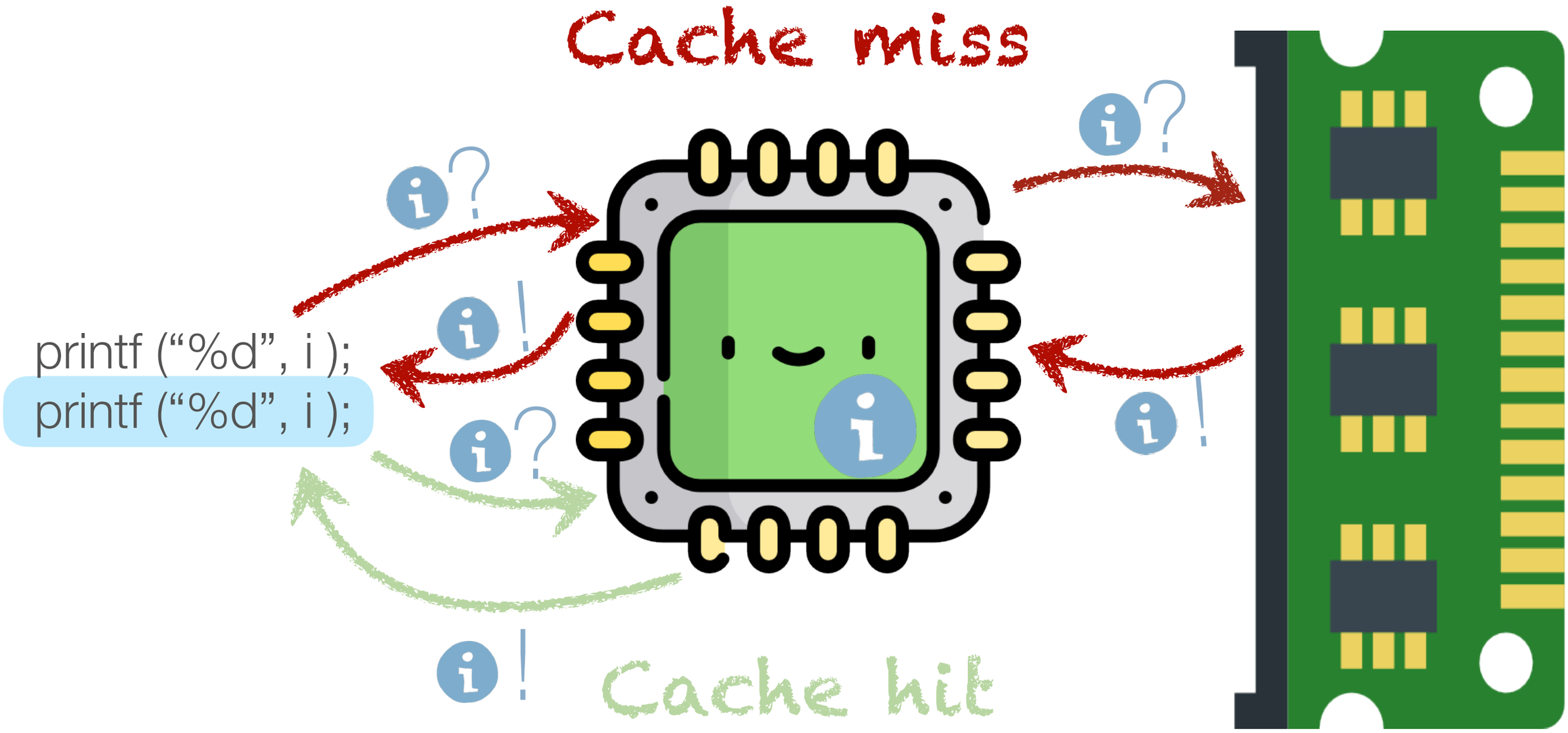
Caches to the rescue!



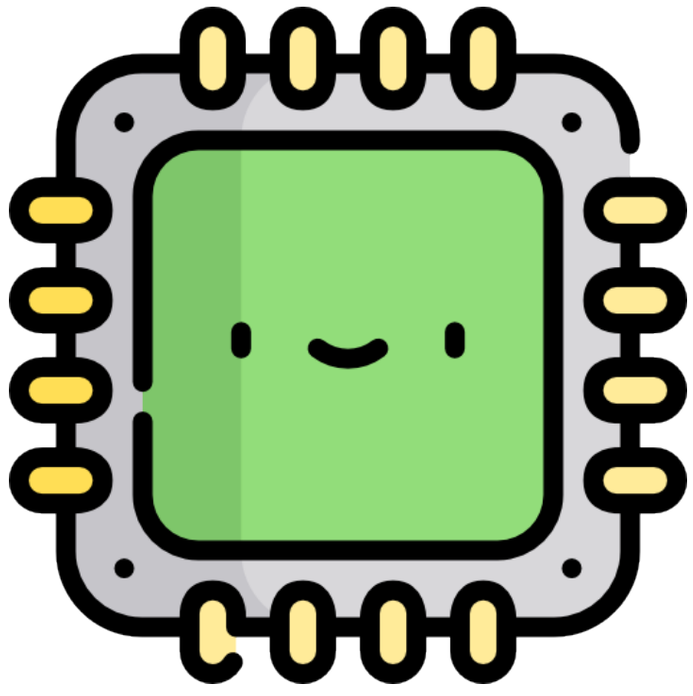
Caches to the rescue!



Caches to the rescue!

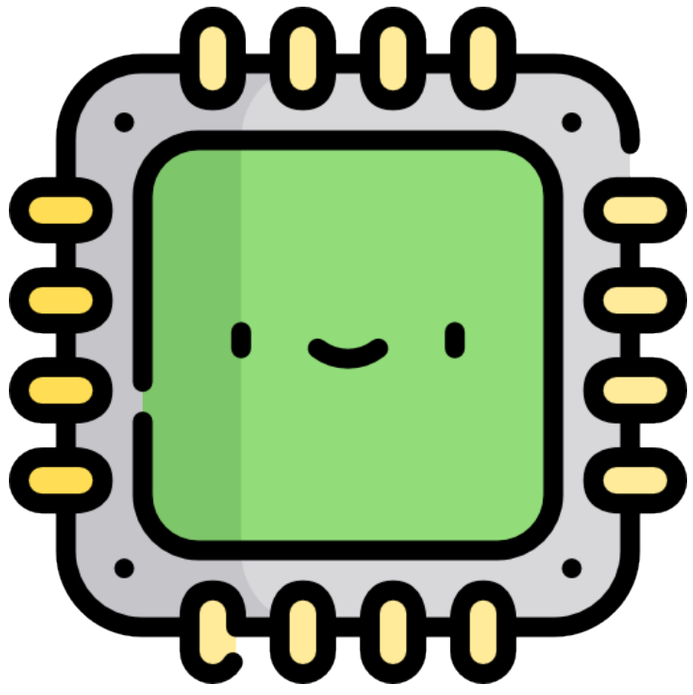


CPU Caches



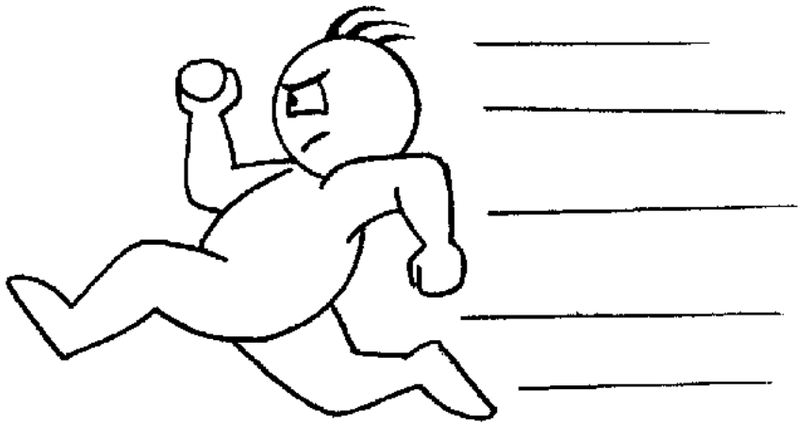
CPU Caches

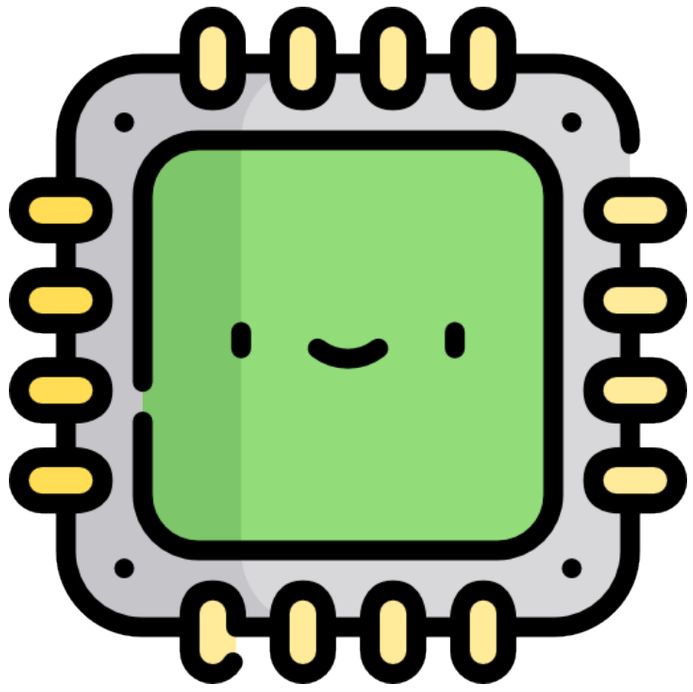
Advantages



CPU Caches

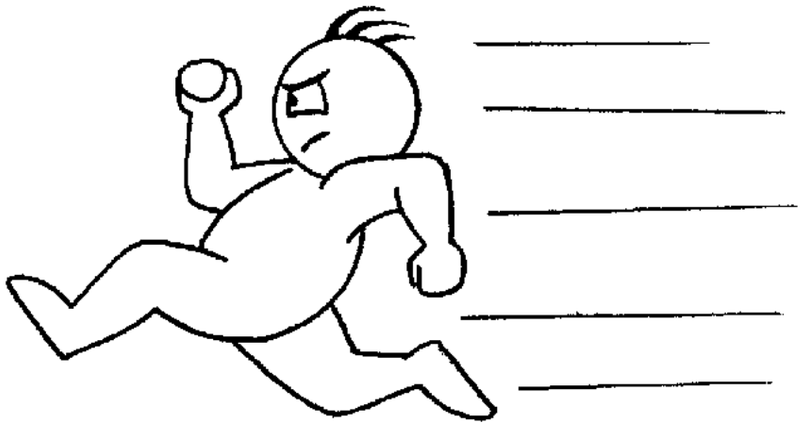
Advantages

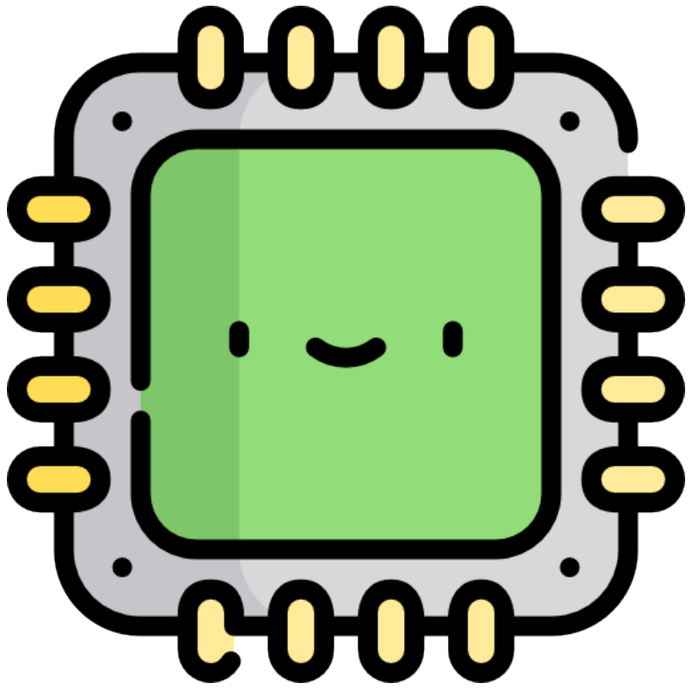




CPU Caches

Advantages

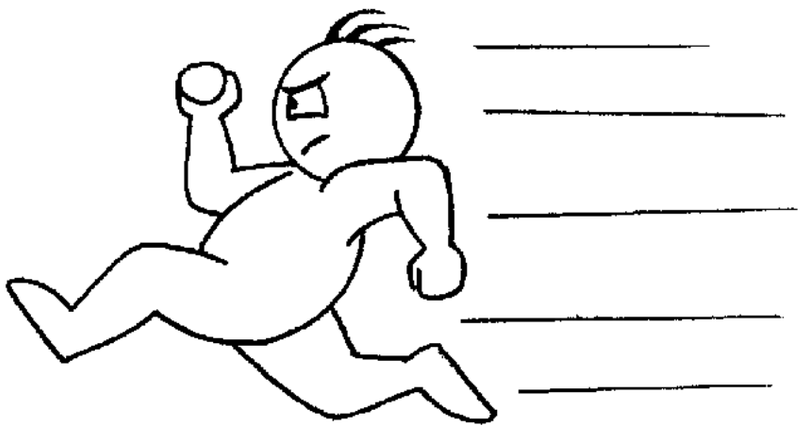


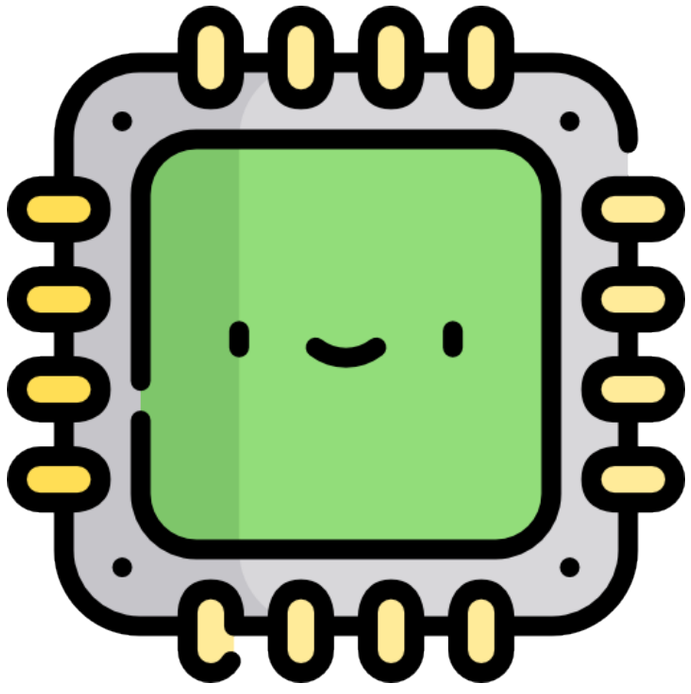


Drawbacks

CPU Caches

Advantages



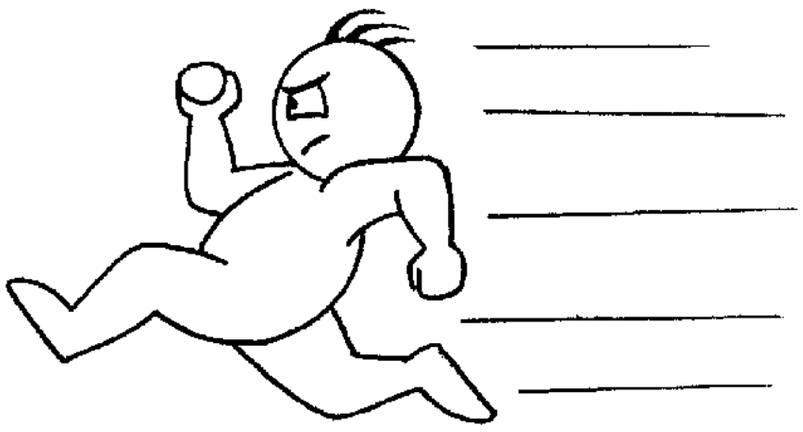


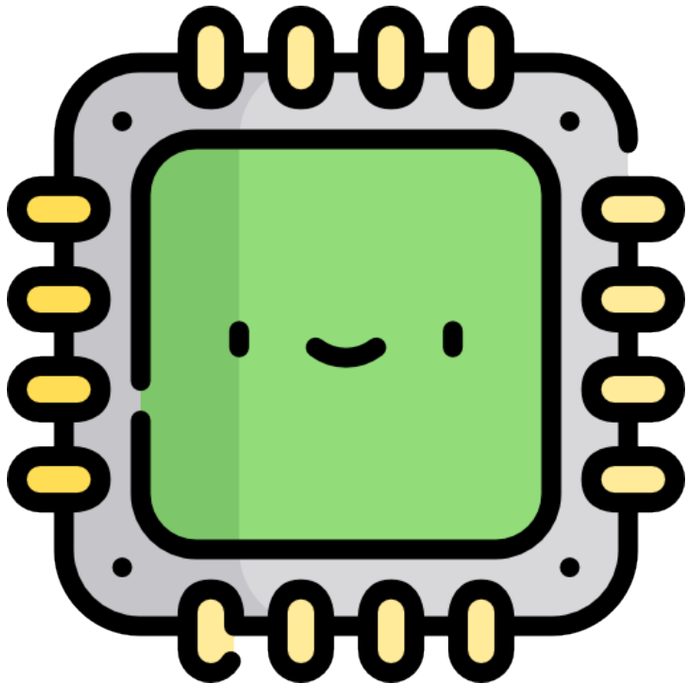
Drawbacks

- Cache hits and misses leak information

CPU Caches

Advantages



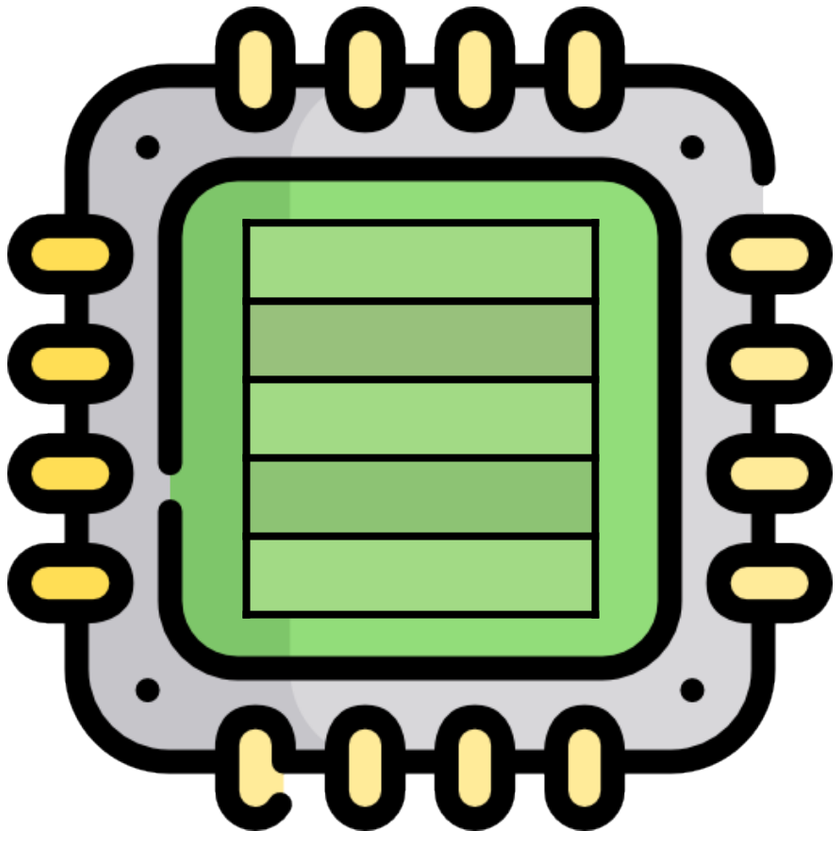


Drawbacks

- Cache hits and misses leak information

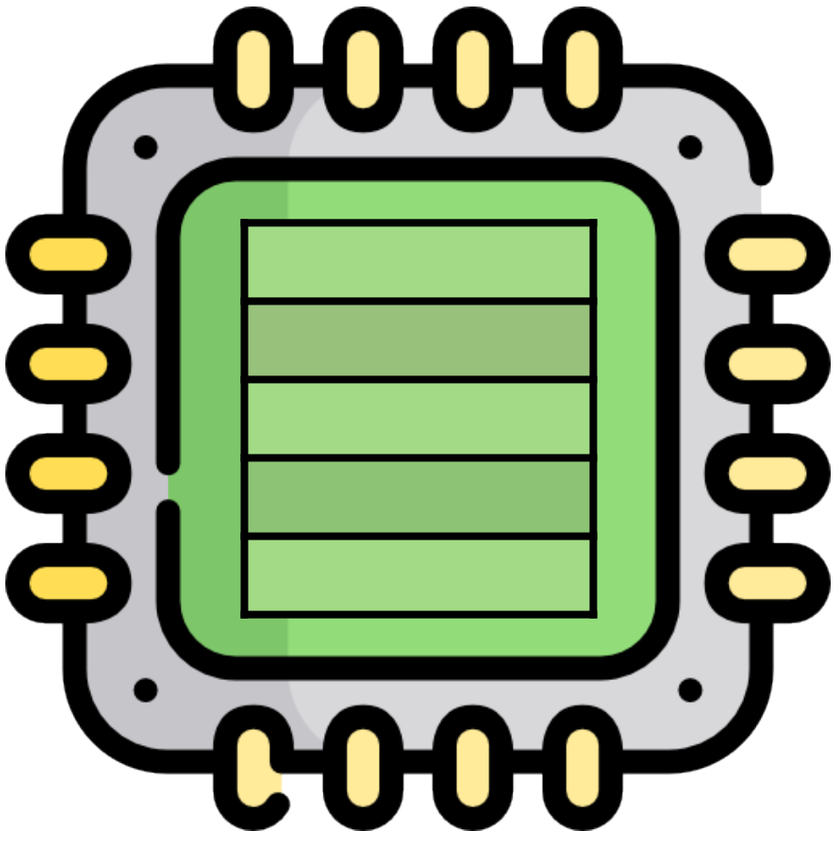
- Attacks
  - Prime+Probe
  - Flush+reload
  - …

Prime + Probe



Prime + Probe

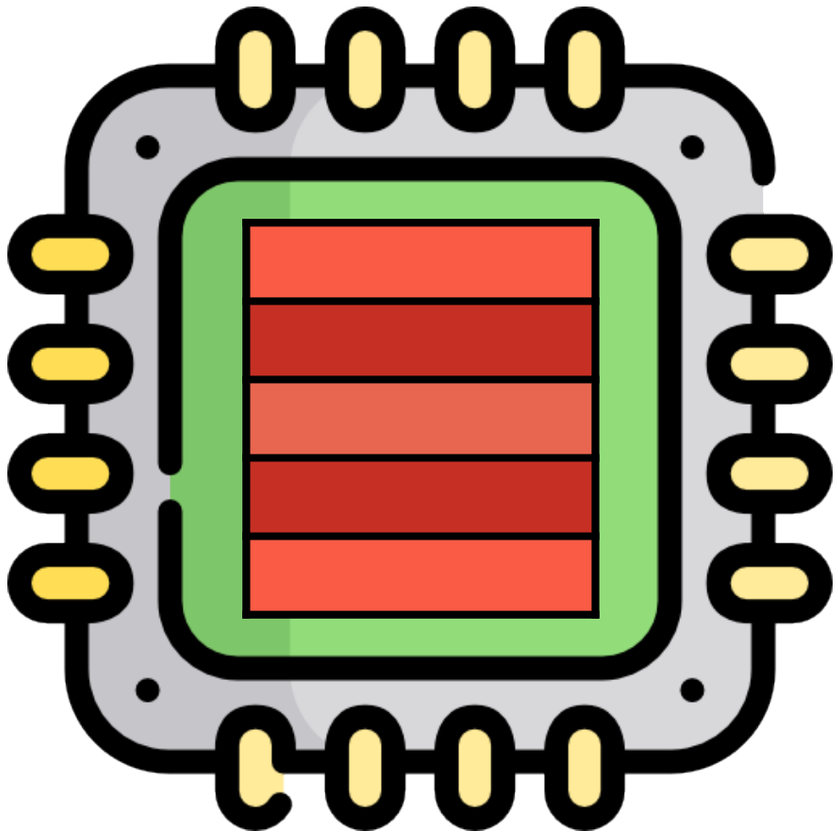
Prime



Prime + Probe

Prime

- Fill the cache with attacker's data

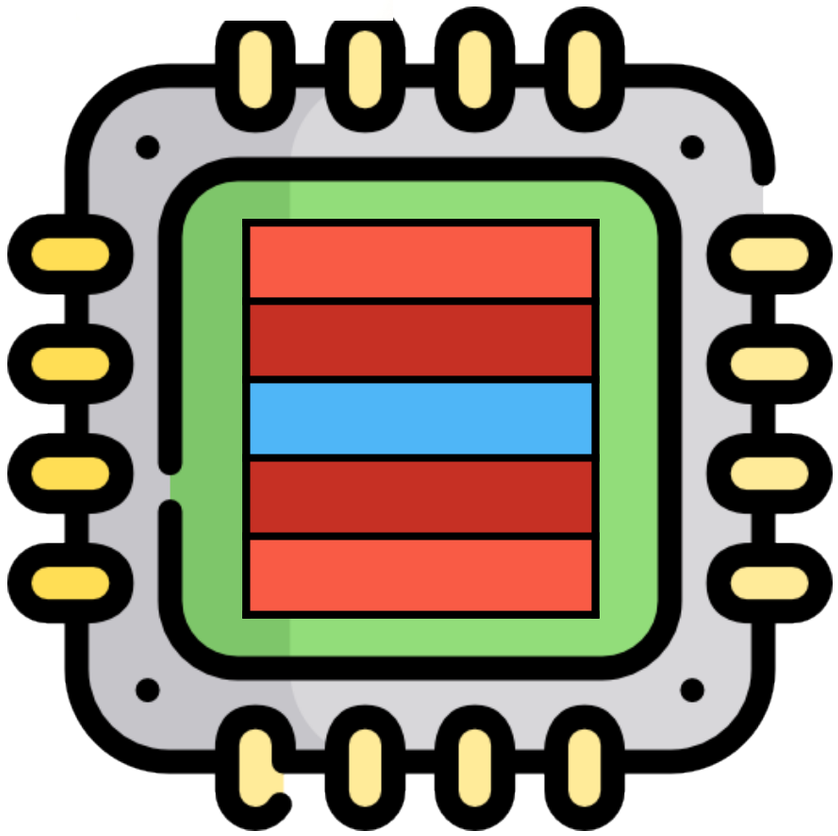


Prime + Probe

Prime

- Fill the cache with attacker's data

Victim executes!

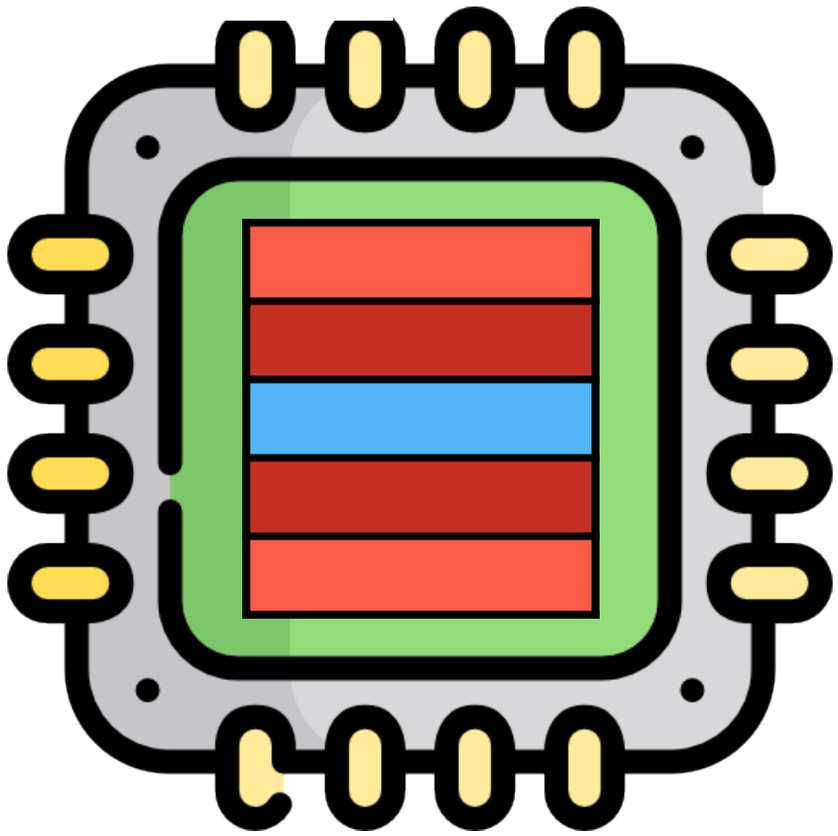


Prime + Probe

Prime

- Fill the cache with attacker's data

Victim executes!



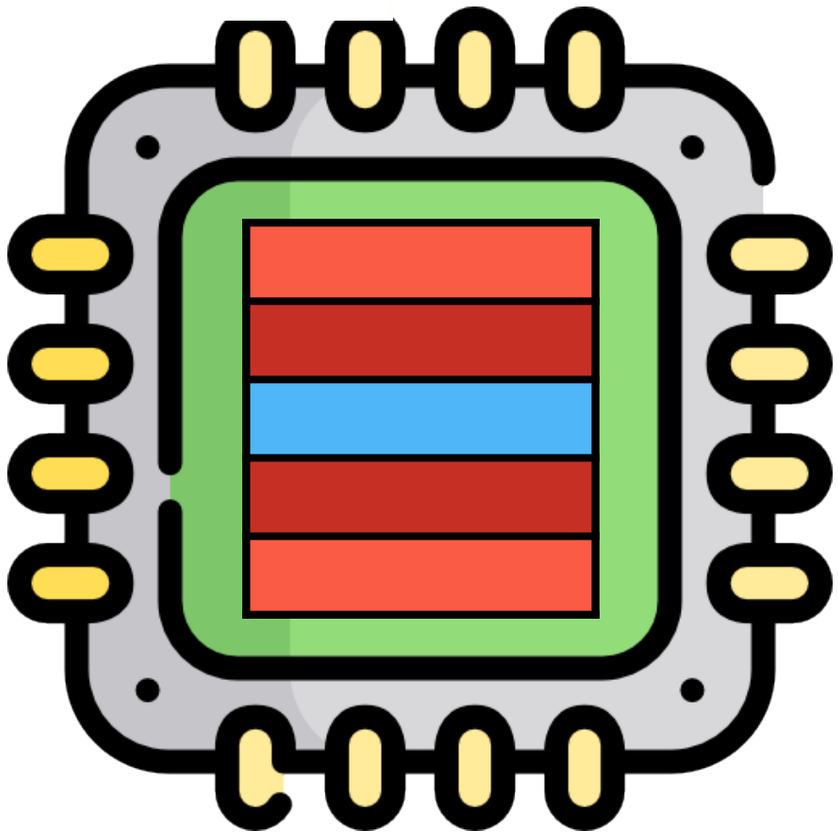
Probe

Prime + Probe

Prime

- Fill the cache with attacker's data

Victim executes!



Probe

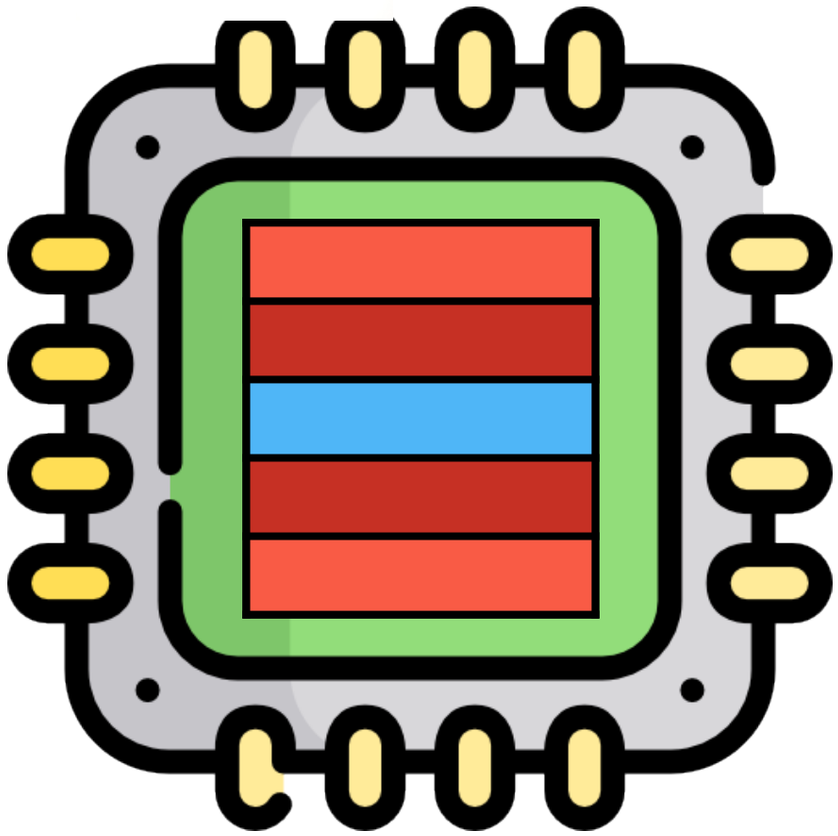
- Attacker accesses the data again

Prime + Probe

Prime

- Fill the cache with attacker's data

Victim executes!



Probe

- Attacker accesses the data again

- Hits and misses indicate what has been access by program

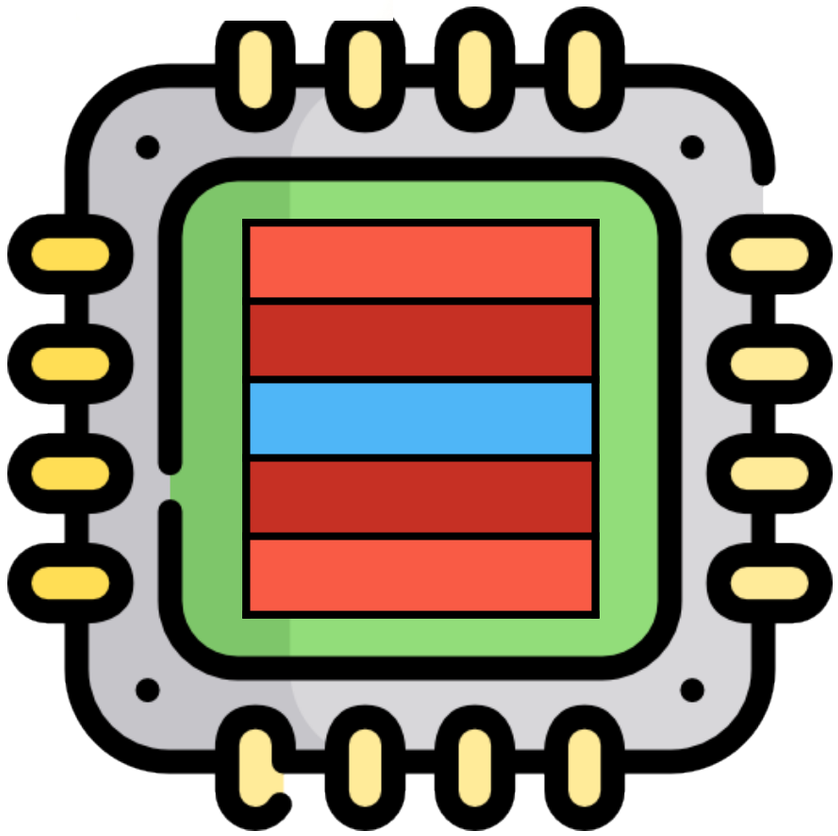
Prime + Probe

Prime

- Fill the cache with attacker's data

Eviction set needed!

Victim executes!



Probe

- Attacker accesses the data again

- Hits and misses indicate what has been access by program

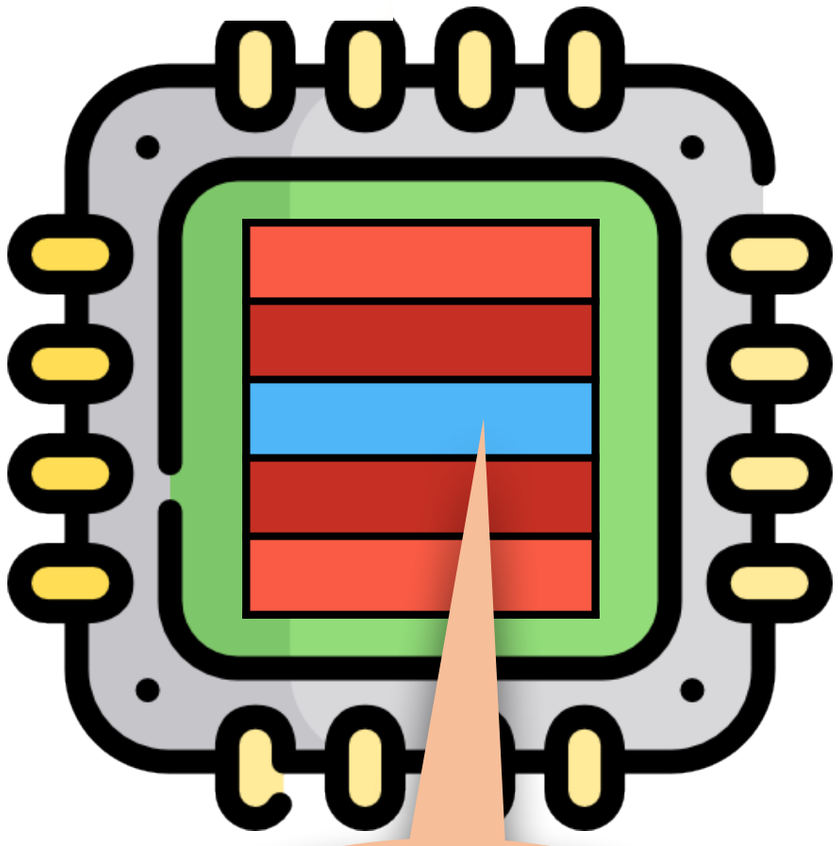
Prime + Probe

Prime

- Fill the cache with attacker's data

Eviction set needed!

Victim executes!



Cache set granularity

Probe

- Attacker accesses the data again

- Hits and misses indicate what has been access by program

Prime + Probe

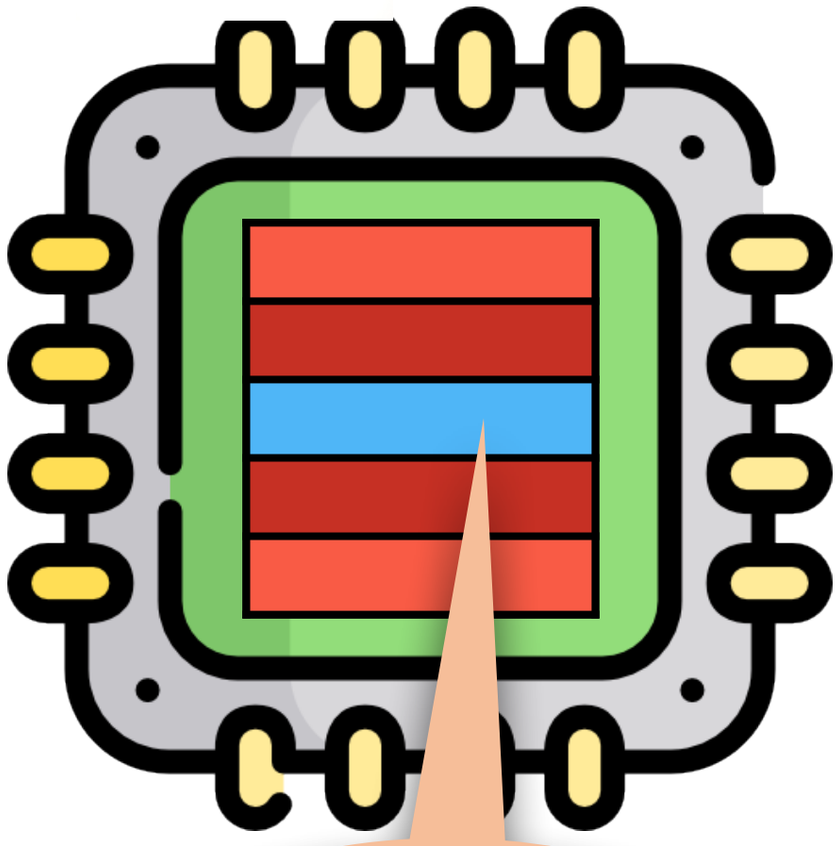
No privileges needed

Prime

- Fill the cache with attacker's data

Eviction set needed!

Victim executes!



Cache set granularity

Probe

- Attacker accesses the data again
- Hits and misses indicate what has been access by program

Instruction pipelining

Instruction pipelining

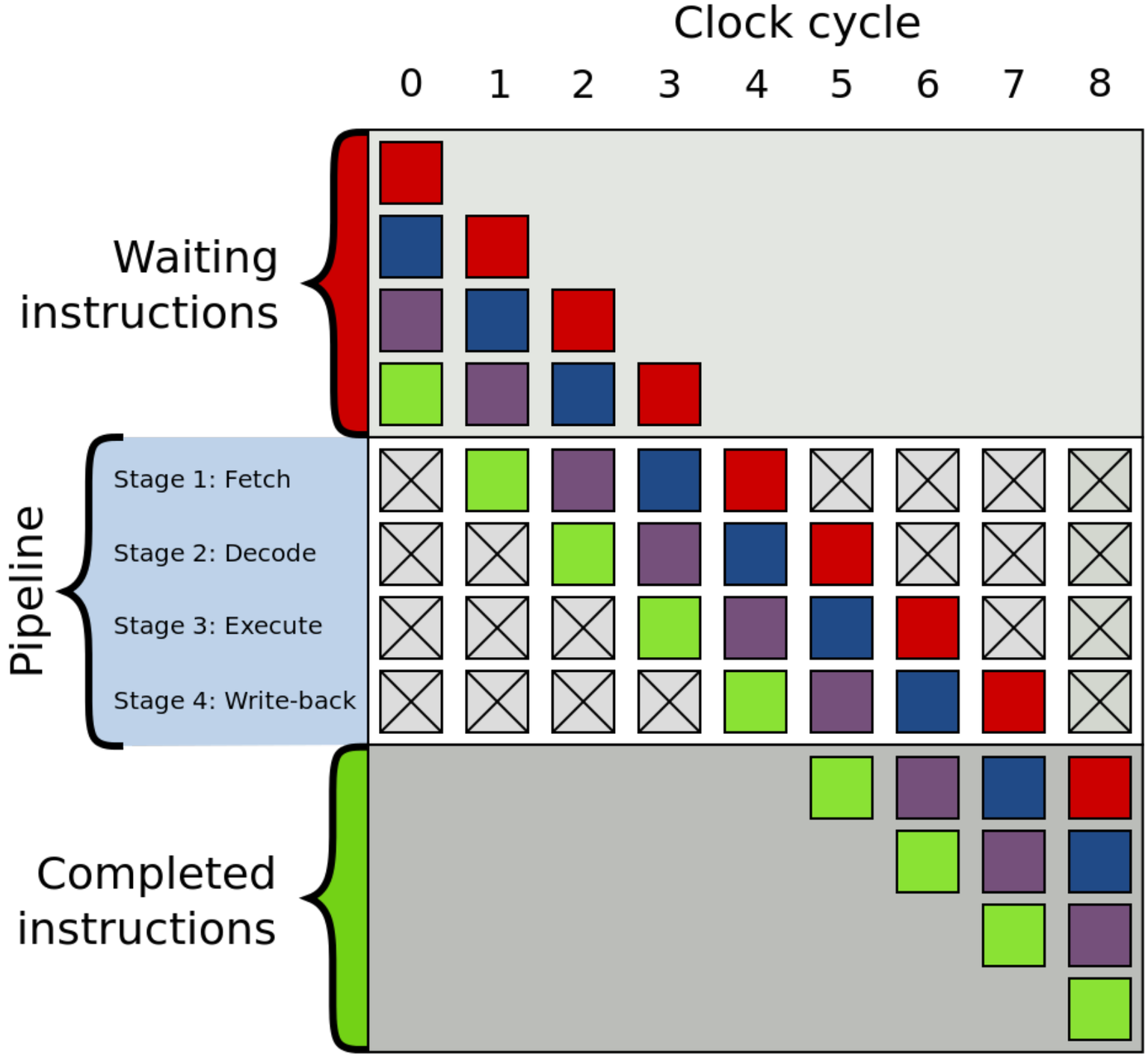


Figure from https://en.wikipedia.org/wiki/Instruction_pipelining

Instruction pipelining

- Instruction-level parallelism on single processors

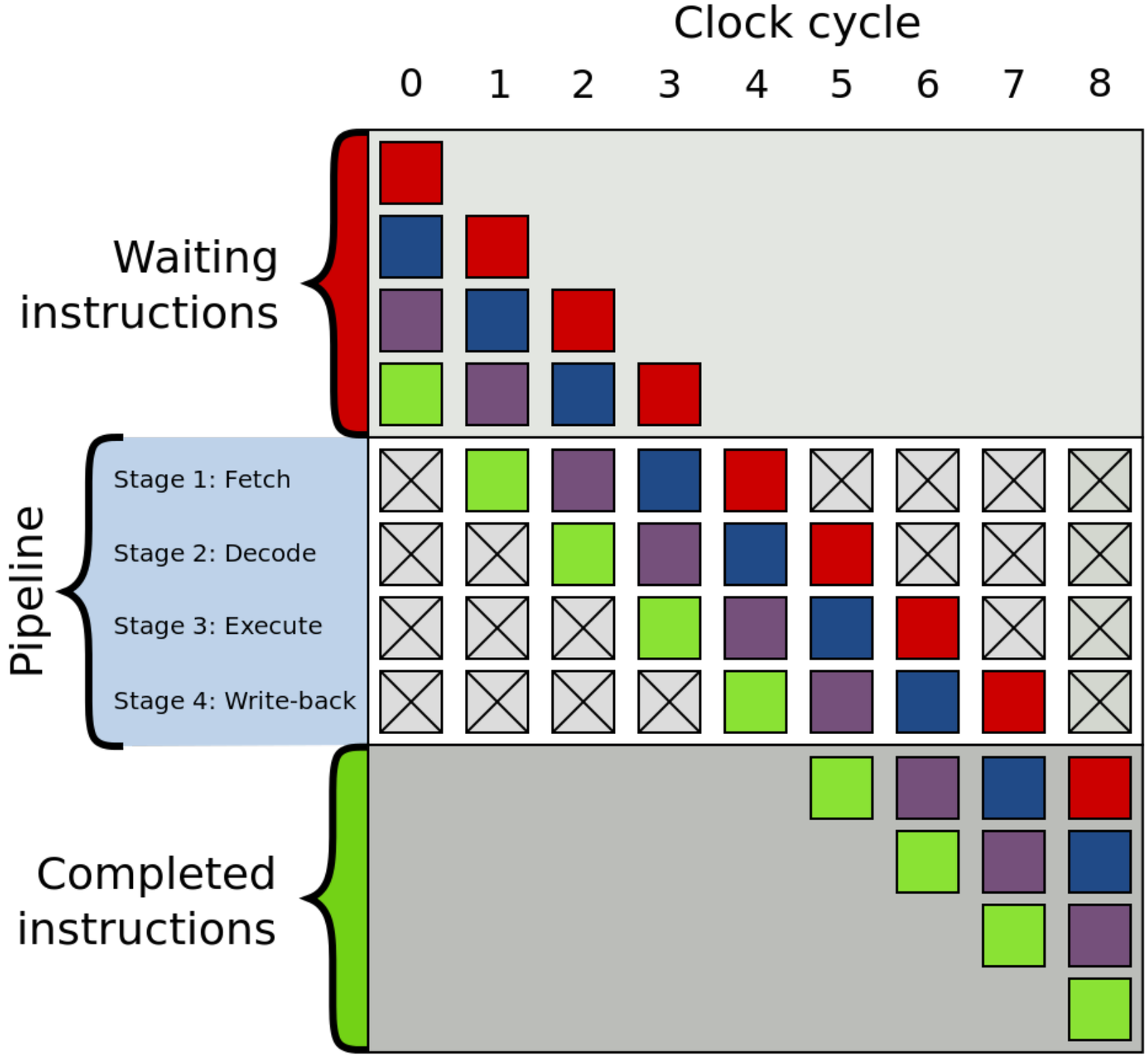


Figure from https://en.wikipedia.org/wiki/Instruction_pipelining

Instruction pipelining

- Instruction-level parallelism on single processors
- Each instruction is executed in steps

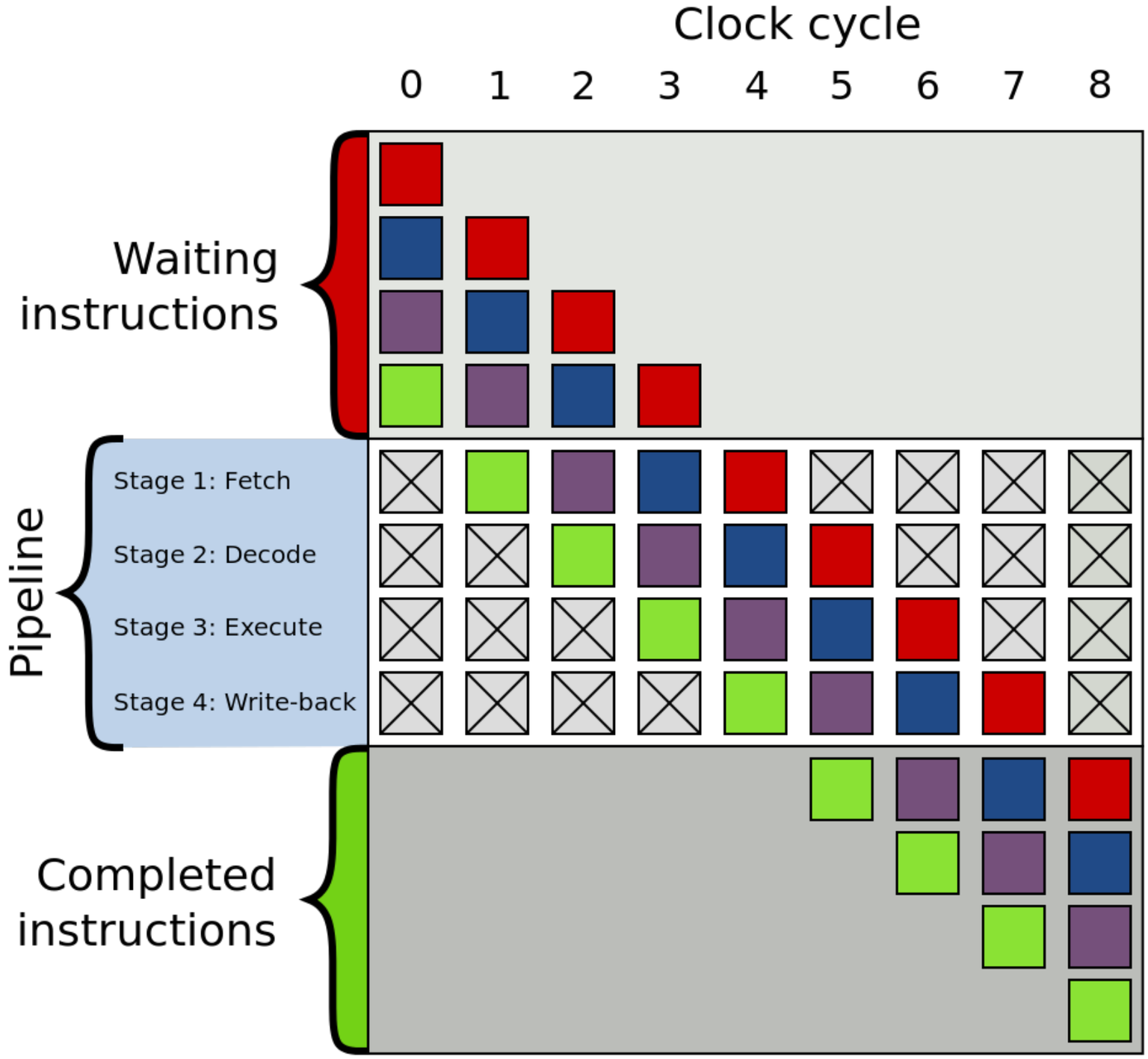


Figure from https://en.wikipedia.org/wiki/Instruction_pipelining

Instruction pipelining

- Instruction-level parallelism on single processors

- Each instruction is executed in steps

- Execute multiple instructions at the same time (in different steps)

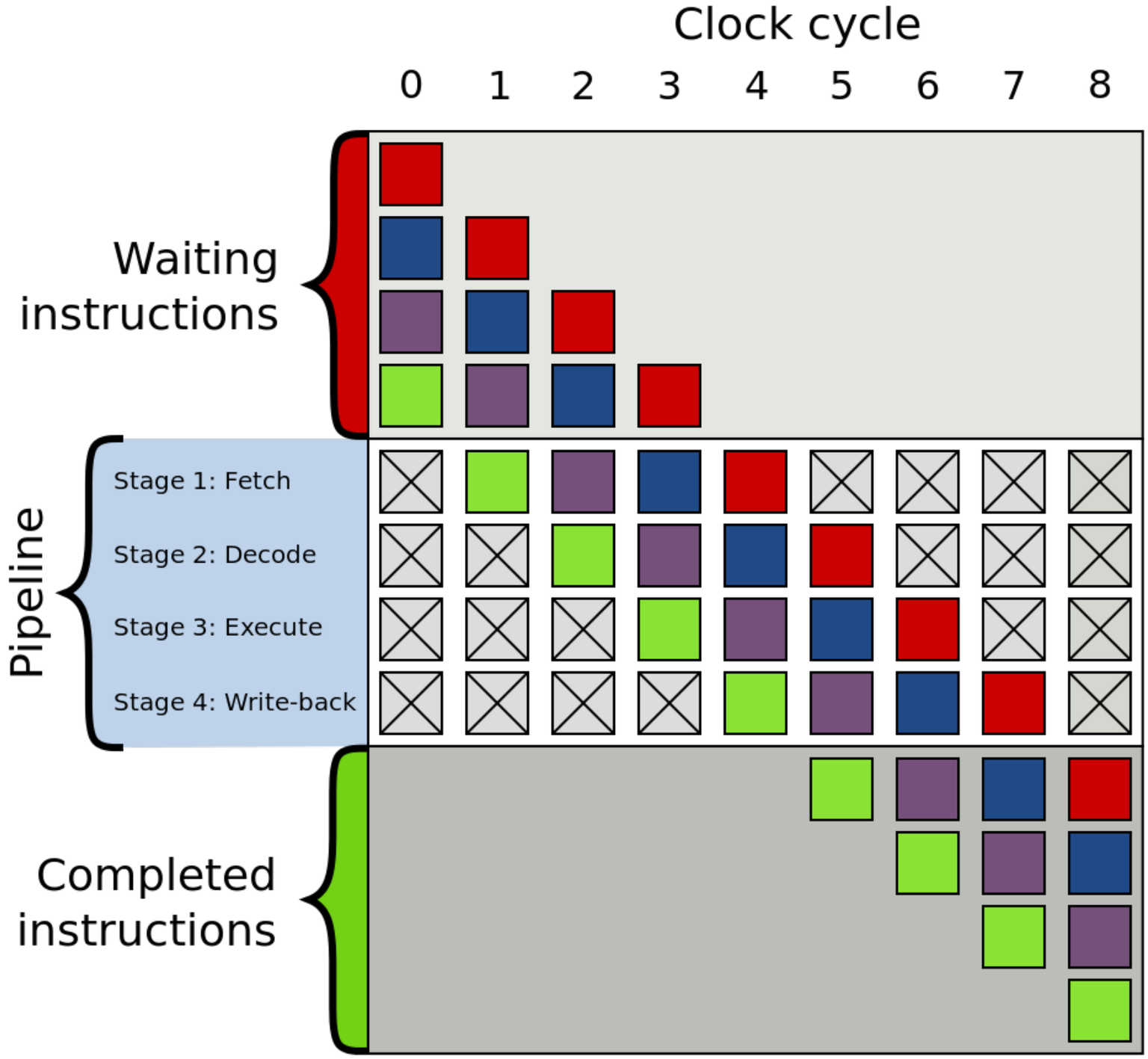


Figure from https://en.wikipedia.org/wiki/Instruction_pipelining

Instruction pipelining

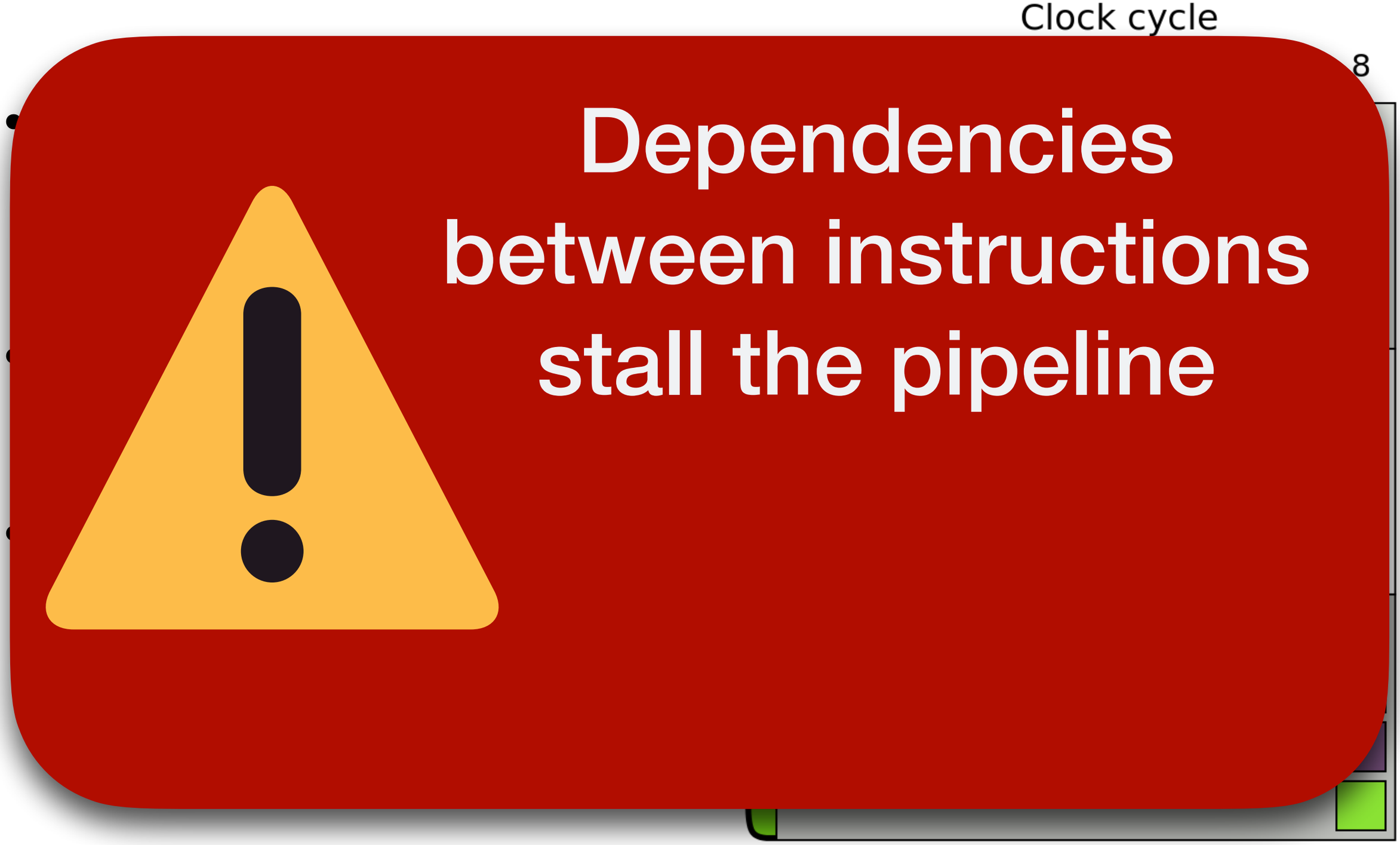


Figure from https://en.wikipedia.org/wiki/Instruction_pipelining

Instruction pipelining

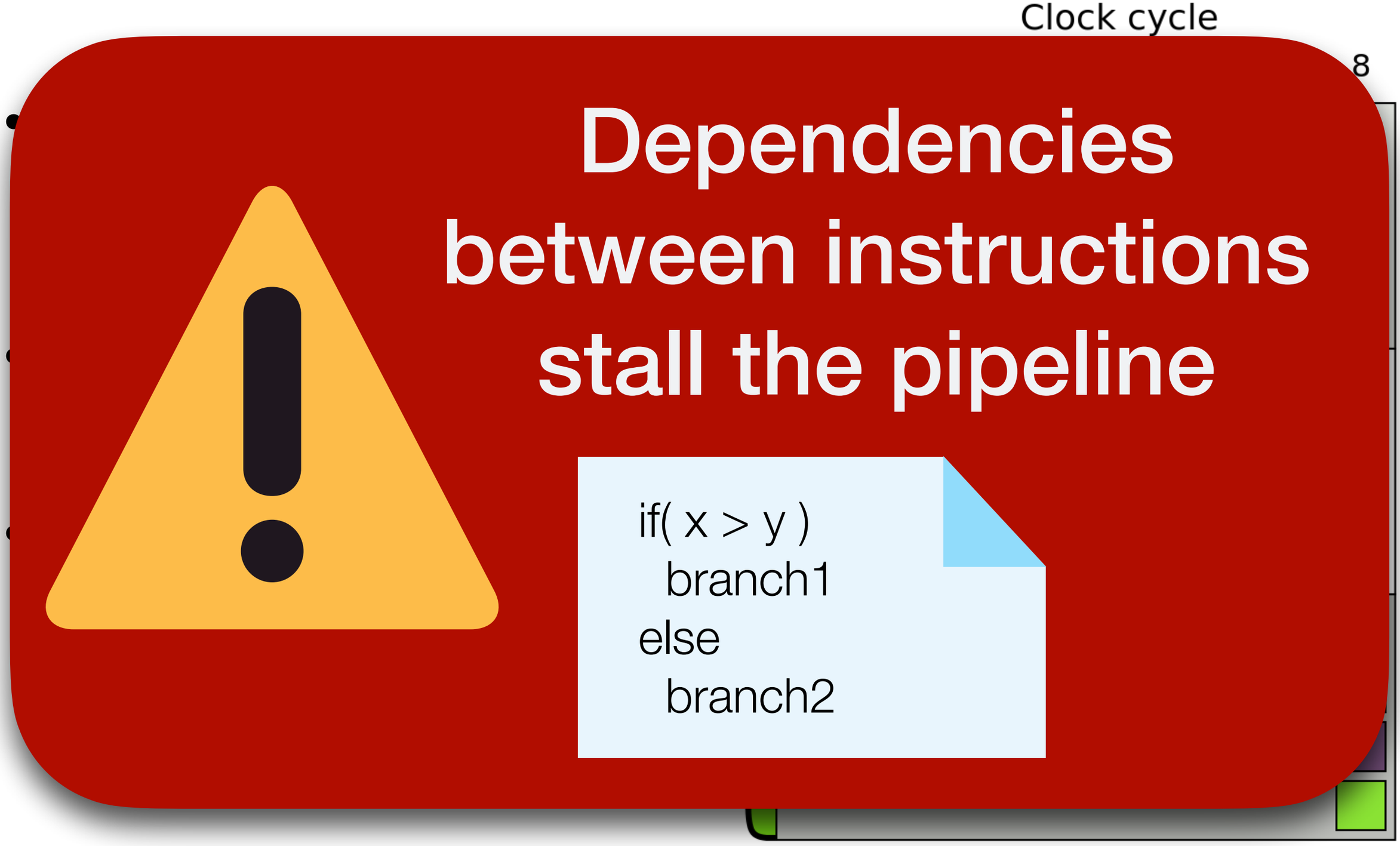


Figure from https://en.wikipedia.org/wiki/Instruction_pipelining

Instruction pipelining

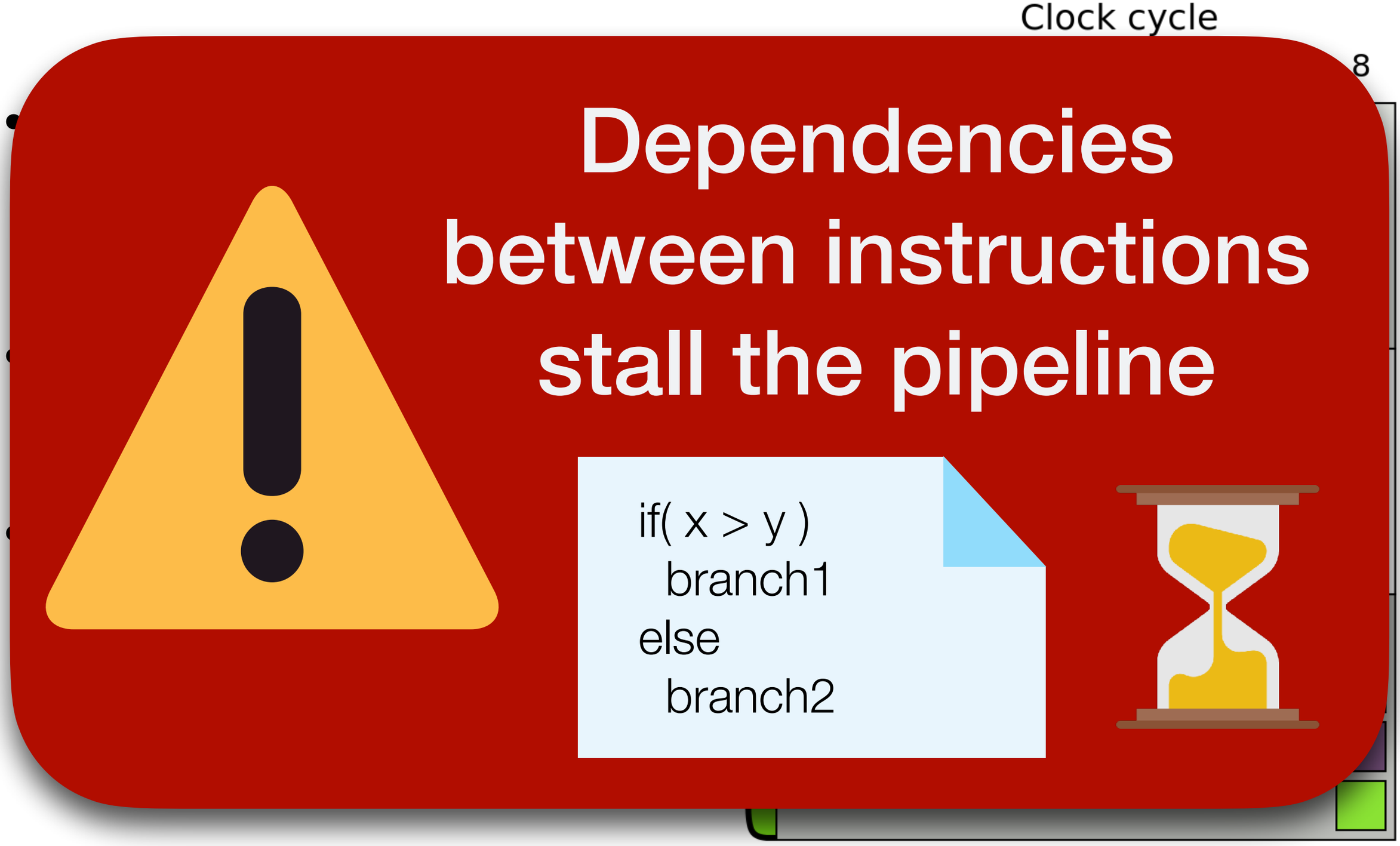


Figure from https://en.wikipedia.org/wiki/Instruction_pipelining

Instruction pipelining

Clock cycle

Figure from https://en.wikipedia.org/wiki/Instruction_pipelining

Speculative execution

Speculative execution

Speculative execution

Speculative execution

Speculative execution

Speculative execution

- Optimization technique

Speculative execution

- Optimization technique
- Instead of waiting for operations to complete, speculate on their results (i.e., guess) and continue the execution

Speculative execution

- Optimization technique
- Instead of waiting for operations to complete, speculate on their results (i.e., guess) and continue the execution
- When the results are available, check if the guess was correct. If not, abort the speculative execution and discard changes

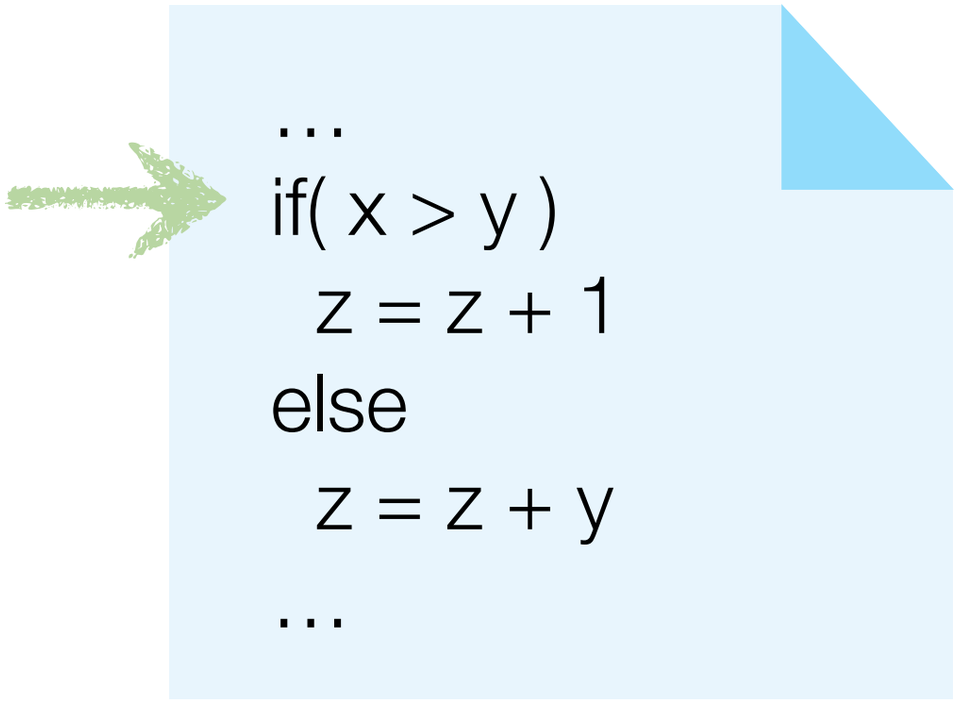
Speculative execution

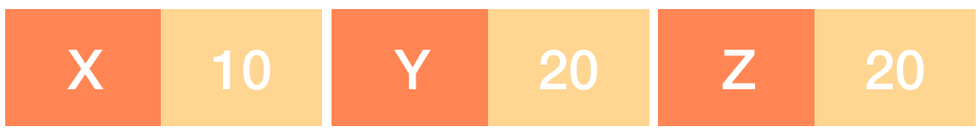
Branching speculatively

```
...
if( x > y )
  z = z + 1
else
  z = z + y
...
```

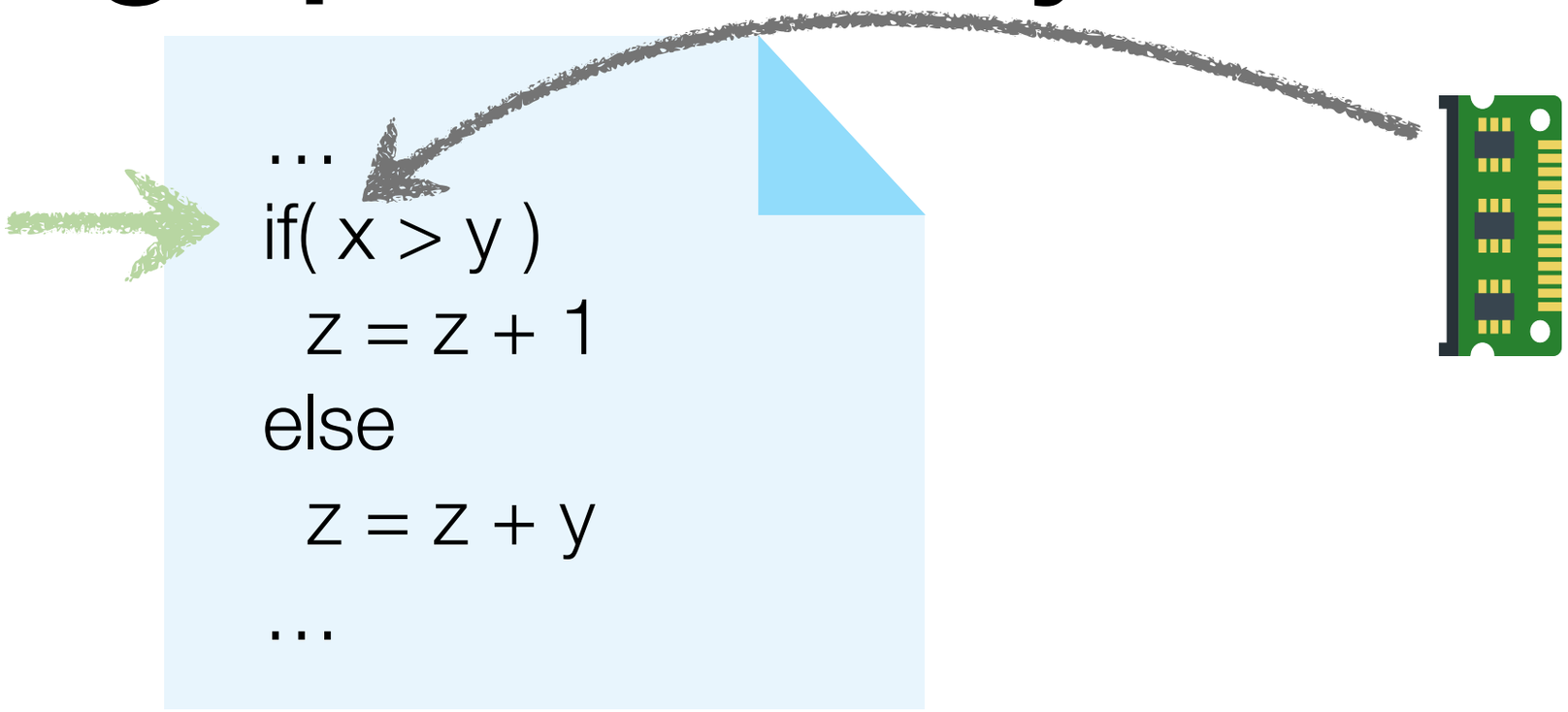
X	10	Y	20	Z	20

Branching speculatively



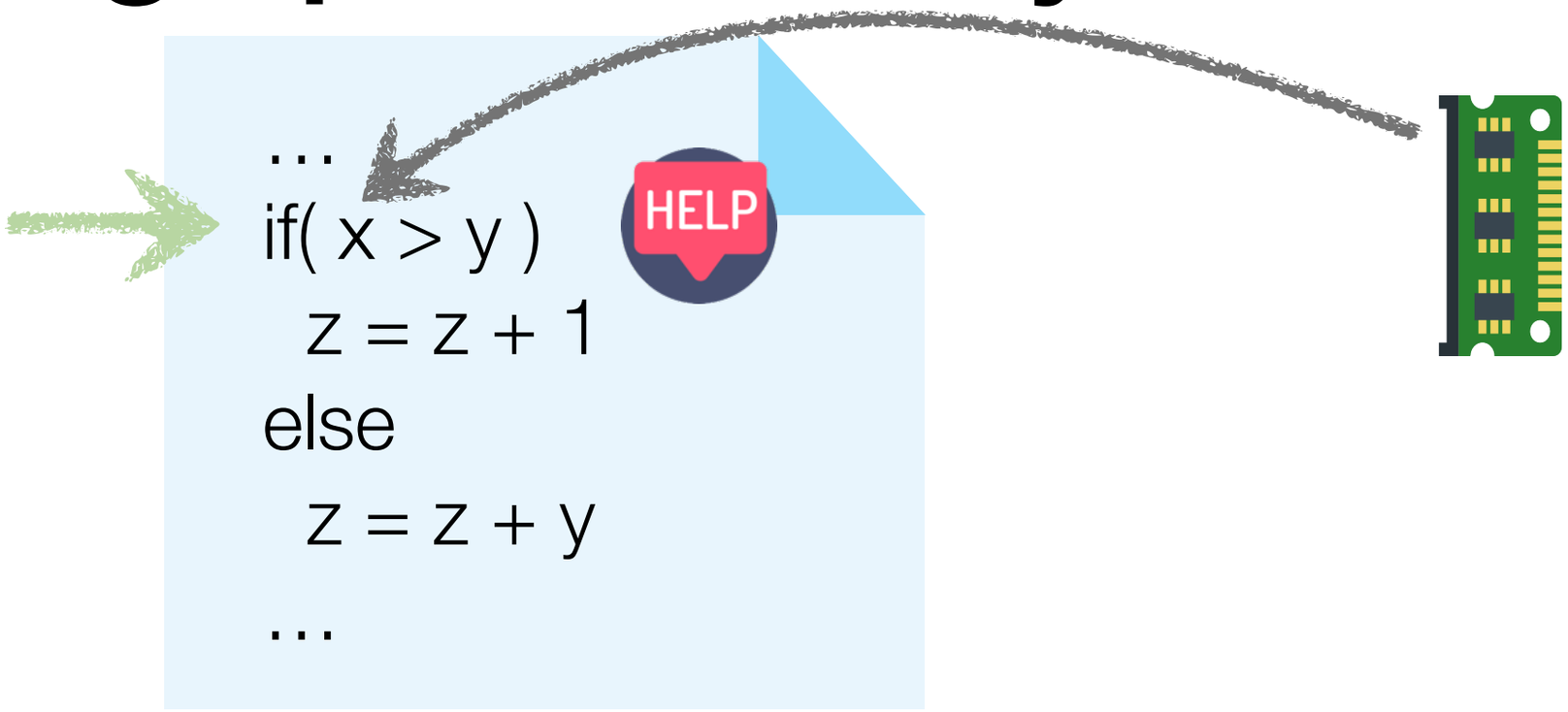


Branching speculatively



X	10	Y	20	Z	20

Branching speculatively

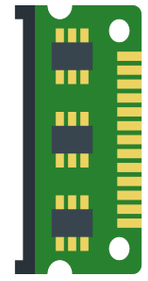


X	10	Y	20	Z	20

Branching speculatively

```
...
if( x > y )
  z = z + 1
else
  z = z + y
...
```

HELP



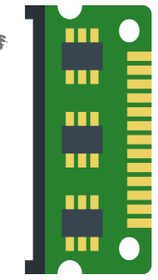
Branch predictor

X	10	Y	20	Z	20

Branching speculatively

```
...
if( x > y )
  z = z + 1
else
  z = z + y
...
```

HELP



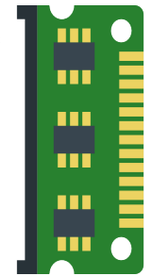
Branch predictor

X	10	Y	20	Z	20

Branching speculatively

```
...
if( x > y )
  z = z + 1
else
  z = z + y
...
```

HELP



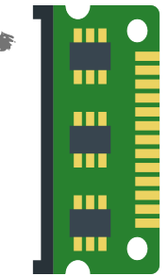
Branch predictor

X	10	Y	20	Z	20

X	10	Y	20	Z	20

Branching speculatively

```
...
if( x > y )
  z = z + 1
else
  z = z + y
...
```

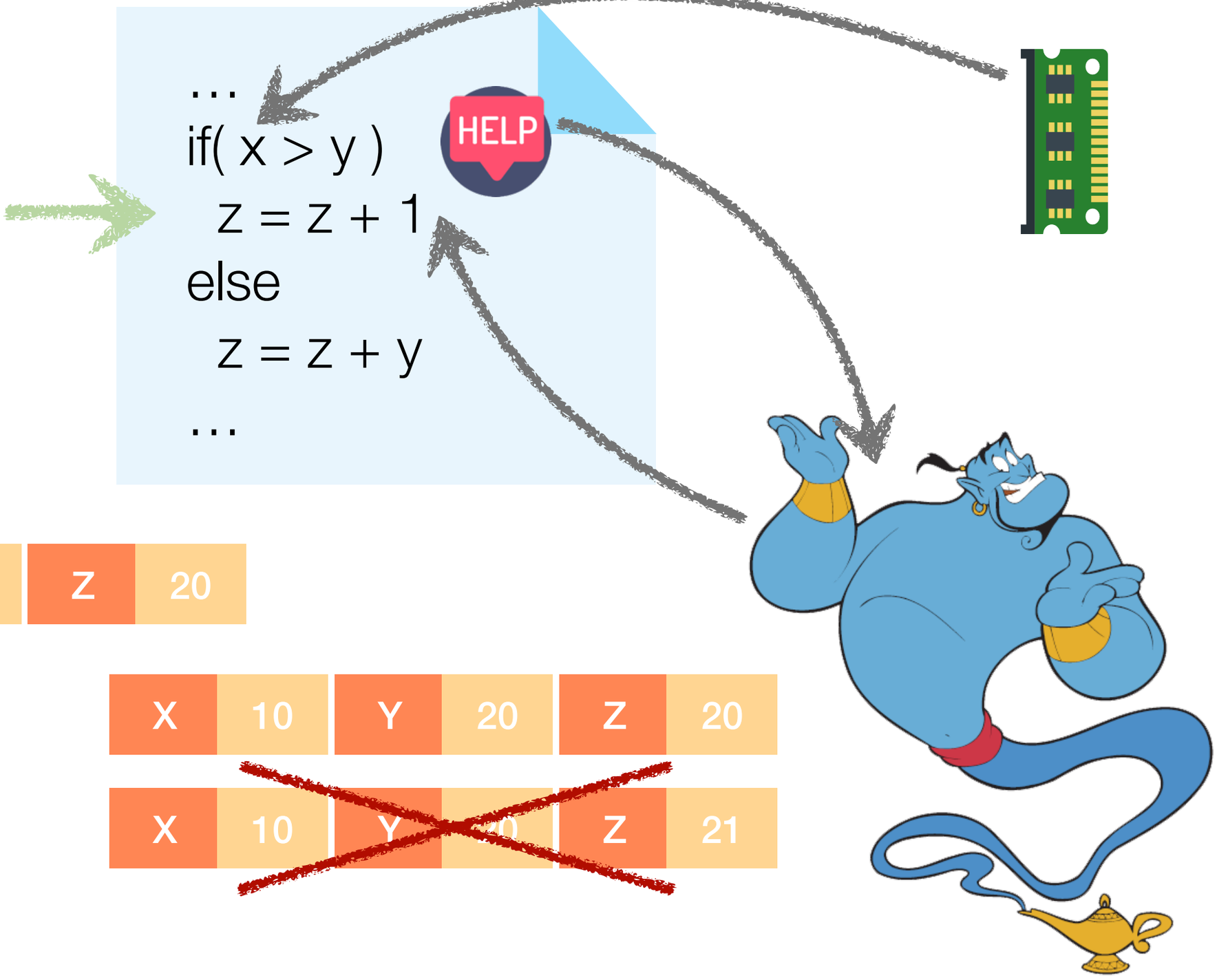


Branch predictor

X	10	Y	20	Z	20

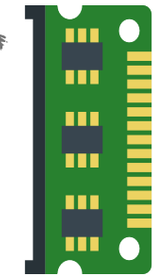
X	10	Y	20	Z	20
X	10	Y	20	Z	21

Branching speculatively



Branching speculatively

```
...
if( x > y )
  z = z + 1
else
  z = z + y
...
```



Branch predictor

X	10	Y	20	Z	20

X	10	Y	20	Z	20
X	10	Y	20	Z	21

X	10	Y	20	Z	40

Aborting speculative execution

Aborting speculative execution

Architectural state

- Registers
- Memory
- CPU flags
- ...

Aborting speculative execution

Architectural state

- Registers
- Memory
- CPU flags
- ...

It may affect the program's semantics

Aborting speculative execution

Architectural state

- Registers
- Memory
- CPU flags
- ...

It may affect the program's semantics

Micro-architectural state

- Cache content
- Activation status of CPU units
- ...

Aborting speculative execution

Architectural state

- Registers
- Memory
- CPU flags
- ...

It may affect the program's semantics

Micro-architectural state

- Cache content
- Activation status of CPU units
- ...

It doesn't affect the program's semantics

Aborting speculative execution

Architectural state

- Registers
- Memory
- CPU flags
- ...

It may affect the program's semantics

Saved and restored during speculative execution

Micro-architectural state

- Cache content
- Activation status of CPU units
- ...

It doesn't affect the program's semantics

Aborting speculative execution

Architectural state

- Registers
- Memory
- CPU flags
- ...

It may affect the program's semantics

Saved and restored during speculative execution

Micro-architectural state

- Cache content
- Activation status of CPU units
- ...

It doesn't affect the program's semantics

Ignored during speculative execution

Branch predictors (BP)

Branch predictors (BP)

- **Static BP** : always the same prediction

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

MOTIVATION:
Loops (backward jumps) are often taken more than once, and the first case of conditionals is more likely to be taken

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

MOTIVATION:
Loops (backward jumps) are often taken more than once, and the first case of conditionals is more likely to be taken

```
Lbl1: …
      cmp r1, r2
      je Lbl1
      mov r2, 5
      …
Lbl2: …
```

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

MOTIVATION:
Loops (backward jumps) are often taken more than once, and the first case of conditionals is more likely to be taken

```
Lbl1: ...
      cmp r1, r2
      je Lbl1
      mov r2, 5
      ...
Lbl2: ...
```

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

MOTIVATION:
Loops (backward jumps) are often taken more than once, and the first case of conditionals is more likely to be taken

```
Lbl1: …
      cmp r1, r2
      je Lbl1
      mov r2, 5
      …
Lbl2: …
```

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

MOTIVATION:
Loops (backward jumps) are often taken more than once, and the first case of conditionals is more likely to be taken

```
Lbl1: …
      cmp r1, r2
      je Lbl1
      mov r2, 5
      …
Lbl2: …
```

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

MOTIVATION:
Loops (backward jumps) are often taken more than once, and the first case of conditionals is more likely to be taken

```
Lbl1: …
      cmp r1, r2
      je Lbl2
      mov r2, 5
      …
Lbl2: …
```

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

MOTIVATION:
Loops (backward jumps) are often taken more than once, and the first case of conditionals is more likely to be taken

```
Lbl1: ...
      cmp r1, r2
      je Lbl2
      mov r2, 5
      ...
Lbl2: ...
```

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

MOTIVATION:
Loops (backward jumps) are often taken more than once, and the first case of conditionals is more likely to be taken

```
Lbl1: …
      cmp r1, r2
      je Lbl2
      mov r2, 5
      …
Lbl2: …
```

Branch predictors (BP)

- **Static BP** : always the same prediction
  - ***BTFNT*** - Backward (jumps) taken, forward not taken

MOTIVATION:
Loops (backward jumps) are often taken more than once, and the first case of conditionals is more likely to be taken

```
Lbl1: ...
      cmp r1, r2
      je Lbl2
      mov r2, 5
      ...
Lbl2: ...
```

HELP

Branch predictors (BP)

Branch predictors (BP)

- **Dynamic BP** : considers history, more precise

Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome

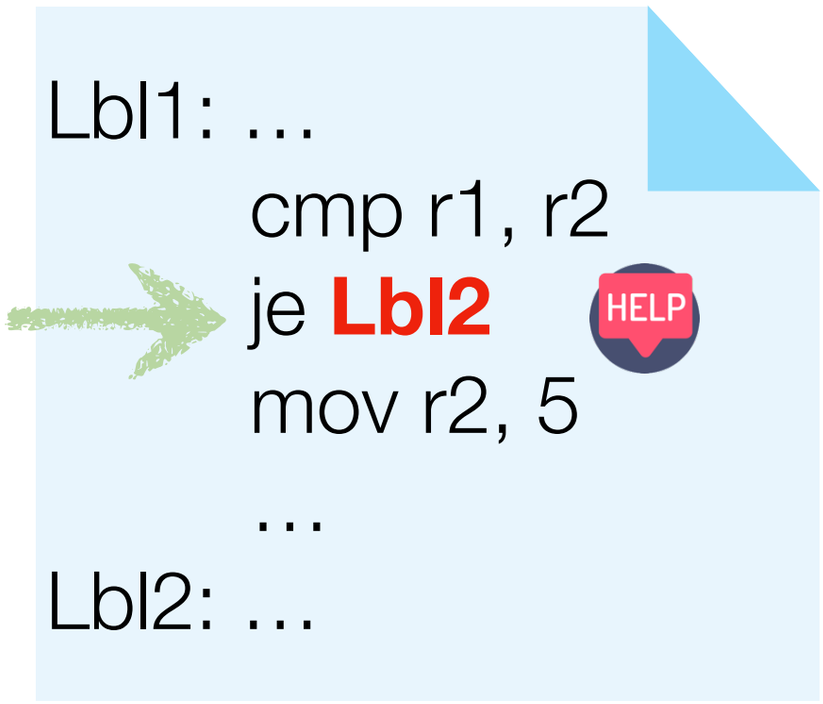
Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome

```
Lbl1: …
      cmp r1, r2
      je Lbl2
      mov r2, 5
      …
Lbl2: …
```

Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome



Branch predictors (BP)

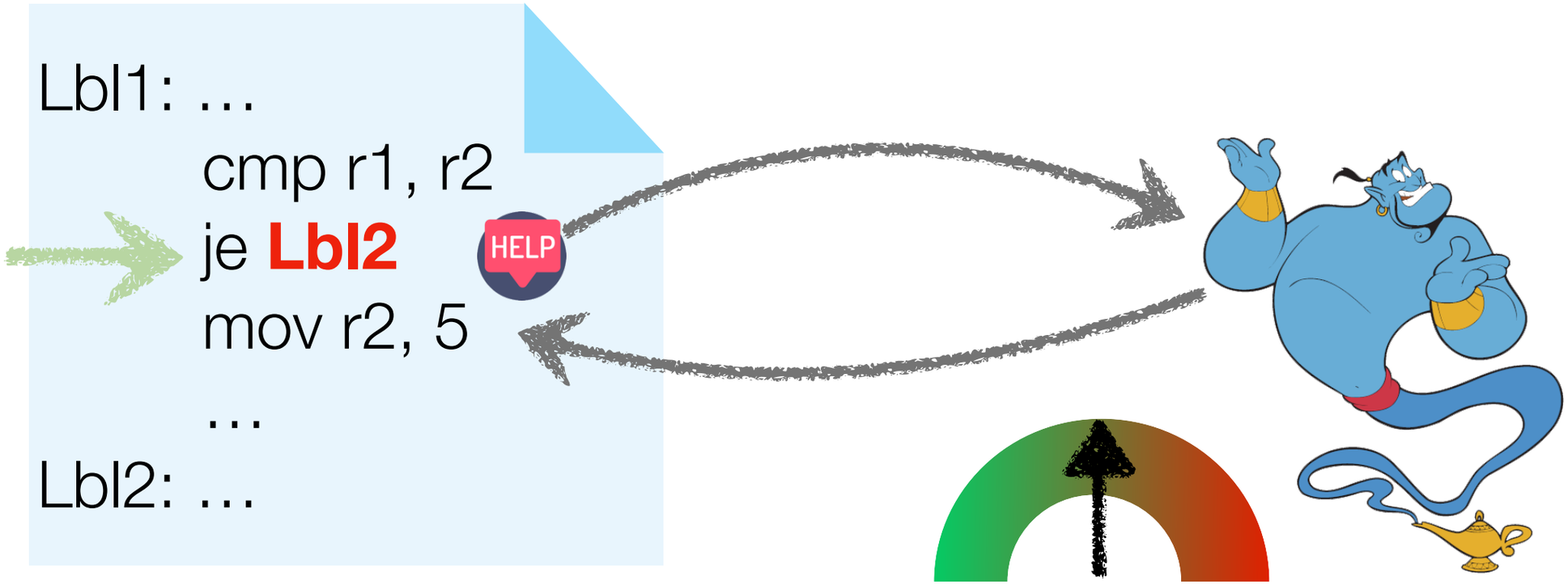
- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome

```
Lbl1: …
      cmp r1, r2
      je Lbl2
      mov r2, 5
      …
Lbl2: …
```

ctr

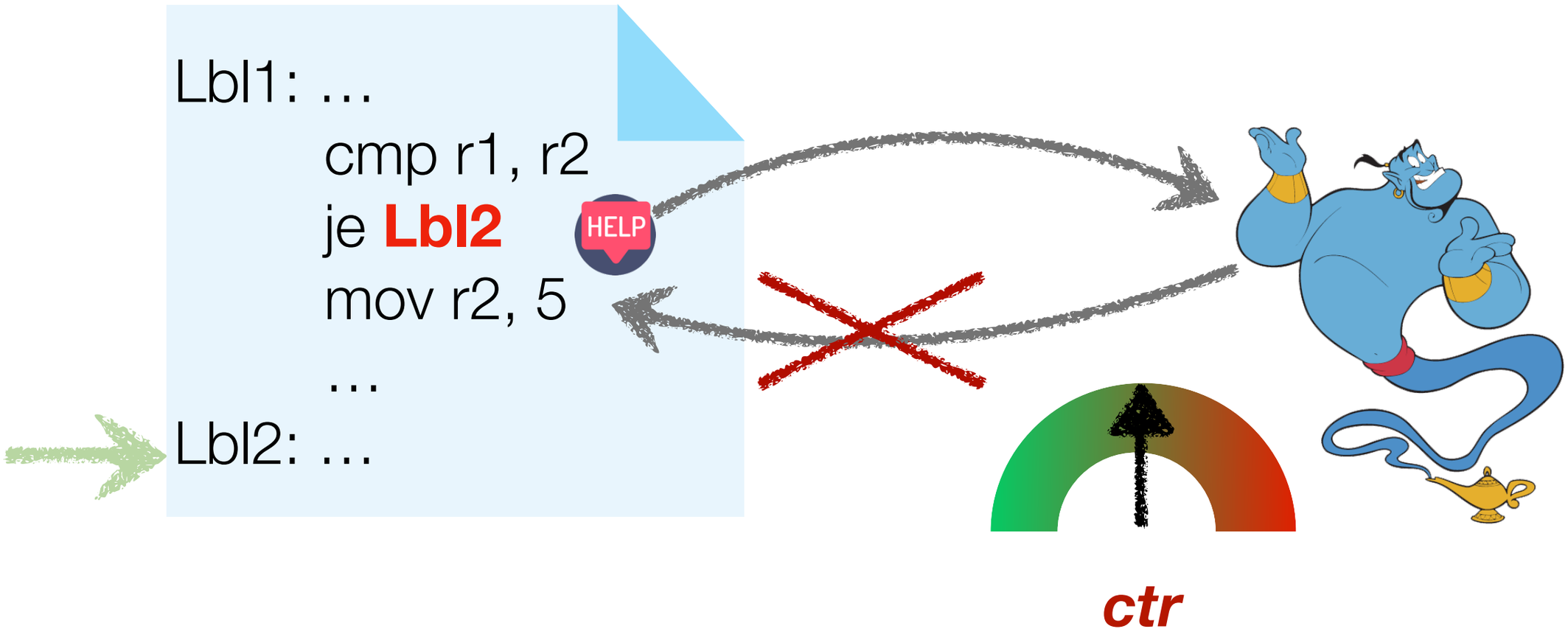
Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome



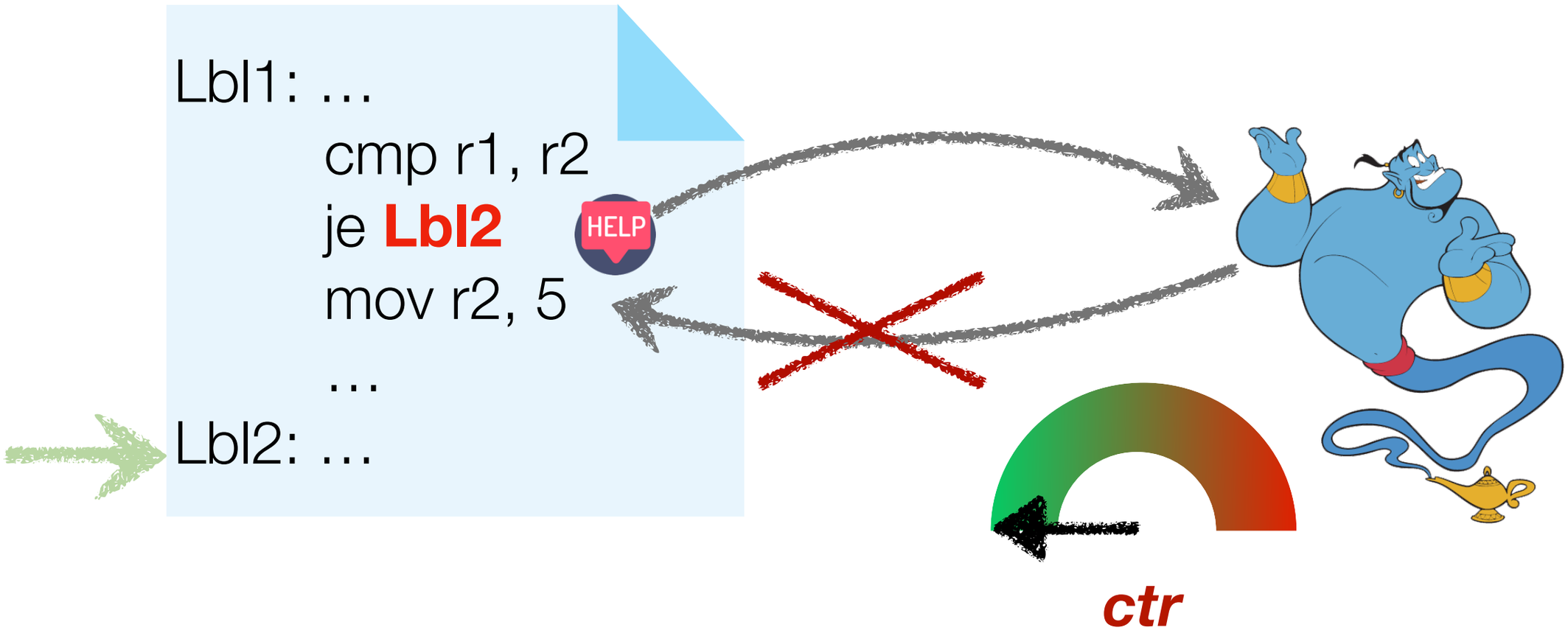
Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome



Branch predictors (BP)

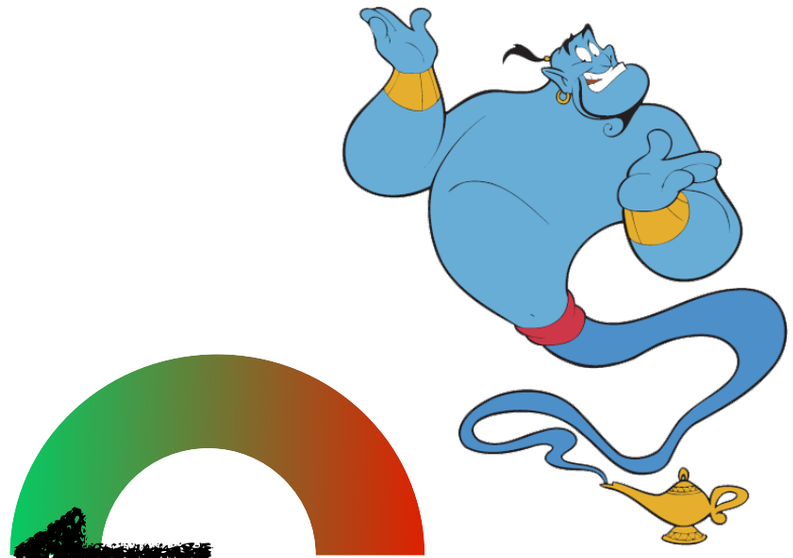
- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome



Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome

```
Lbl1: …
      cmp r1, r2
      je Lbl2
      mov r2, 5
      …
Lbl2: …
```



Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome

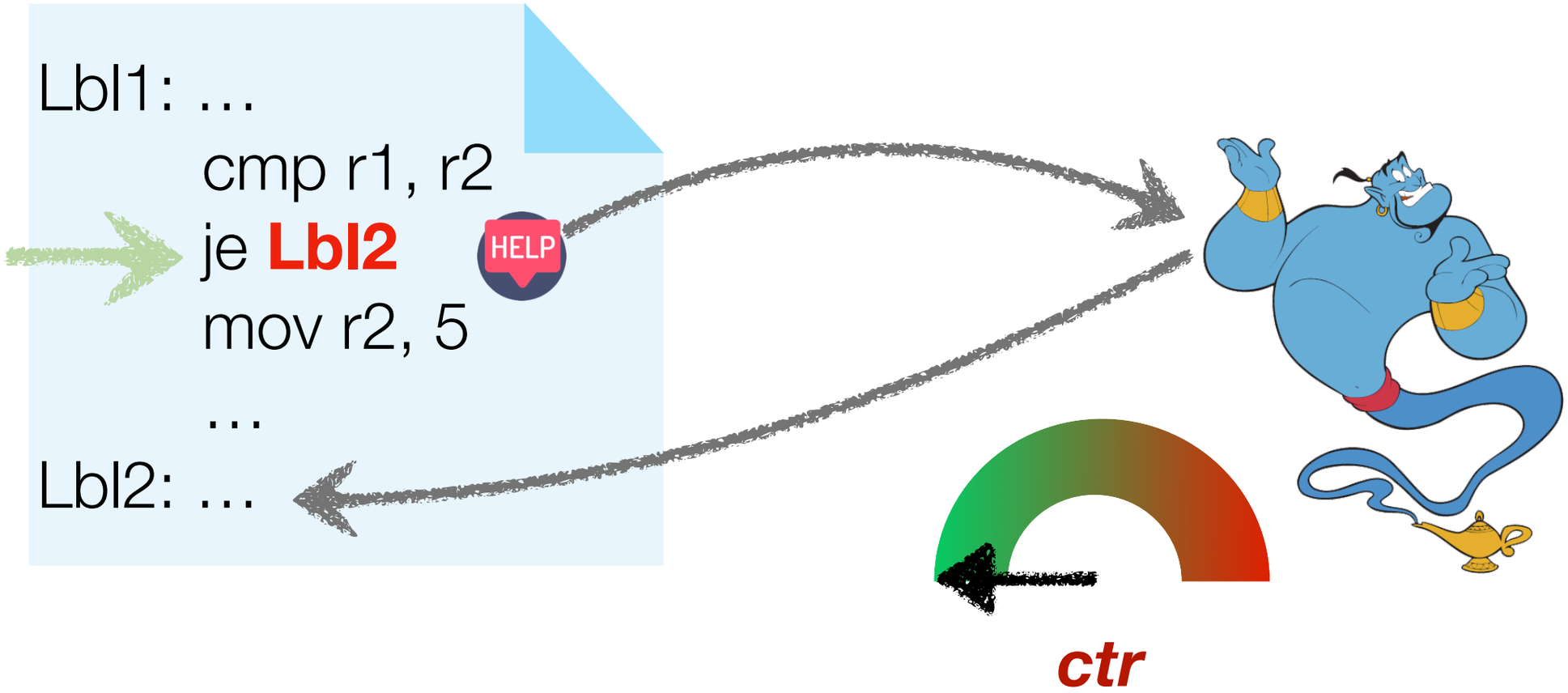
```
Lbl1: …
      cmp r1, r2
      je Lbl2
      mov r2, 5
      …
Lbl2: …
```

HELP

ctr

Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 1-bit counter ***ctr*** : prediction based on branch's last outcome



Branch predictors (BP)

- **Dynamic BP** : considers history, more precise

Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 2-bit counter ***ctr*** : prediction based on last 2 outcomes

Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 2-bit counter ***ctr*** : prediction based on last 2 outcomes
  - Branch taken ***ctr ++***, branch not taken ***ctr --***

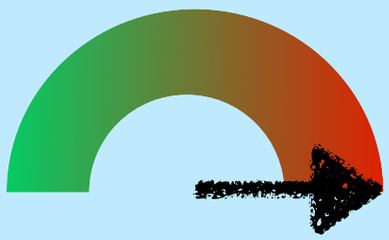
Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 2-bit counter ***ctr*** : prediction based on last 2 outcomes
  - Branch taken ***ctr ++***, branch not taken ***ctr --***

Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 2-bit counter ***ctr*** : prediction based on last 2 outcomes
  - Branch taken ***ctr ++***, branch not taken ***ctr --***

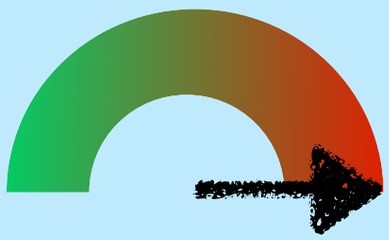
0 → strongly not taken



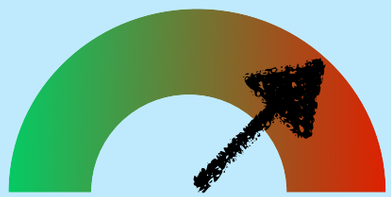
Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 2-bit counter ***ctr*** : prediction based on last 2 outcomes
  - Branch taken ***ctr ++***, branch not taken ***ctr --***

0 → strongly not taken



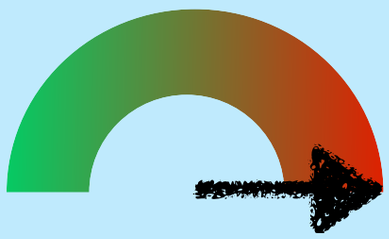
1 → weakly not taken



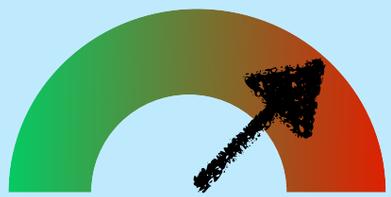
Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 2-bit counter ***ctr*** : prediction based on last 2 outcomes
  - Branch taken ***ctr ++***, branch not taken ***ctr --***

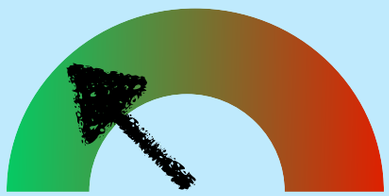
0 → strongly not taken



1 → weakly not taken



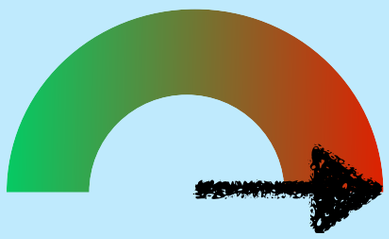
2 → weakly taken



Branch predictors (BP)

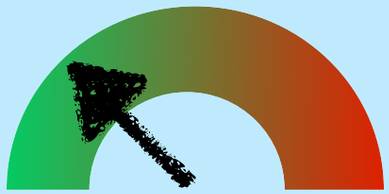
- **Dynamic BP** : considers history, more precise
  - 2-bit counter ***ctr*** : prediction based on last 2 outcomes
  - Branch taken ***ctr ++***, branch not taken ***ctr --***

0 → strongly not taken

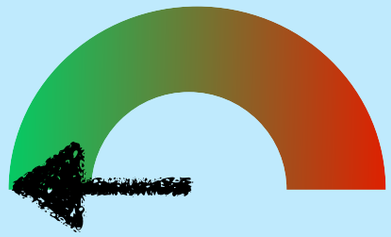


1 → weakly not taken

2 → weakly taken



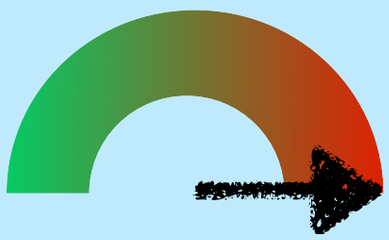
3 → strongly taken



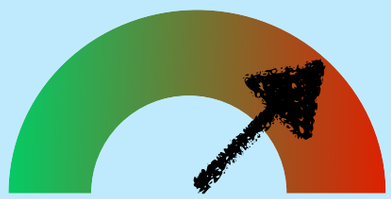
Branch predictors (BP)

- **Dynamic BP** : considers history, more precise
  - 2-bit counter ***ctr*** : prediction based on last 2 outcomes
  - Branch taken ***ctr ++***, branch not taken ***ctr --***

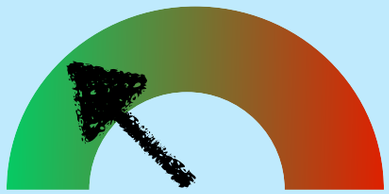
0 → strongly not taken



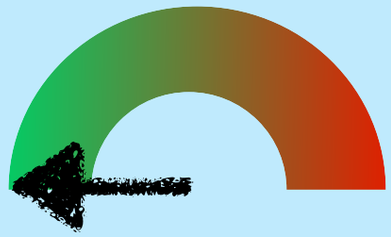
1 → weakly not taken



2 → weakly taken



3 → strongly taken



Attacks (SPECTRE V1)

Speculative execution attacks

Speculative execution attacks

Speculative execution attacks

Trigger speculation

Speculative execution attacks

Trigger speculation

Target micro-architectural component

Speculative execution attacks

Trigger speculation

Target micro-architectural component

Recover data from micro-architecture

Speculative execution attacks

Trigger speculation

Target micro-architectural component

Encode data into micro-architecture

Recover data from micro-architecture

Speculative execution attacks

Trigger speculation

Influence the speculation target

Target micro-architectural component

Encode data into micro-architecture

Recover data from micro-architecture

Setting

Attacker

Victim

Setting

Attacker

Victim

Setting

Attacker

Victim

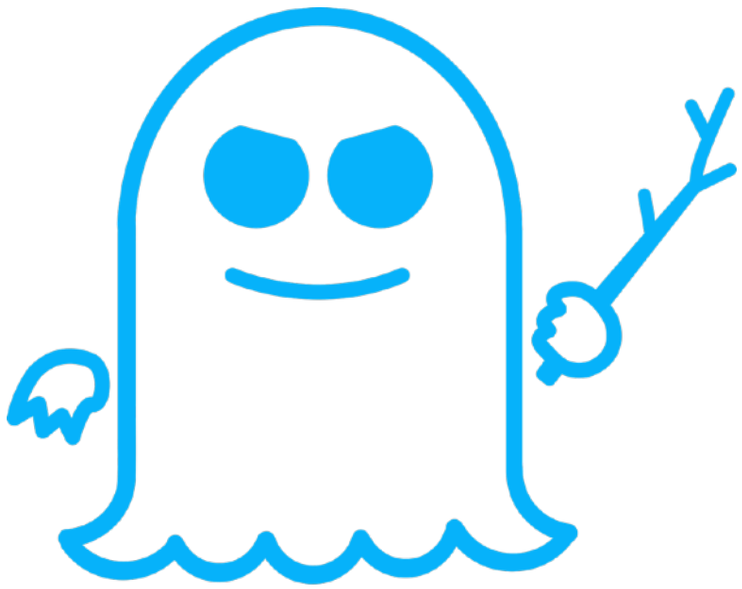
Setting

Attacker

Victim

SPECTRE (Variant 1)

SPECTRE (Variant 1)



- **Speculation** : using branch commands and BPs

SPECTRE (Variant 1)

- **Speculation** : using branch commands and BPs
- **Micro-architectural state** : caches
- **Encode** data through memory accesses
- **Recover** data using Prime+Probe, Flush+reload, …

SPECTRE (Variant 1)

- **Speculation** : using branch commands and BPs
- **Micro-architectural state** : caches
  - **Encode** data through memory accesses
  - **Recover** data using Prime+Probe, Flush+reload, …
- **Influence speculation target** : exploit dynamic BP

SPECTRE (Variant 1)

Victim's code

```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

SPECTRE (Variant 1)


```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

SPECTRE (Variant 1)

What is in A[5]?

```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

SPECTRE (Variant 1)

What is in A[5]?

```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

1) Training

SPECTRE (Variant 1)

What is in A[5]?

```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

1) Training

SPECTRE (Variant 1)

What is in A[5]?

```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

1) Training

f(0);

SPECTRE (Variant 1)

What is in A[5]?

```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

1) Training

f(0); f(1); f(2);

SPECTRE (Variant 1)

What is in A[5]?

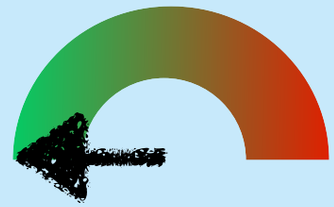
```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

1) Training

f(0); f(1); f(2);



SPECTRE (Variant 1)

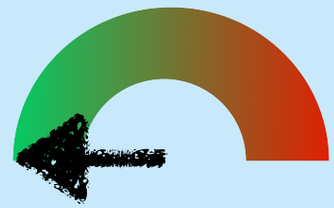
Victim's code

```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

4

1) Training

f(0); f(1); f(2);



2) Prepare

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?

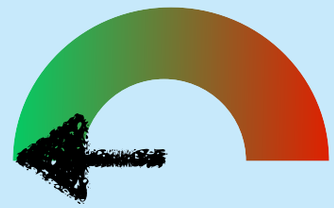
Victim's code

```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

4

1) Training

f(0); f(1); f(2);

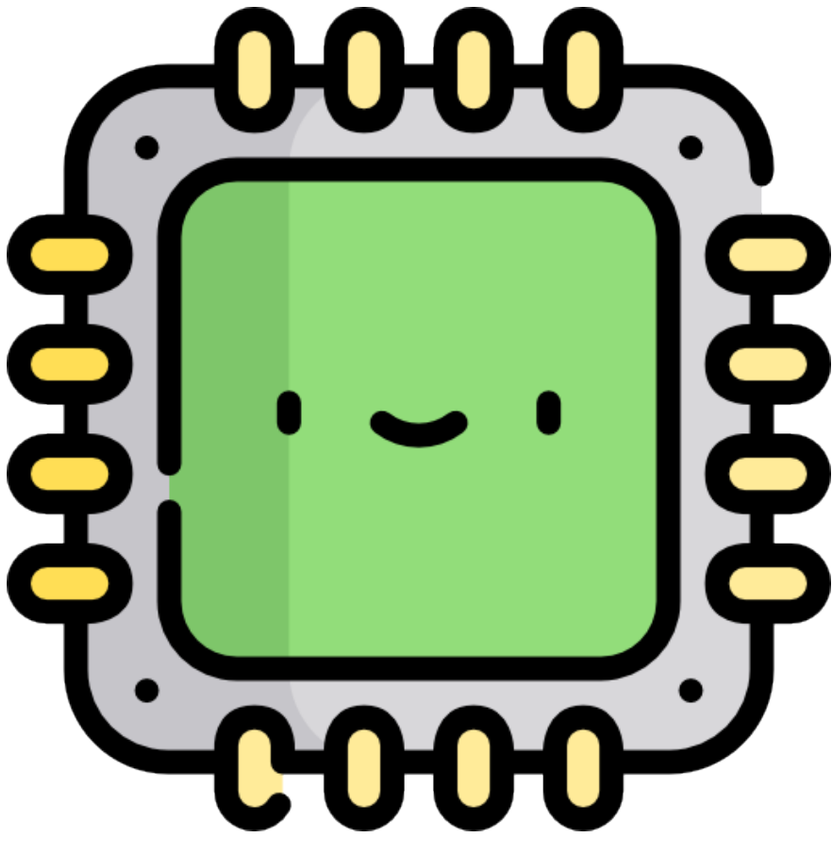


2) Prepare

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?



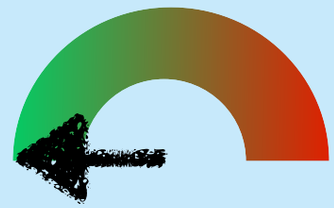
```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

1) Training

f(0); f(1); f(2);

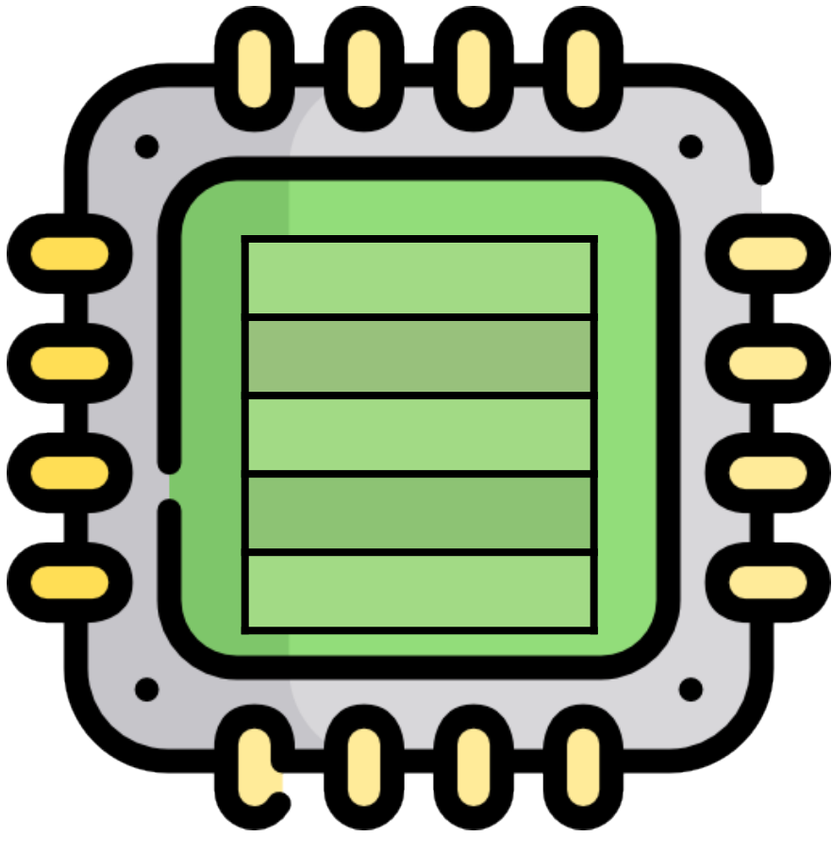


2) Prepare

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?



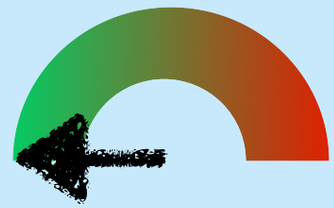
```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

1) Training

f(0); f(1); f(2);

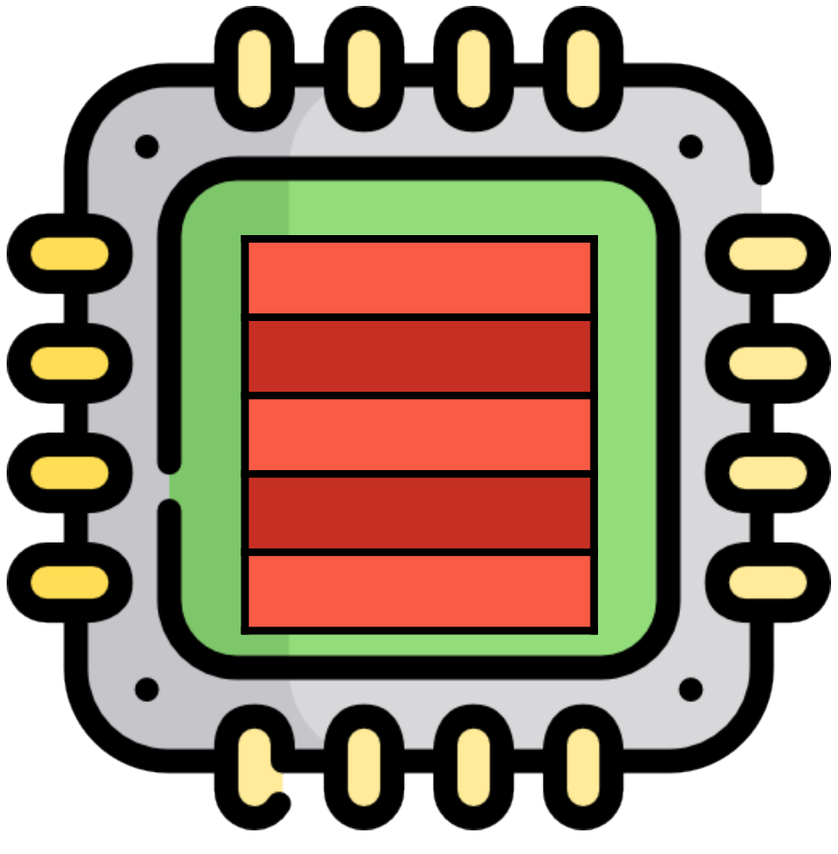


2) Prepare

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?



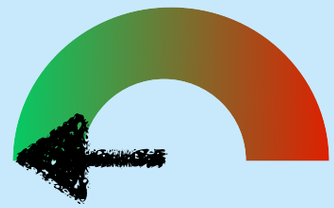
```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

1) Training

f(0); f(1); f(2);

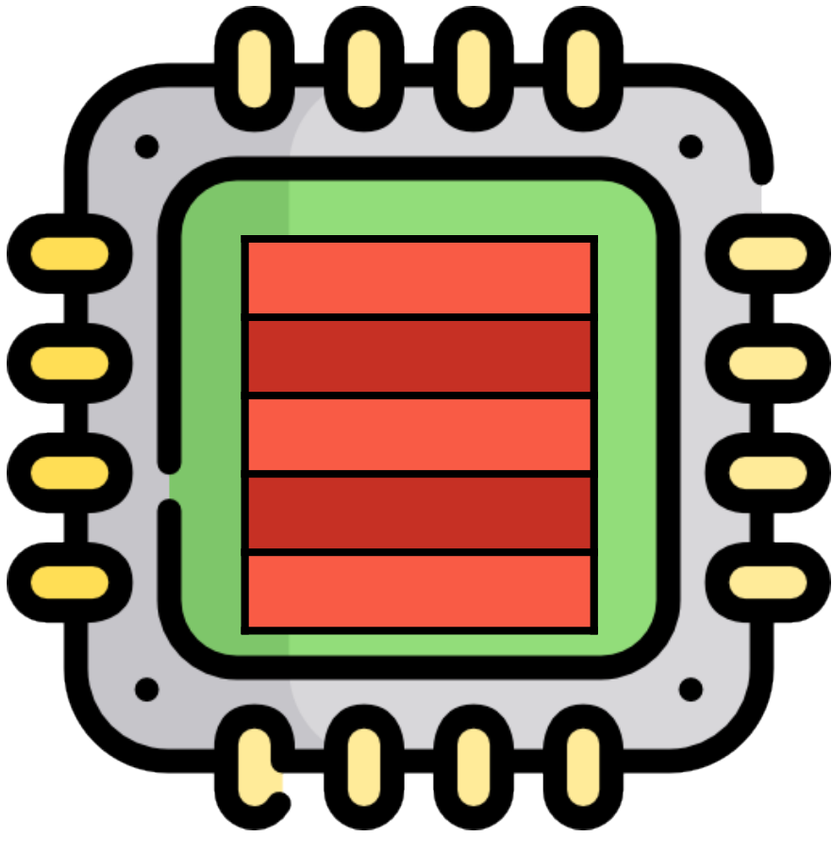


2) Prepare Fill cache with attacker's data

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?

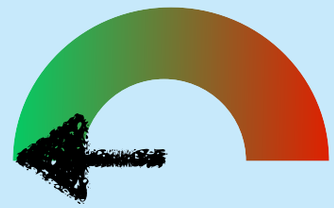


```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

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



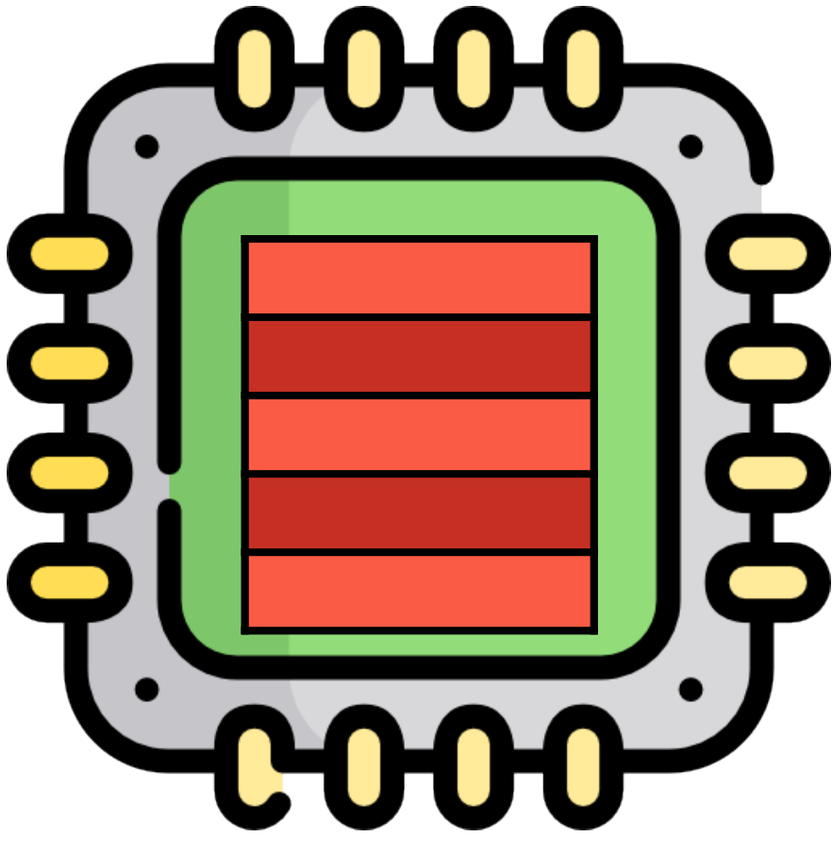
2) Prepare Fill cache with attacker's data

3) Attack

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?

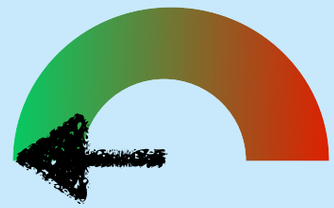


```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

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



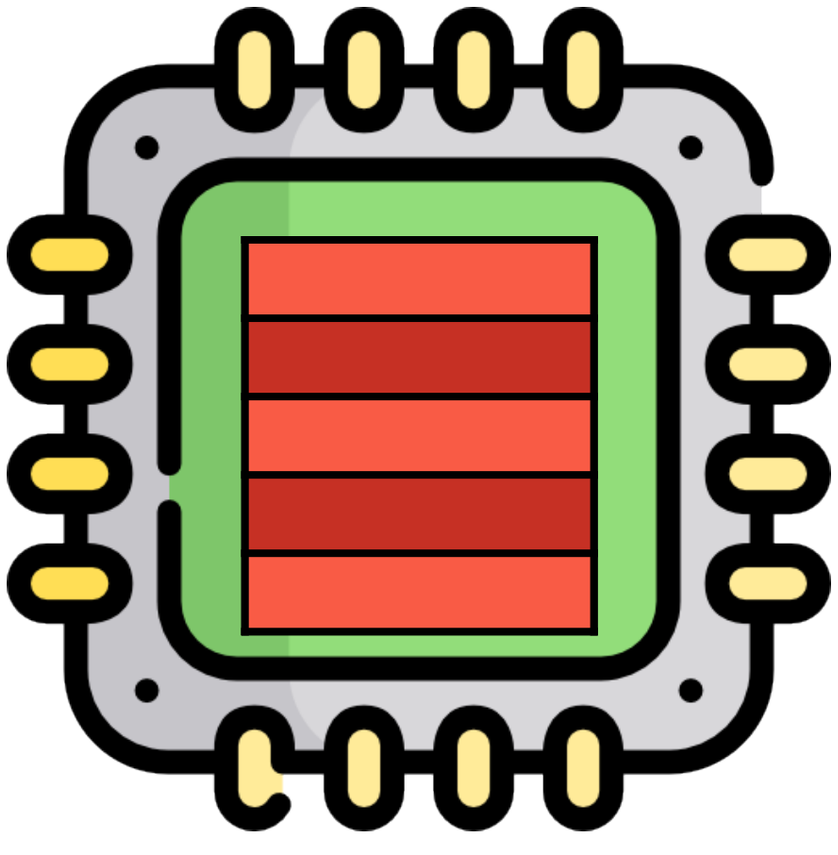
2) Prepare Fill cache with attacker's data

3) Attack f(5);

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?

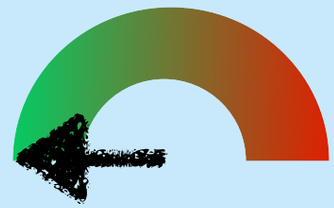


```
void f(x) {
  if  (x < A_size)
        y = B[ A[x] * 4096]
}
```

Victim's code

4

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



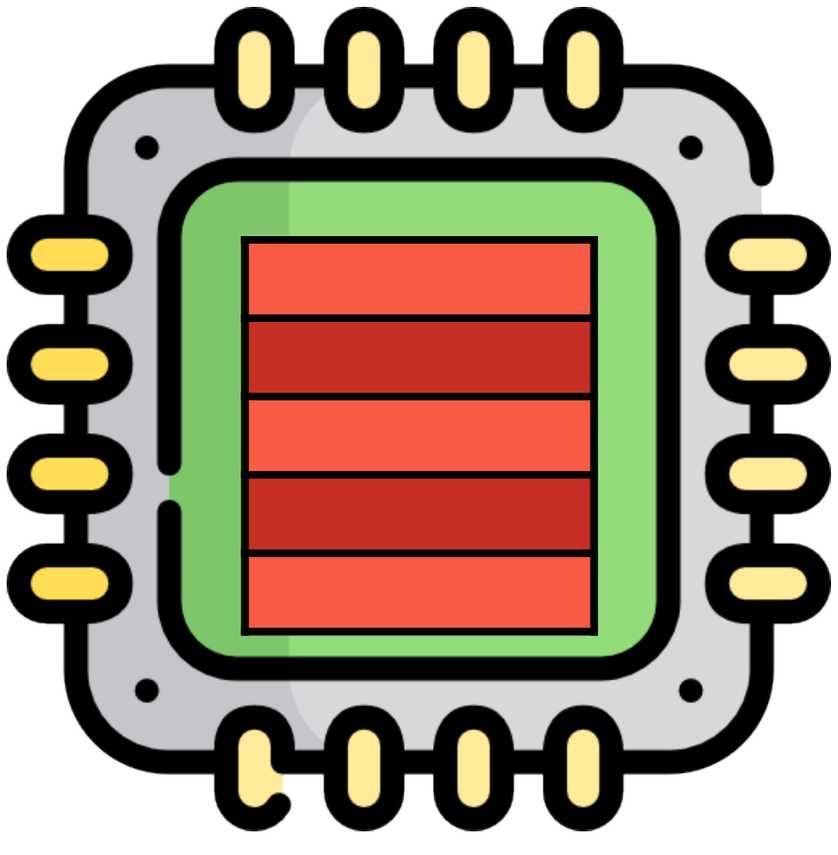
2) Prepare Fill cache with attacker's data

3) Attack f(5);

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?



```
void f(x) {
  if  (x < A_size)
      y = B[ A[x] * 4096]
}
```

Victim's code

4

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

2) Prepare Fill cache with attacker's data

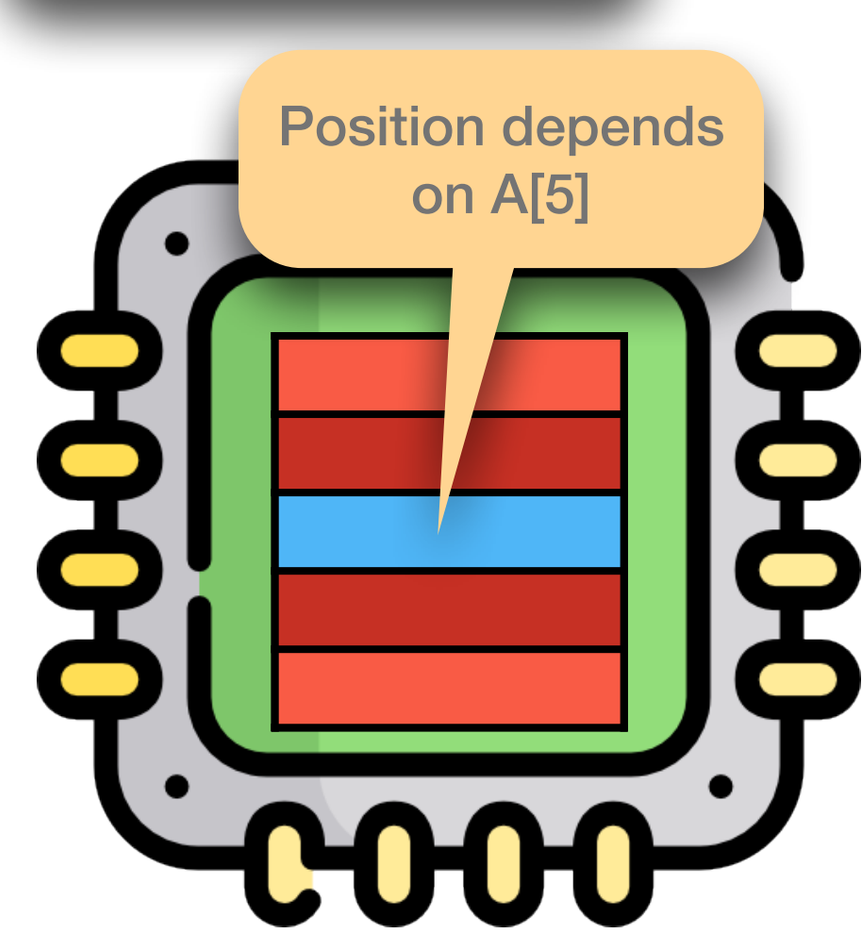
3) Attack f(5);

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?

Position depends on A[5]



```
void f(x) {
  if  (x < A_size)
      y = B[ A[x] * 4096]
}
```

Victim's code

4

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

2) Prepare Fill cache with attacker's data

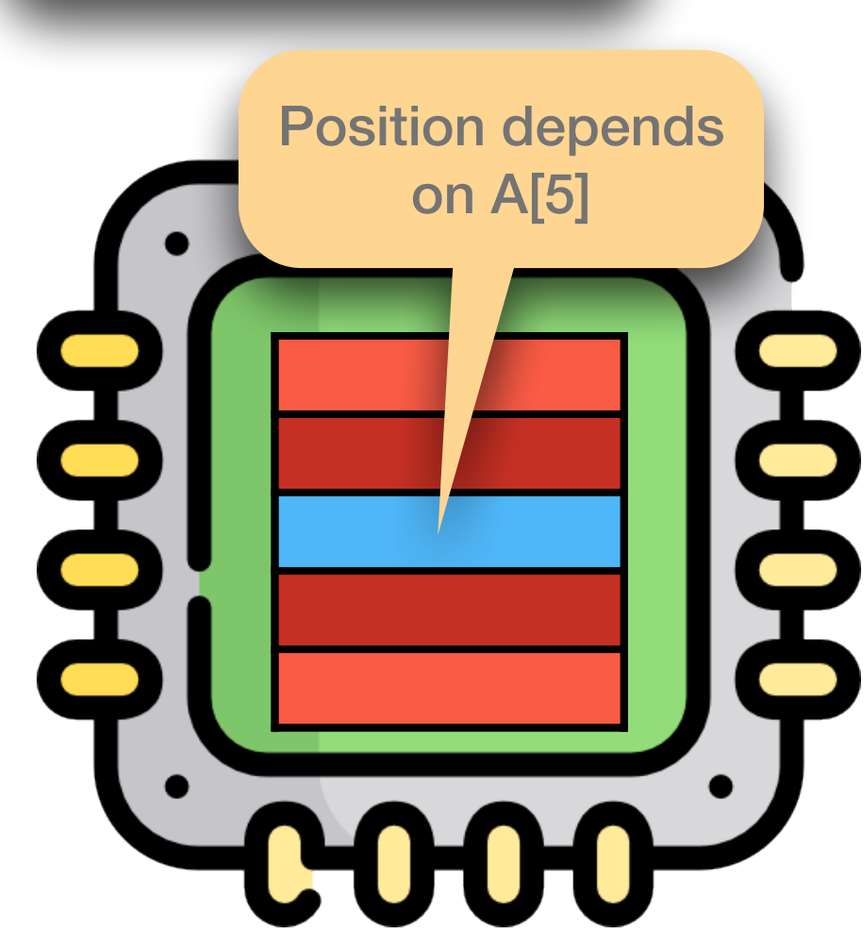
3) Attack f(5);

SPECTRE (Variant 1)

& Prime + Probe

What is in A[5]?

Position depends on A[5]



```
void f(x) {
  if  (x < A_size)
      y = B[ A[x] * 4096]
}
```

Victim's code

4

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

2) Prepare Fill cache with attacker's data

3) Attack f(5);

4) Extract

SPECTRE (Variant 1)

& Prime + Probe

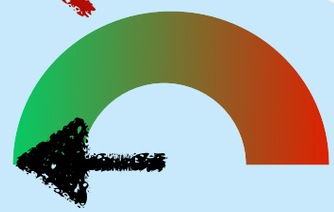
What is in A[5]?

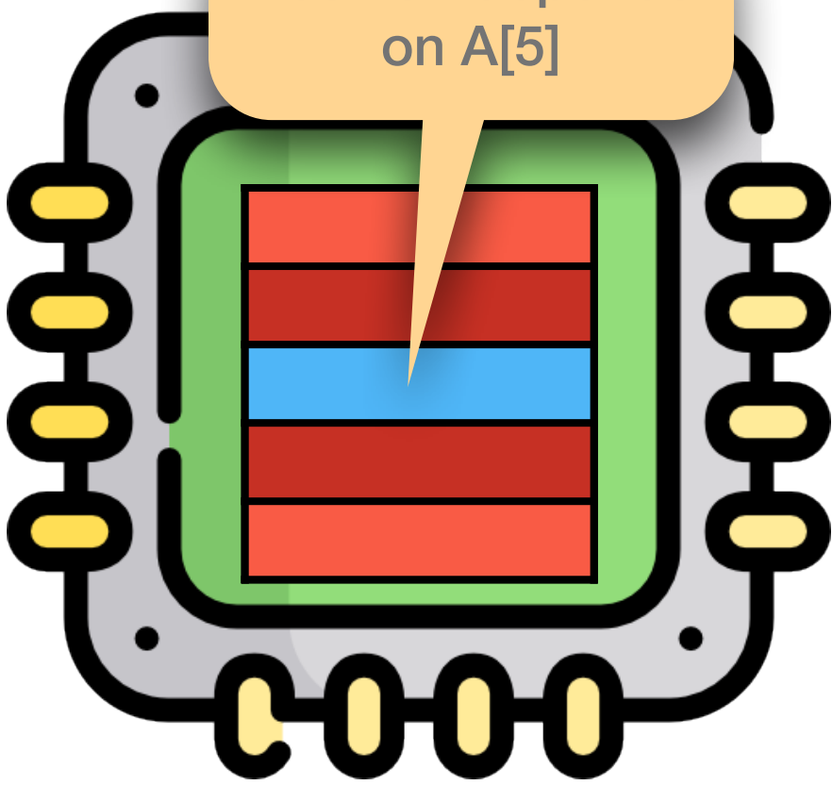
```
void f(x) {
  if  (x < A_size)
      y = B[ A[x] * 4096]
}
```

Victim's code

4

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





2) Prepare Fill cache with attacker's data

3) Attack f(5);

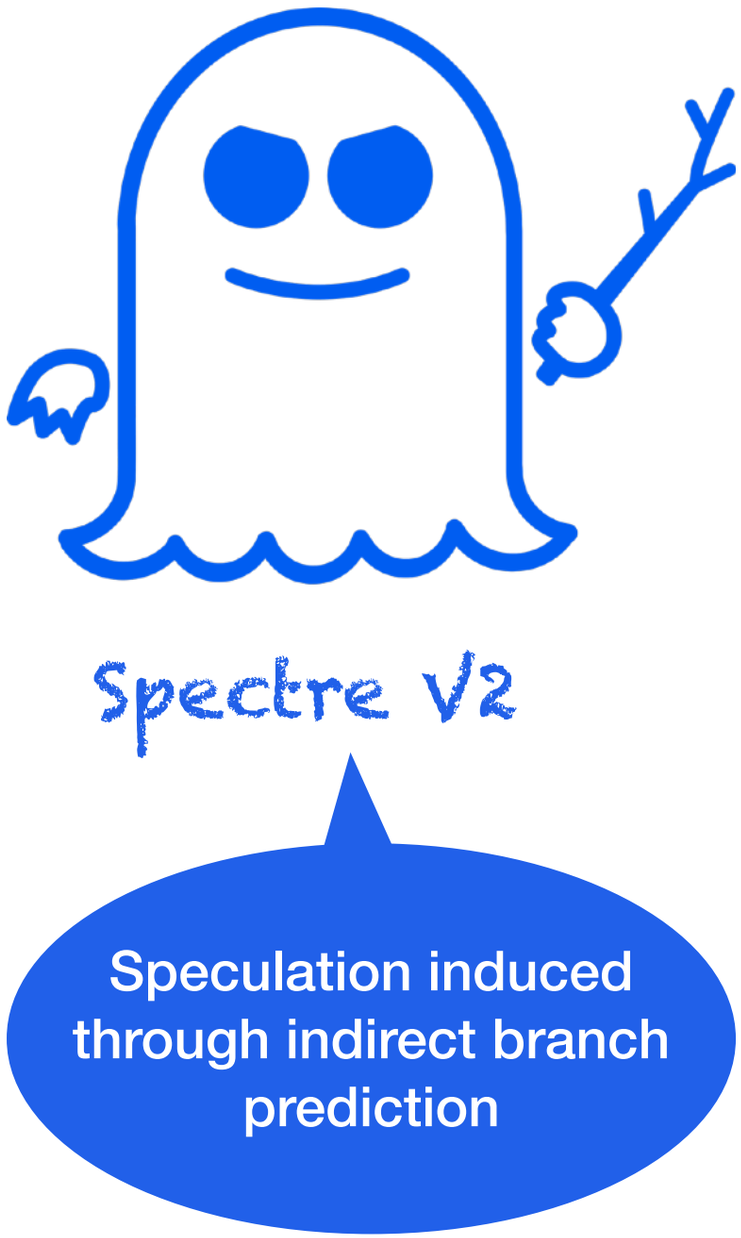
4) Extract "K-th block causes cache miss only if A[5] = Z"

… and his brothers

… and his brothers

Spectre V2

… and his brothers



… and his brothers

Spectre V2

Ret2spec

… and his brothers

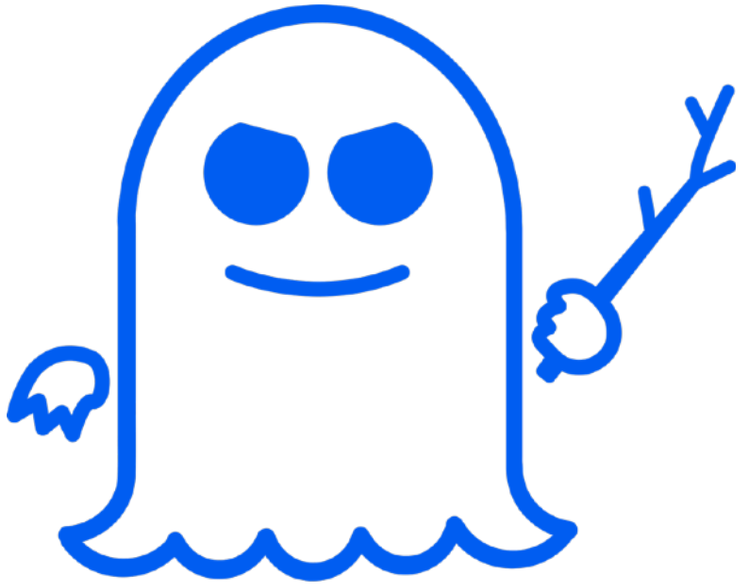
Spectre V2

Ret2spec

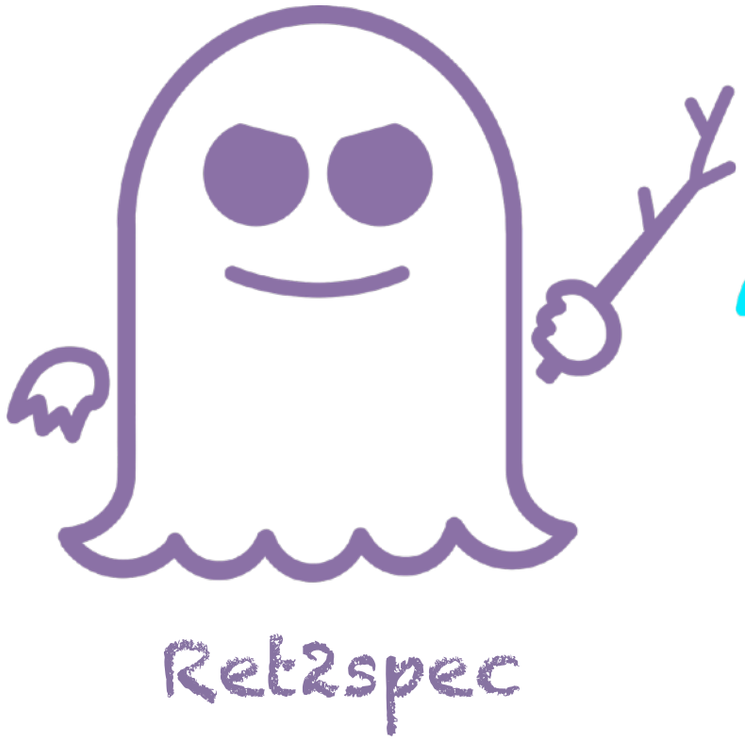
Speculation induced through return stack buffers

… and his brothers

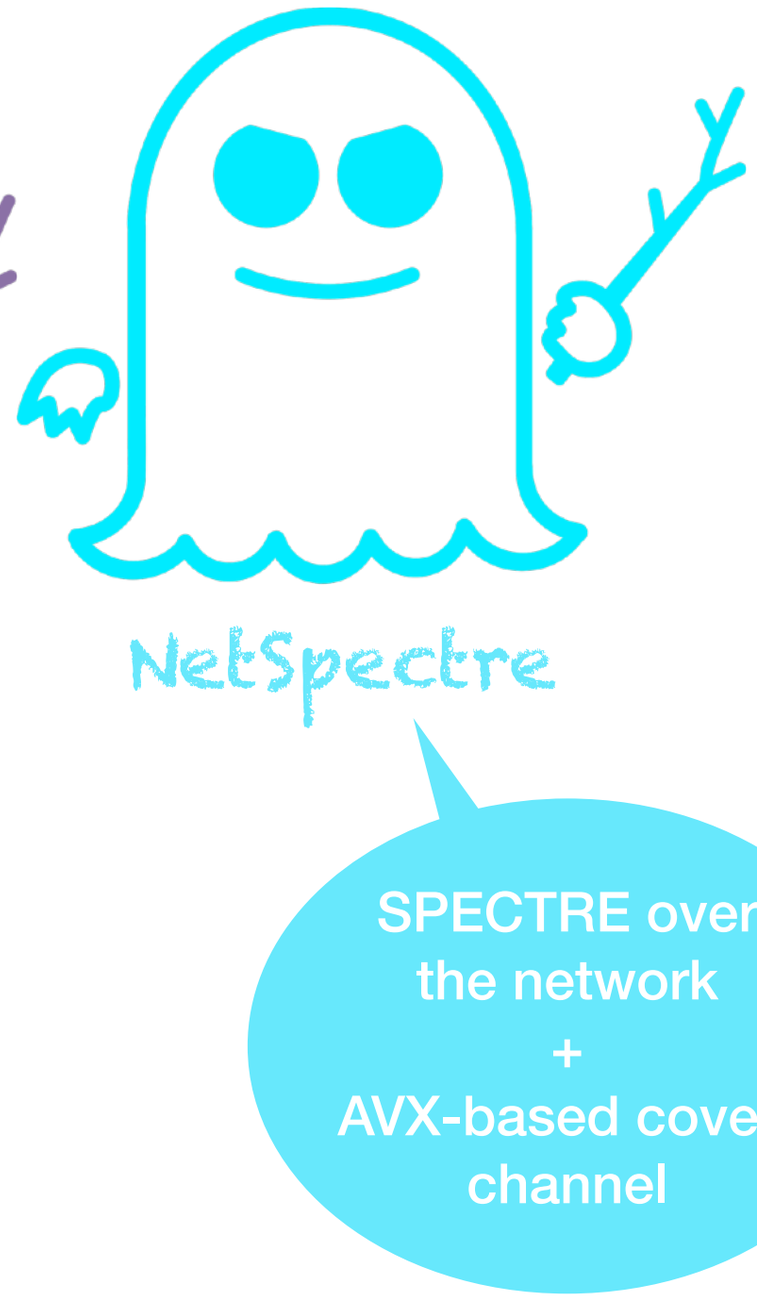
… and his brothers



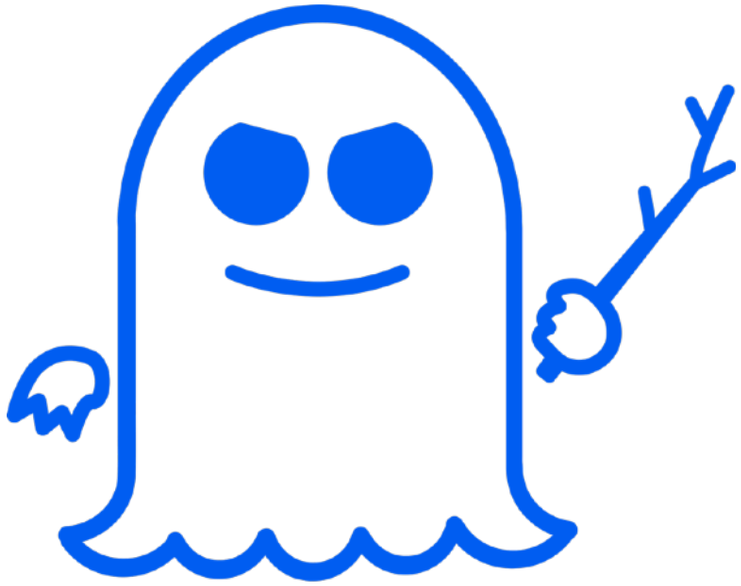
Spectre V2



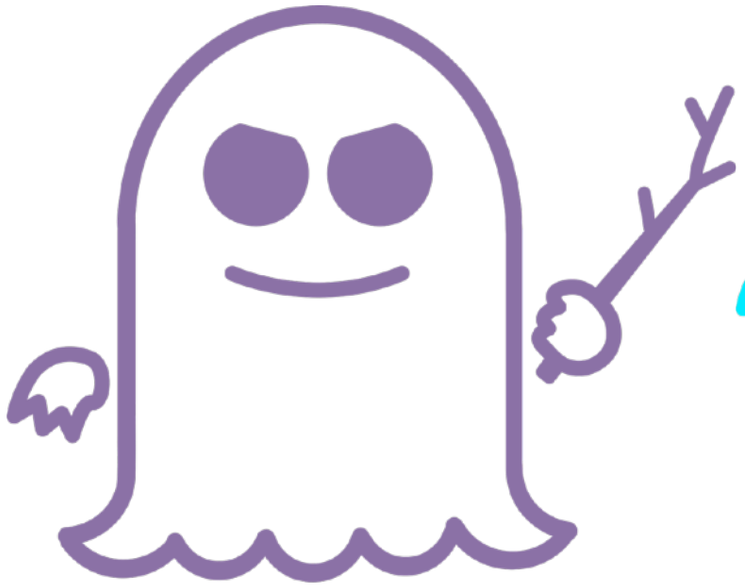
Ret2spec



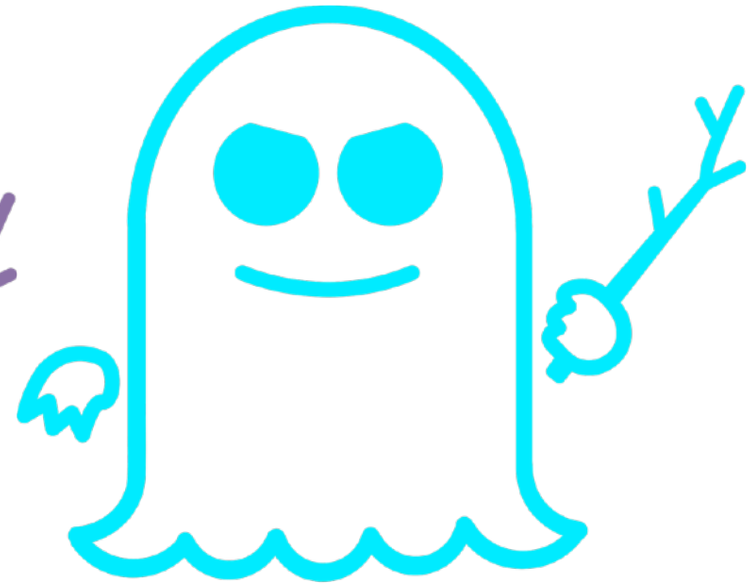
… and his brothers



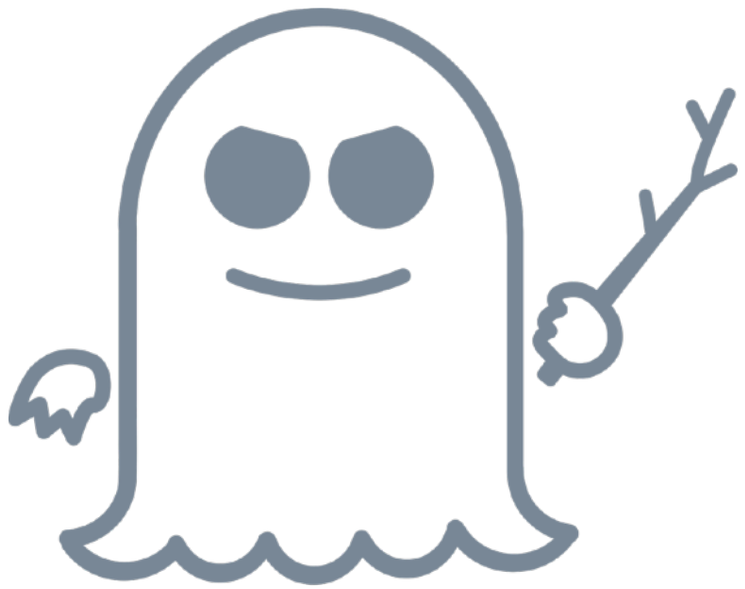
Spectre V2



Ret2spec

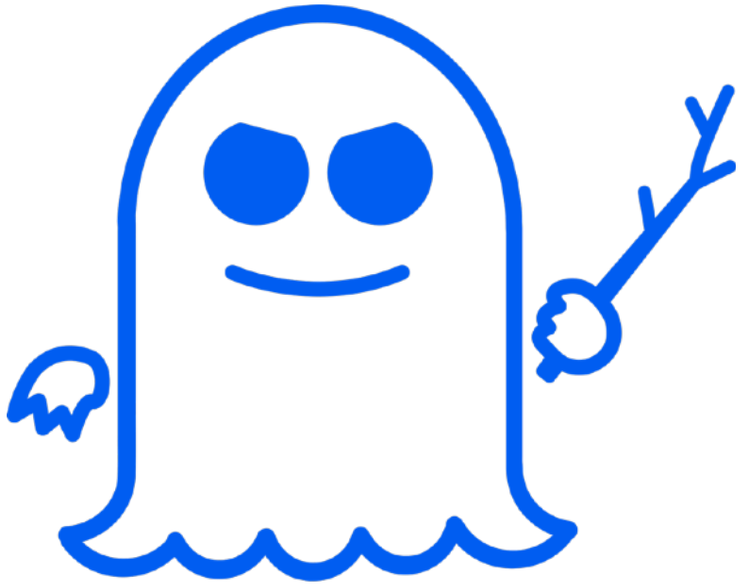


NetSpectre

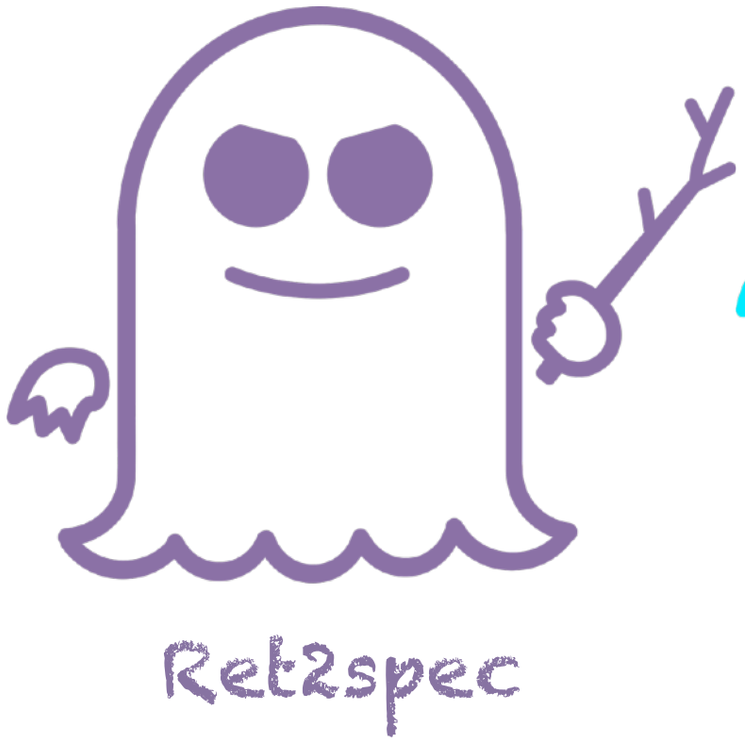


Spectre V4

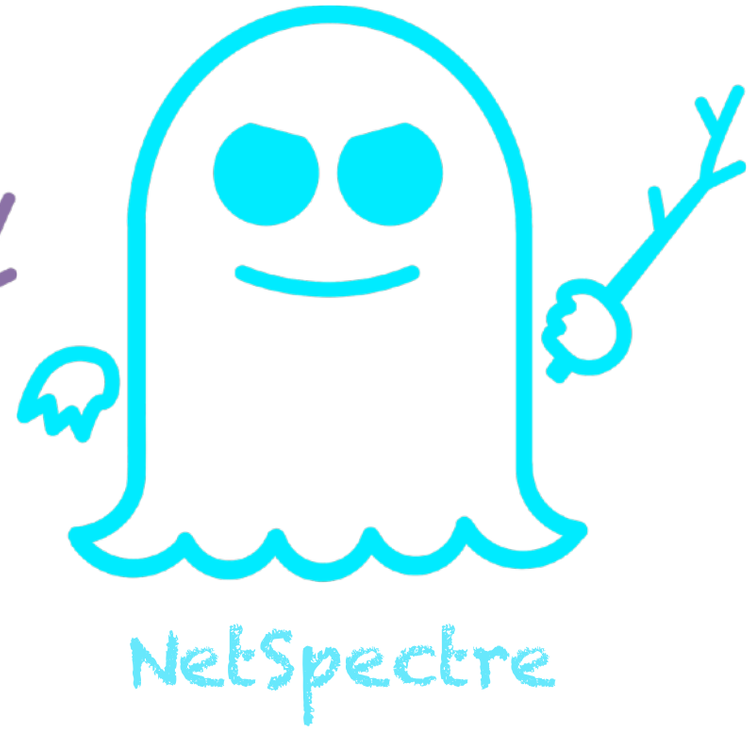
… and his brothers



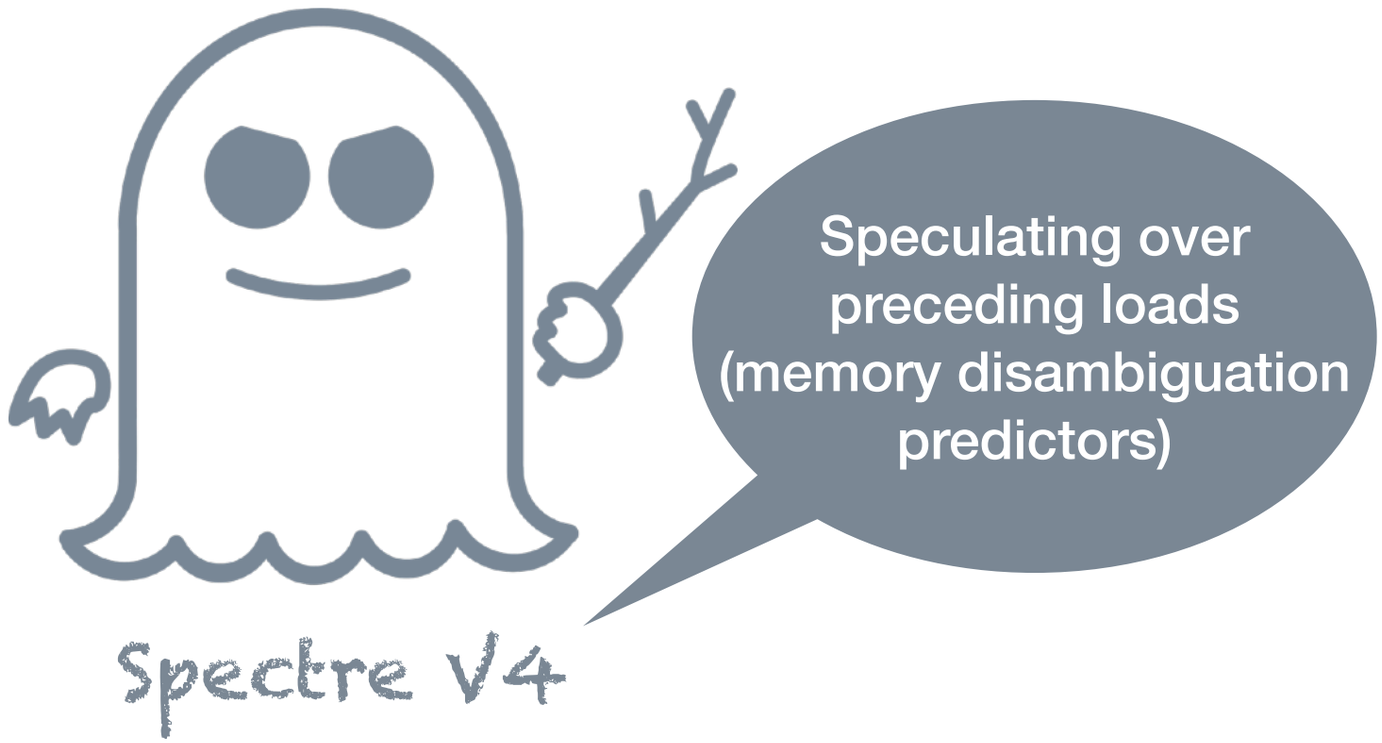
Spectre V2



Ret2spec

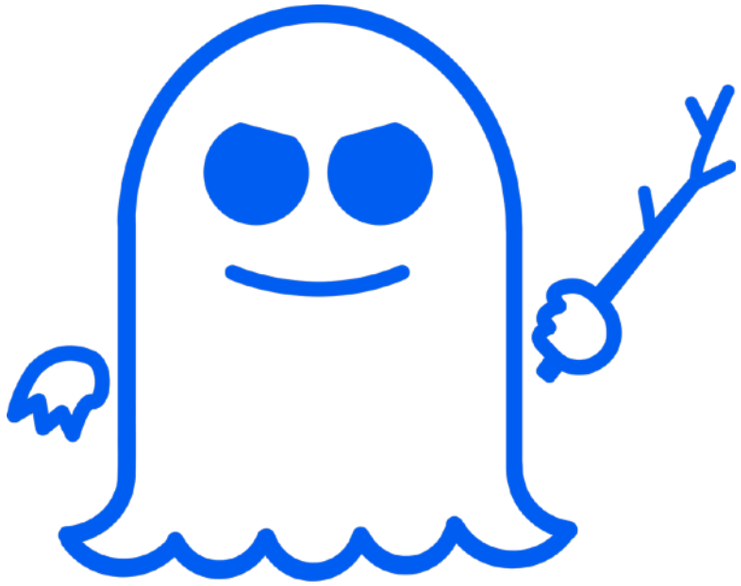


NetSpectre

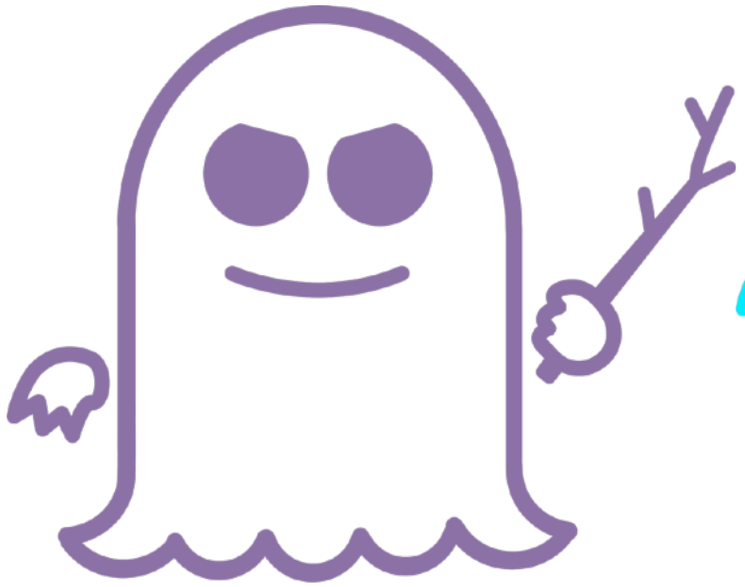


Spectre V4

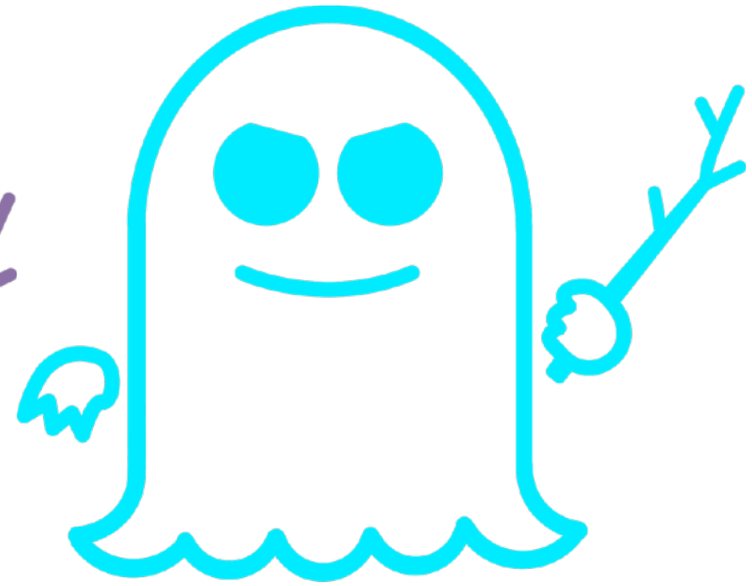
… and his brothers



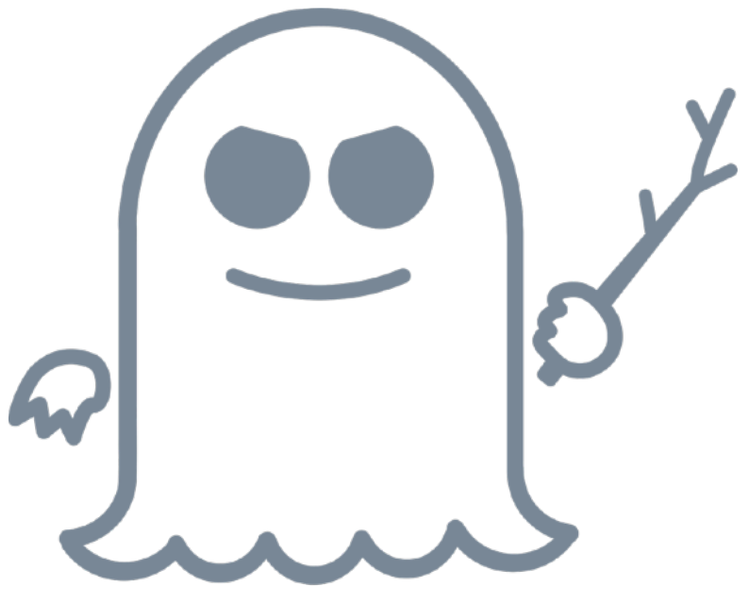
Spectre V2



Ret2spec



NetSpectre



Spectre V4

Countermeasures (SPECTRE V1)

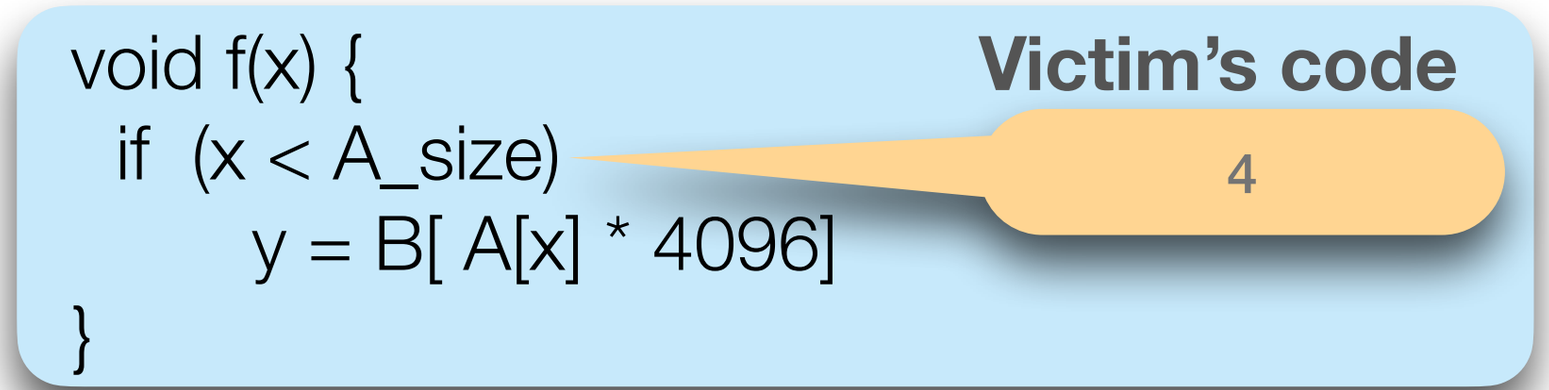
Countermeasures SPECTRE v1

Countermeasures SPECTRE v1

- Disable speculative execution in CPU

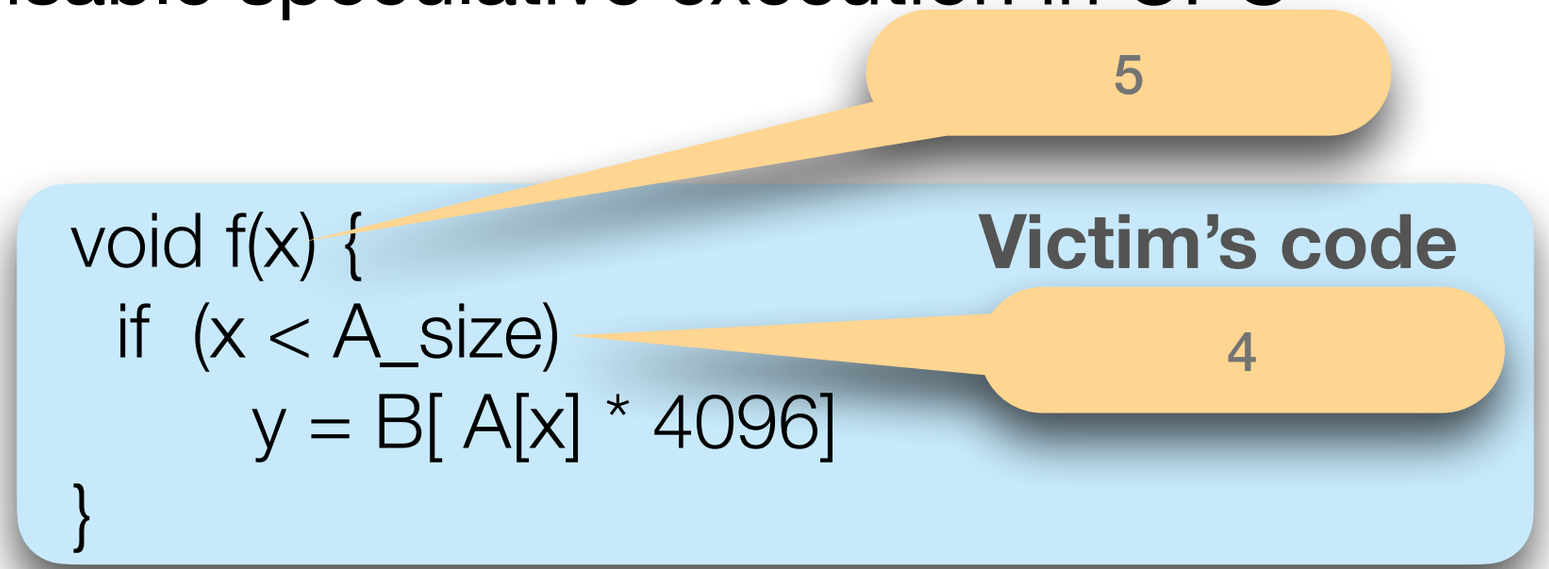
Countermeasures SPECTRE v1

- Disable speculative execution in CPU



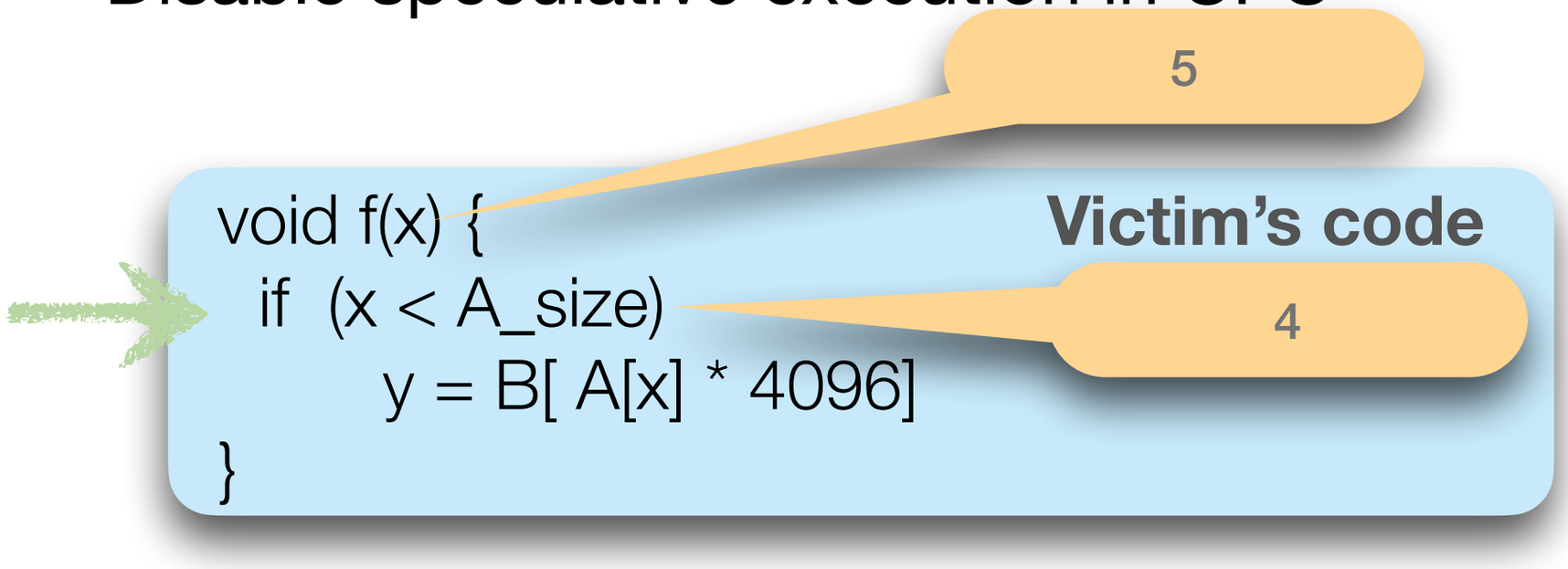
Countermeasures SPECTRE v1

- Disable speculative execution in CPU



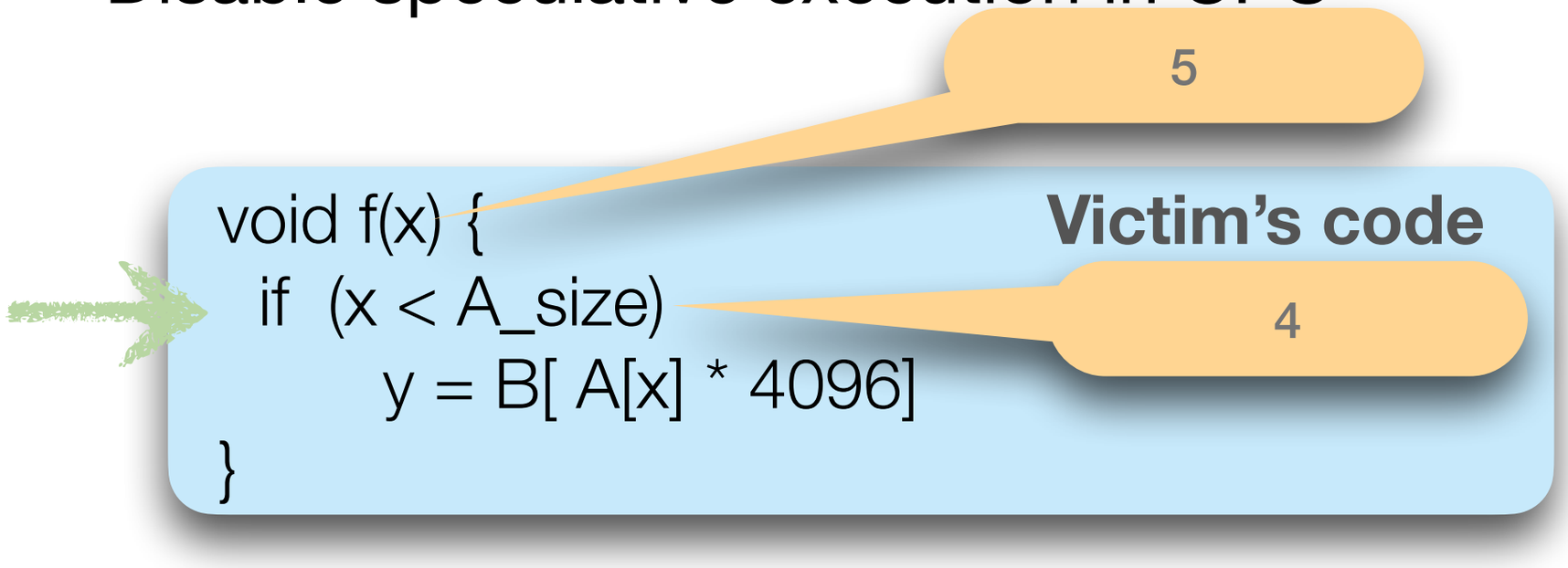
Countermeasures SPECTRE v1

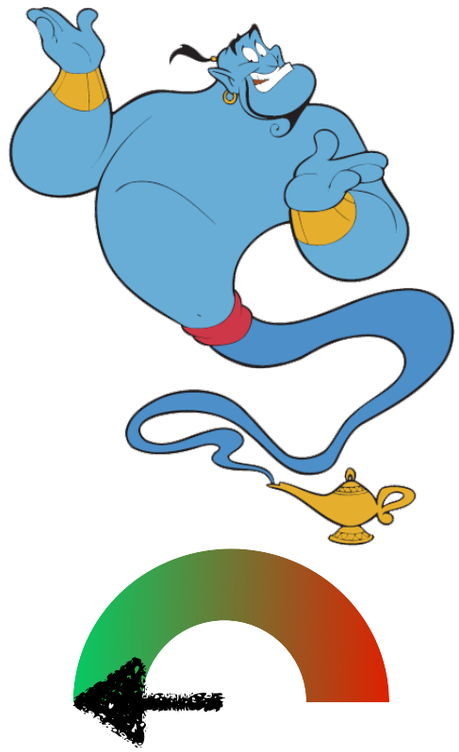
- Disable speculative execution in CPU



Countermeasures SPECTRE v1

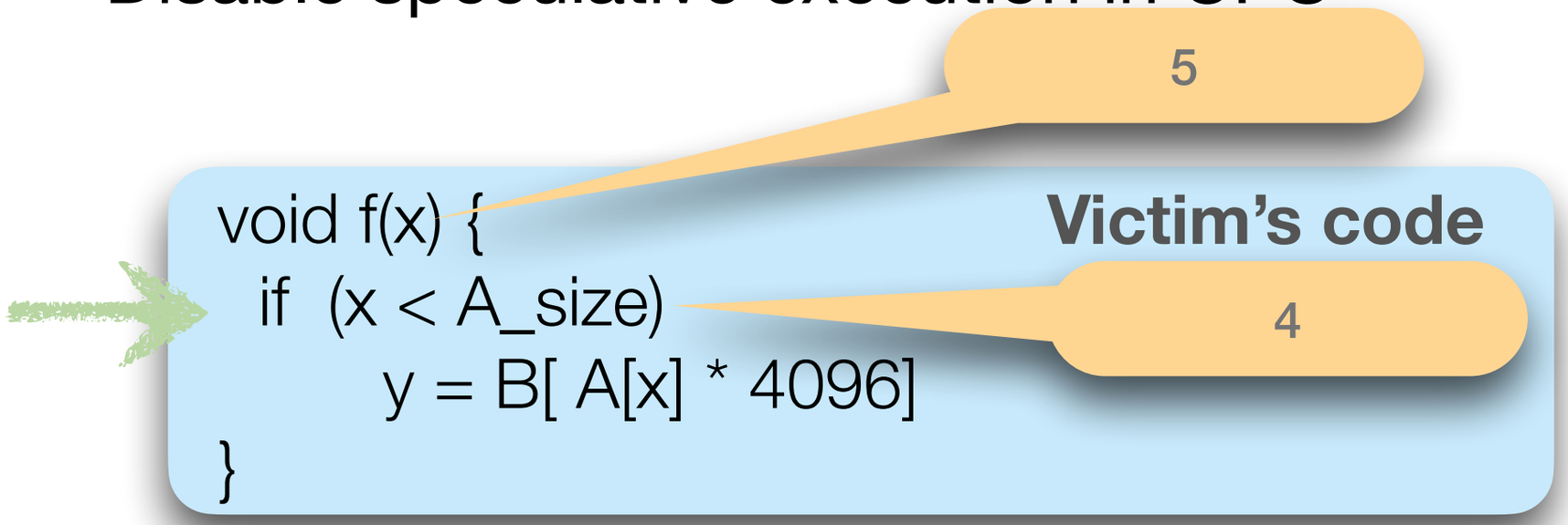
- Disable speculative execution in CPU

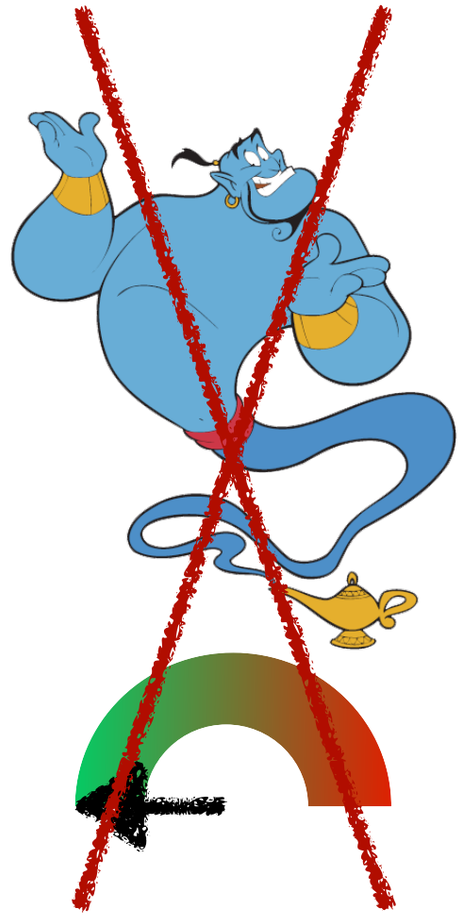




Countermeasures SPECTRE v1

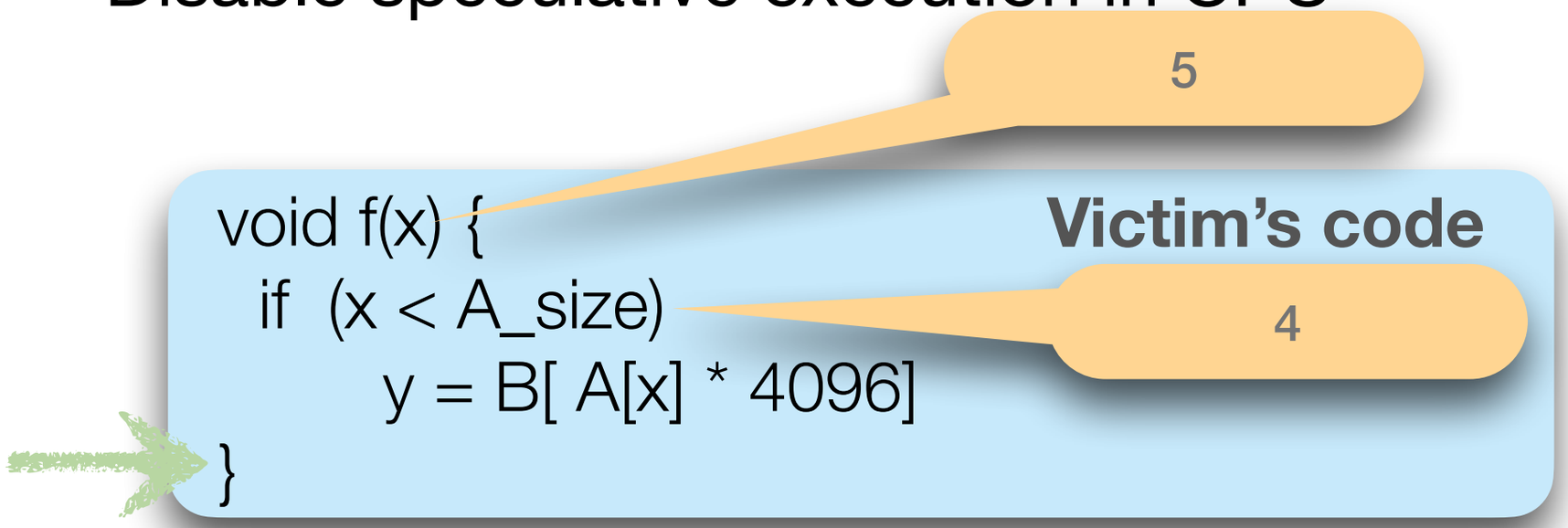
- Disable speculative execution in CPU

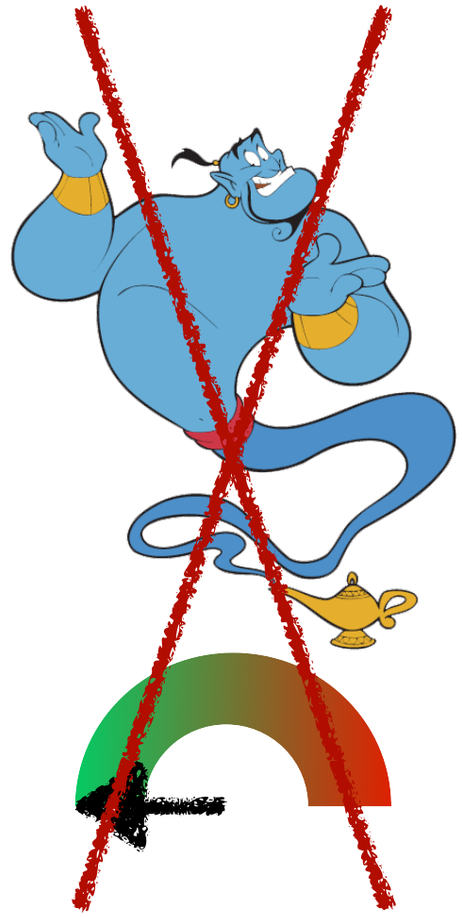




Countermeasures SPECTRE v1

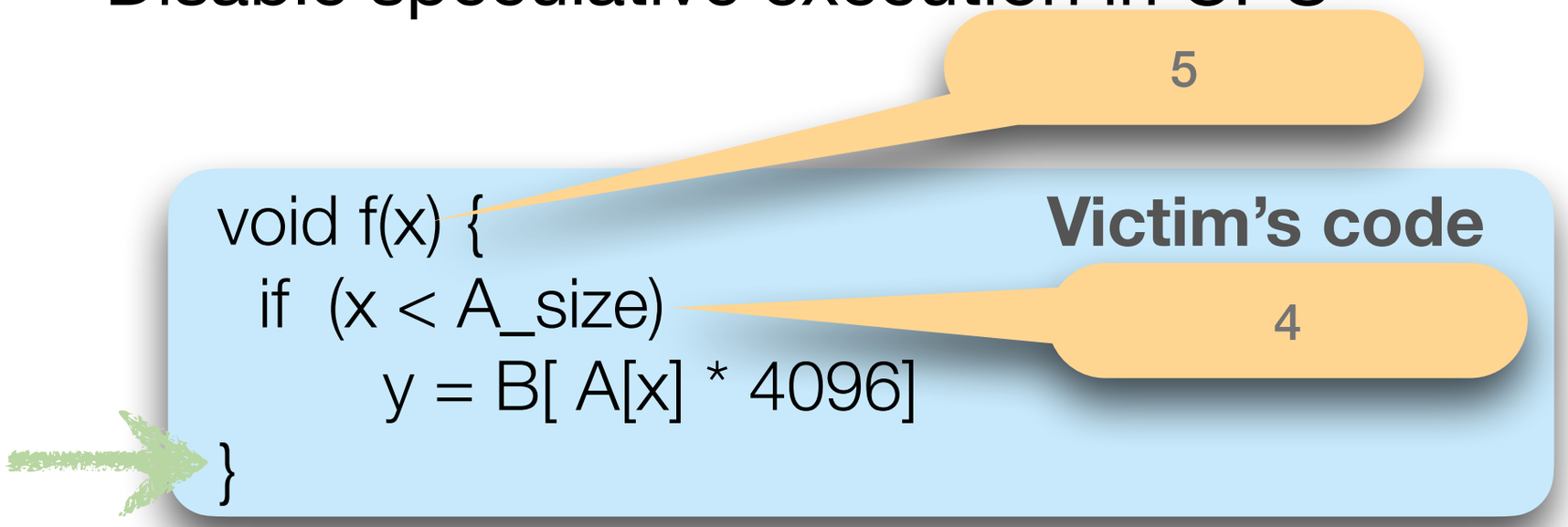
- Disable speculative execution in CPU

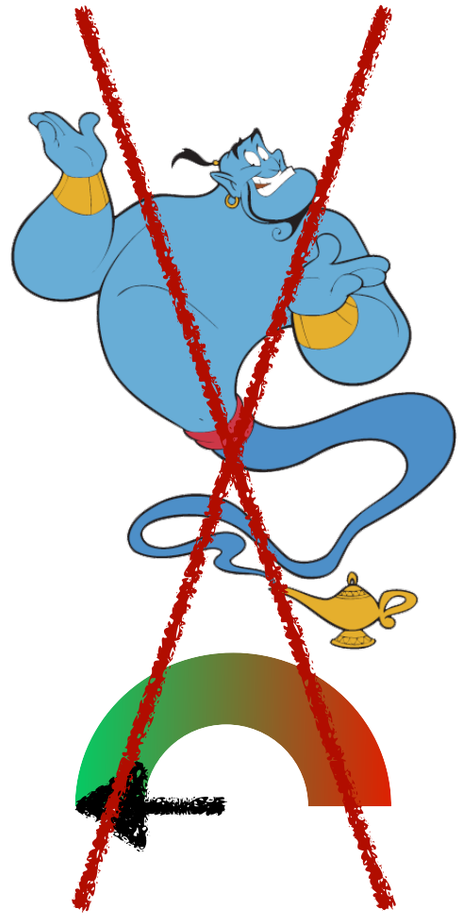




Countermeasures SPECTRE v1

- Disable speculative execution in CPU

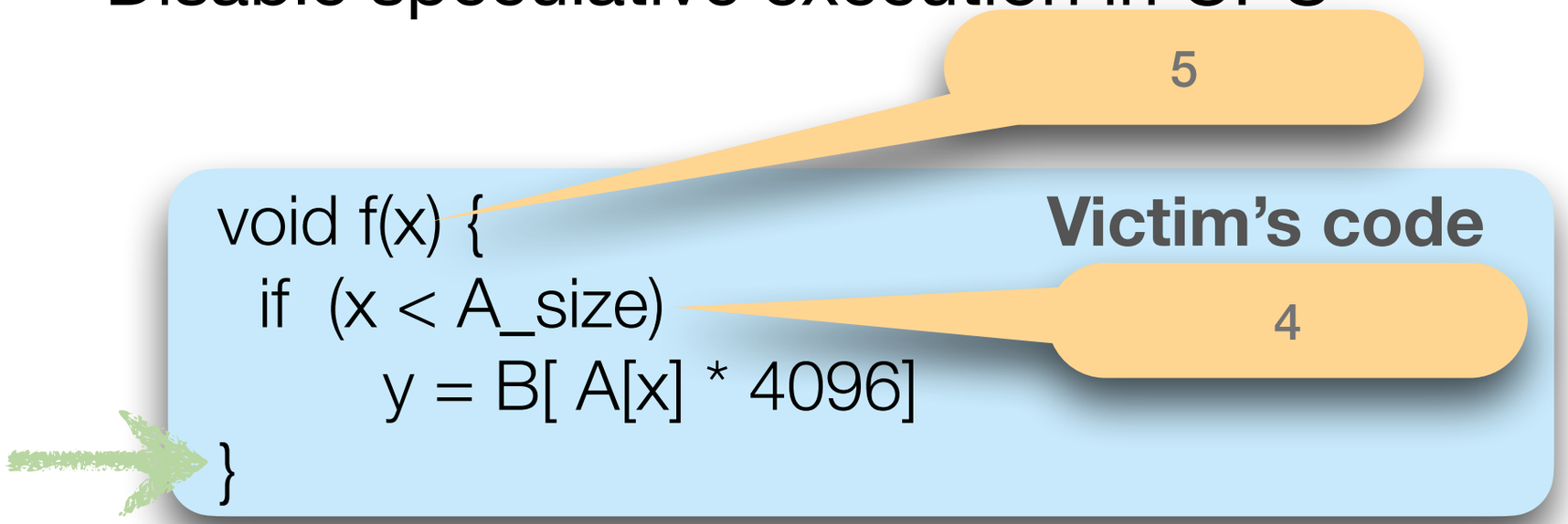


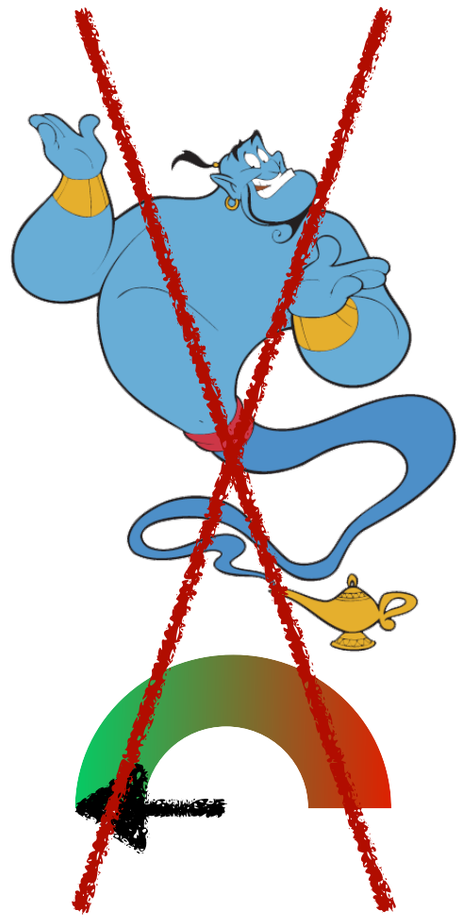


- **SECURE**

Countermeasures SPECTRE v1

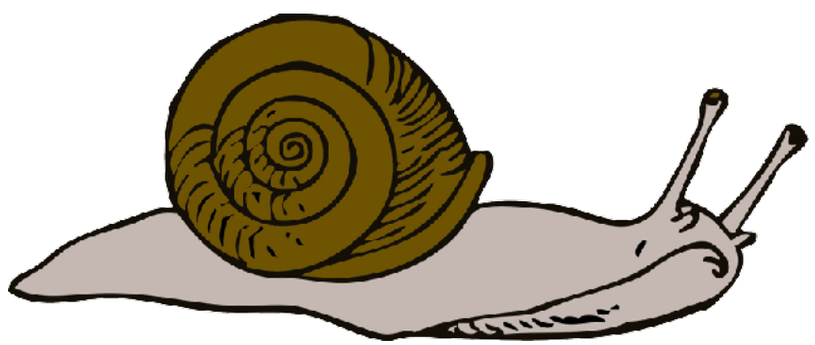
- Disable speculative execution in CPU





- **SECURE**

- **PROBLEM**: Performance



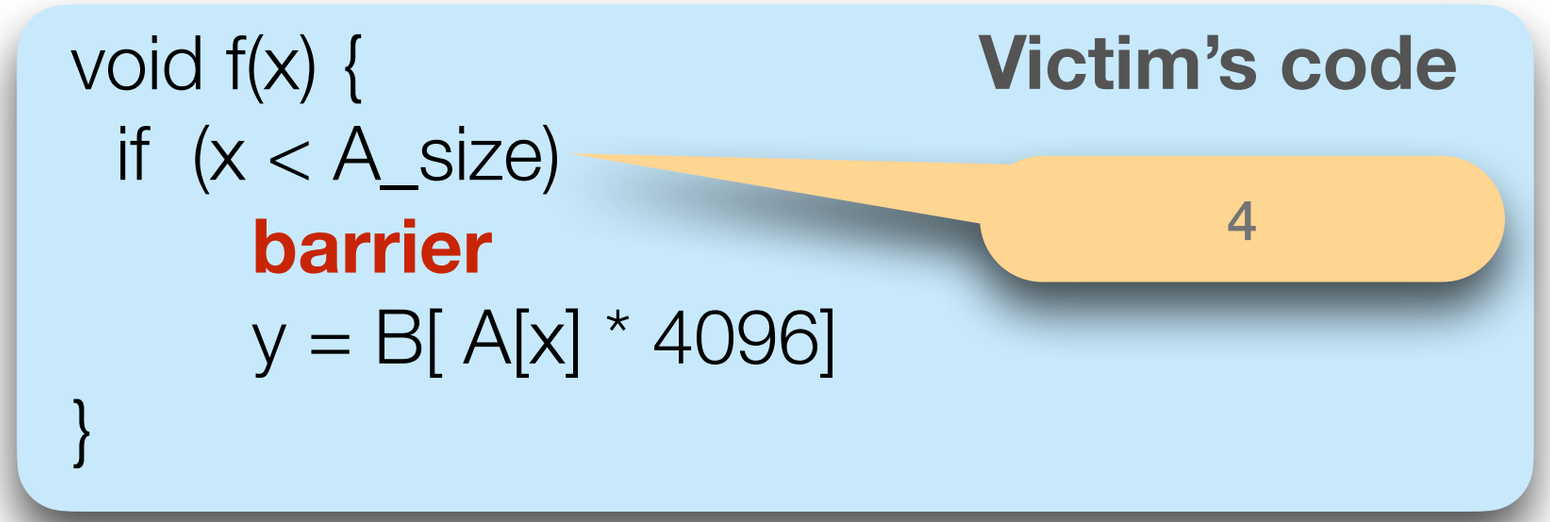
Countermeasures SPECTRE v1

Countermeasures SPECTRE v1

- Speculation barriers

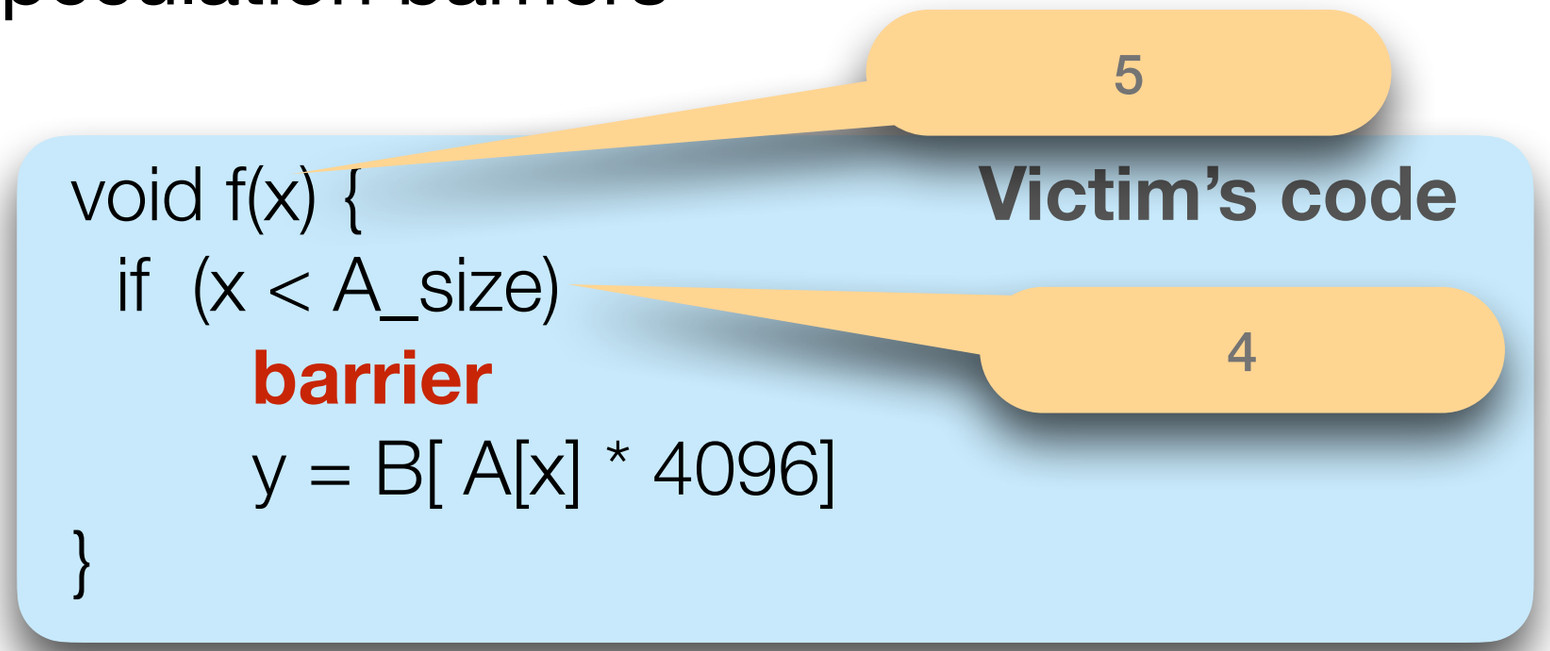
Countermeasures SPECTRE v1

- Speculation barriers



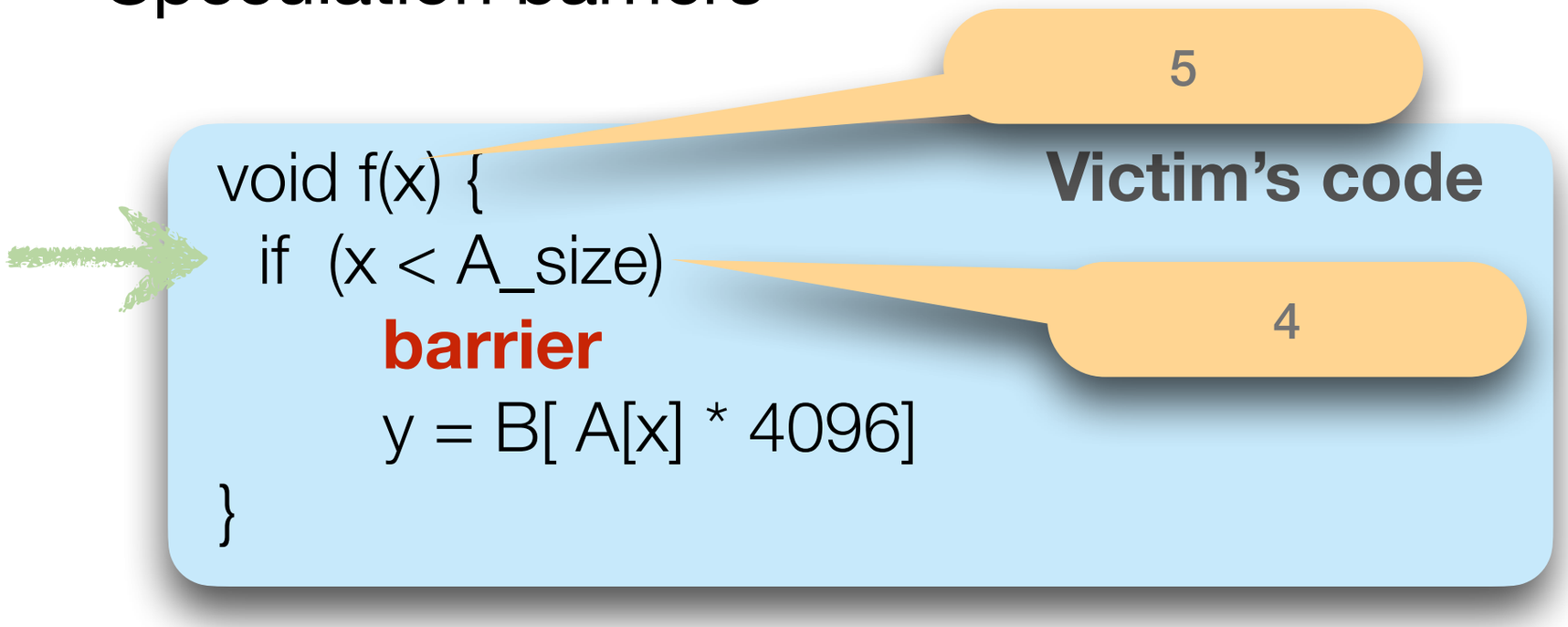
Countermeasures SPECTRE v1

- Speculation barriers



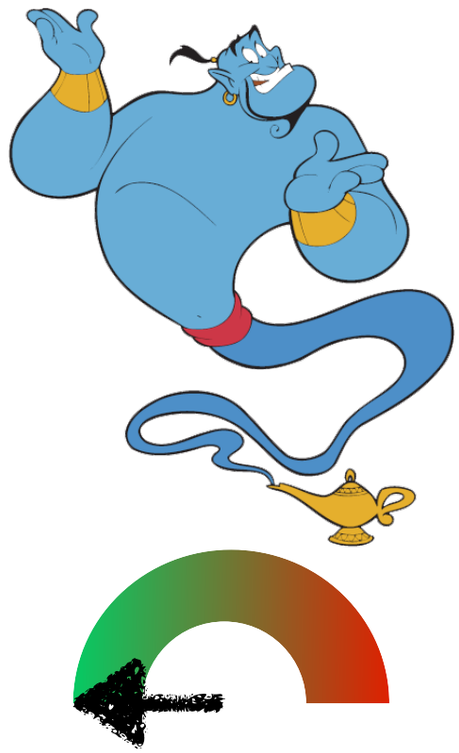
Countermeasures SPECTRE v1

- Speculation barriers



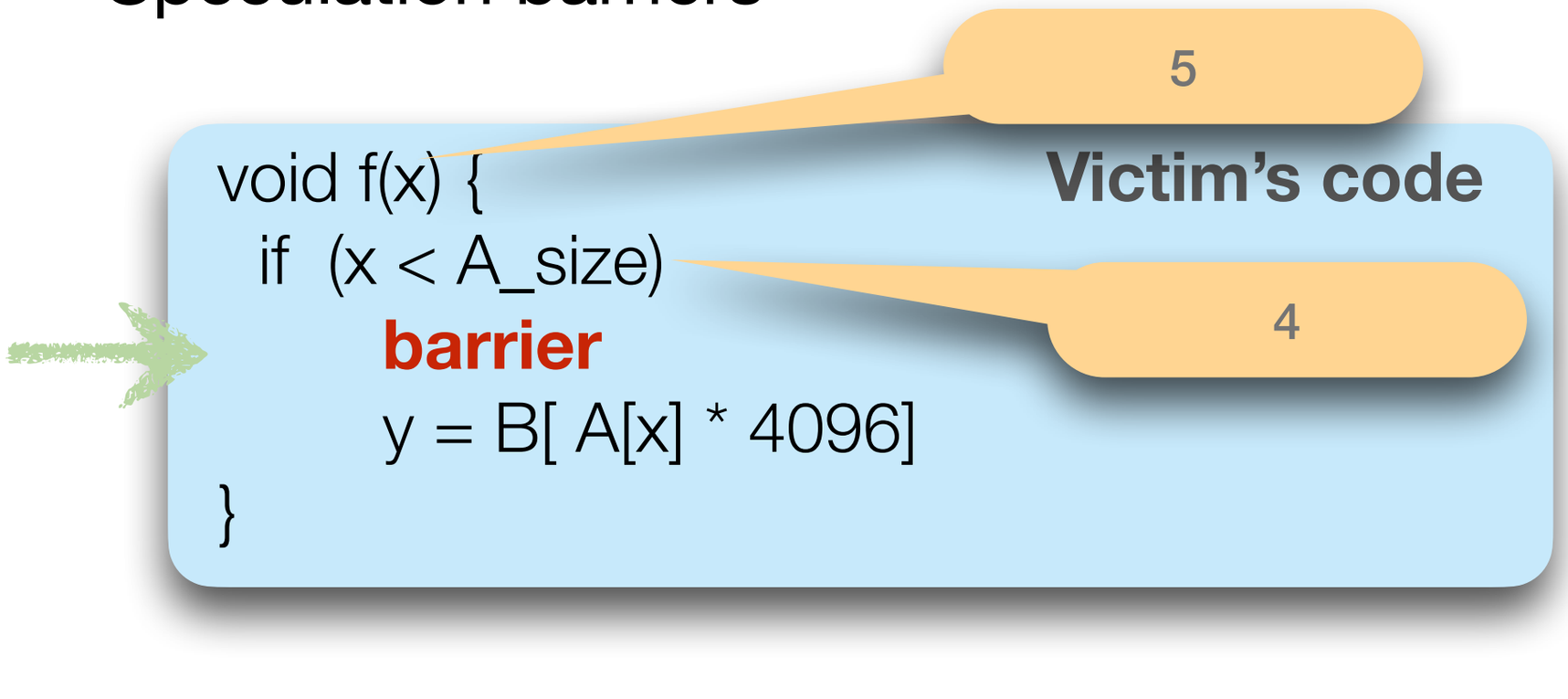
Countermeasures SPECTRE v1

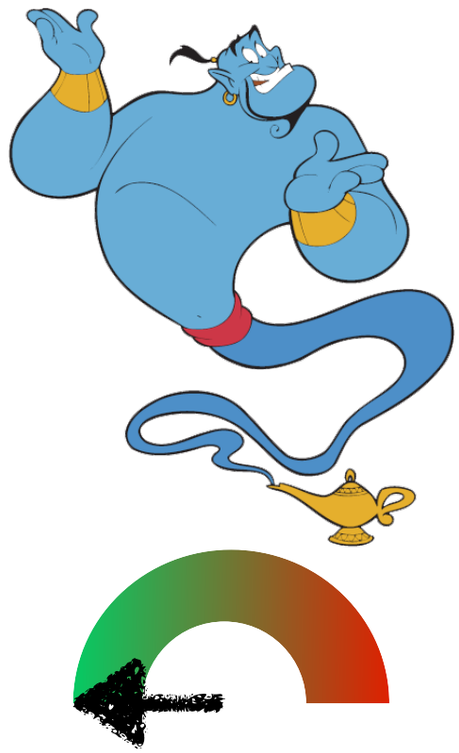
- Speculation barriers



Countermeasures SPECTRE v1

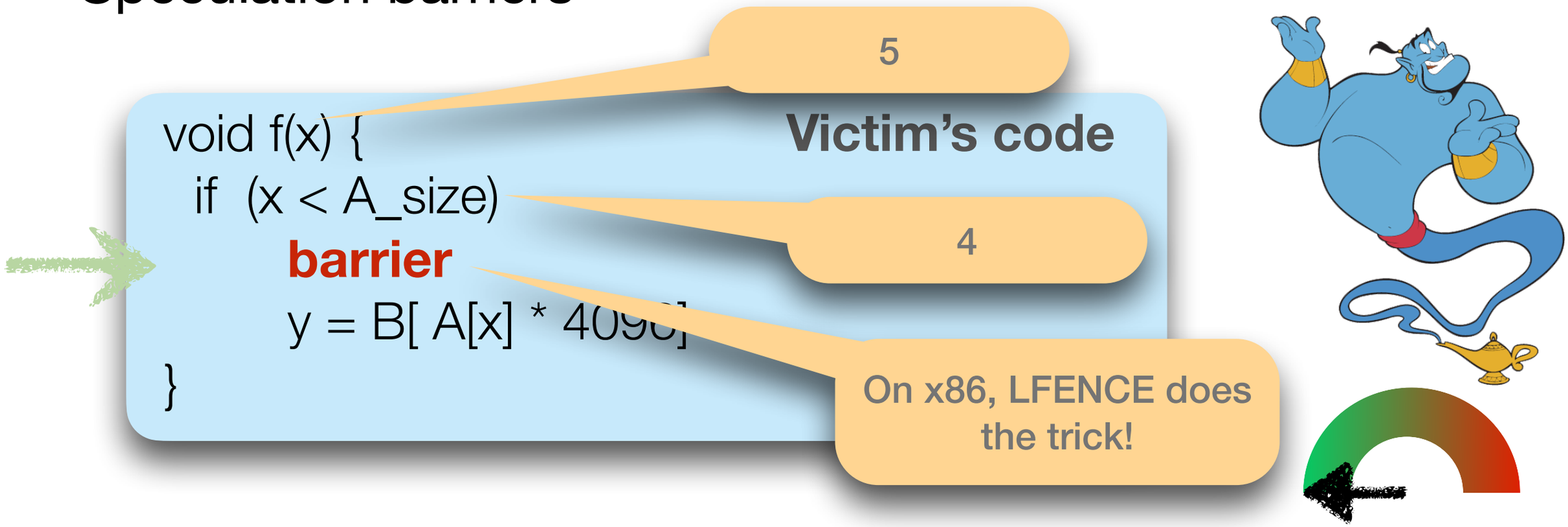
- Speculation barriers





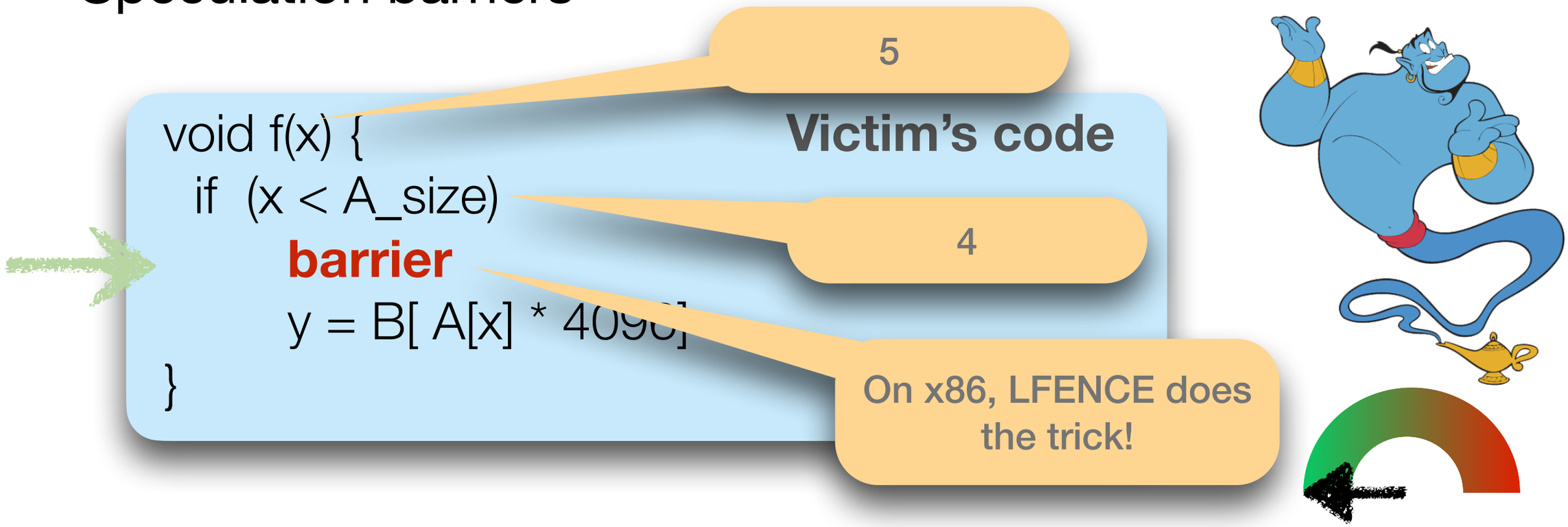
Countermeasures SPECTRE v1

- Speculation barriers



Countermeasures SPECTRE v1

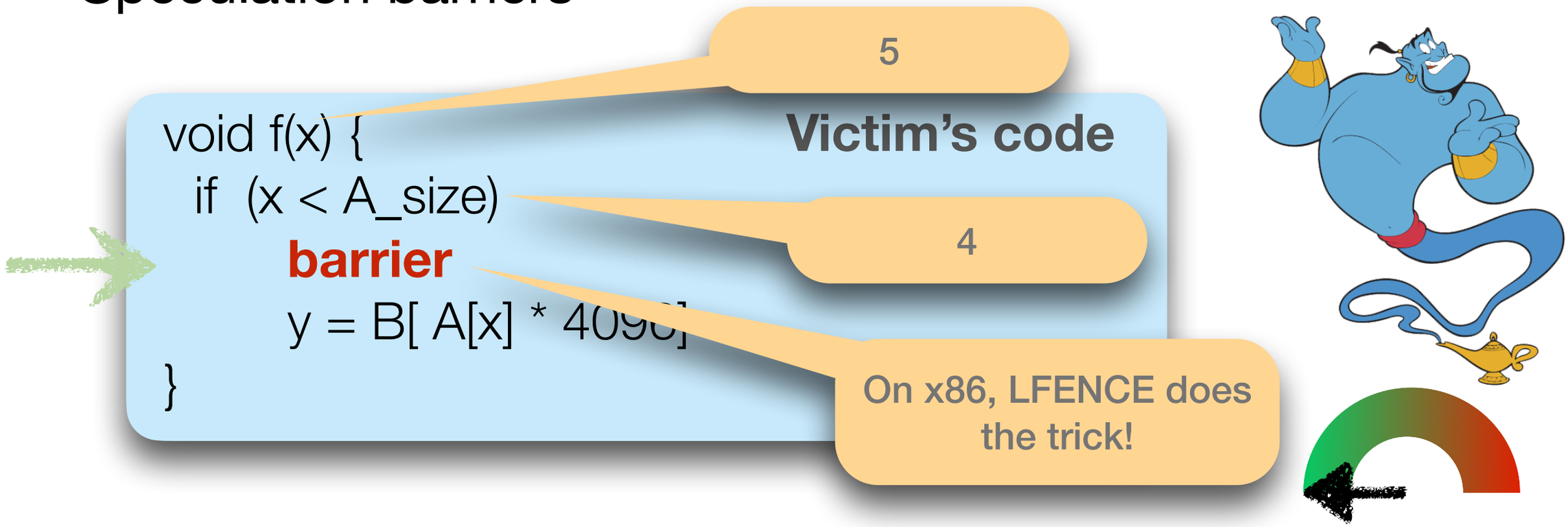
- Speculation barriers



- **PROBLEM**: How do we place barriers?

Countermeasures SPECTRE v1

- Speculation barriers



- **PROBLEM**: How do we place barriers?
  - Trade-off between security and performance

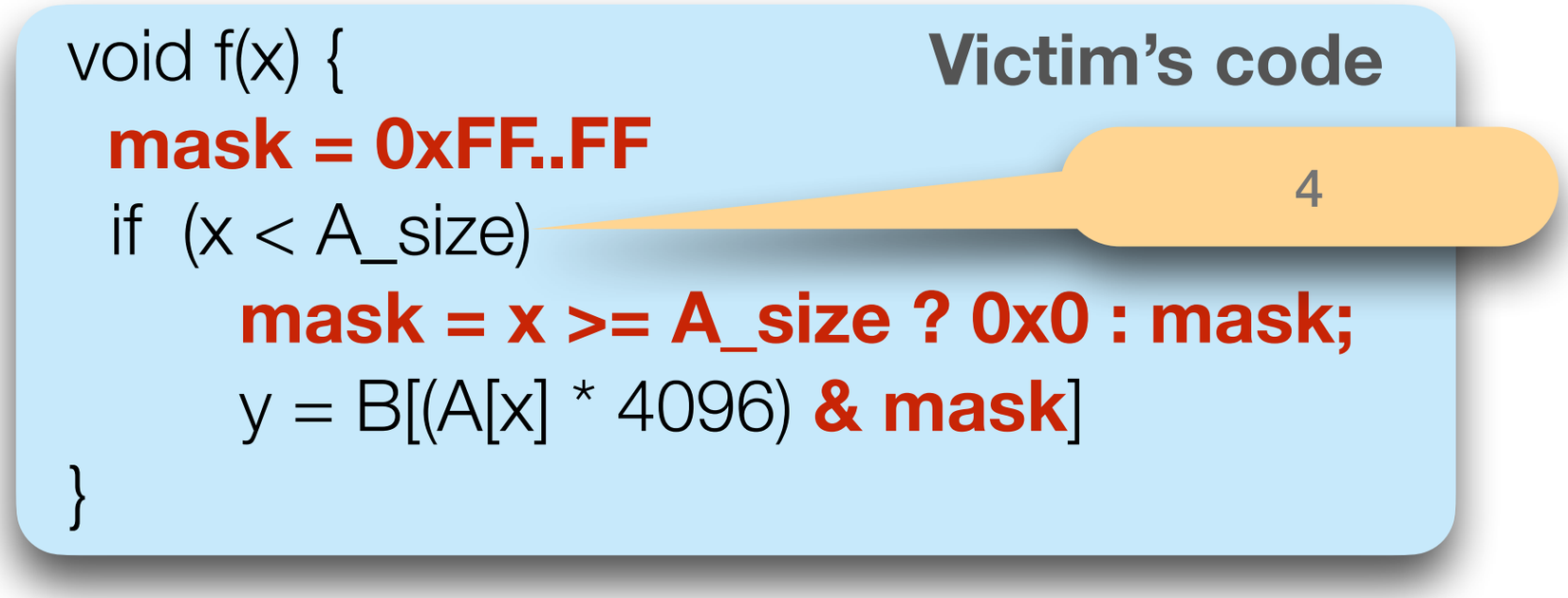
Countermeasures SPECTRE v1

Countermeasures SPECTRE v1

- Speculative load hardening

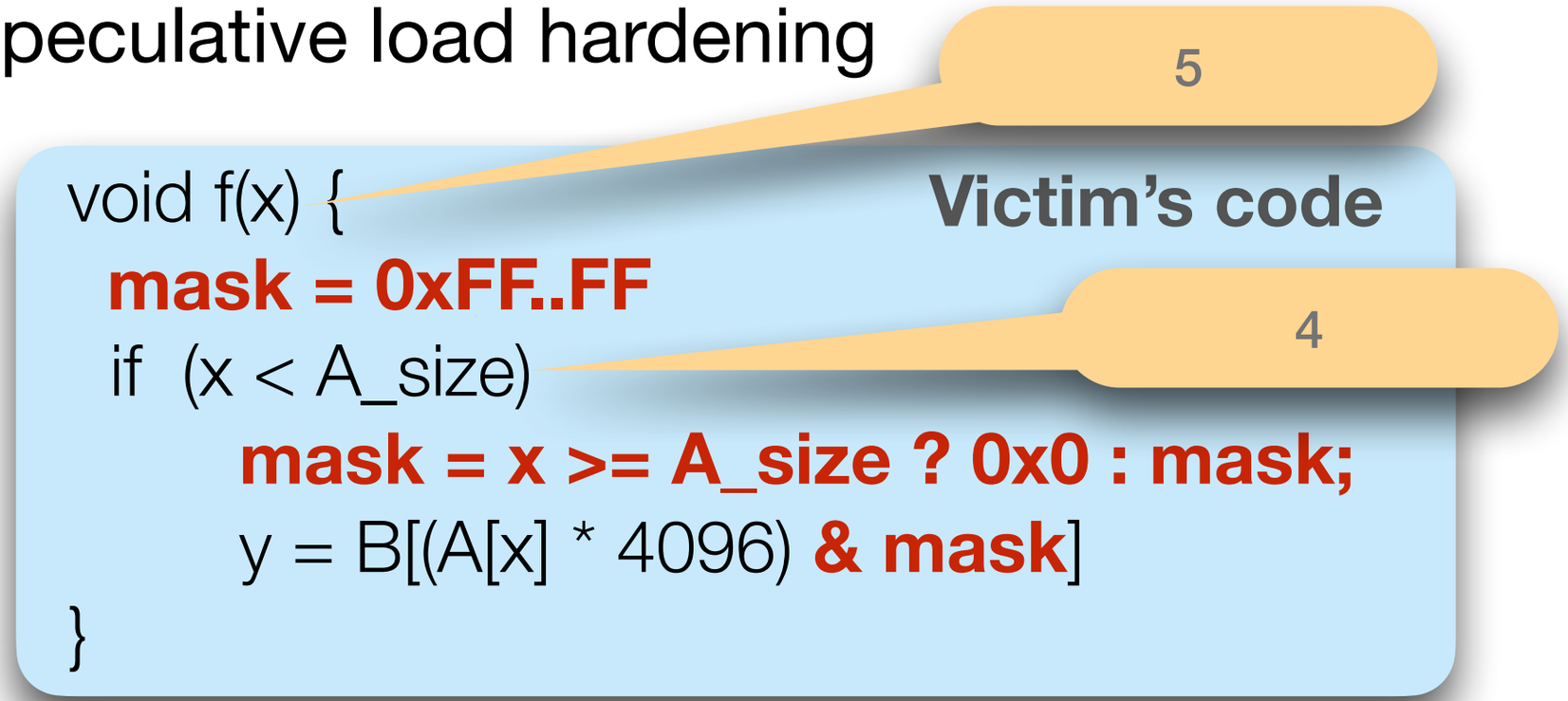
Countermeasures SPECTRE v1

- Speculative load hardening



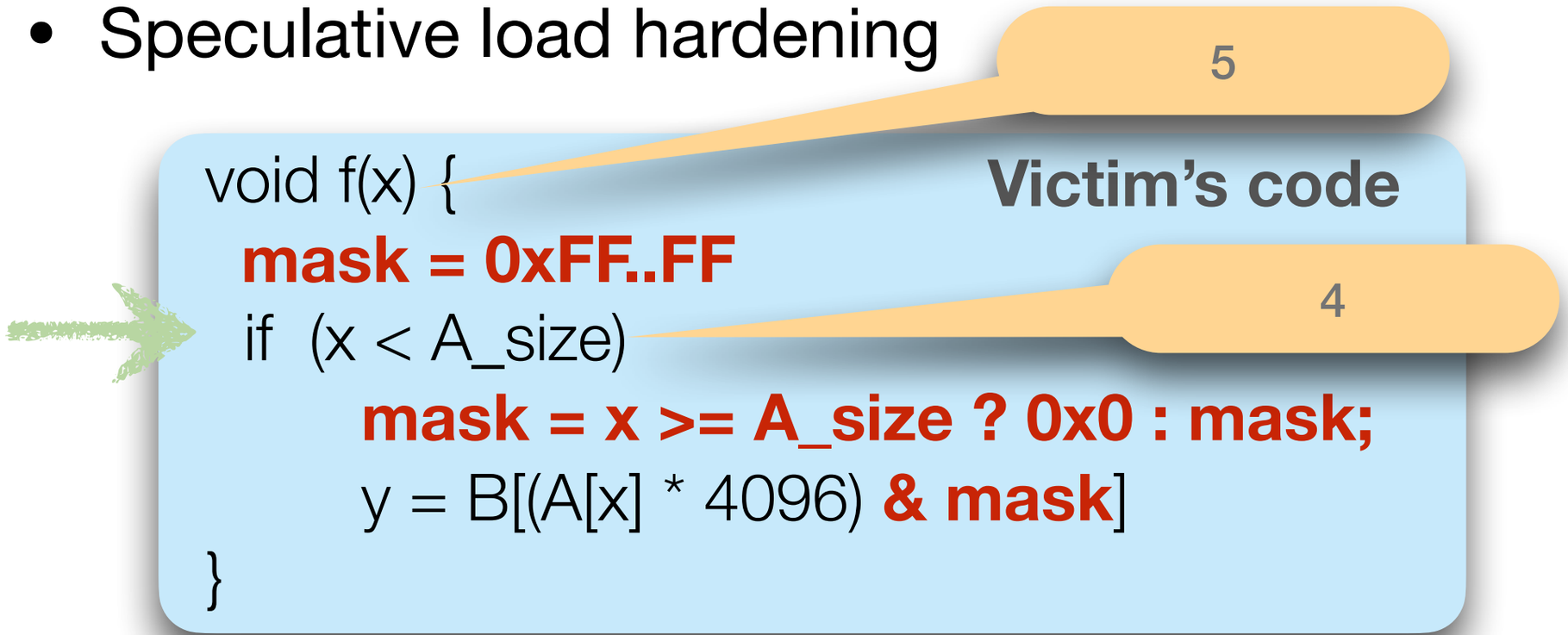
Countermeasures SPECTRE v1

- Speculative load hardening



Countermeasures SPECTRE v1

- Speculative load hardening



Countermeasures SPECTRE v1

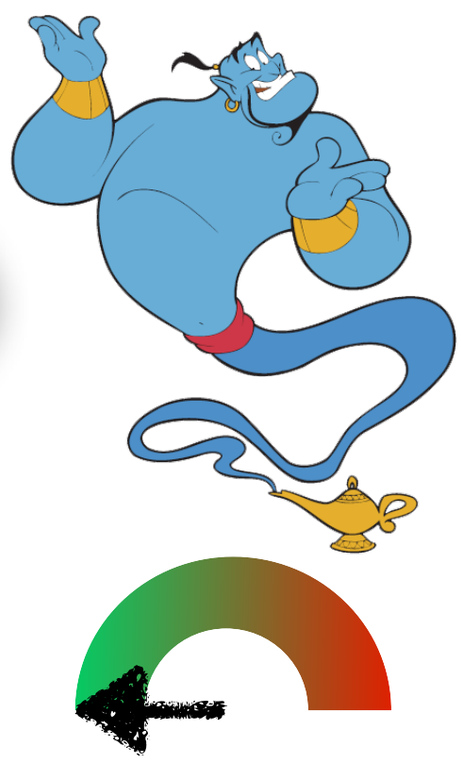
- Speculative load hardening

5

Victim's code

```
void f(x) {
  mask = 0xFF..FF
  if  (x < A_size)
      mask = x >= A_size ? 0x0 : mask;
      y = B[(A[x] * 4096) & mask]
}
```

4



Countermeasures SPECTRE v1

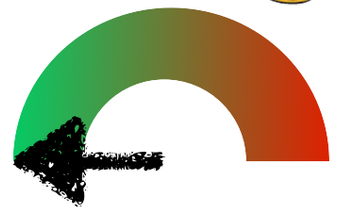
- Speculative load hardening

5

Victim's code

```
void f(x) {
  mask = 0xFF..FF
  if  (x < A_size)
      mask = x >= A_size ? 0x0 : mask;
      y = B[(A[x] * 4096) & mask]
}
```

4



Countermeasures SPECTRE v1

- Speculative load hardening

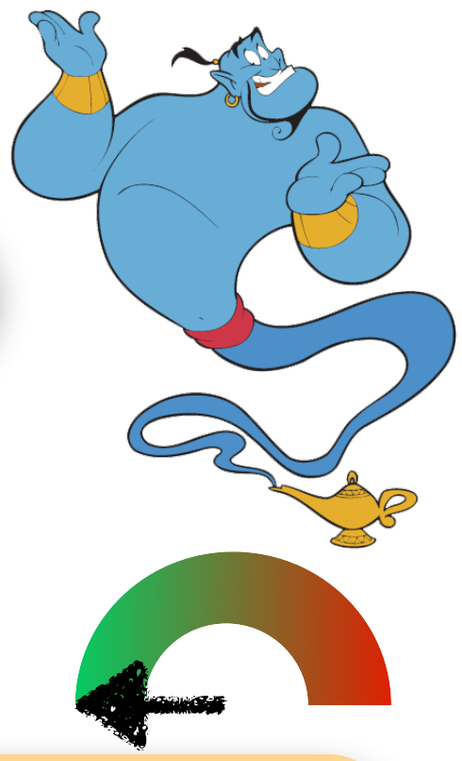
```
void f(x) {
  mask = 0xFF..FF
  if  (x < A_size)
      mask = x >= A_size ? 0x0 : mask;
      y = B[(A[x] * 4096) & mask]
}
```

5

Victim's code

4

Blocks the leak into cache



Countermeasures SPECTRE v1

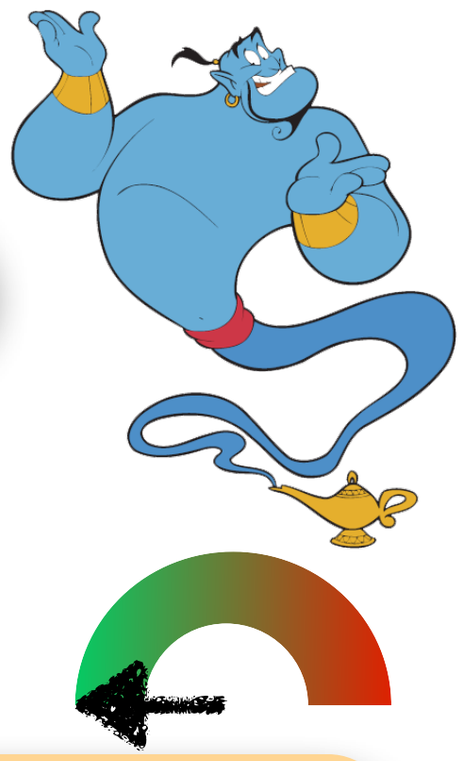
- Speculative load hardening

5

Victim's code

```
void f(x) {
  mask = 0xFF..FF
  if  (x < A_size)
      mask = x >= A_size ? 0x0 : mask;
      y = B[(A[x] * 4096) & mask]
}
```

4



Blocks the leak into cache

- **PROBLEM**:
  - How can we find the code to harden?

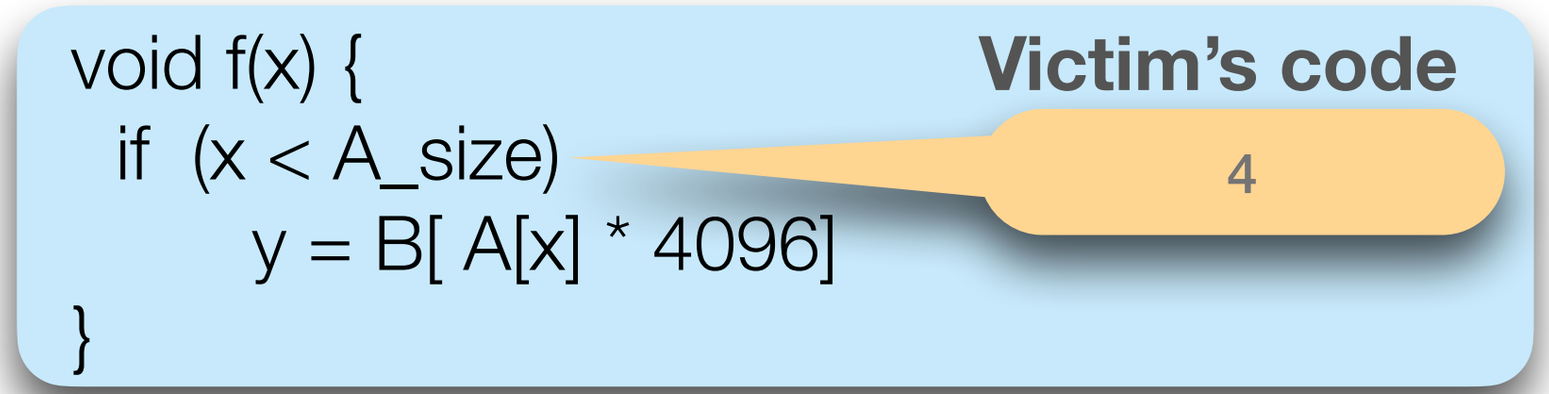
Countermeasures SPECTRE v1

Countermeasures SPECTRE v1

- Roll-back micro-architectural state when speculative execution aborts

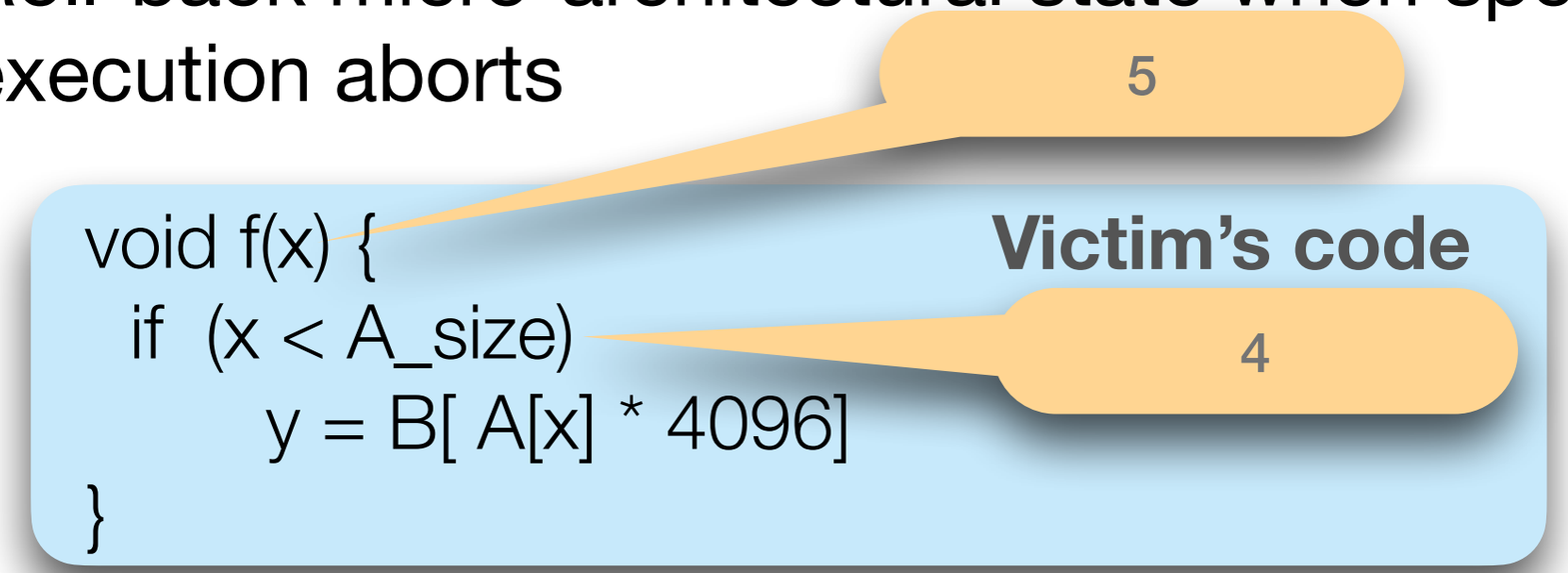
Countermeasures SPECTRE v1

- Roll-back micro-architectural state when speculative execution aborts



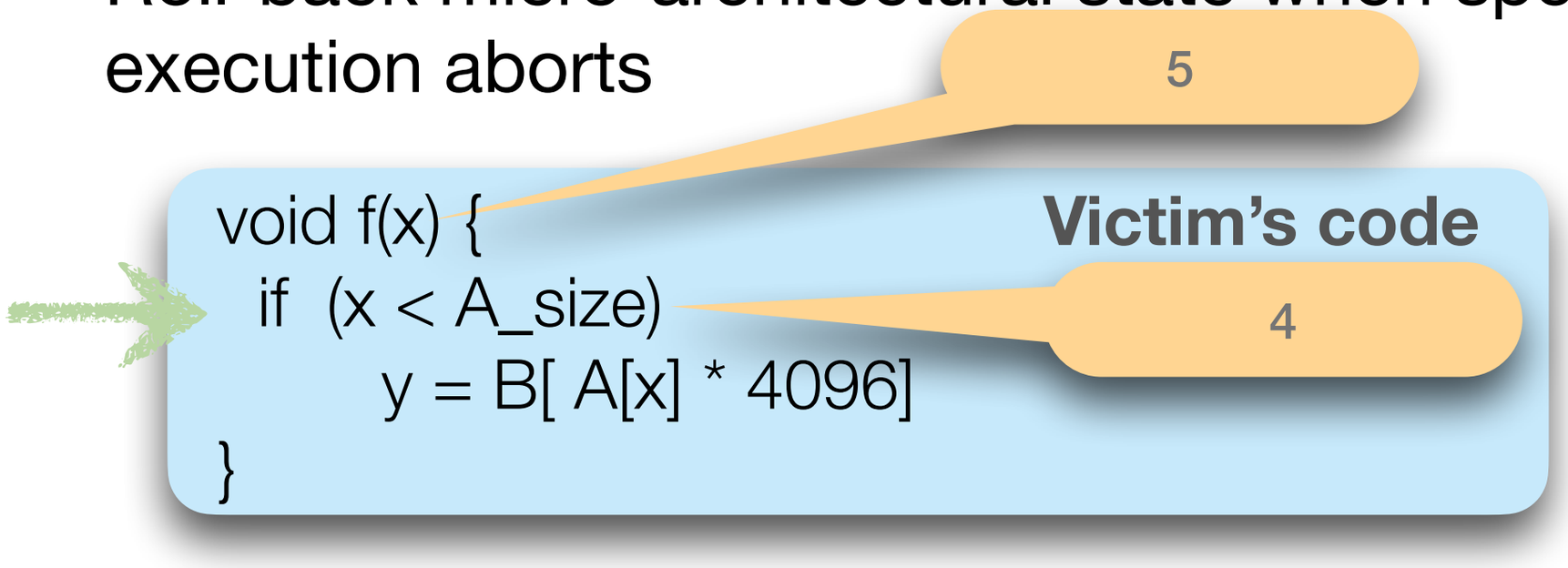
Countermeasures SPECTRE v1

- Roll-back micro-architectural state when speculative execution aborts



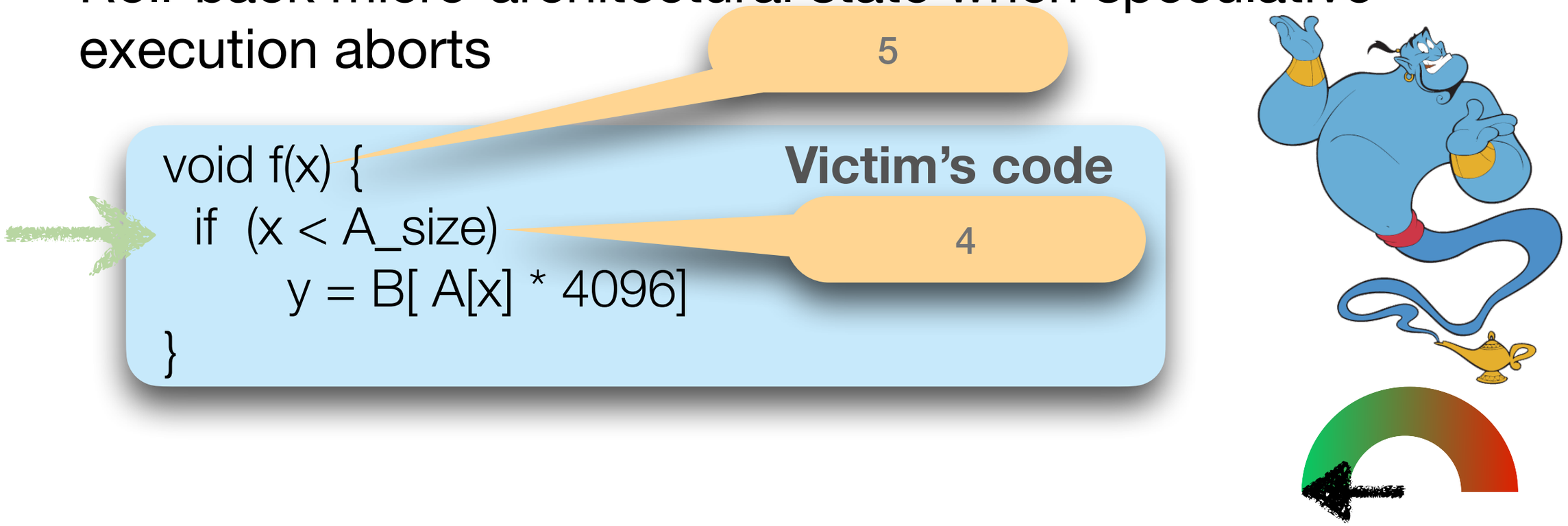
Countermeasures SPECTRE v1

- Roll-back micro-architectural state when speculative execution aborts



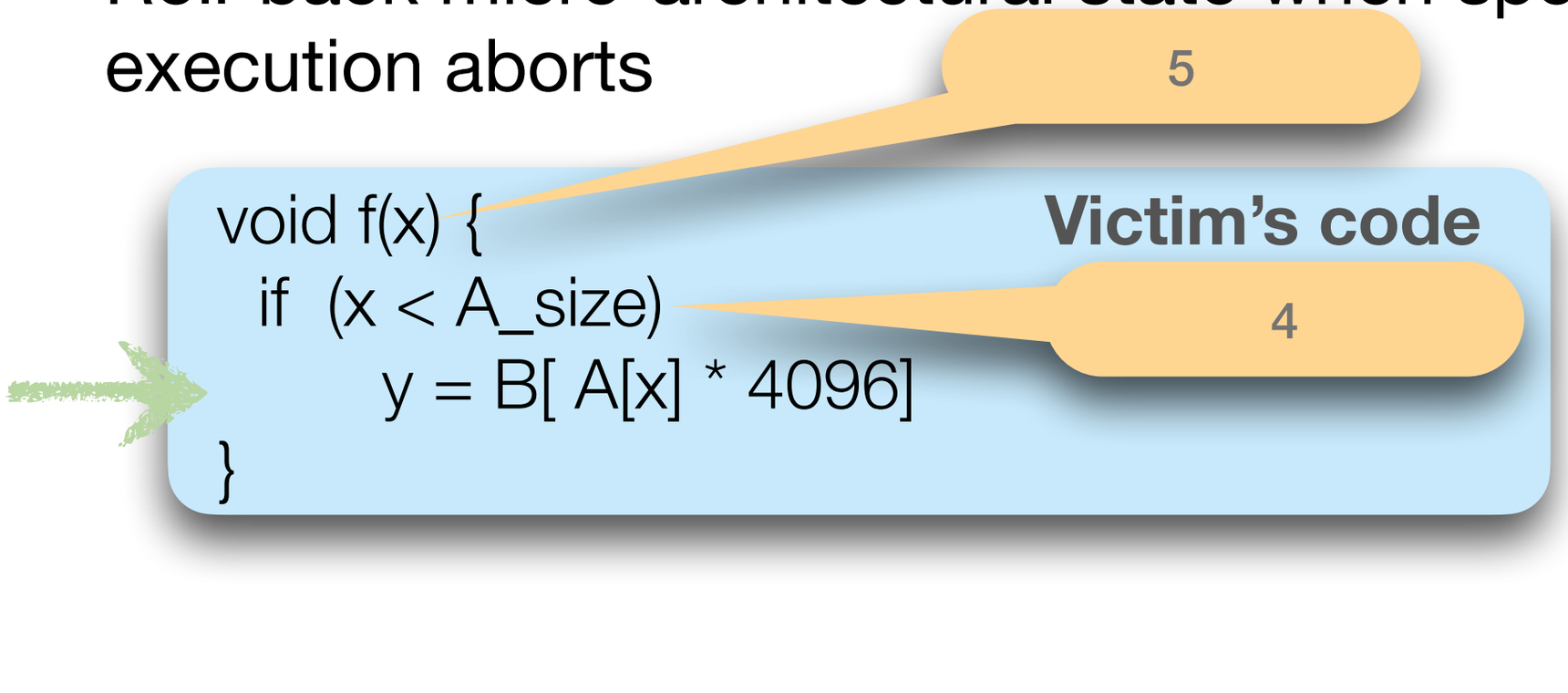
Countermeasures SPECTRE v1

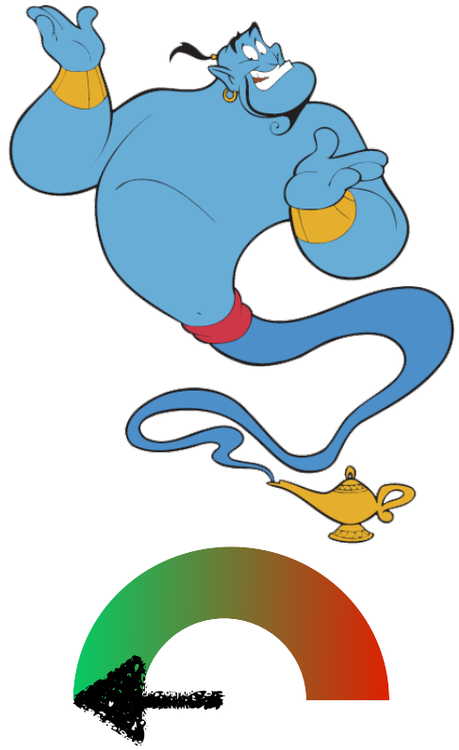
- Roll-back micro-architectural state when speculative execution aborts



Countermeasures SPECTRE v1

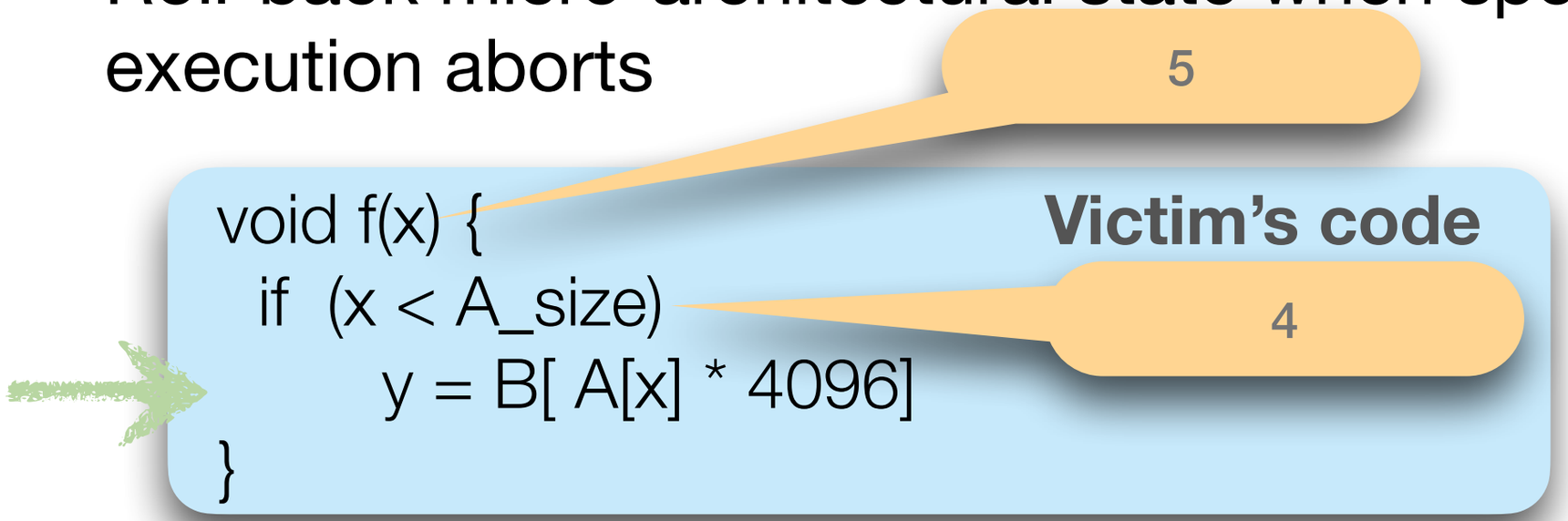
- Roll-back micro-architectural state when speculative execution aborts

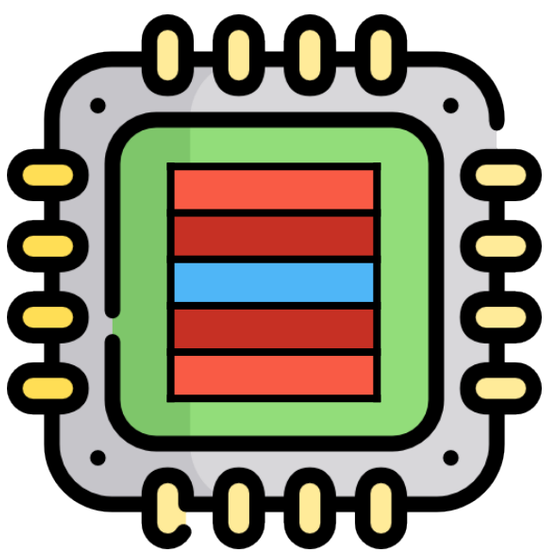




Countermeasures SPECTRE v1

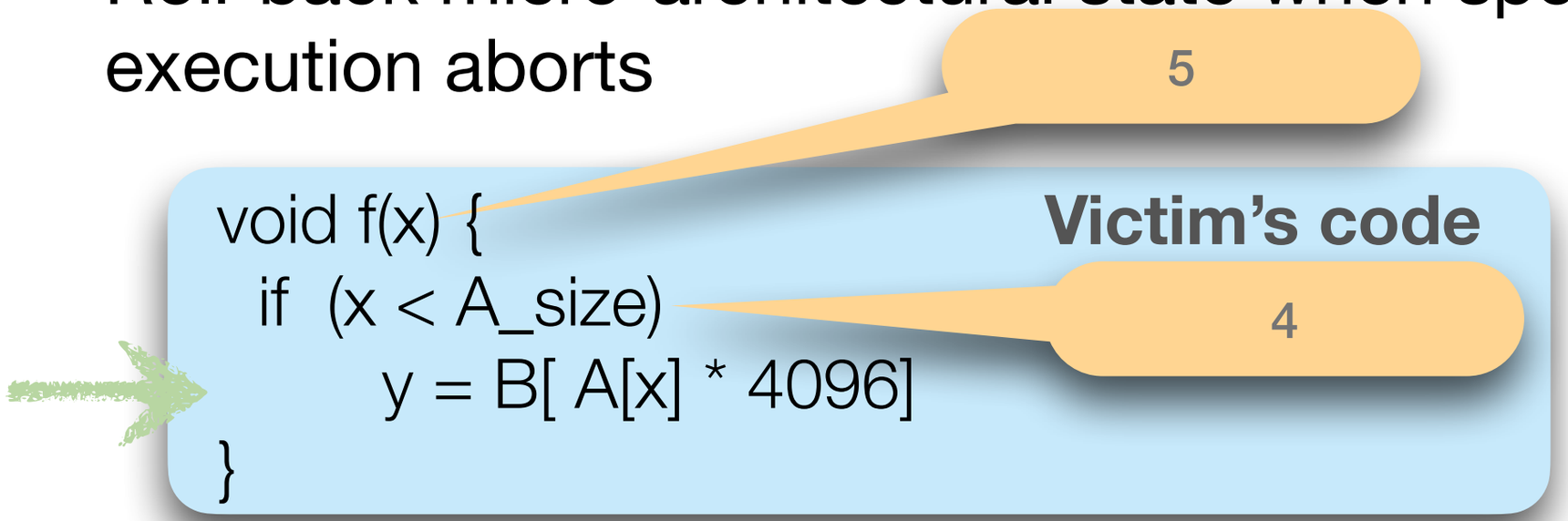
- Roll-back micro-architectural state when speculative execution aborts

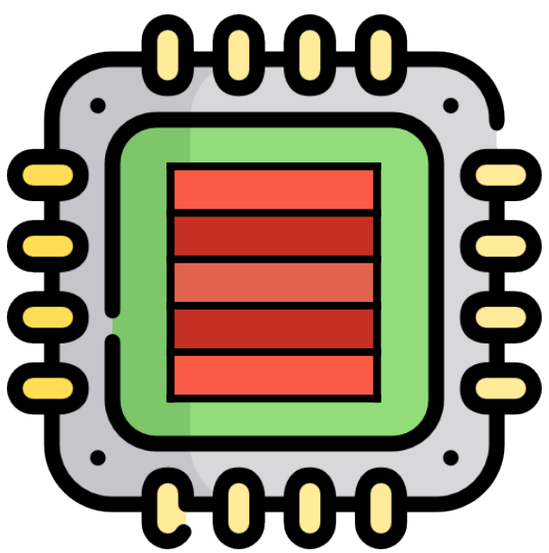




Countermeasures SPECTRE v1

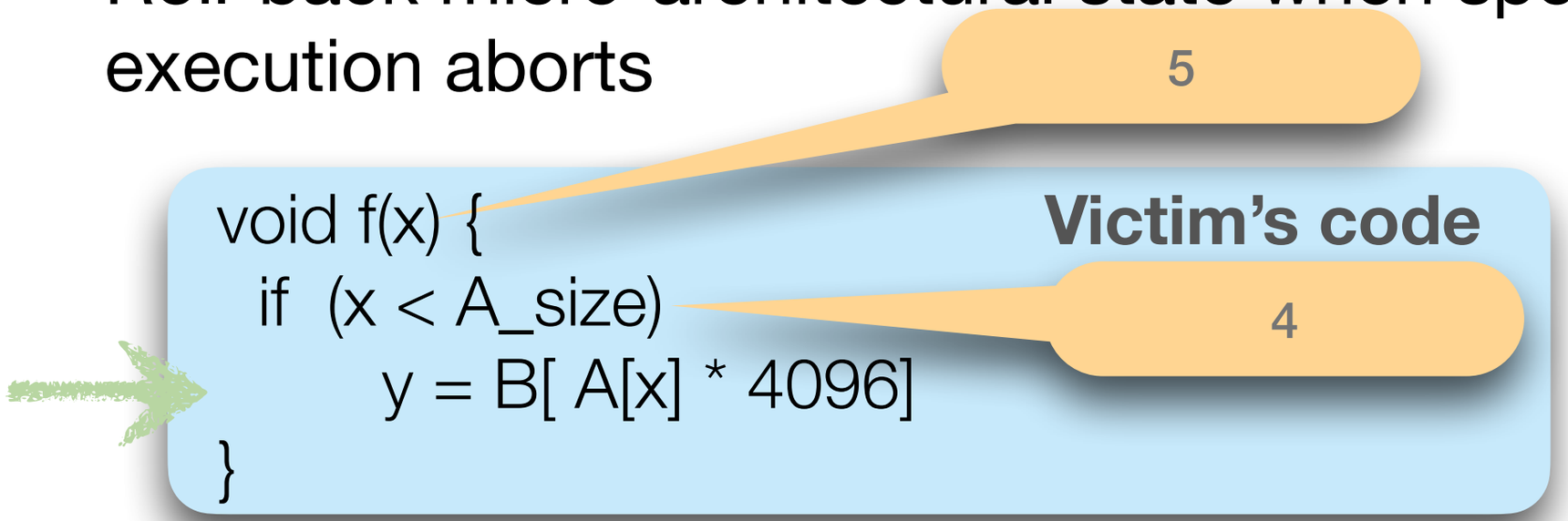
- Roll-back micro-architectural state when speculative execution aborts

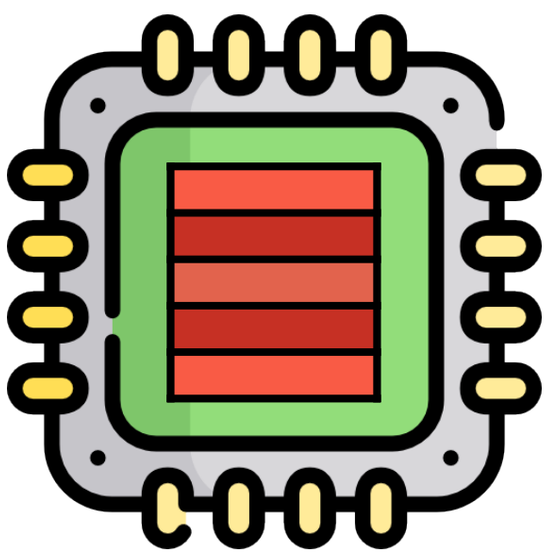




Countermeasures SPECTRE v1

- Roll-back micro-architectural state when speculative execution aborts

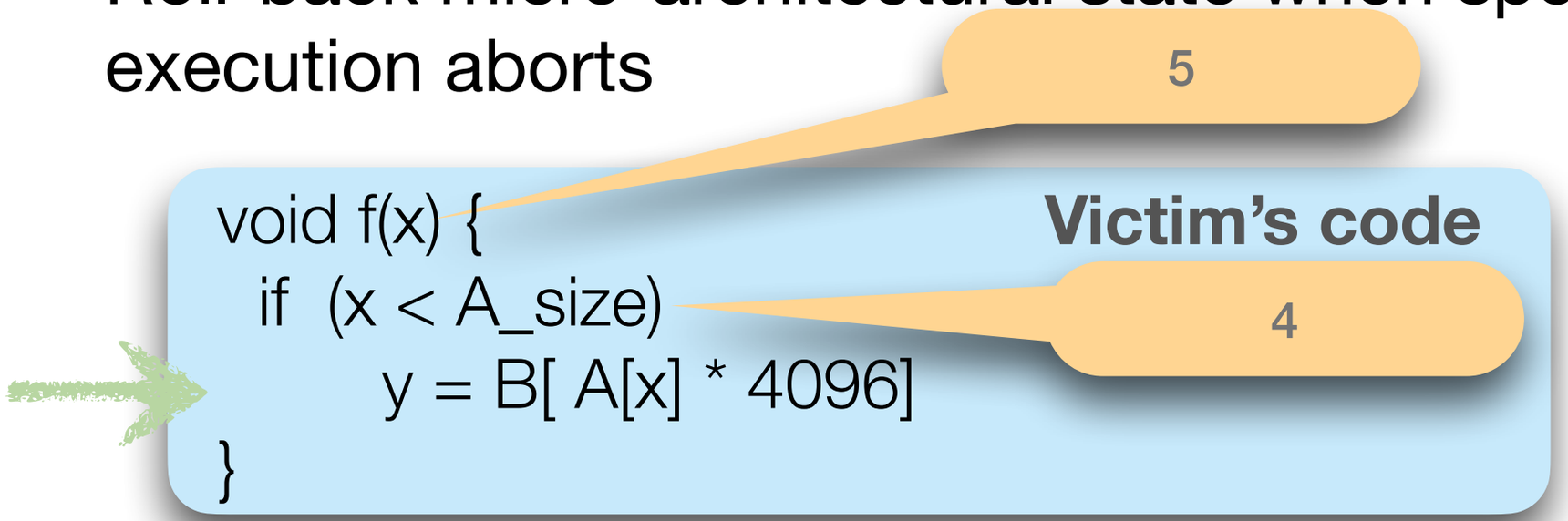


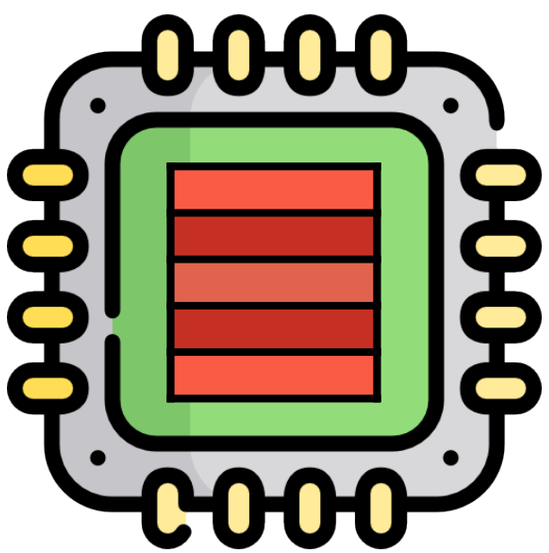


- **PROBLEMS**
  - Requires changes in HW

Countermeasures SPECTRE v1

- Roll-back micro-architectural state when speculative execution aborts

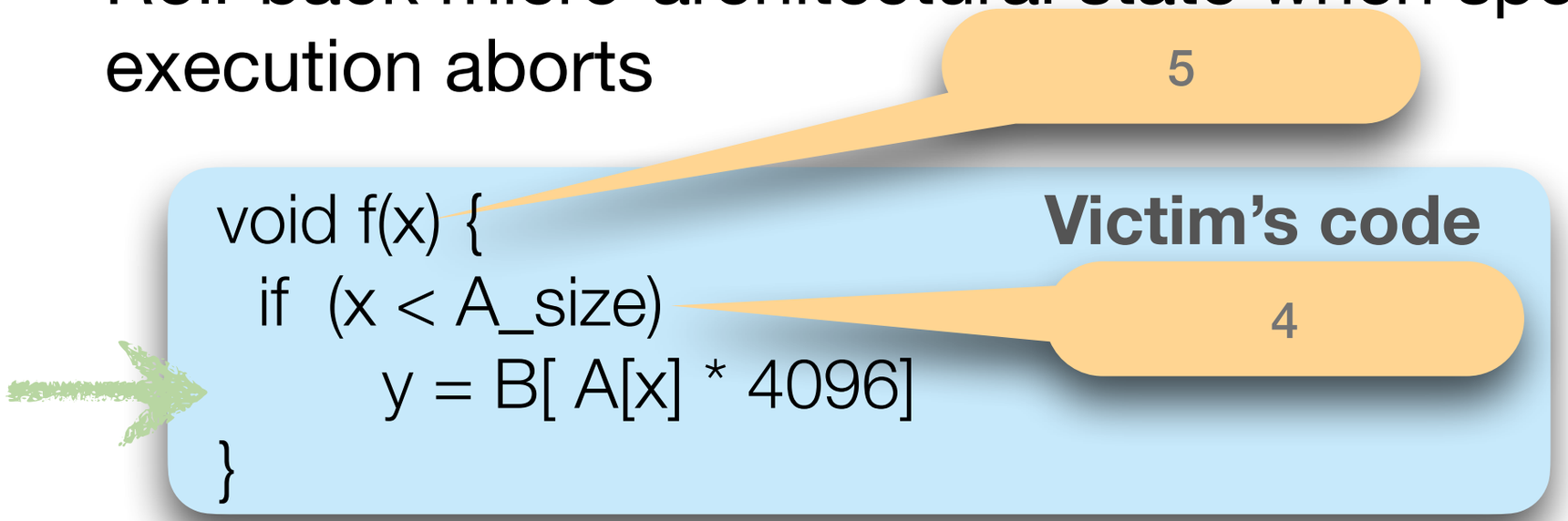


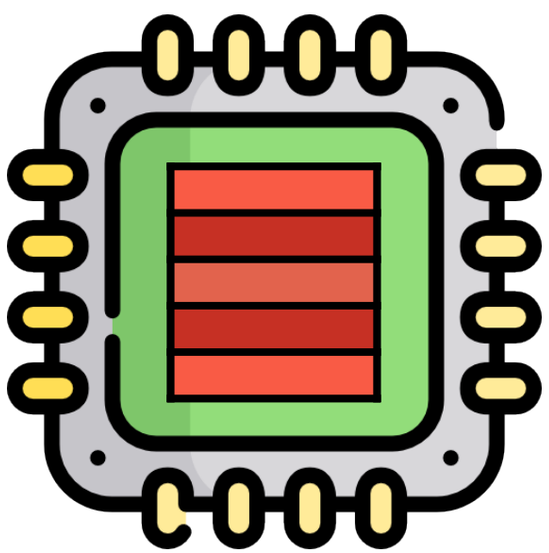


- **PROBLEMS**
  - Requires changes in HW
  - Attacker may observe micro-architectural state during speculation

Countermeasures SPECTRE v1

- Roll-back micro-architectural state when speculative execution aborts





- **PROBLEMS**
  - Requires changes in HW
  - Attacker may observe micro-architectural state during speculation
  - What is micro-architectural state?

SPECTRE again (Variant 2)

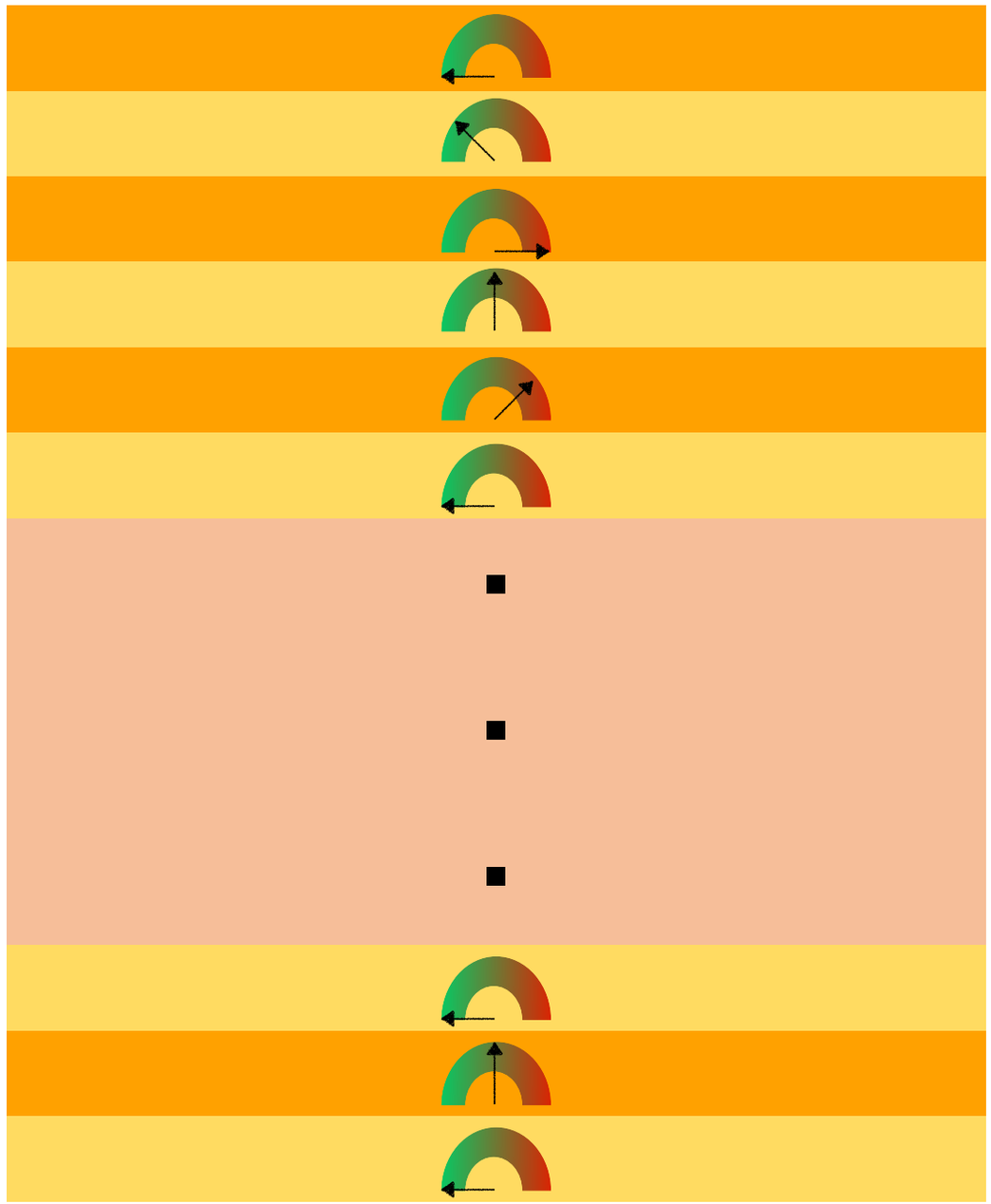
Return of the branch predictor

- Branch history buffer
- Branch target buffer
- Pattern history table
- Global history counter

Branch history buffer

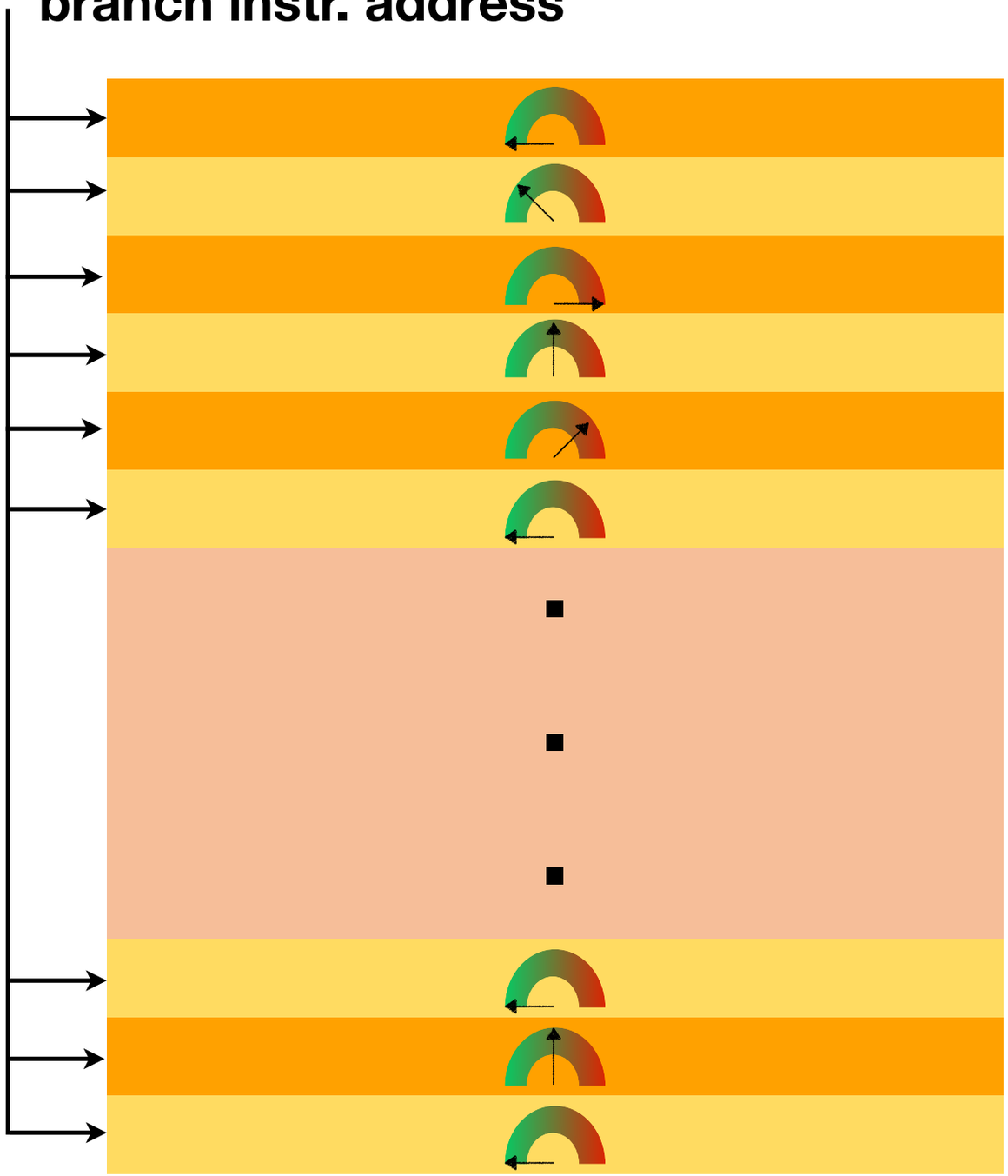
Branch history buffer

Branch history buffer



How do we handle multiple branch instructions with dynamic BP?

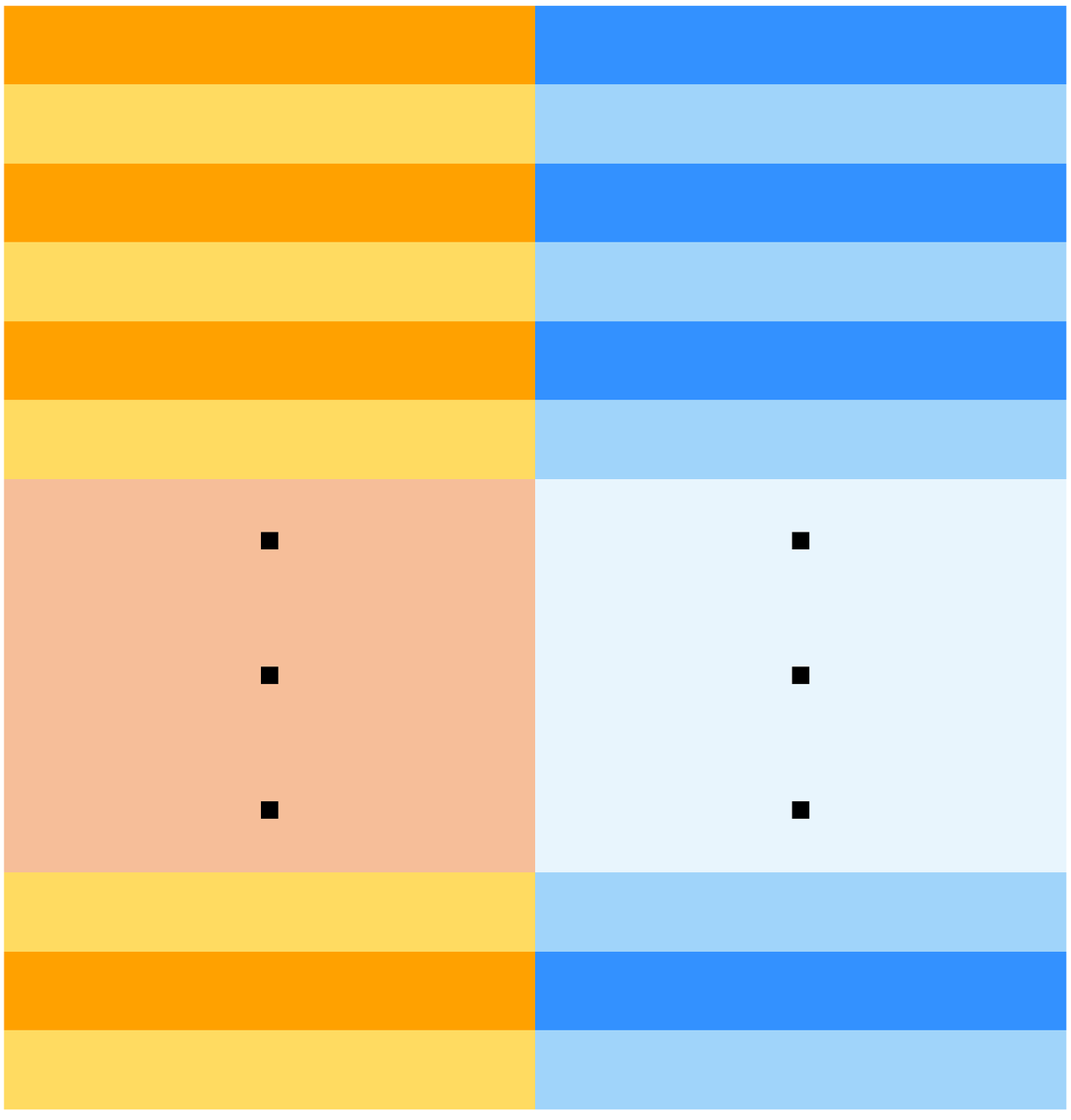
Branch history buffer



Branch target buffer

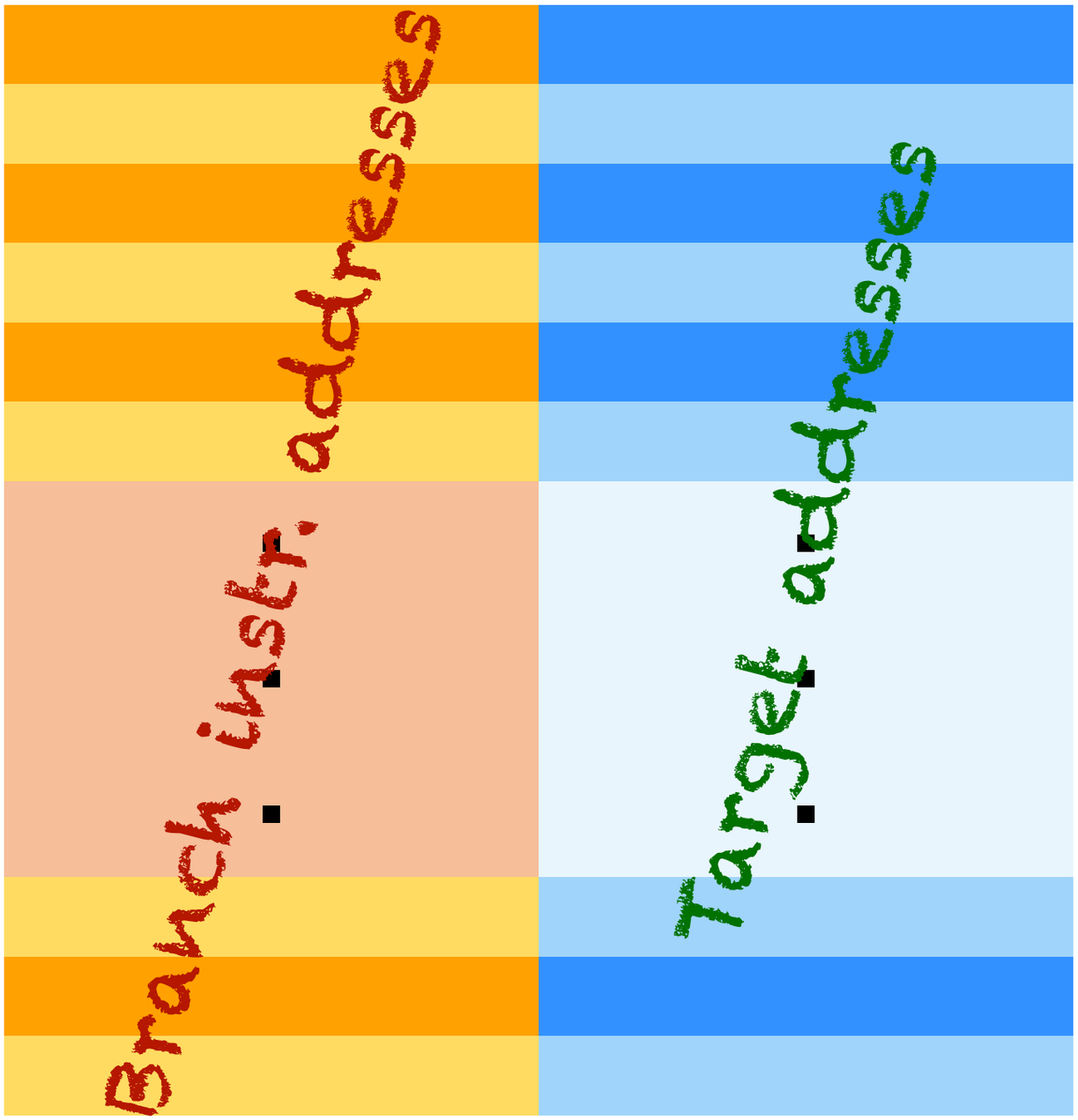
Branch target buffer

Branch target buffer



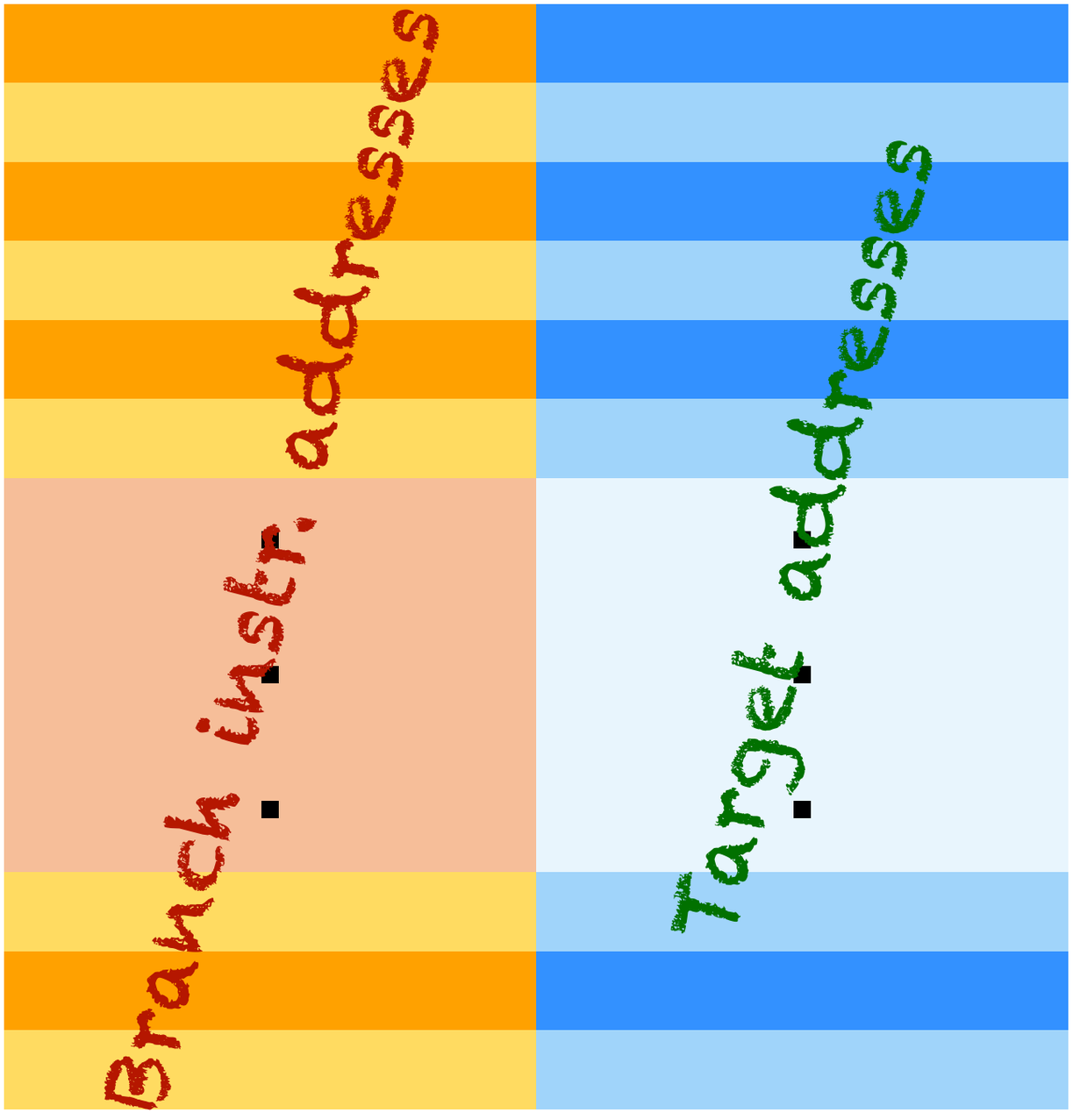
Associative memory

Branch target buffer

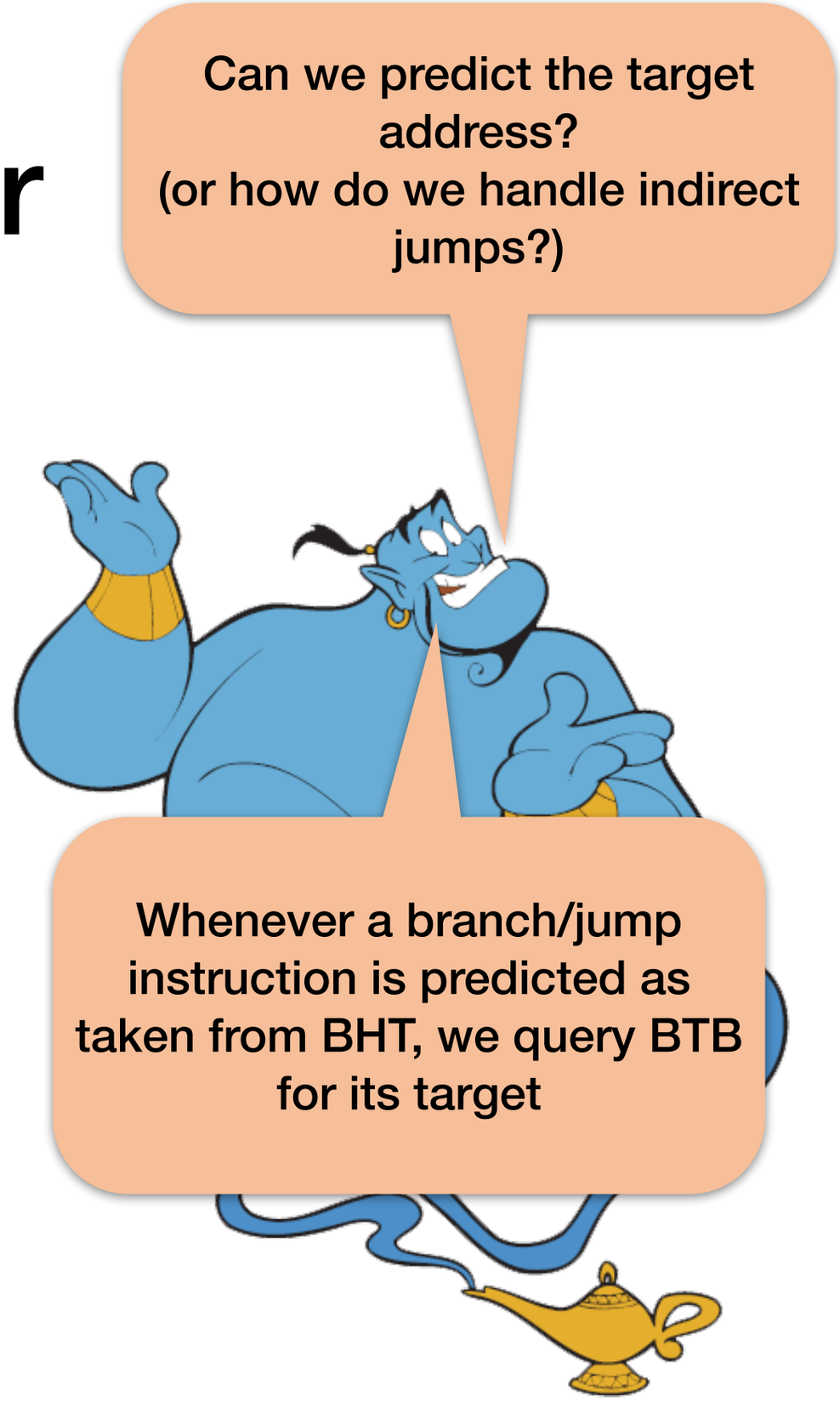


Associative memory

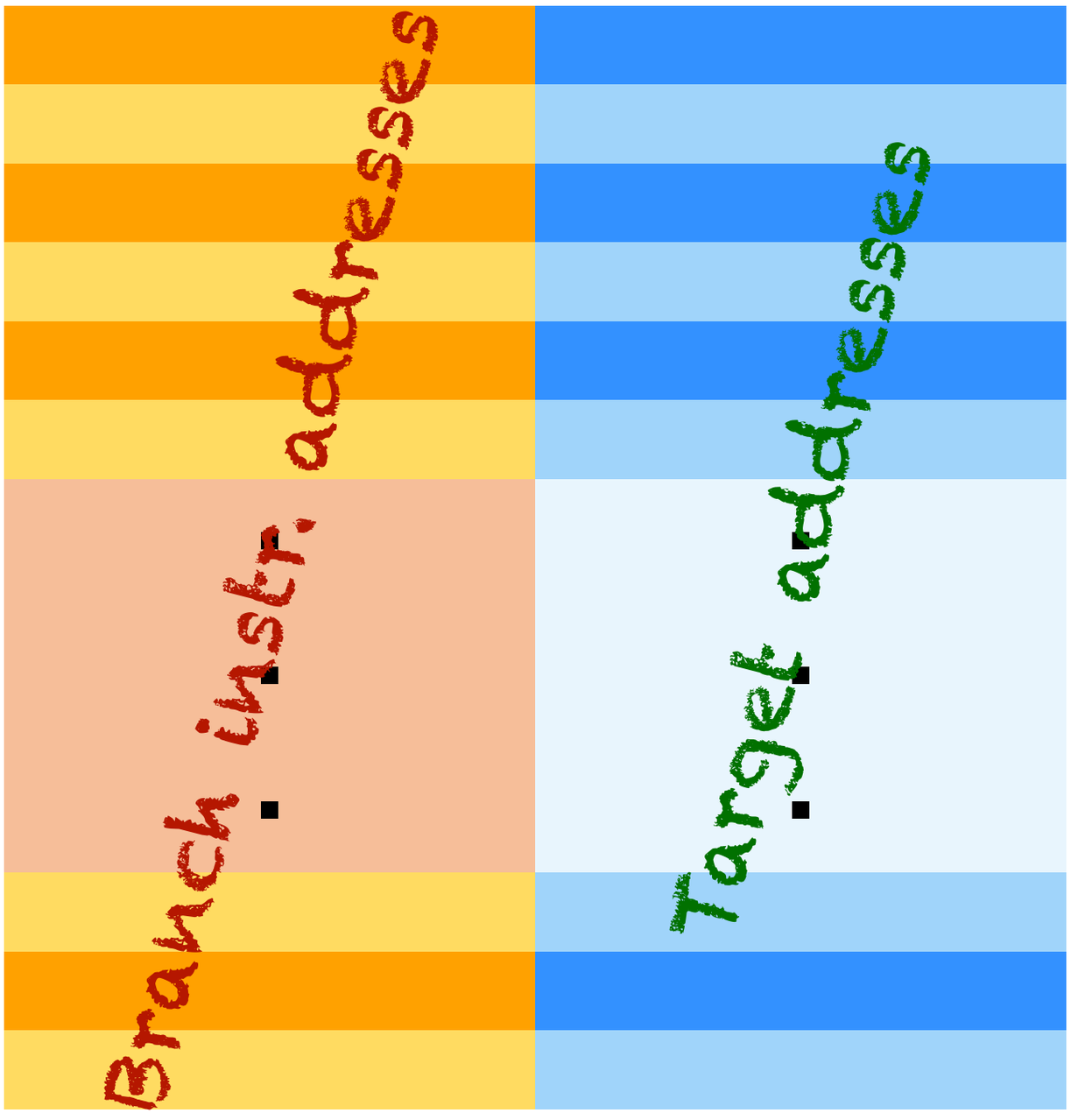
Branch target buffer



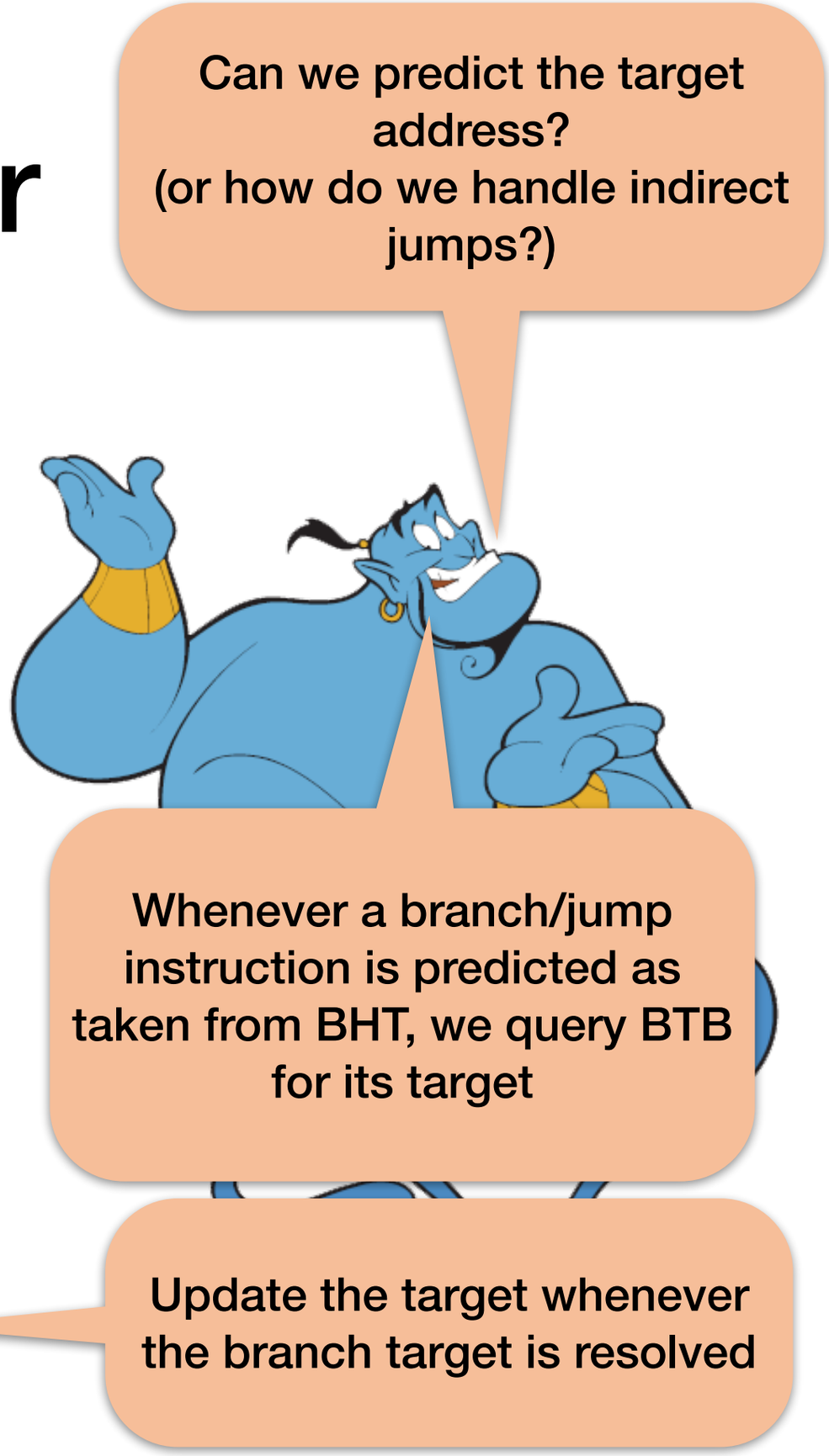
Associative memory



Branch target buffer



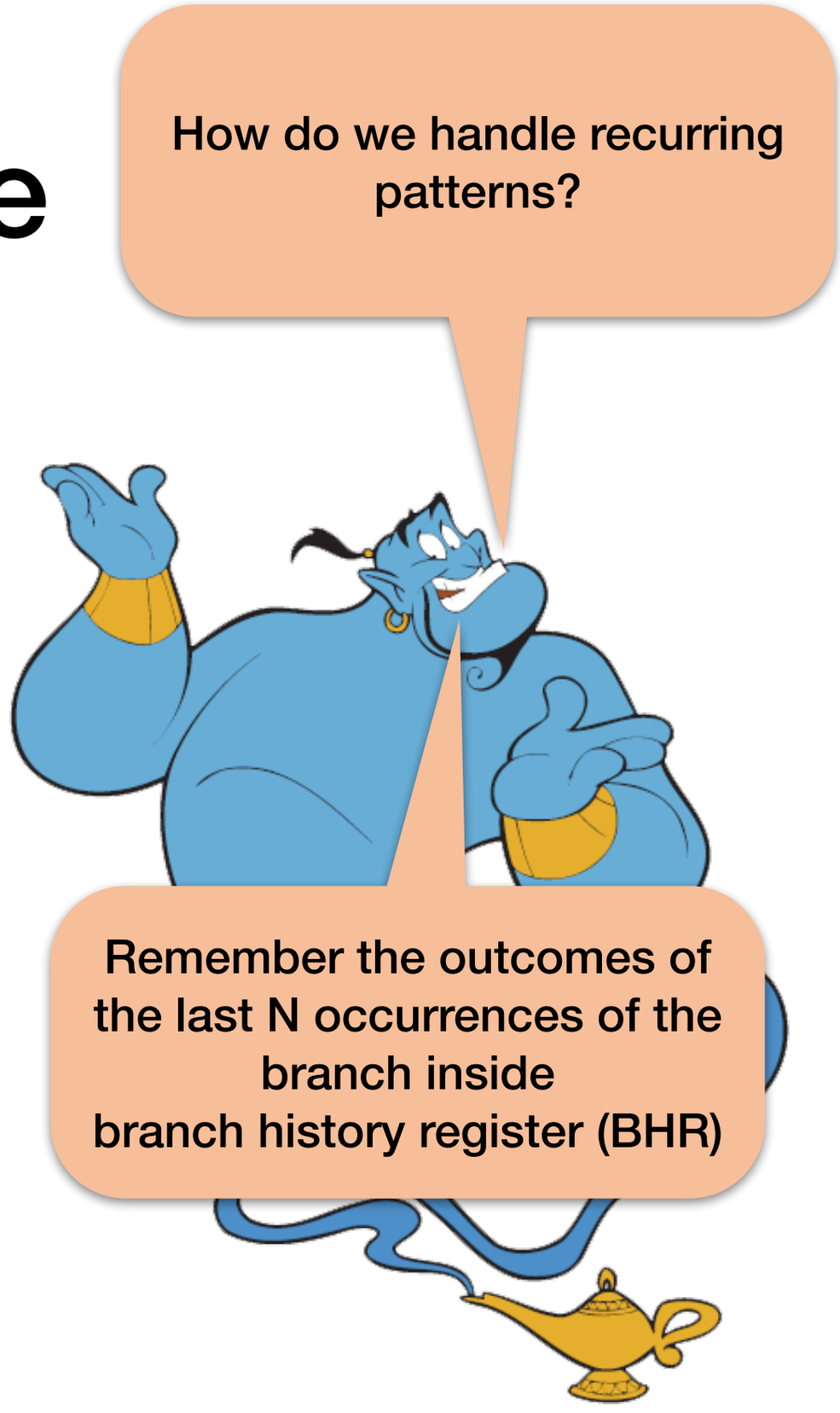
Associative memory



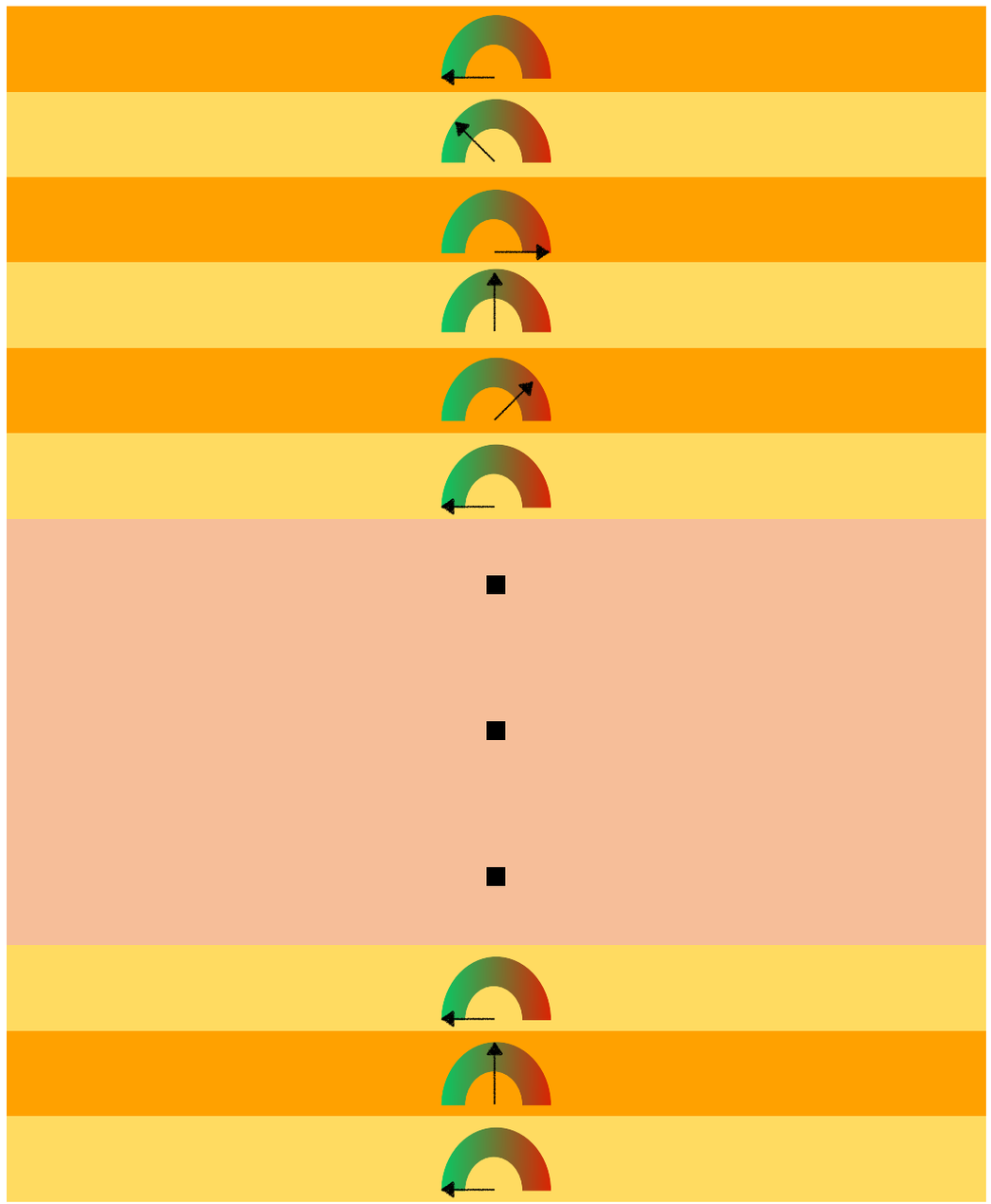
Pattern history table

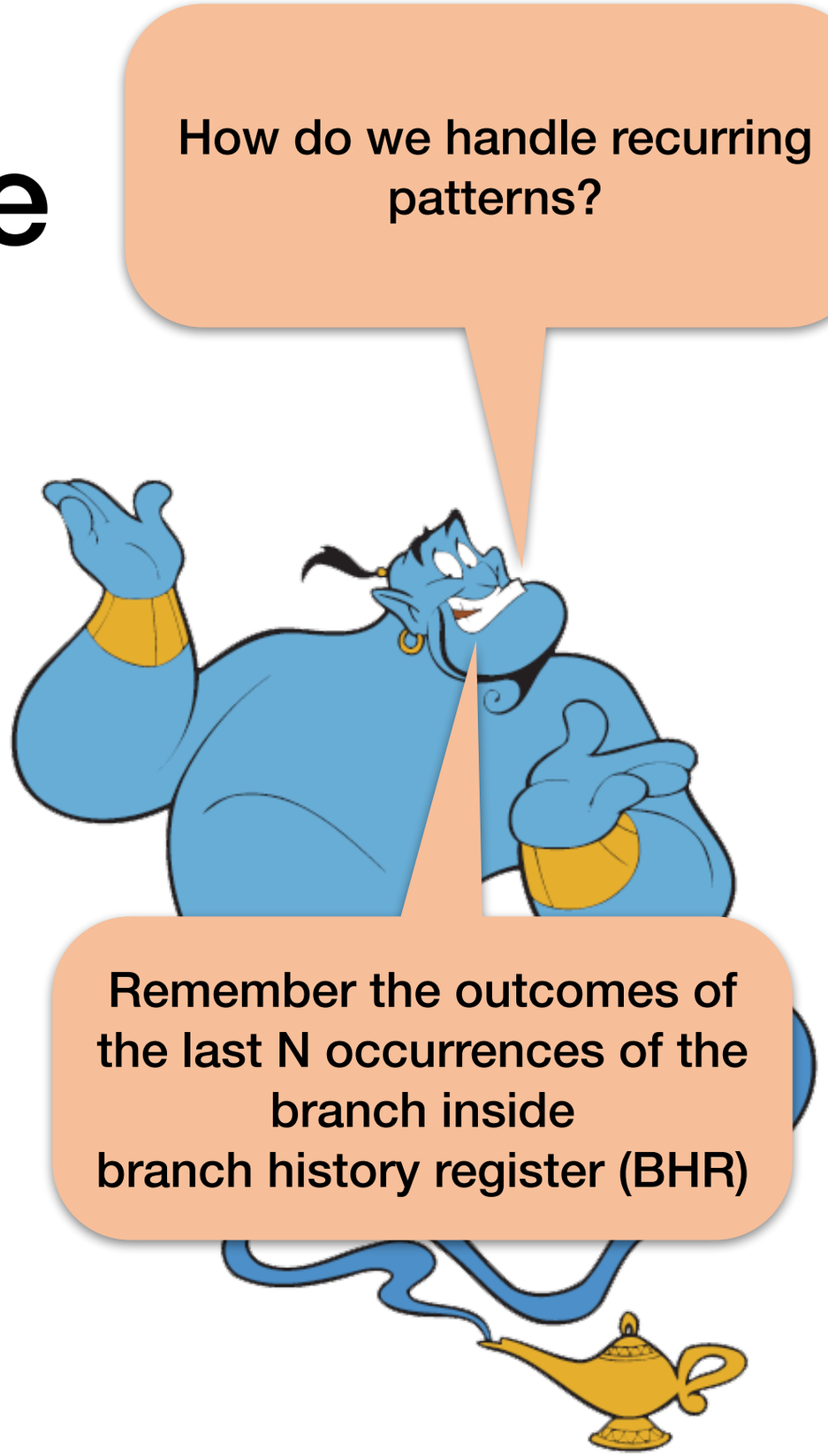
Pattern history table

Pattern history table

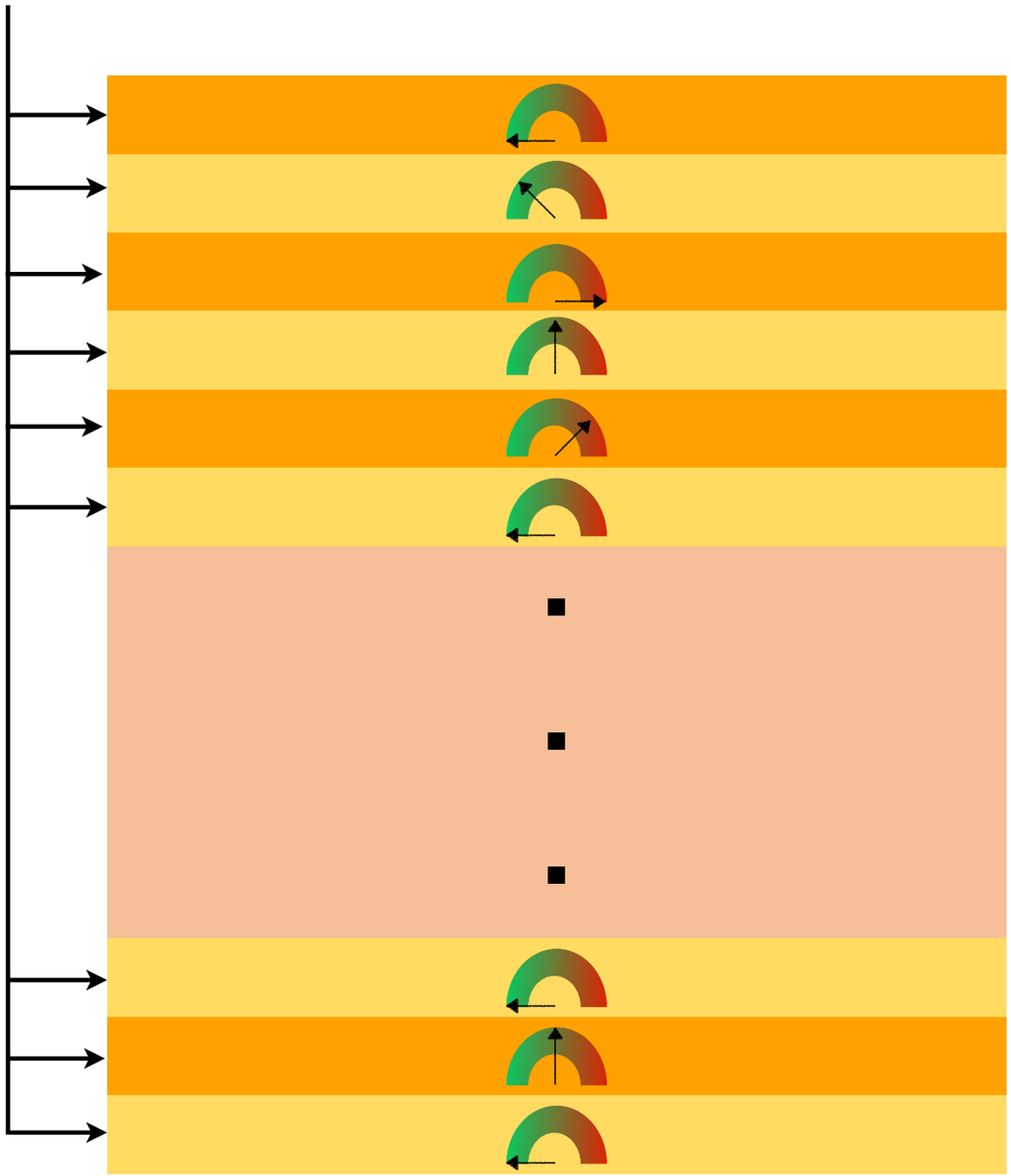


Pattern history table





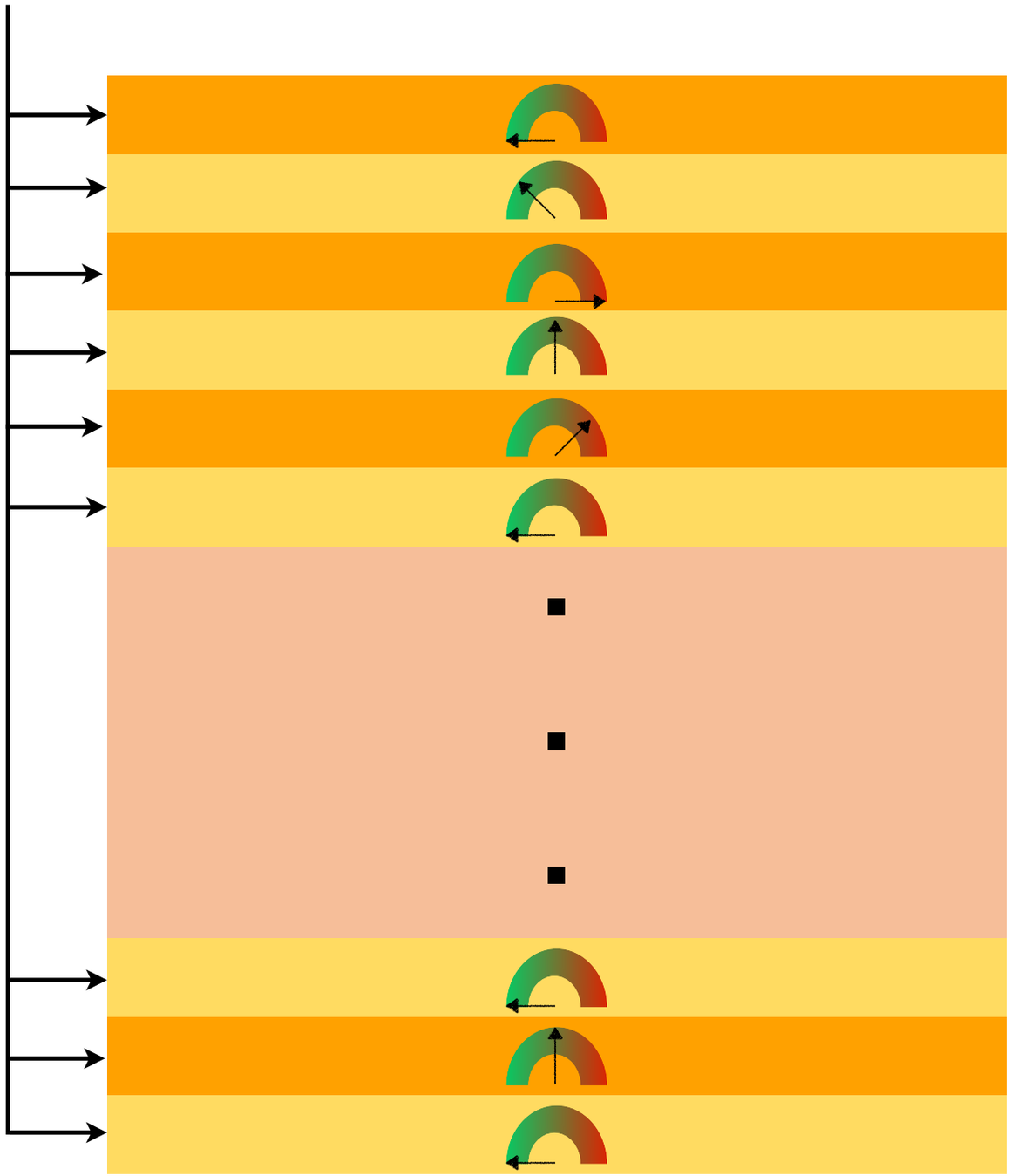
Pattern history table



How do we handle recurring patterns?

Remember the outcomes of the last N occurrences of the branch inside branch history register (BHR)

Pattern history table



How do we handle recurring patterns?

Remember the outcomes of the last N occurrences of the branch inside branch history register (BHR)

For each branch, we need N bits for BHR + 2^N * |ctr| bits

Global history buffer

Global history buffer

Global history buffer

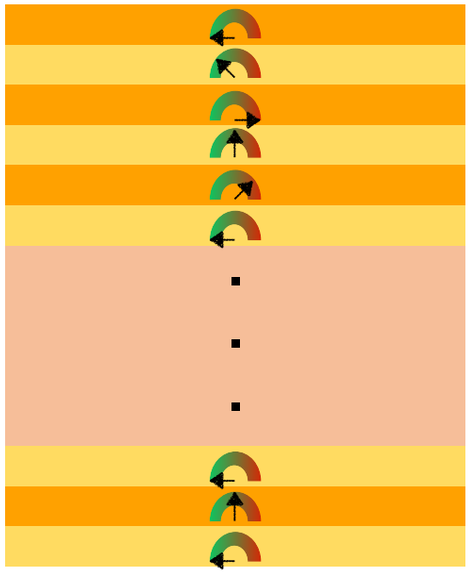
PHT grows exponentially in the size of BHR

We'll share PHT and BHR among all branch instructions

Global history buffer

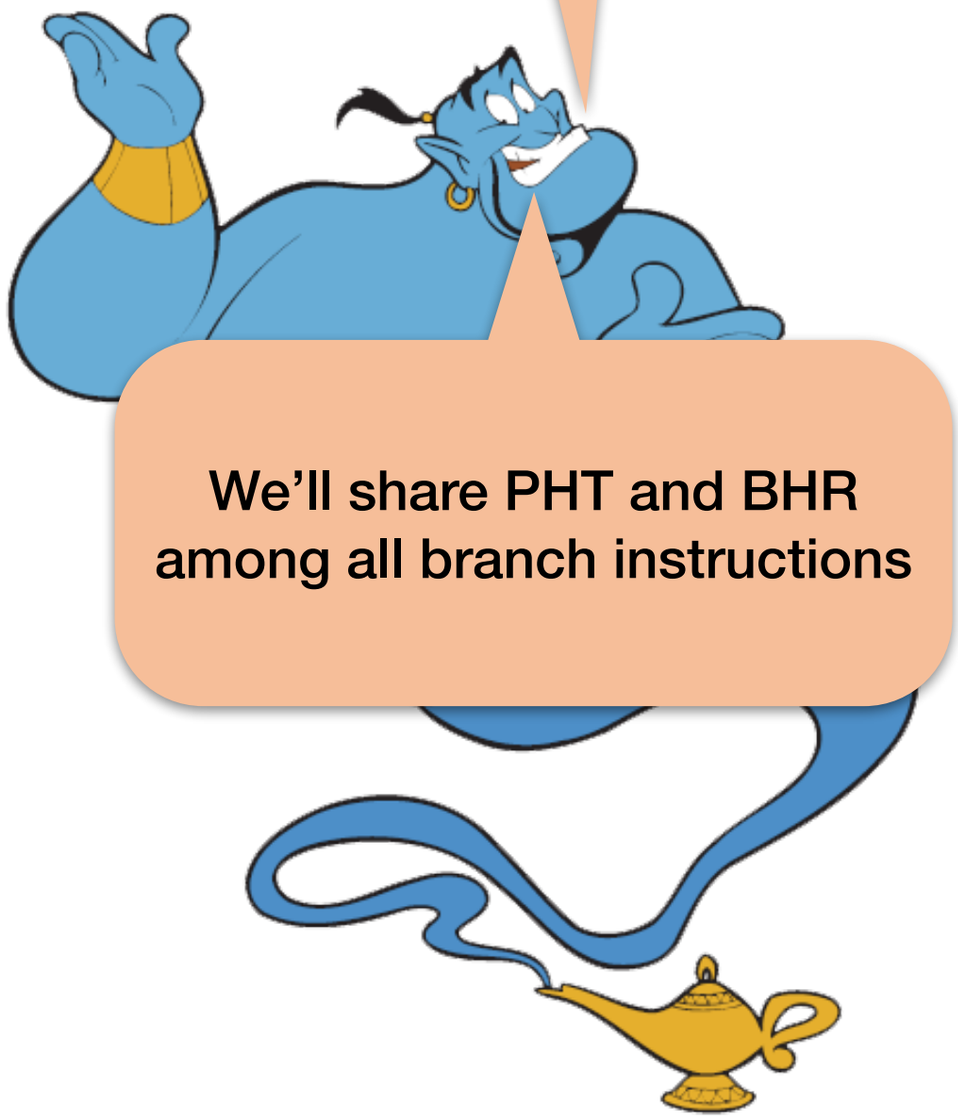
Global PHT (2^m entries)

Global
BHR
(m bits)



PHT grows exponentially in the size of BHR

We'll share PHT and BHR among all branch instructions

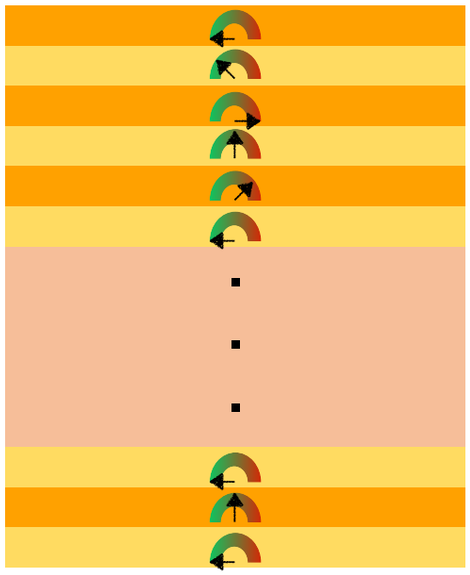


Global history buffer

Global PHT (2^m entries)

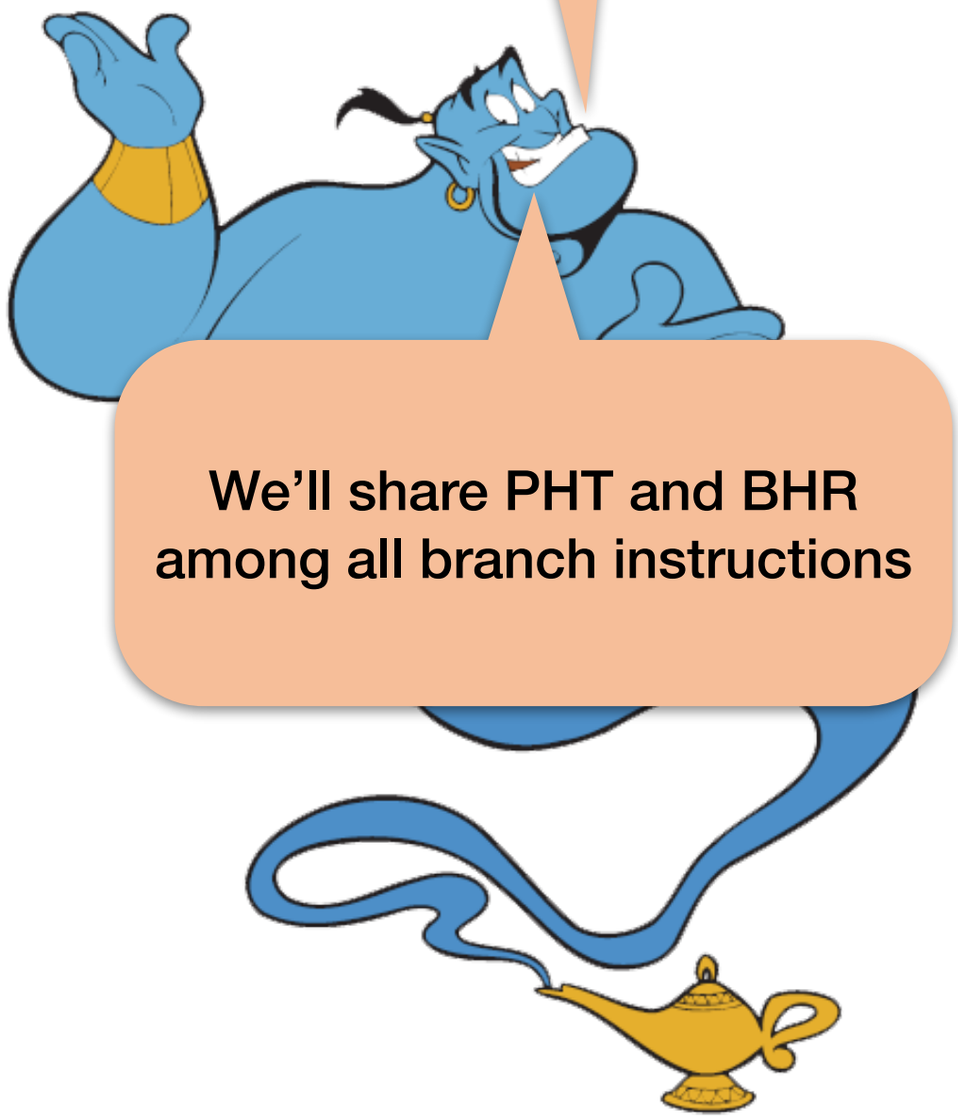
Global
BHR
(m bits)

Branch
address

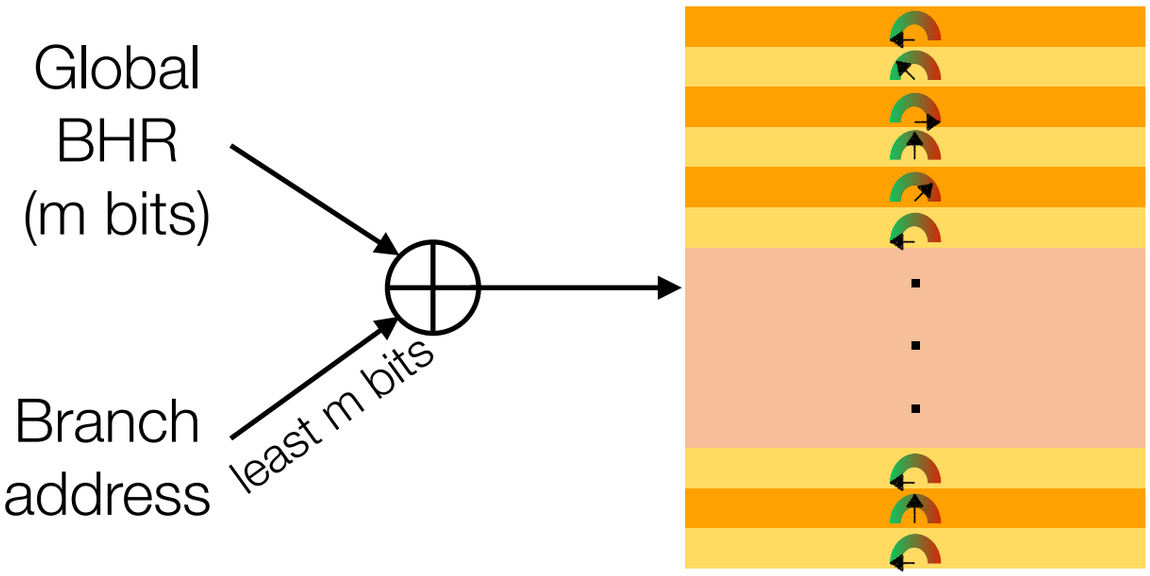


PHT grows exponentially in the size of BHR

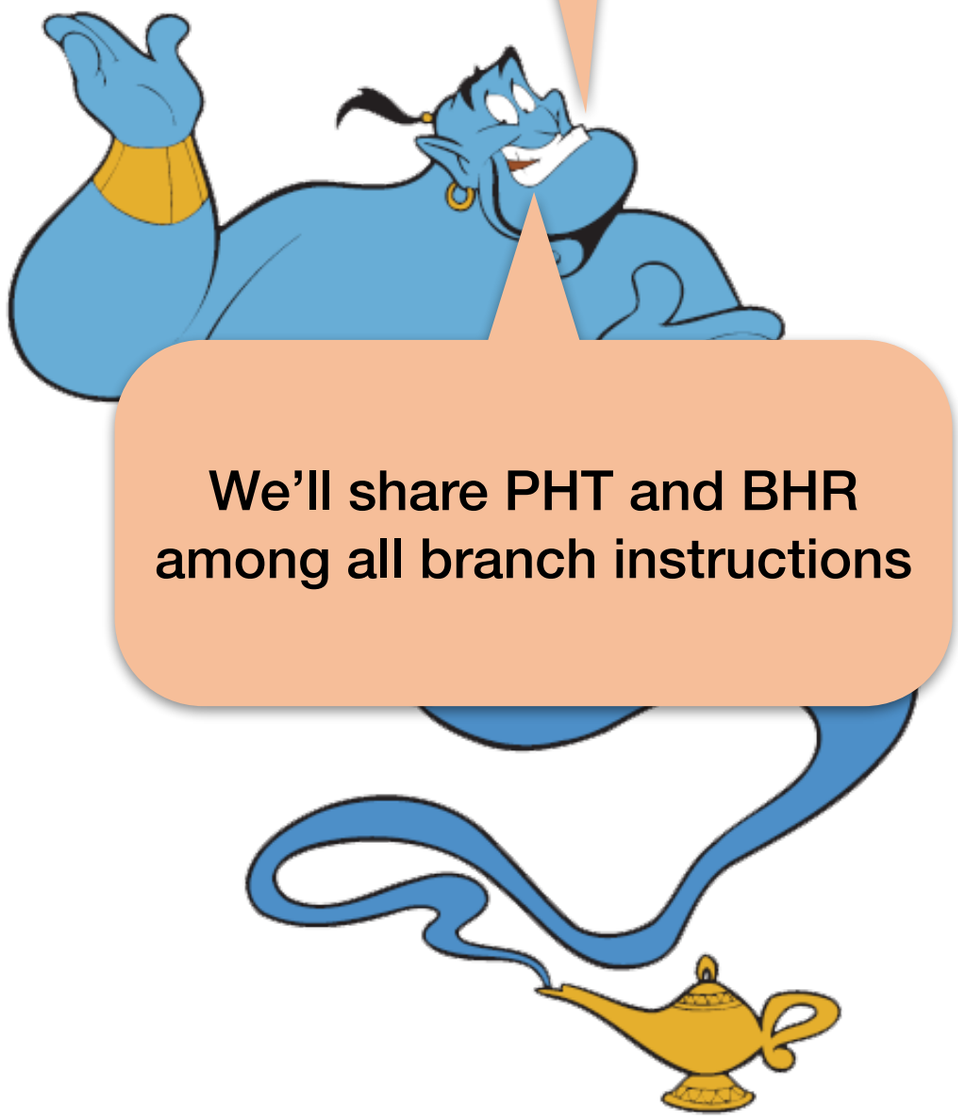
We'll share PHT and BHR among all branch instructions



Global history buffer



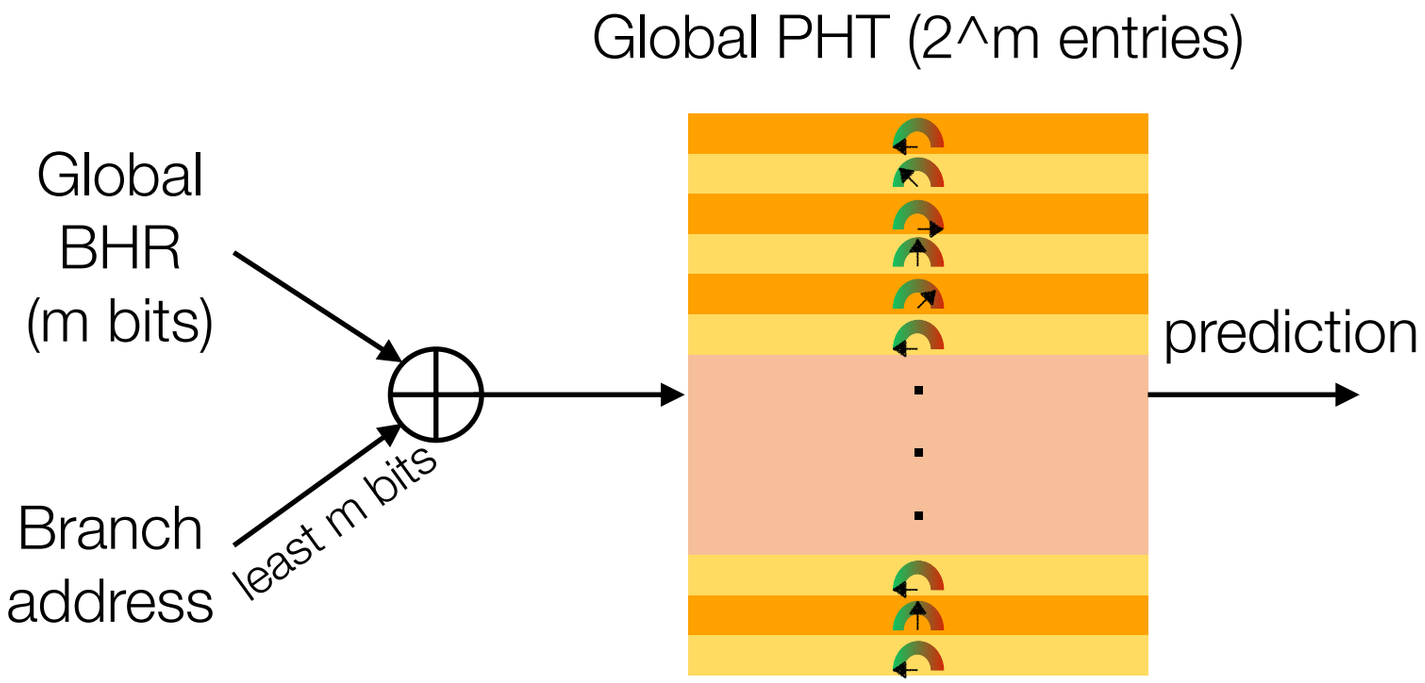
PHT grows exponentially in the size of BHR

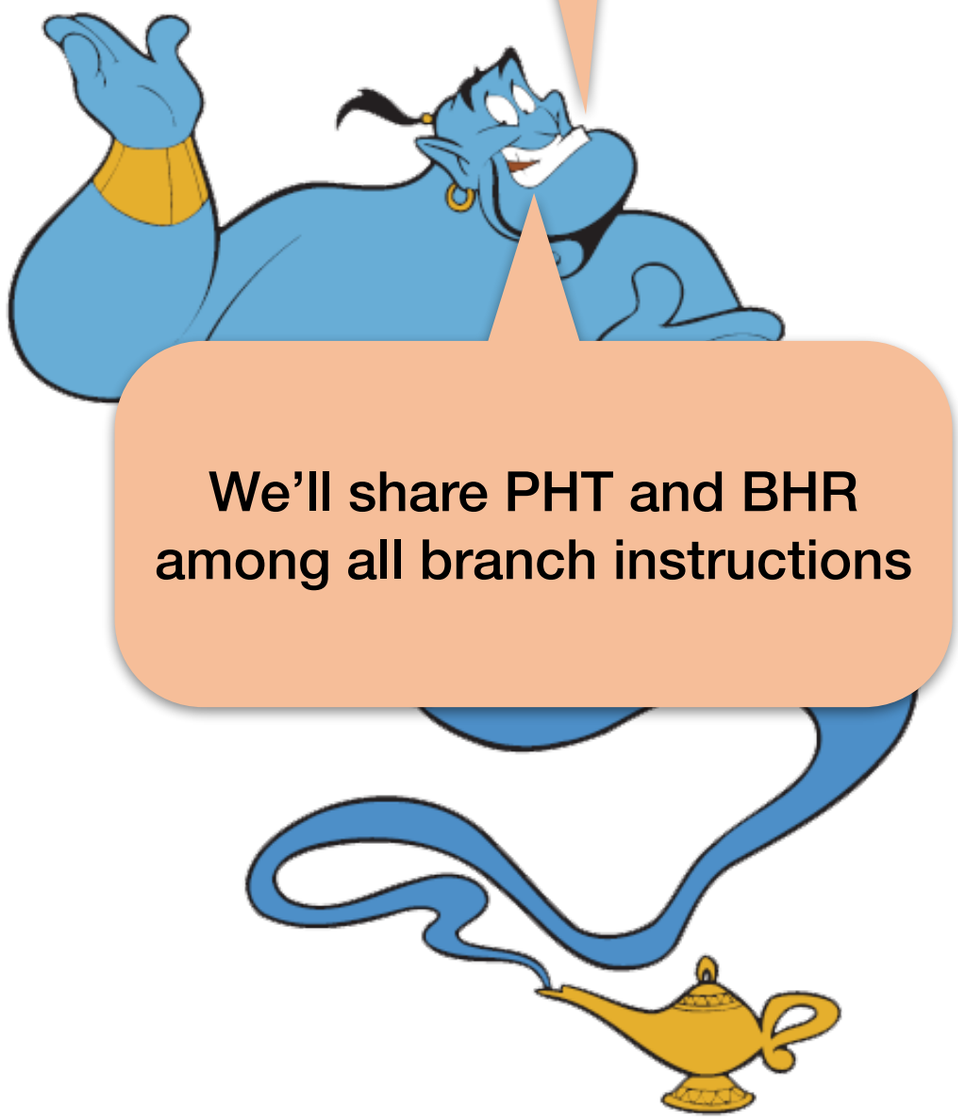


We'll share PHT and BHR among all branch instructions

Global history buffer

PHT grows exponentially in the size of BHR

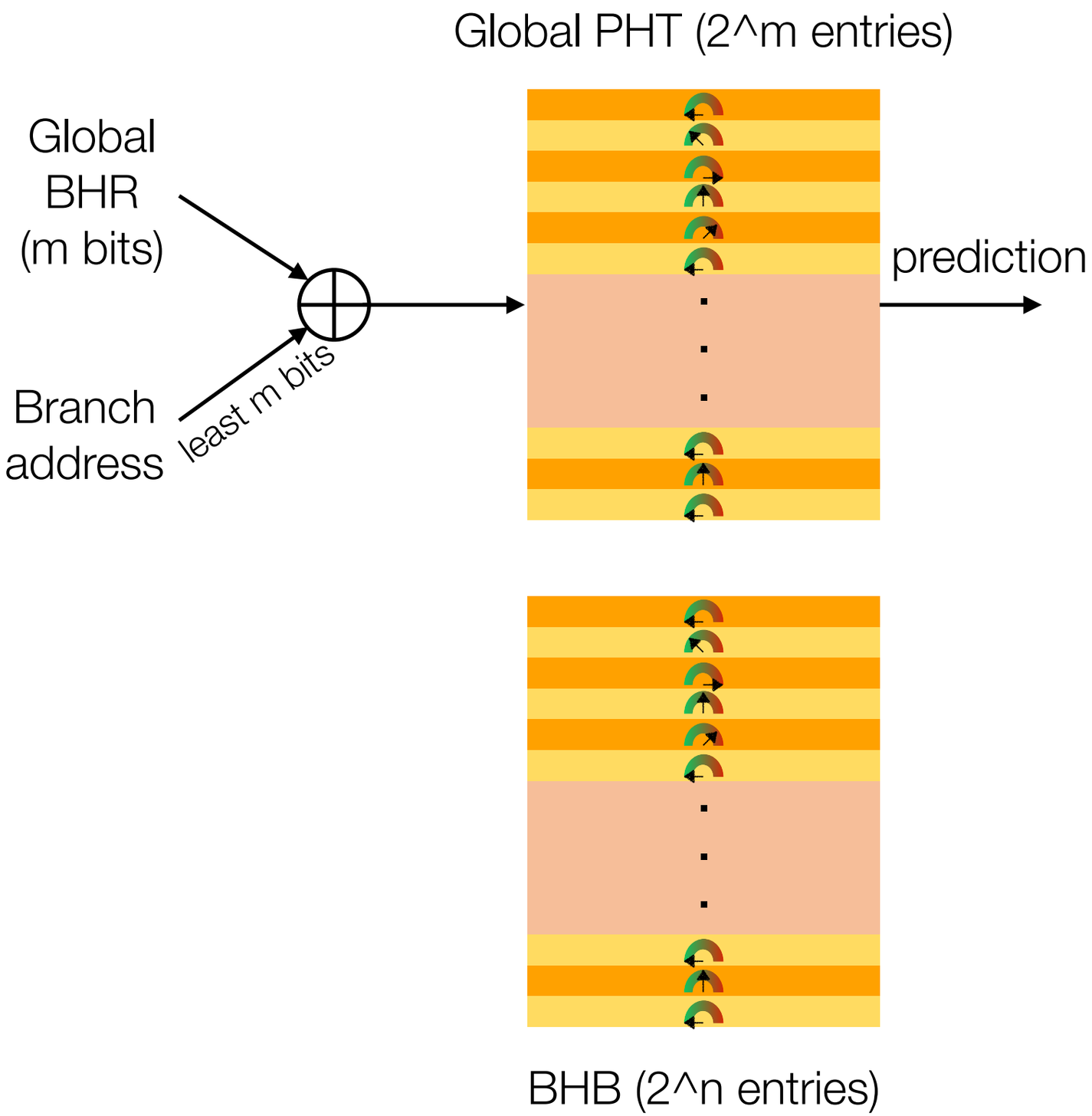


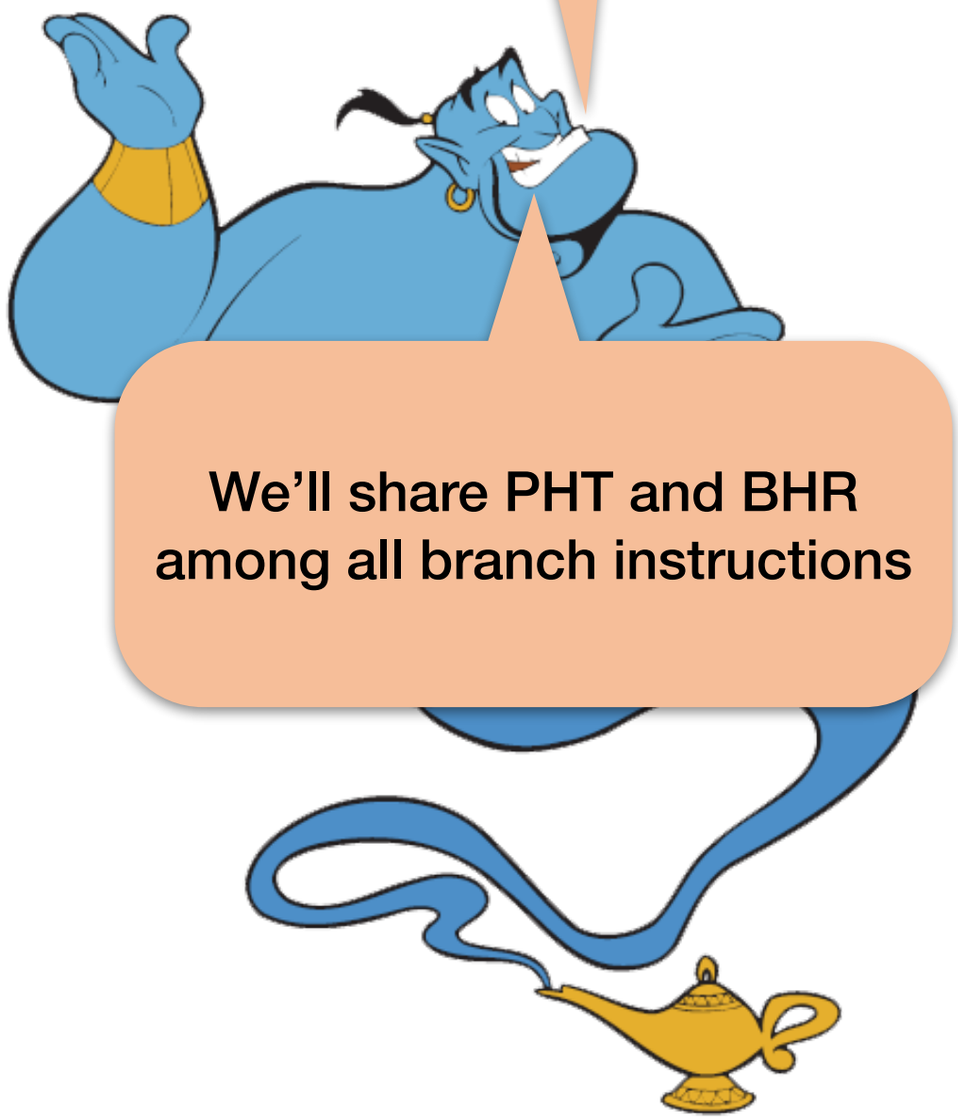


We'll share PHT and BHR among all branch instructions

Global history buffer

PHT grows exponentially in the size of BHR

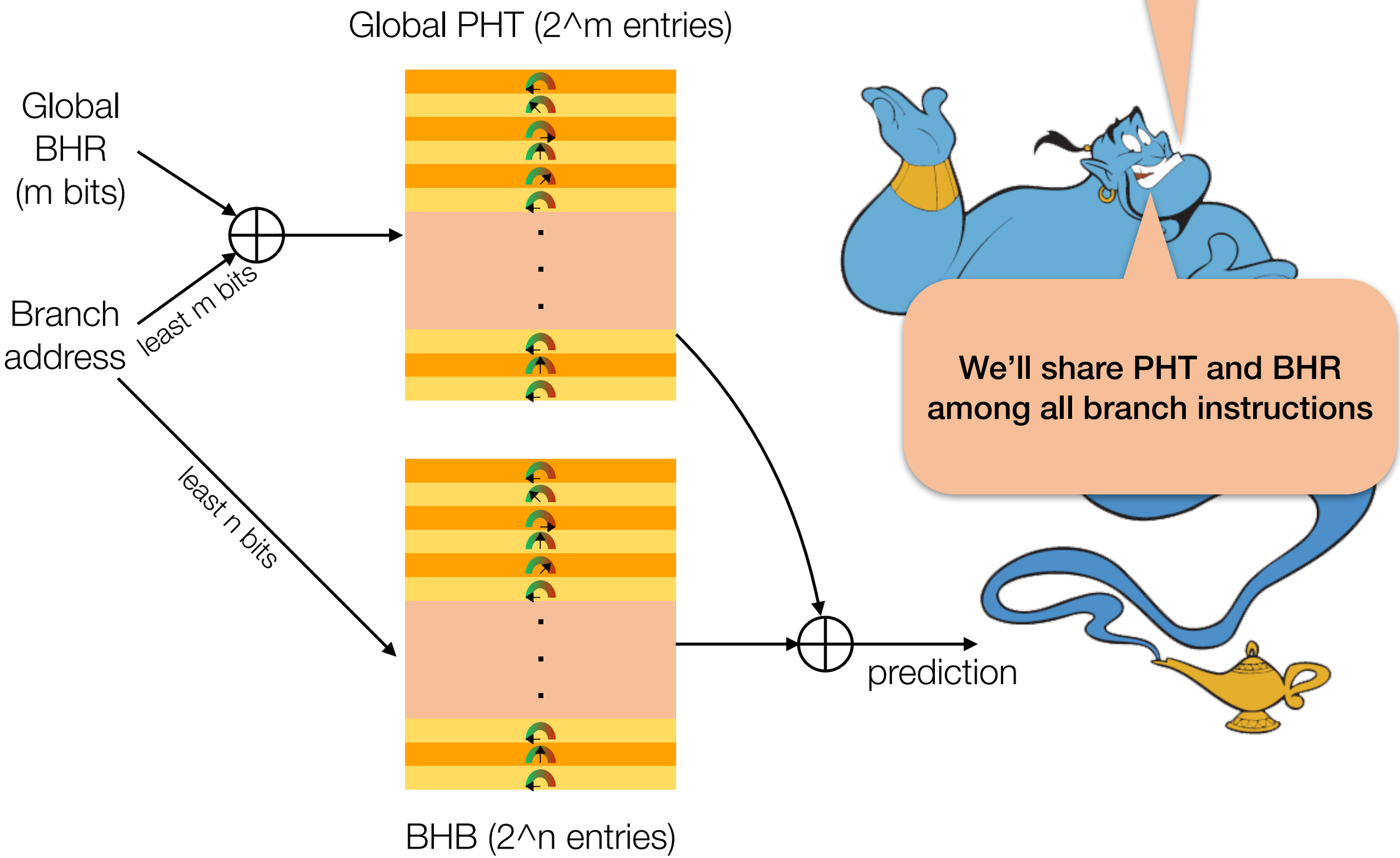




We'll share PHT and BHR among all branch instructions

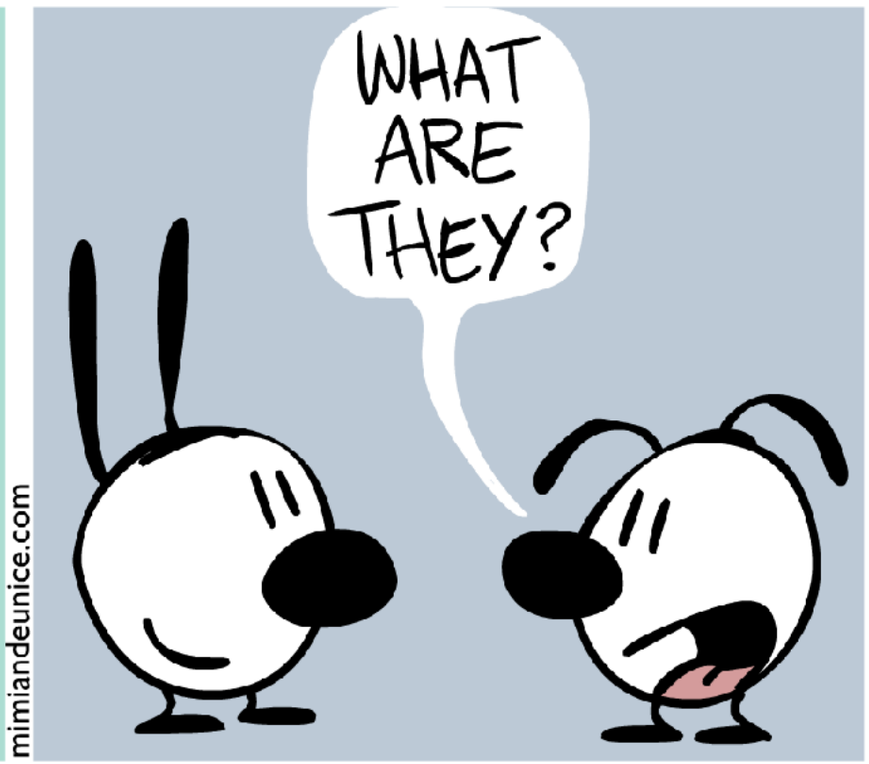
Global history buffer

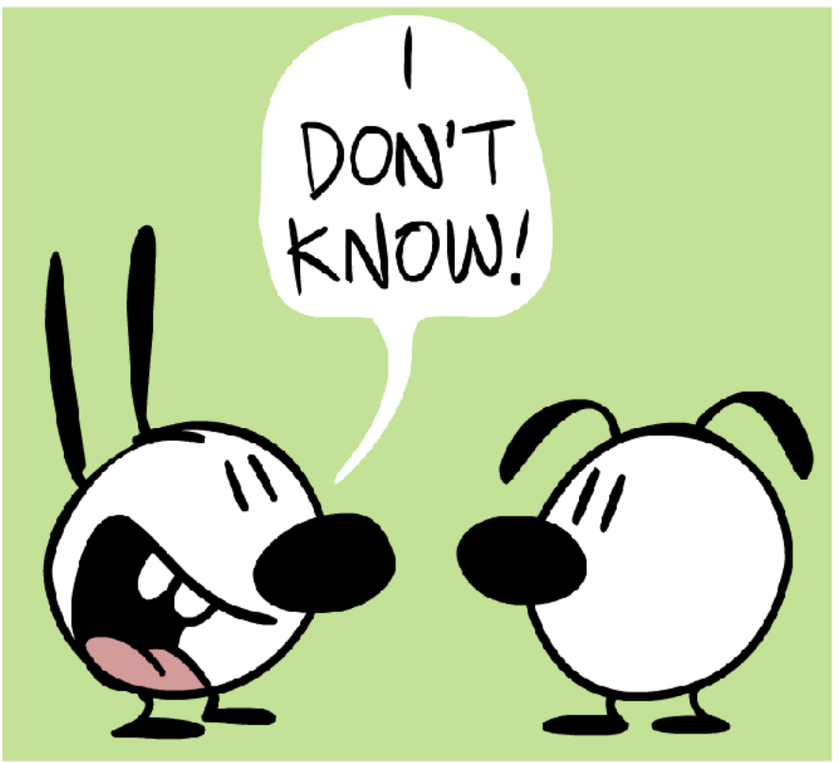
PHT grows exponentially in the size of BHR



We'll share PHT and BHR among all branch instructions


THREE MAGIC WORDS:


WHAT ARE THEY?
mimiandeunice.com


I DON'T KNOW!

THREE MAGIC

WHAT ARE

I DON'T KNOW!

58-bit Branch History Buffer

58 bit

Indirect branch prediction

Source

subset of 12 LSB

XOR folding

BTB lookup

64-bit destination

fallback

Taken branches

Source

Destination

bits 4-19

bits 0-5

XOR folding (cf. Listing 4)

16 bit

Direct branch prediction

XOR folding

bits 0-30

Source

BTB lookup

32-bit destination

4 GB straddle bit

cond. ±4 GB adjust

bit 31

bits 32-63

64-bit destination

See Spectre paper for details

SPECTRE (Variant 2)

SPECTRE (Variant 2)

- **Speculation** : using indirect jump commands (jmp [expr], call [fnct],…)

SPECTRE (Variant 2)

- **Speculation** : using indirect jump commands (jmp [expr], call [fnct],…)
- **Micro-architectural state** : caches
- **Encode** data through memory accesses
- **Recover** data using Prime+Probe, Flush+reload, …

SPECTRE (Variant 2)

- **Speculation** : using indirect jump commands (jmp [expr], call [fnct],…)
- **Micro-architectural state** : caches
  - **Encode** data through memory accesses
  - **Recover** data using Prime+Probe, Flush+reload, …
- **Influence speculation target** : exploit dynamic BP (BTB)

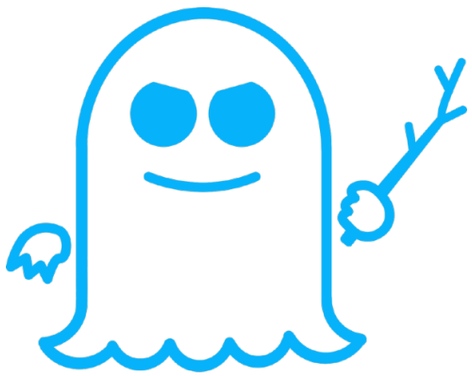
SPECTRE (Variant 2)

SPECTRE (Variant 2)

Victim program

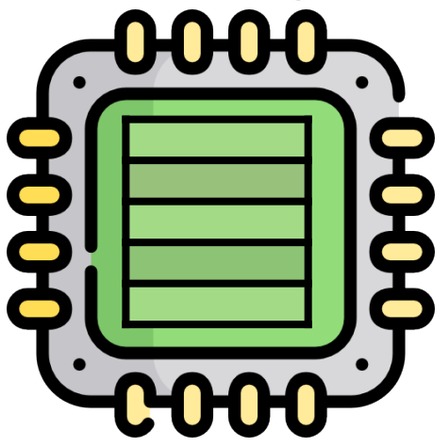
SPECTRE (Variant 2)

& Prime + Probe



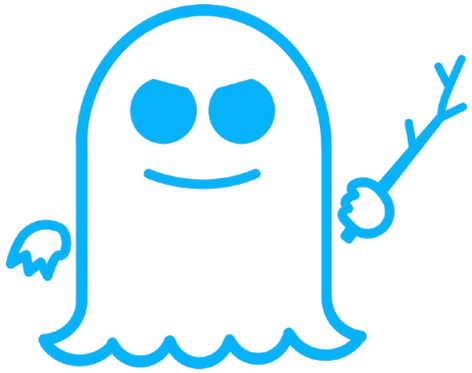
SPECTRE

Victim program



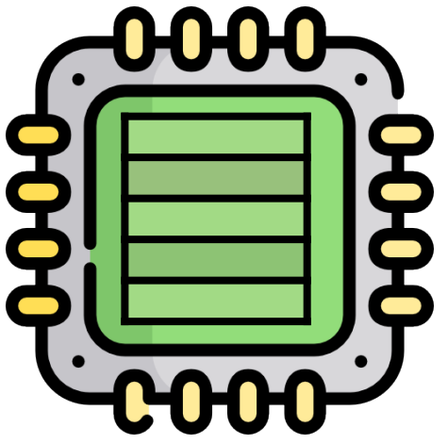
SPECTRE (Variant 2)

& Prime + Probe



1) Find SPECTRE gadget

Victim program



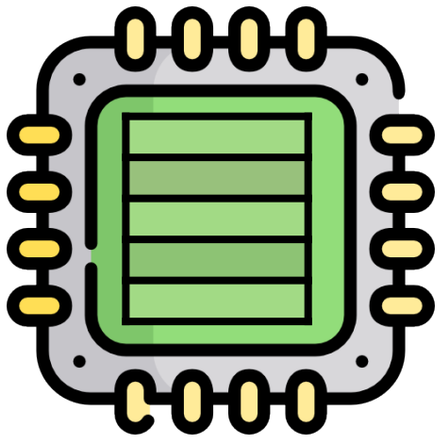
SPECTRE (Variant 2)

& Prime + Probe

Locate a gadget in victim's executable memory to leak into the micro-architecture

1) Find SPECTRE gadget

Victim program



SPECTRE (Variant 2)

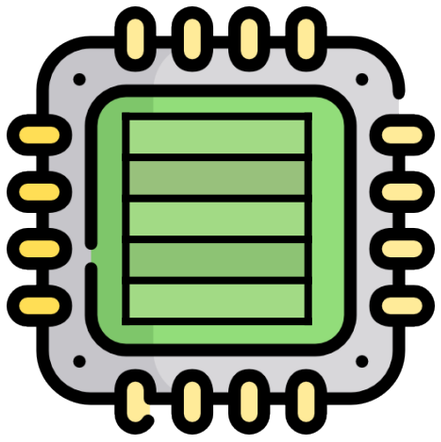
& Prime + Probe

Locate a gadget in victim's executable memory to leak into the micro-architecture

y = B[A[**x**] * 4096]

1) Find SPECTRE gadget

Victim program



SPECTRE (Variant 2)

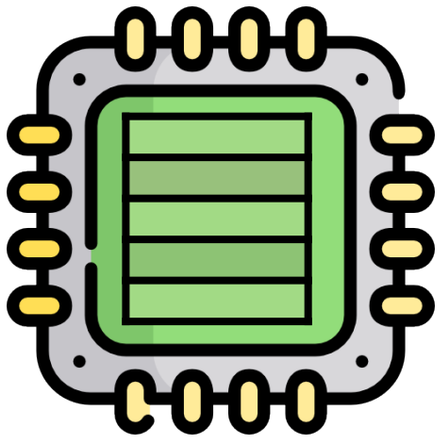
& Prime + Probe

Locate a gadget in victim's executable memory to leak into the micro-architecture

load R2, **R1**
load R3, R2

1) Find SPECTRE gadget

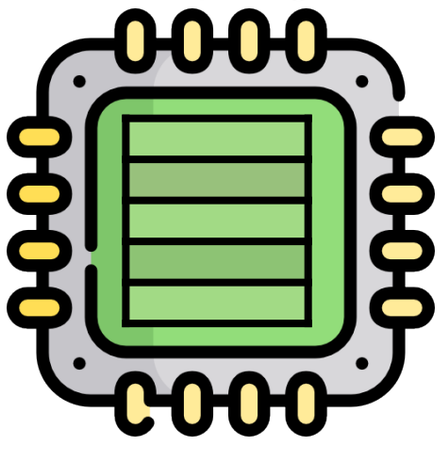
Victim program



SPECTRE (Variant 2)

& Prime + Probe

Victim program



1) Find SPECTRE gadget

2) Training

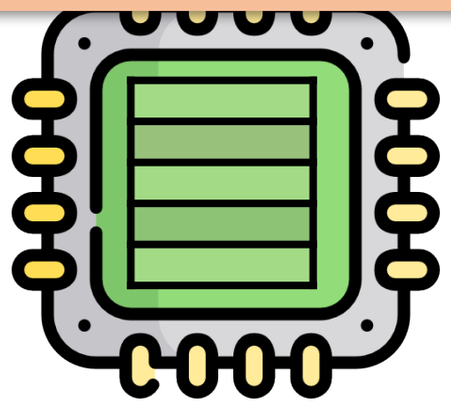
SPECTRE (Variant 2)

& Prime + Probe

1. Identify an **indirect jump** (e.g., jmp [expr] or call [fnct]) in victim's program

1) Find SPECTRE gadget

2) Training



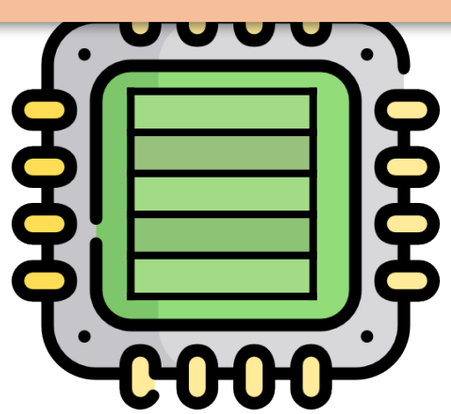
SPECTRE (Variant 2)

& Prime + Probe

1. Identify an **indirect jump** (e.g., jmp [expr] or call [fnct]) in victim's program
2. **Copy** victim's program and replace jmp [expr] with jmp 

1) Find SPECTRE gadget 

2) Training 



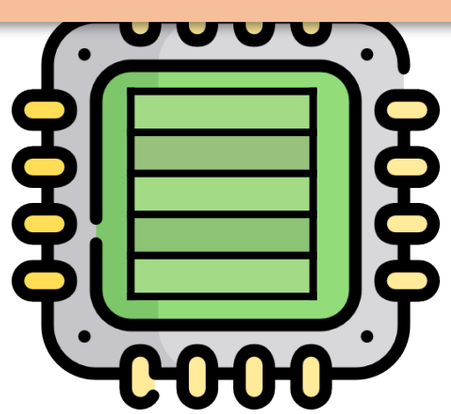
SPECTRE (Variant 2)

& Prime + Probe

1. Identify an **indirect jump** (e.g., jmp [expr] or call [fnct]) in victim's program
2. **Copy** victim's program and replace jmp [expr] with jmp 
3. **Mimic** victim's program **branch history** up to jmp 

1) Find SPECTRE gadget 

2) Training 



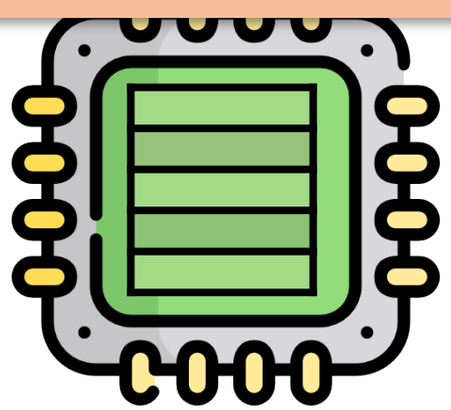
SPECTRE (Variant 2)

& Prime + Probe

1. Identify an **indirect jump** (e.g., jmp [expr] or call [fnct]) in victim's program
2. **Copy** victim's program and replace jmp [expr] with jmp 
3. **Mimic** victim's program **branch history** up to jmp 
4. **Repeat** 3 several times

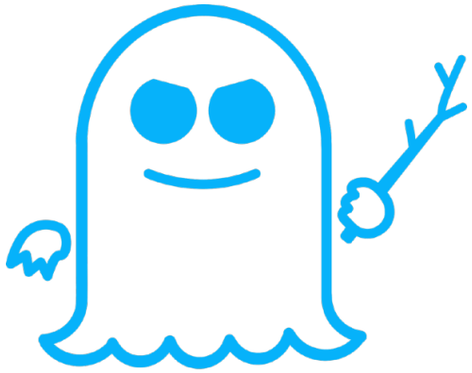
1) Find SPECTRE gadget 

2) Training 



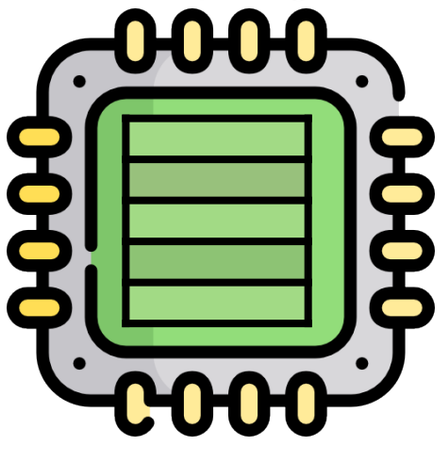
SPECTRE (Variant 2)

& Prime + Probe



SPECTRE

Victim program



1) Find SPECTRE gadget

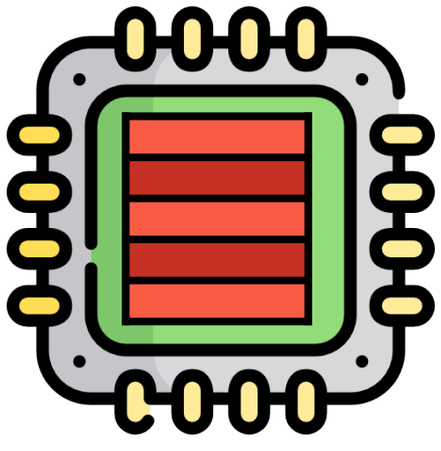
2) Training

3) Prepare

SPECTRE (Variant 2)

& Prime + Probe

Victim program



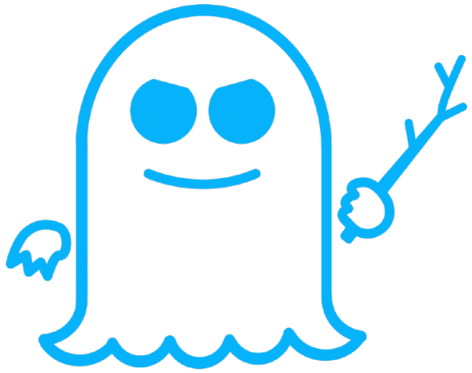
1) Find SPECTRE gadget

2) Training

3) Prepare Fill cache with attacker's data

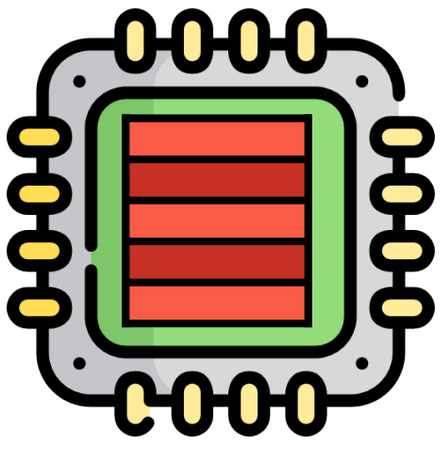
SPECTRE (Variant 2)

& Prime + Probe



SPECTRE

Victim program



1) Find SPECTRE gadget

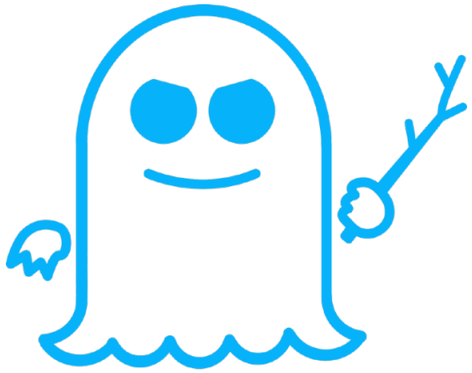
2) Training

3) Prepare Fill cache with attacker's data

4) Attack

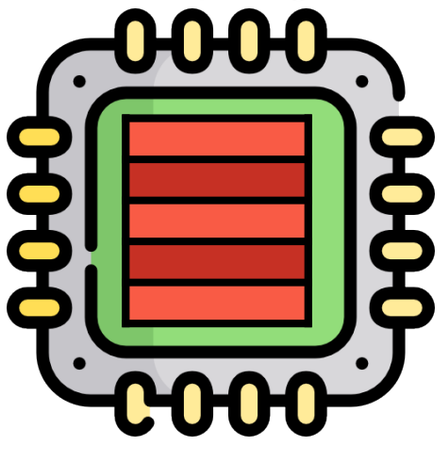
SPECTRE (Variant 2)

& Prime + Probe



SPECTRE

Victim program



1) Find SPECTRE gadget

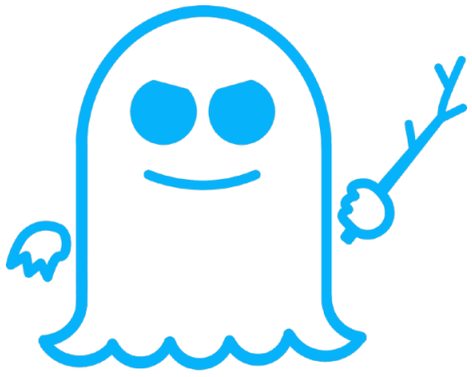
2) Training

3) Prepare Fill cache with attacker's data

4) Attack Run

SPECTRE (Variant 2)

& Prime + Probe



SPECTRE

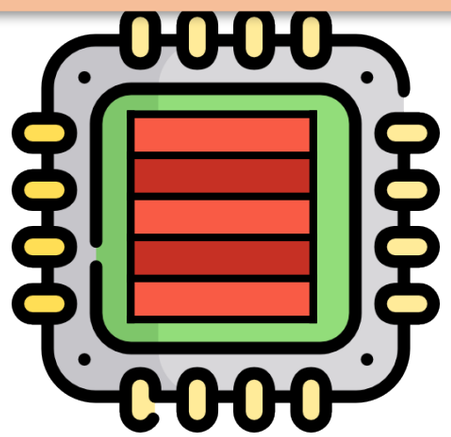
1. **Ensure that** jmp [expr] **triggers spec. ex.** (e.g., [expr] **not in cache)**

1) Find SPECTRE gadget

2) Training

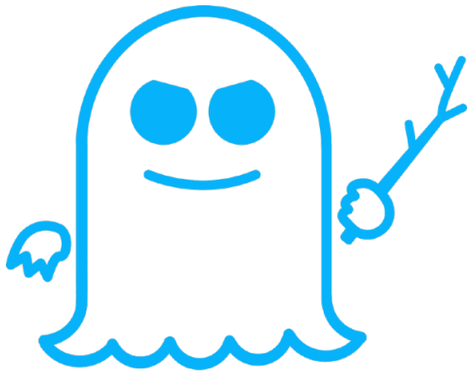
3) Prepare Fill cache with attacker's data

4) Attack Run



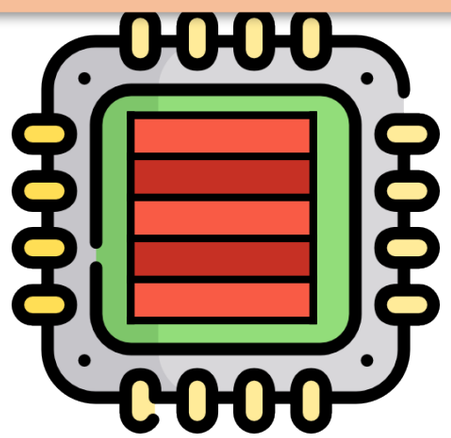
SPECTRE (Variant 2)

& Prime + Probe



SPECTRE

1. **Ensure that** jmp [expr] **triggers spec. ex.** (e.g., [expr] not in cache)
2. **Run** 



1) Find SPECTRE gadget 

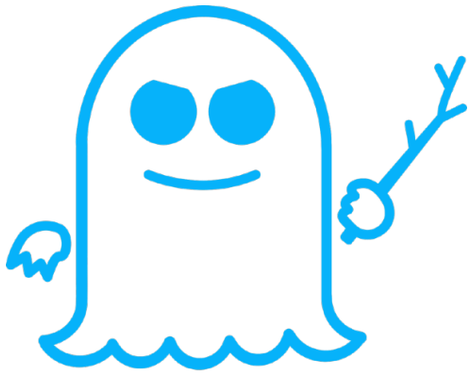
2) Training 

3) Prepare Fill cache with attacker's data

4) Attack Run 

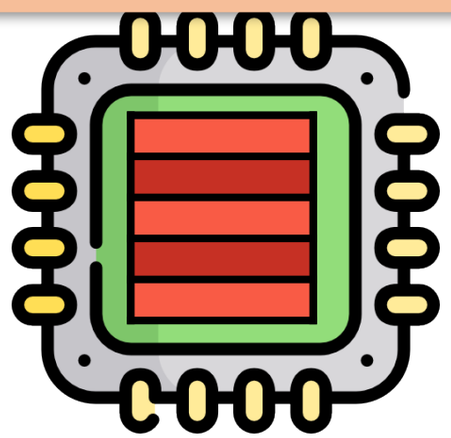
SPECTRE (Variant 2)

& Prime + Probe



SPECTRE

1. **Ensure that** jmp [expr] **triggers spec. ex.** (e.g., [expr] **not in cache)**
2. **Run** 
3. **Now** jmp [expr] **spec. executes** 



1) Find SPECTRE gadget 

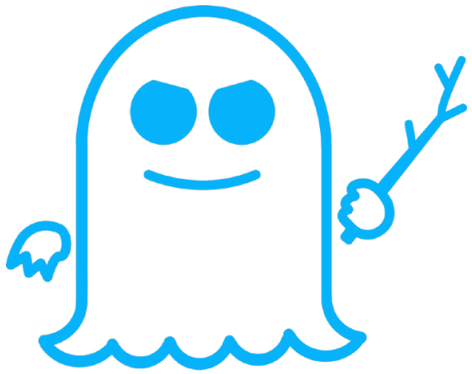
2) Training 

3) Prepare Fill cache with attacker's data

4) Attack Run 

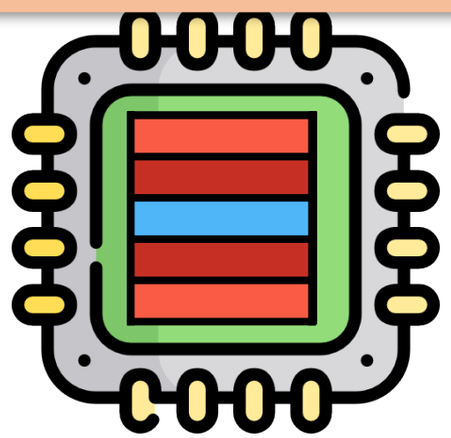
SPECTRE (Variant 2)

& Prime + Probe



SPECTRE

1. **Ensure that** jmp [expr] **triggers spec. ex.** (e.g., [expr] **not in cache)**
2. **Run** 
3. **Now** jmp [expr] **spec. executes** 



1) Find SPECTRE gadget 

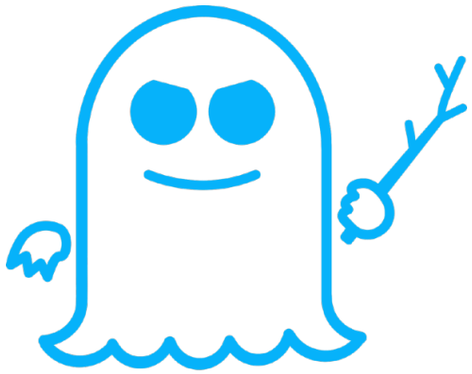
2) Training 

3) Prepare Fill cache with attacker's data

4) Attack Run 

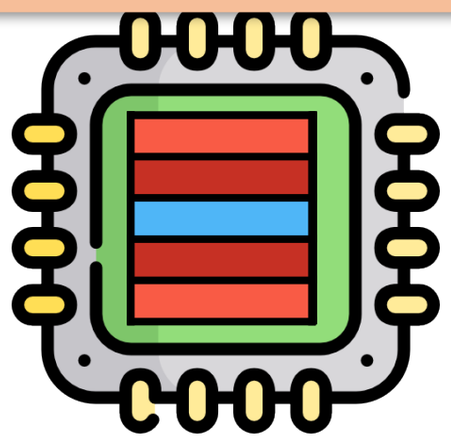
SPECTRE (Variant 2)

& Prime + Probe



SPECTRE

1. **Ensure that** jmp [expr] **triggers spec. ex.** (e.g., [expr] not in cache)
2. **Run** 
3. **Now** jmp [expr] **spec. executes** 



1) Find SPECTRE gadget 

2) Training 

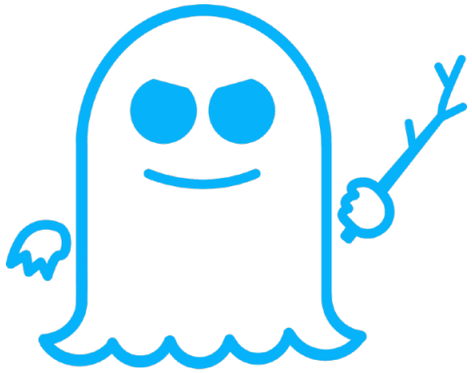
3) Prepare Fill cache with attacker's data

4) Attack Run 

5) Extract

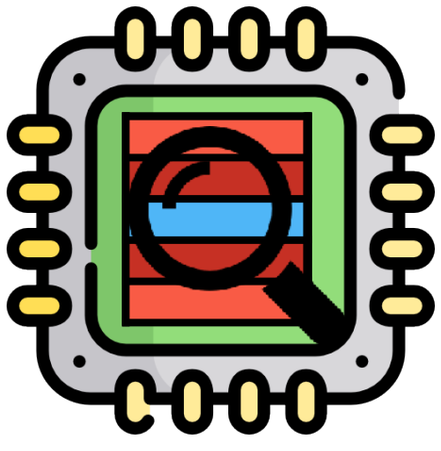
SPECTRE (Variant 2)

& Prime + Probe



SPECTRE

Victim program



1) Find SPECTRE gadget

2) Training

3) Prepare Fill cache with attacker's data

4) Attack Run

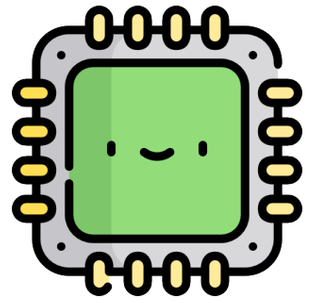
5) Extract Probe values in cache

Conclusions

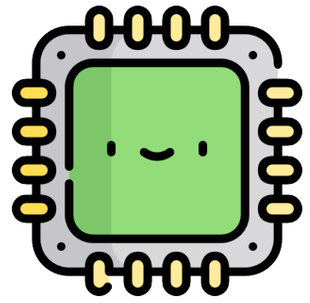
Conclusions

Conclusions

- Speculative execution key part of modern CPUs

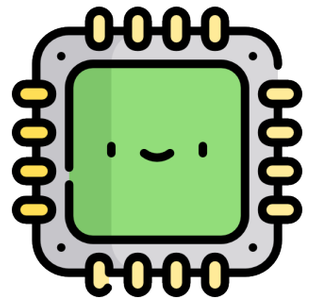


Conclusions



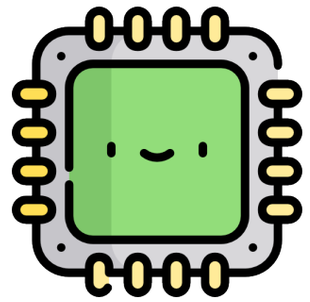
- Speculative execution key part of modern CPUs
- Rolling back only the architectural state is not enough to avoid attacks!

Conclusions



- Speculative execution key part of modern CPUs
- Rolling back only the architectural state is not enough to avoid attacks!
- We cannot simply get rid of speculative execution

Conclusions



- Speculative execution key part of modern CPUs
- Rolling back only the architectural state is not enough to avoid attacks!
- We cannot simply get rid of speculative execution

- Complex problem to solve
  - Promising solutions involve HW/SW co-design

Further reading

- *Kocher et al.*, Spectre attacks: exploiting speculative execution
- *Maisuradze and Rossow*, Speculose: analyzing the security implications of speculative execution in CPUs
- *Schwarz et al.*, NetSpectre: Read arbitrary memory over the network
- *Maisuradze and Rossow*, ret2spec: speculative execution using return stack buffers