CASPER

CS773-2022-Autumn: Computer Architecture for Performance and Security

Lecture 5: Time it to leak it (covert/side channels)

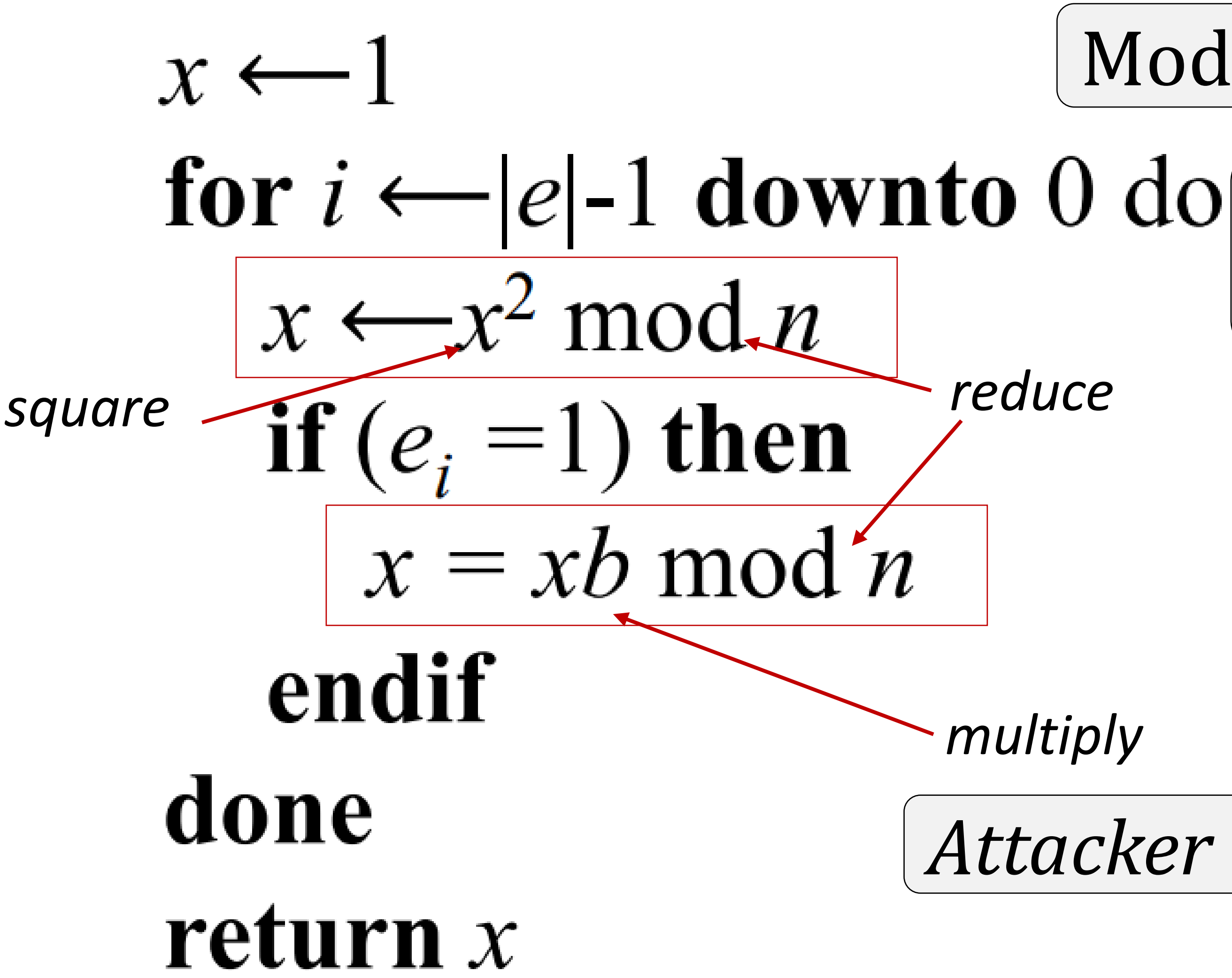
https://www.cse.iitb.ac.in/~biswa/

Phones on silence, please

Thank You

Information leakage

Modular exponentiation, b^e mod n



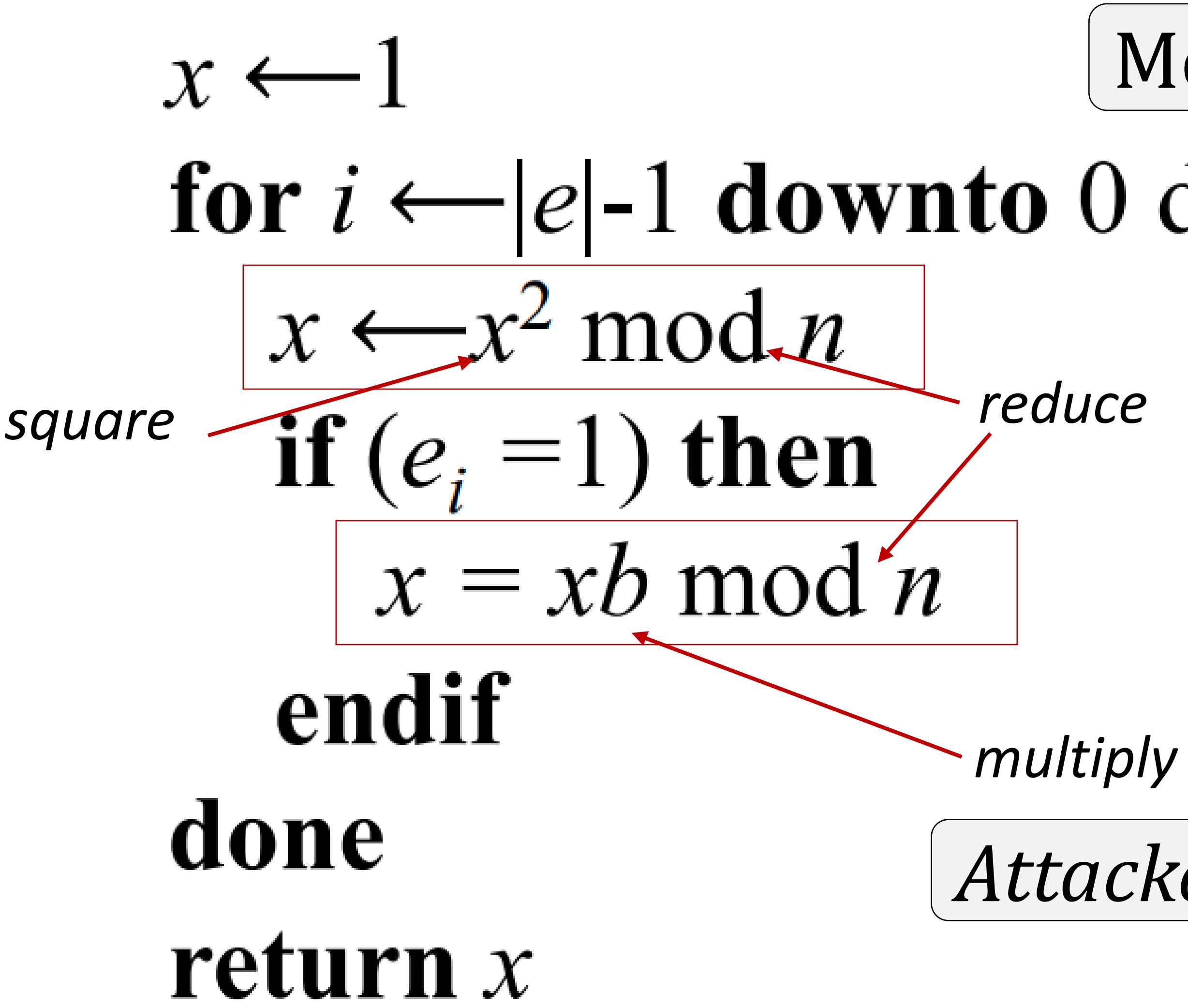
```
$x \leftarrow 1$
for $i \leftarrow |e|-1$ downto 0 do
    $x \leftarrow x^2 \bmod n$
    if ($e_i = 1$) then
        $x = xb \bmod n$
    endif
done
return $x$
```

Exponent *e* is used for decryption

Attacker tries to get the e

Information leakage

Modular exponentiation, b^e mod n



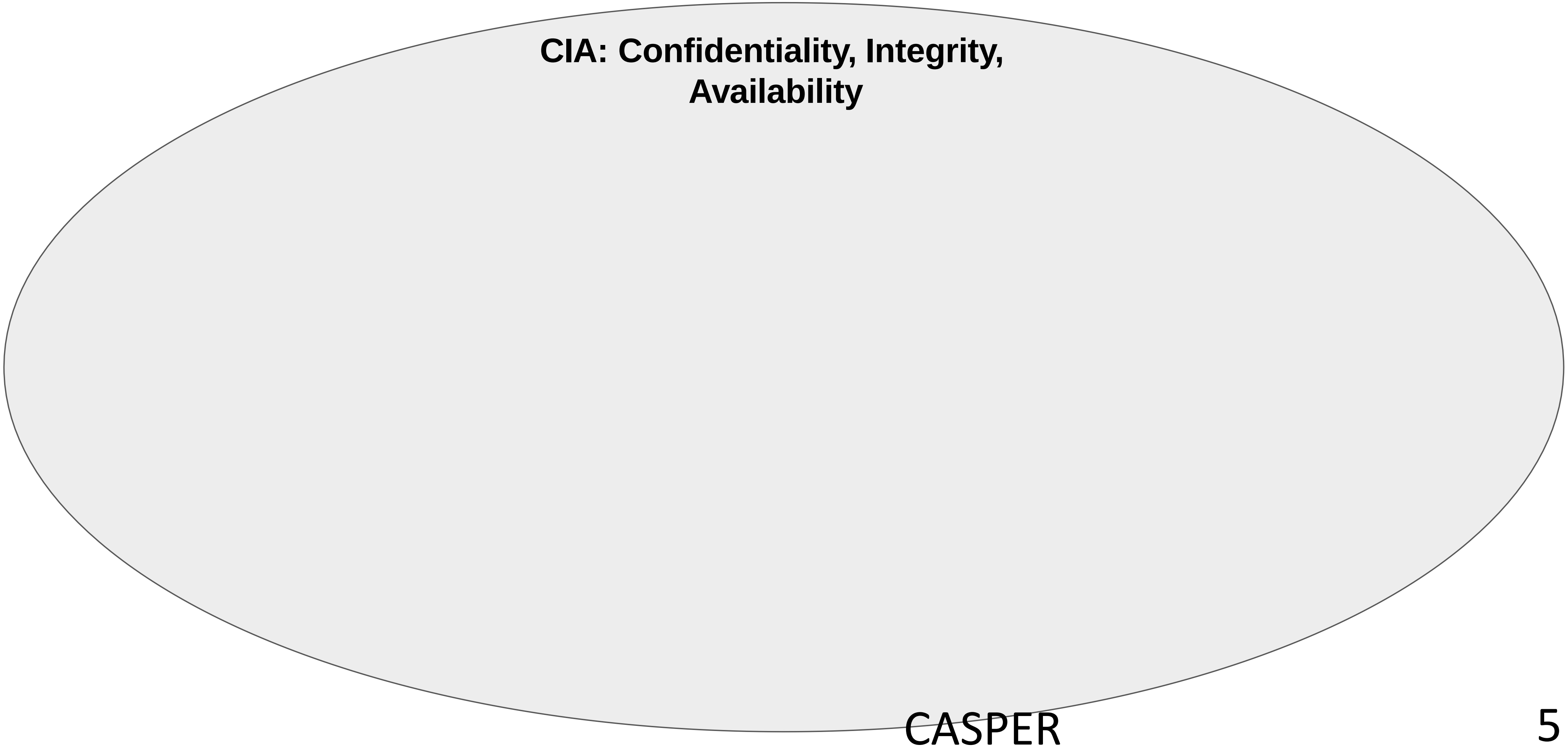
```
$x \leftarrow 1$
**for** $i \leftarrow |e|-1$ **downto** 0 do
    $x \leftarrow x^2 \bmod n$
    **if** $(e_i = 1)$ **then**
        $x = xb \bmod n$
    **endif**
**done**
**return** $x$
```

Exponent *e* is used for decryption

e_i = 0, Square Reduce (SR)
e_i = 1, SRMR

Attacker tries to get the e

Information leakage



CIA: Confidentiality, Integrity, Availability

Information leakage

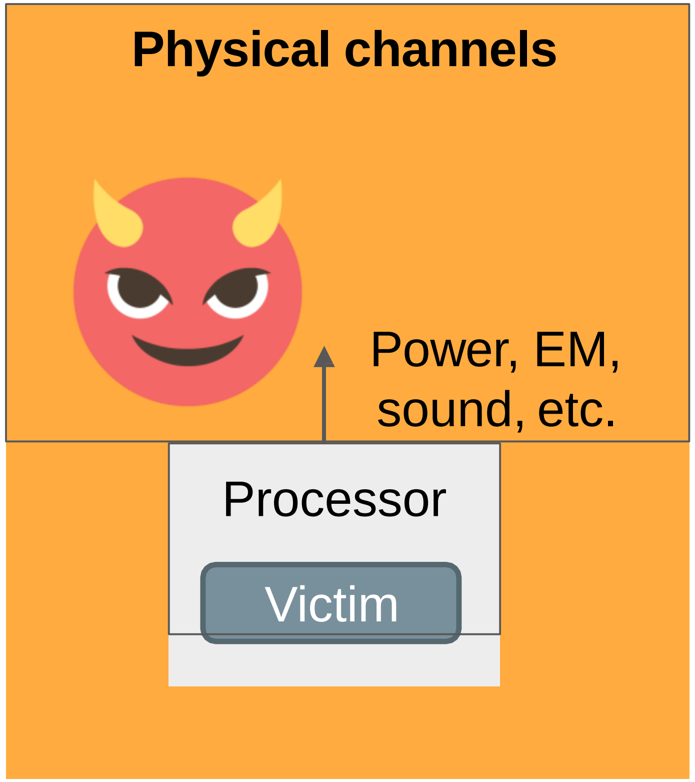
CIA: Confidentiality, Integrity, Availability

Confidentiality: was data being computed upon not revealed to an un-permitted party?

Integrity: was the computation performed correctly, returning the correct result?

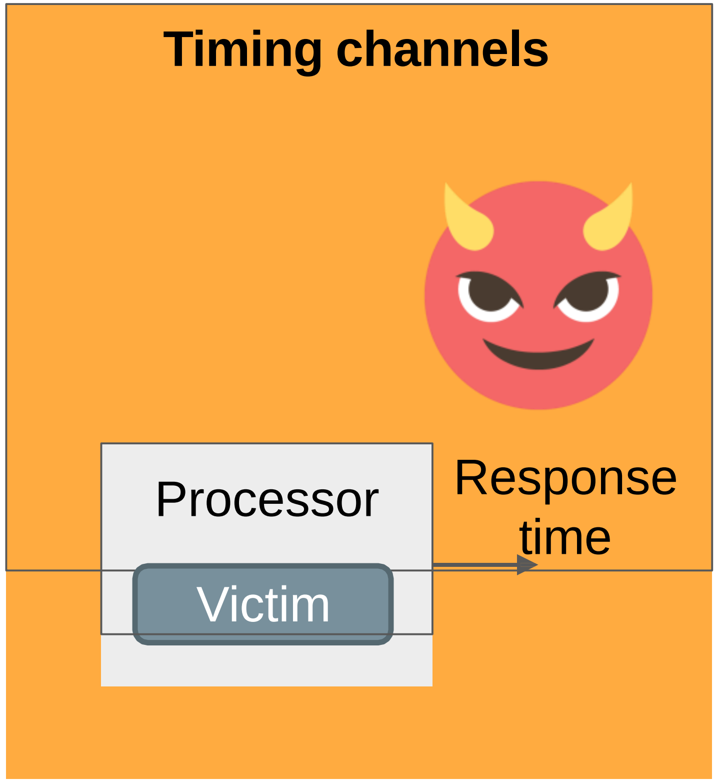
Availability: did the computational resource carry out the task at all?

Channels of Interest



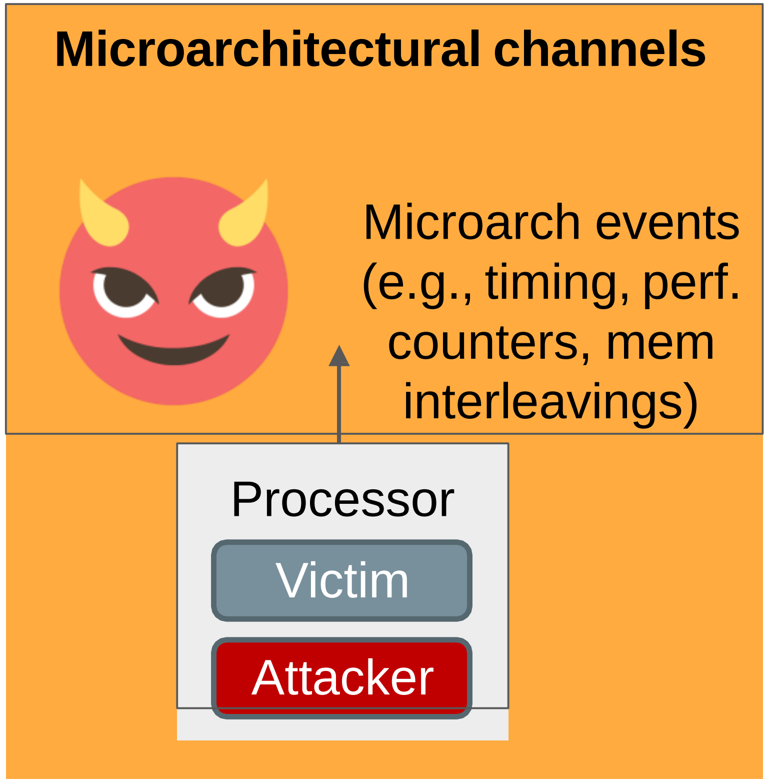
Physical channels

Attacker requires measurement equipment → physical access



Timing channels

Attacker may be remote (e.g., over an internet connection)



Microarchitectural channels

Microarch events (e.g., timing, perf. counters, mem interleavings)
Processor
Victim
Attacker

Attacker may be remote, or be co-located

Side/Covert Channel

- A **side channel** is an *unintended communication* between two or more parties
- A **covert channel** is an *intended communication* between two or more parties

(you upload a video on YouTube to communicate some information to your friends, if Gmail, whatsapp, call is not allowed)

In both cases:

- Communication should not be possible, following system semantics
- The physical channel used for the communication can be the same

Side channels → unintended → need de-noising
Covert channels can show “best case” leakage

Scope of these channels

- Inter-process(application) communication that can violate privilege boundaries

- Infer information from application’s data-dependent HW resource usage

Side/covert channels not in any interface specification (e.g. ISA).

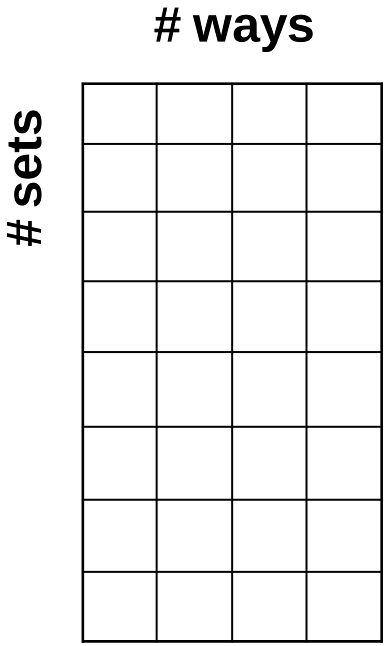
Therefore stealthy

- Sophisticated mechanisms needed to detect channel
- No permanent indication one has been exploited

Let’s try to send a bit

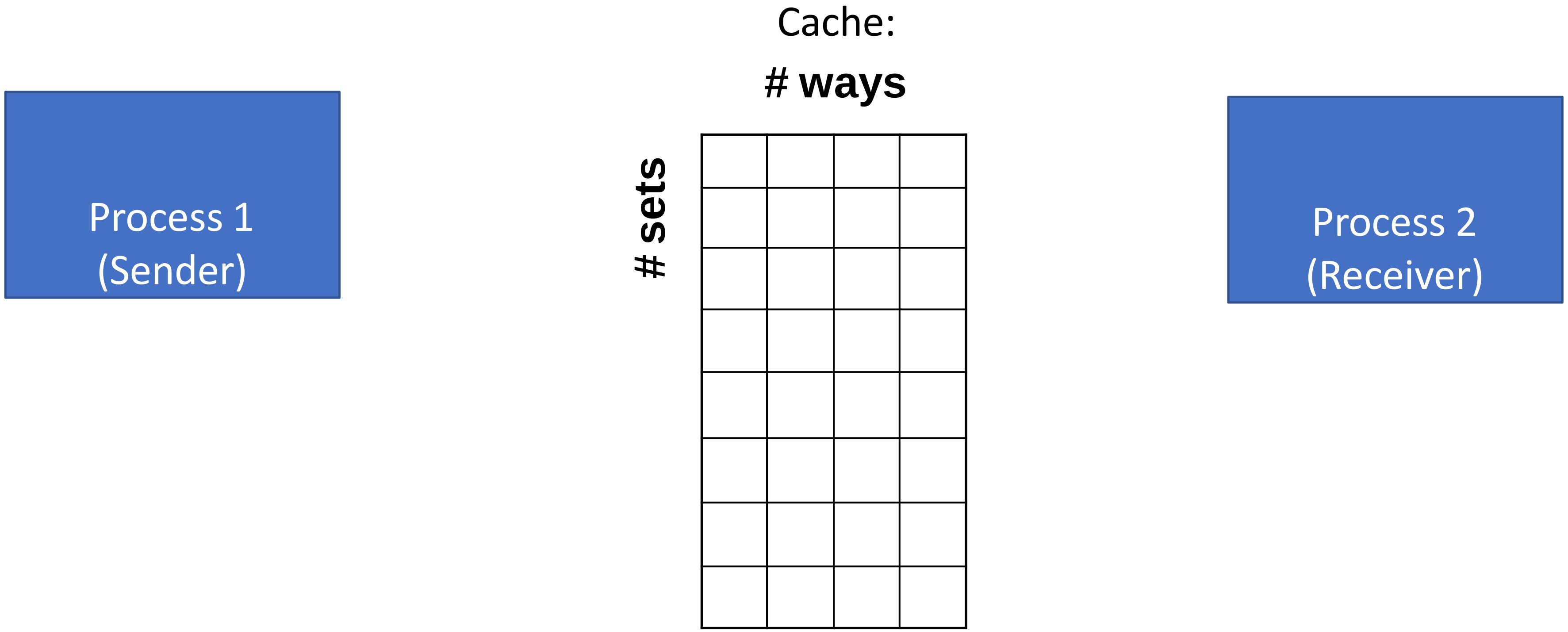
Two processes can agree on “dead drops”

Cache:



Let’s try to send a bit

Two processes can agree on “dead drops”



Process 1 (Sender)

Cache:

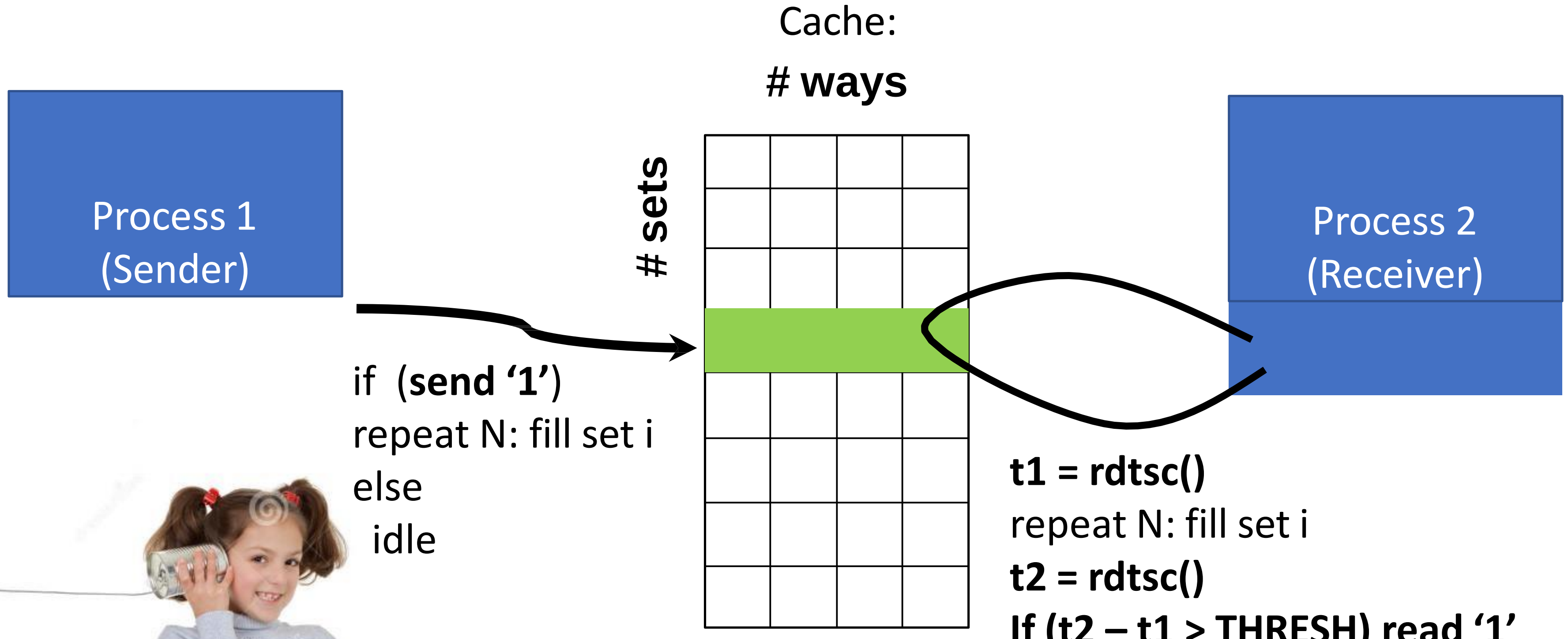
# ways

sets

Process 2 (Receiver)

Let’s try to send a bit

Two processes can agree on “dead drops”



```
if  (send ‘1’)
repeat N: fill set i
else
  idle
```

```
t1 = rdtsc()
repeat N: fill set i
t2 = rdtsc()
If (t2 – t1 > THRESH) read ‘1’
Else read ‘0’
```

How is it different from legitimate communication

send(msg)
Channel
recv()

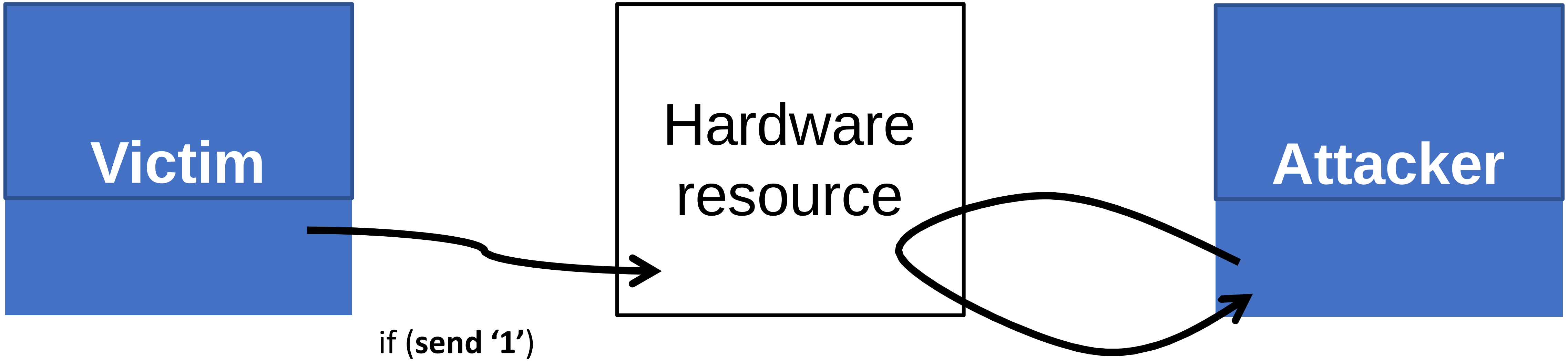
Normal communication

```
include <socket.h>

void send(bit msg) {
  socket.send(msg);
}

bit recv() {
  return socket.recv(msg);
}
```

Covert Channel communication

```
void send(bit msg) {
  // pressure on cache
}

bit recv() {
  st = time();
  // pressure on cache
  return time() – st > THRESH;
}
```

From Covert to Side Channel



Covert channel:

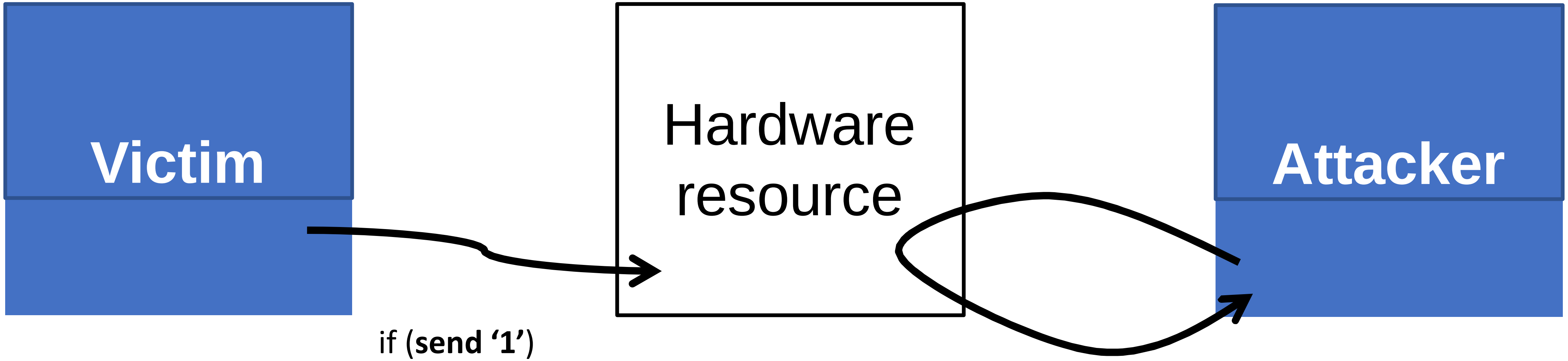
```
if (send '1')
Use resource
else
  idle
```

Side channel:

```
if (secret)
  Use resource
else
  idle
```

```
t1 = rdtsc()
Use resource
t2 = rdtsc()

if (t2 – t1 > THRESH) read '1'
else read '0'
```

From Covert to Side Channel



Covert channel:

```
if (send '1')
Use resource
else
  idle
```

Side channel:

```
if (secret)
  Use resource
else
  idle
```

```
t1 = rdtsc()
Use resource
t2 = rdtsc()

if (t2 – t1 > THRESH) read '1'
else read '0'
```

Information leakage: Again…

```
$x \leftarrow 1$
for $i \leftarrow |e|-1$ downto 0 do
    $x \leftarrow x^2 \bmod n$
    if $(e_i = 1)$ then
        $x = xb \bmod n$
    endif
done
return $x$
```

square

reduce

multiply

Modular exponentiation, $b^e \bmod n$

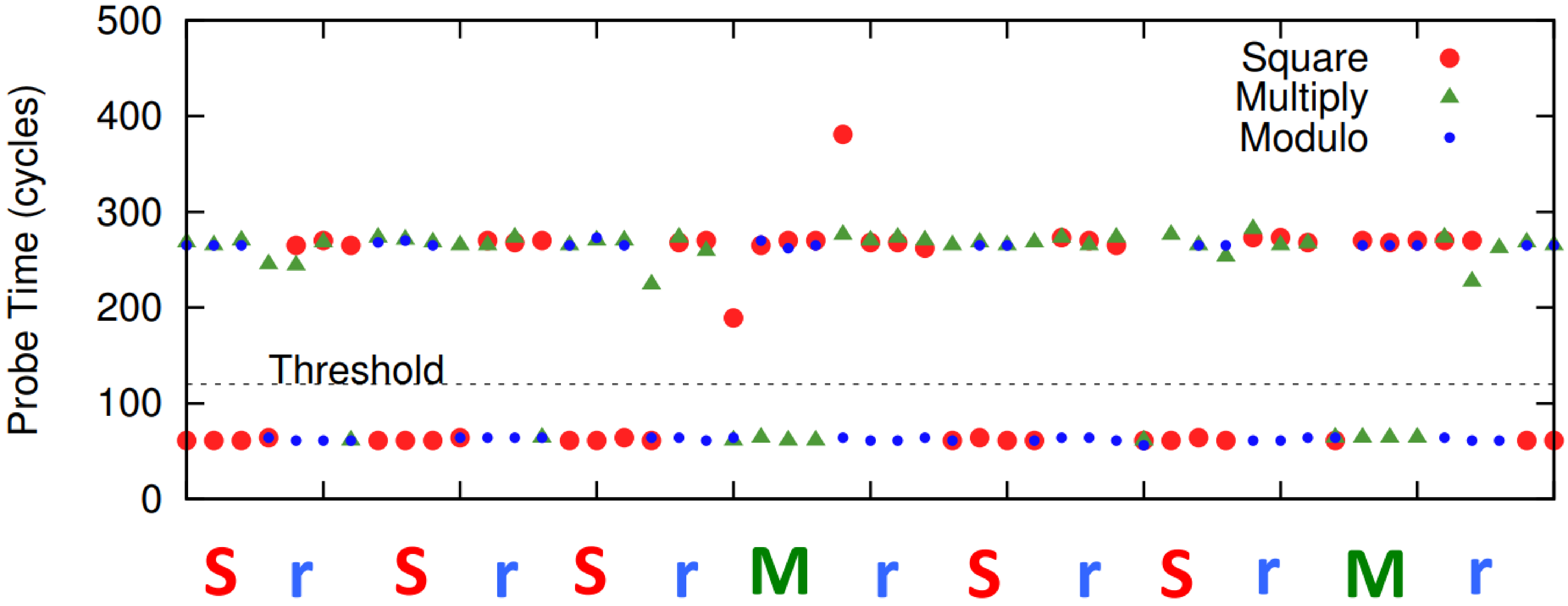
Exponent e is used for decryption

e_i = 0, Square Reduce (SR)

e_i = 1, SRMR

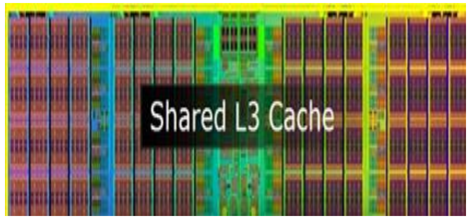
Attacker tries to get the e

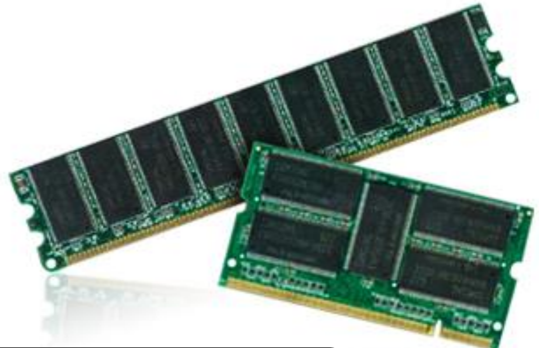
Timing Channel



Side/Covert Channel: Summary

Spy

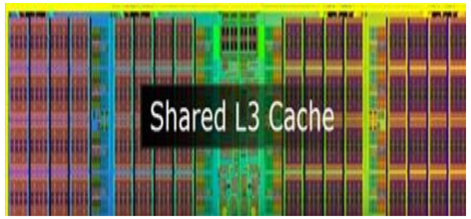




Victim

Side-channel attacks

Let’s play



Oh Yes!!

Covert-channel attacks

PAUSE

Flush based attacks

Victim

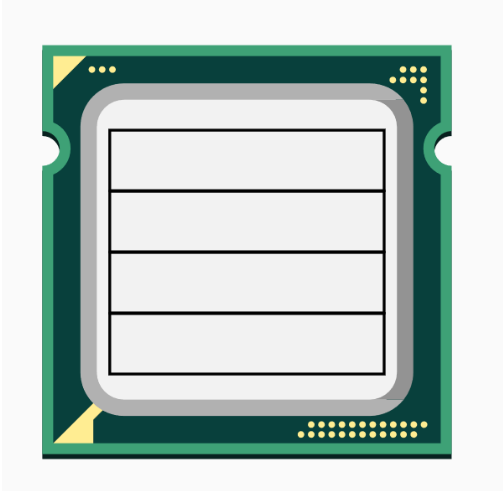
```
If secret=1 do
    access(&a)
else // secret=0
    no-access
```

Attacker

```
    flush(&a)
t1=start_timer
    access(&a)
t2=end_timer
```

Fast – 1

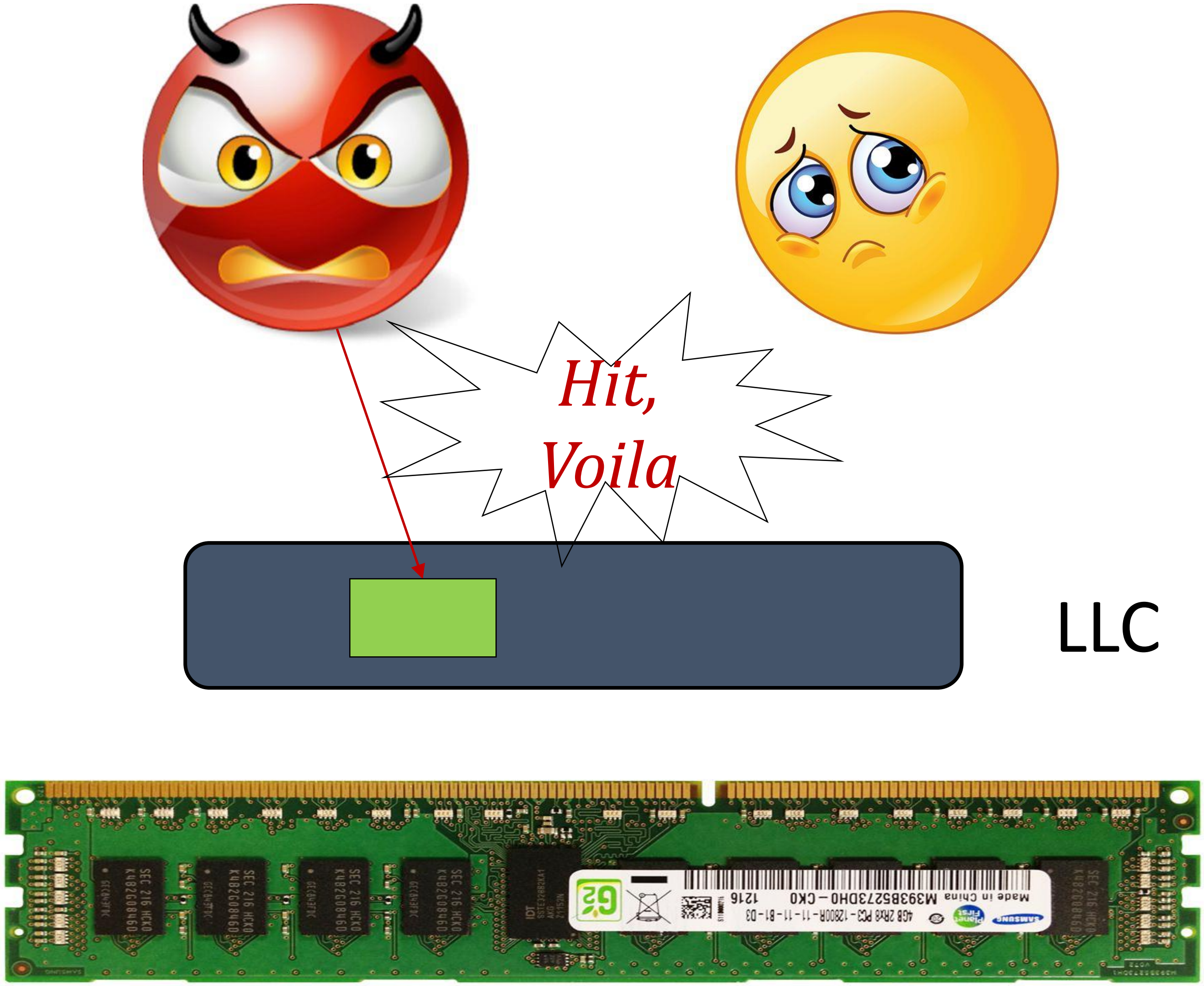
Slow – 0



Clflush instruction

Invalidates from every level of the cache hierarchy in the cache coherence domain the cache line that contains the linear address specified with the memory operand. If that cache line contains modified data at any level of the cache hierarchy, that data is written back to memory. The source operand is a byte memory location.

Clflush instruction



Step 0:Spy *maps* the shared library, shared in the cache

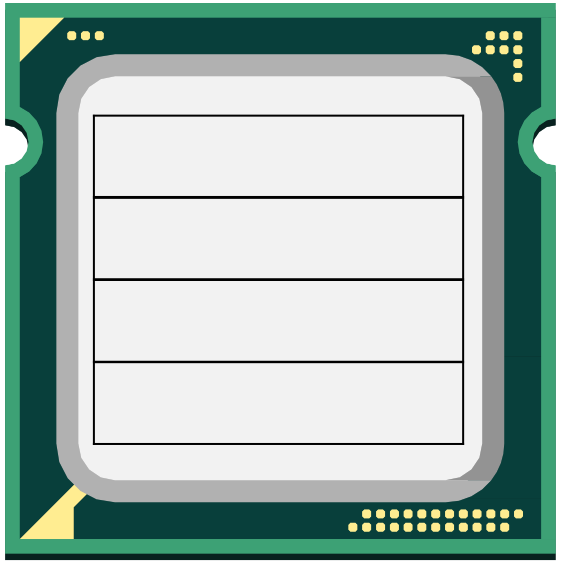
Step 1:Spy *flushes* the cache block

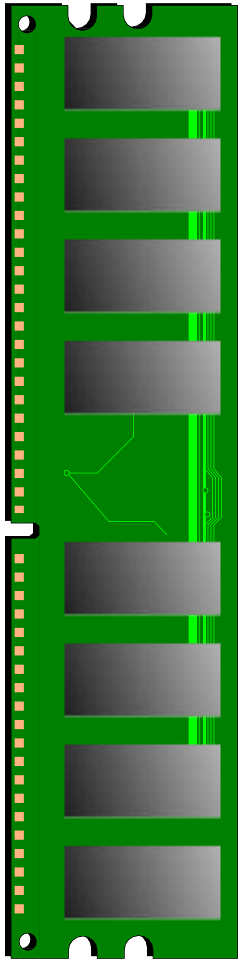
Step 2: Victim *reloads* the cache block

Step 3: Spy *reloads* the cache block (hit/miss)

Let’s see step by step

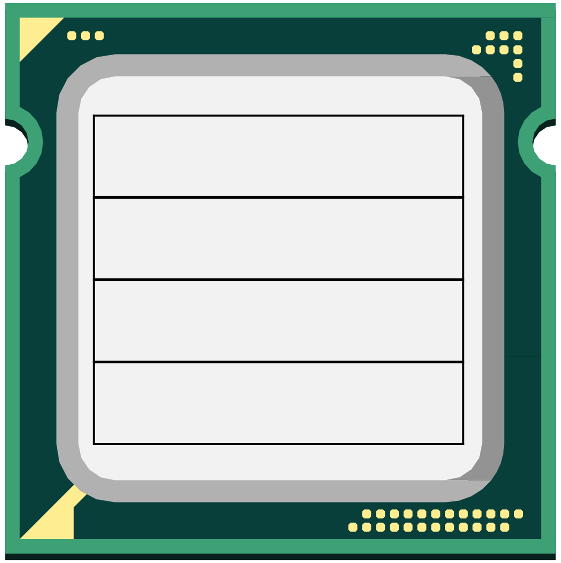
```
printf("%d", i);
printf("%d", i);
```

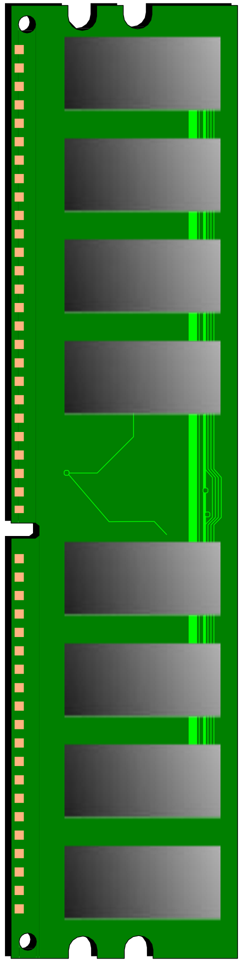




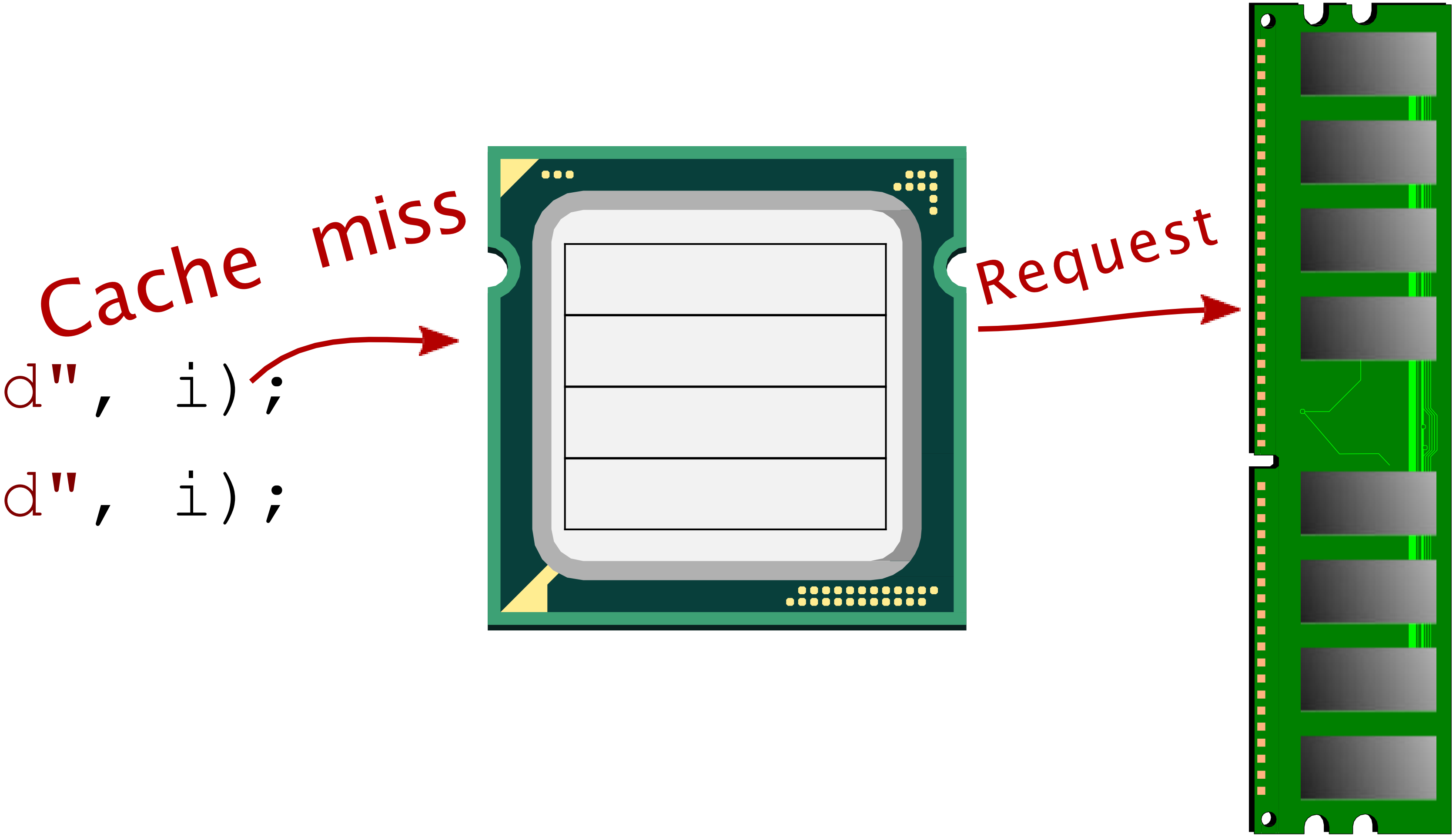
Cache miss

```
printf("%d", i);
printf("%d", i);
```





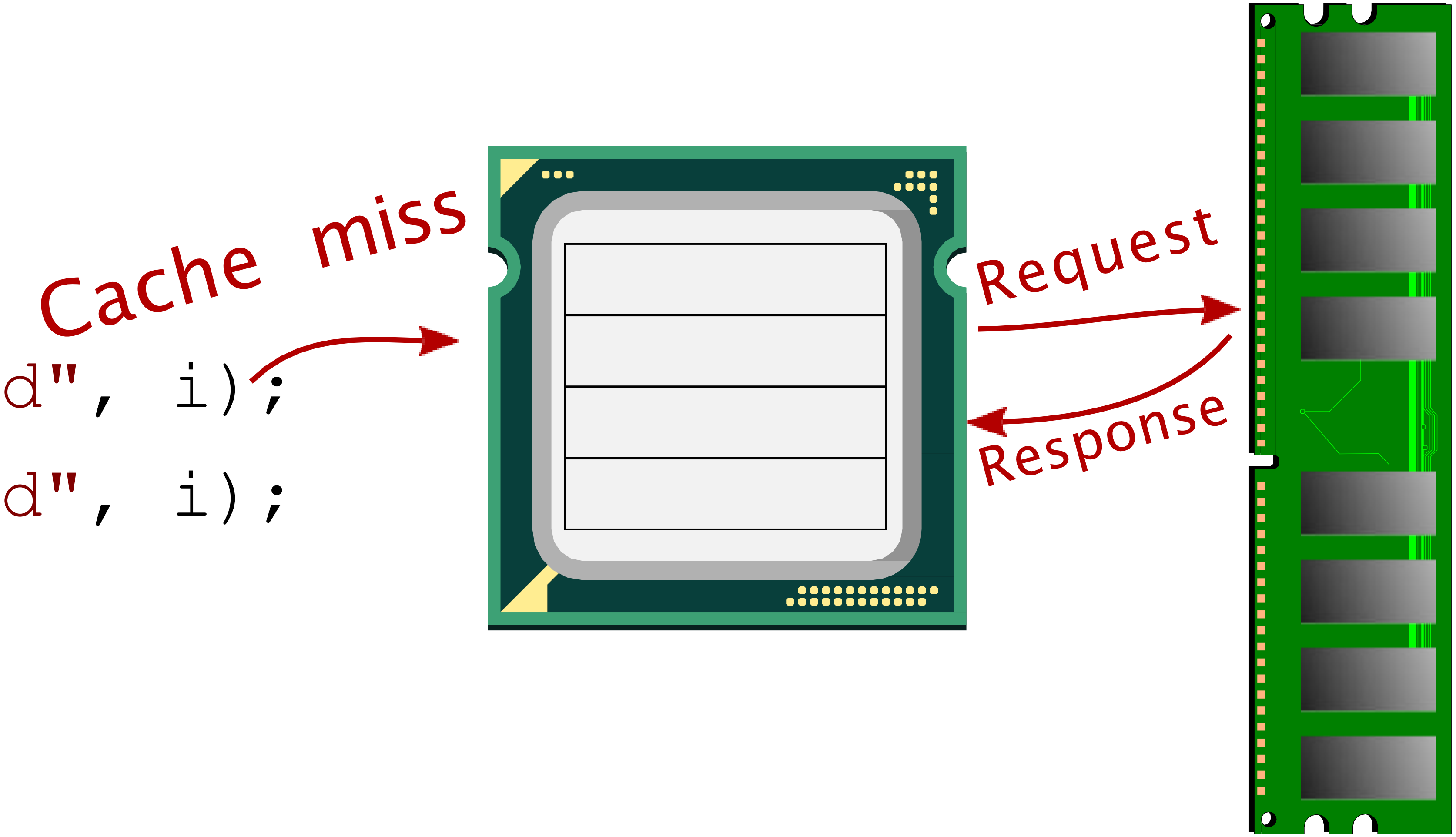
```
printf("%d", i);
printf("%d", i);
```



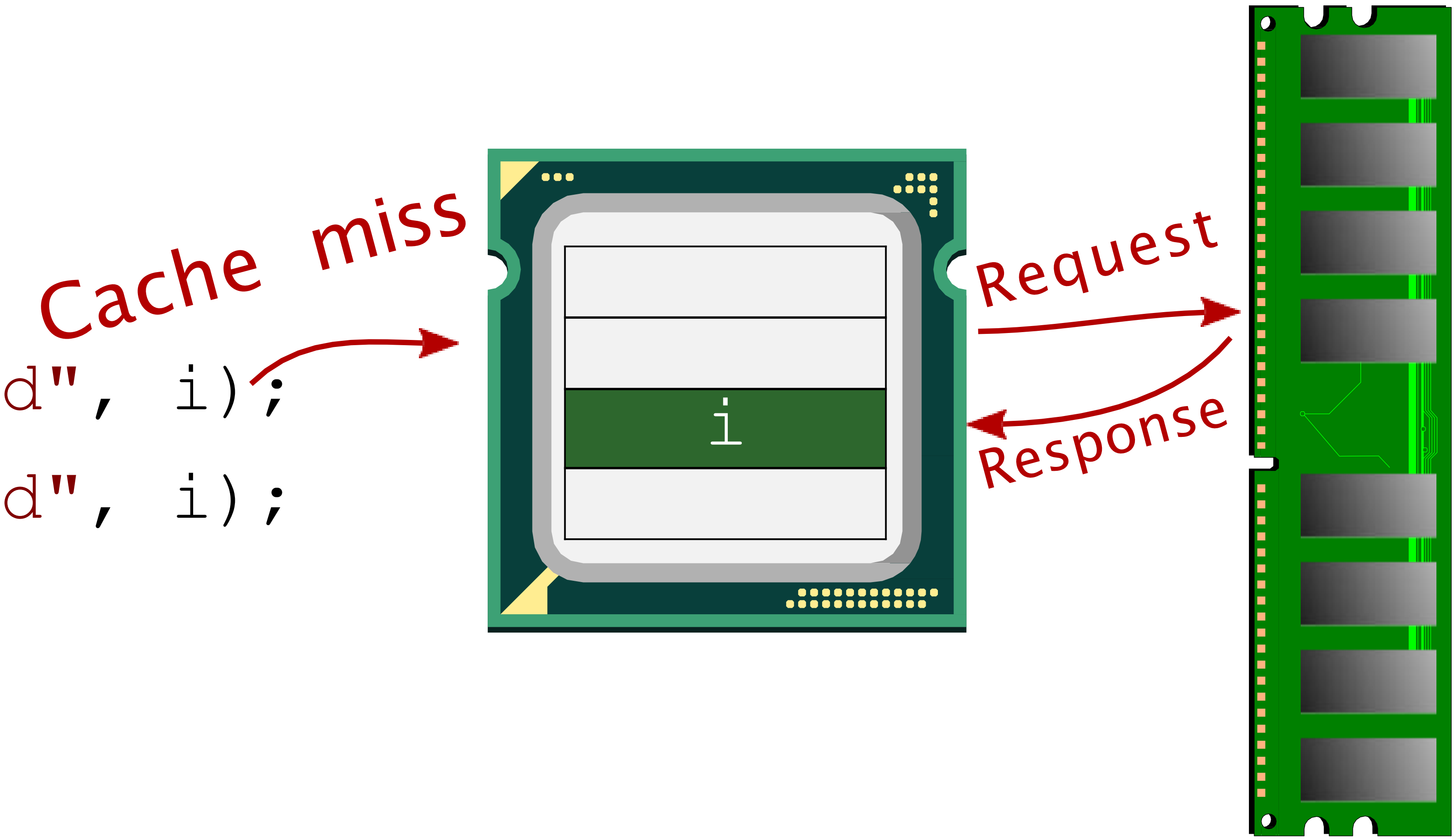
Cache miss

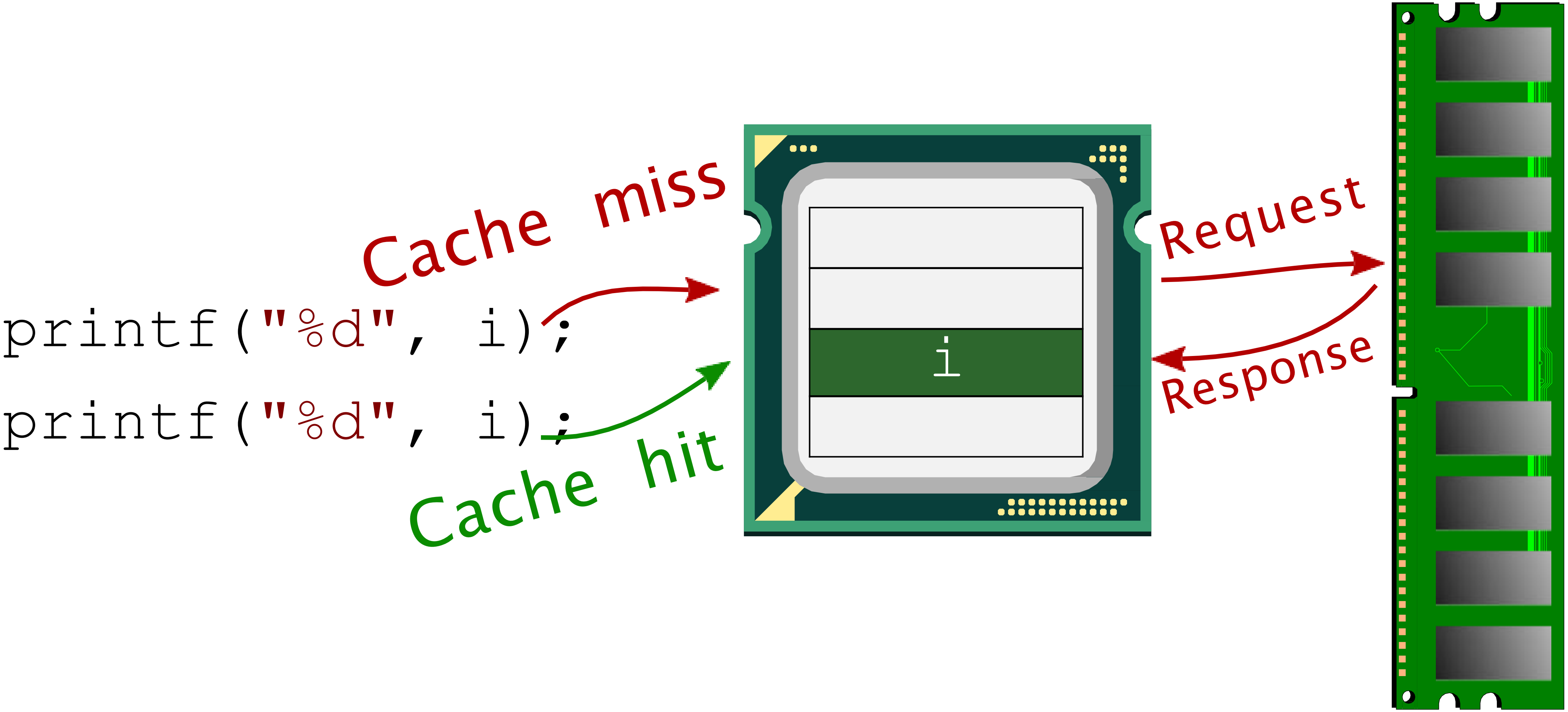
Request

```
printf("%d", i);
printf("%d", i);
```



```
printf("%d", i);
printf("%d", i);
```





```
printf("%d", i);
printf("%d", i);
```

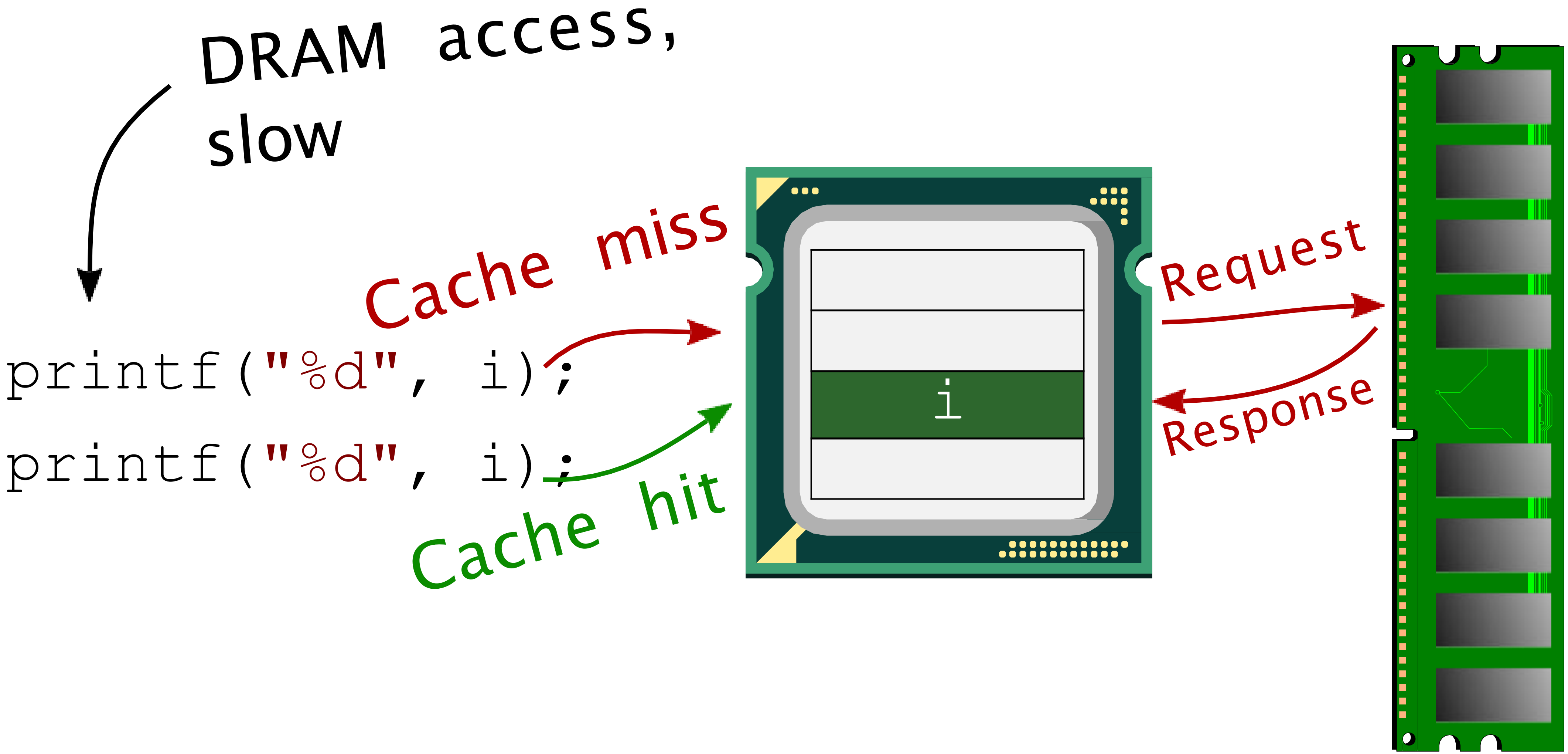
Cache miss

Cache hit

i

Request

Response



DRAM access, slow

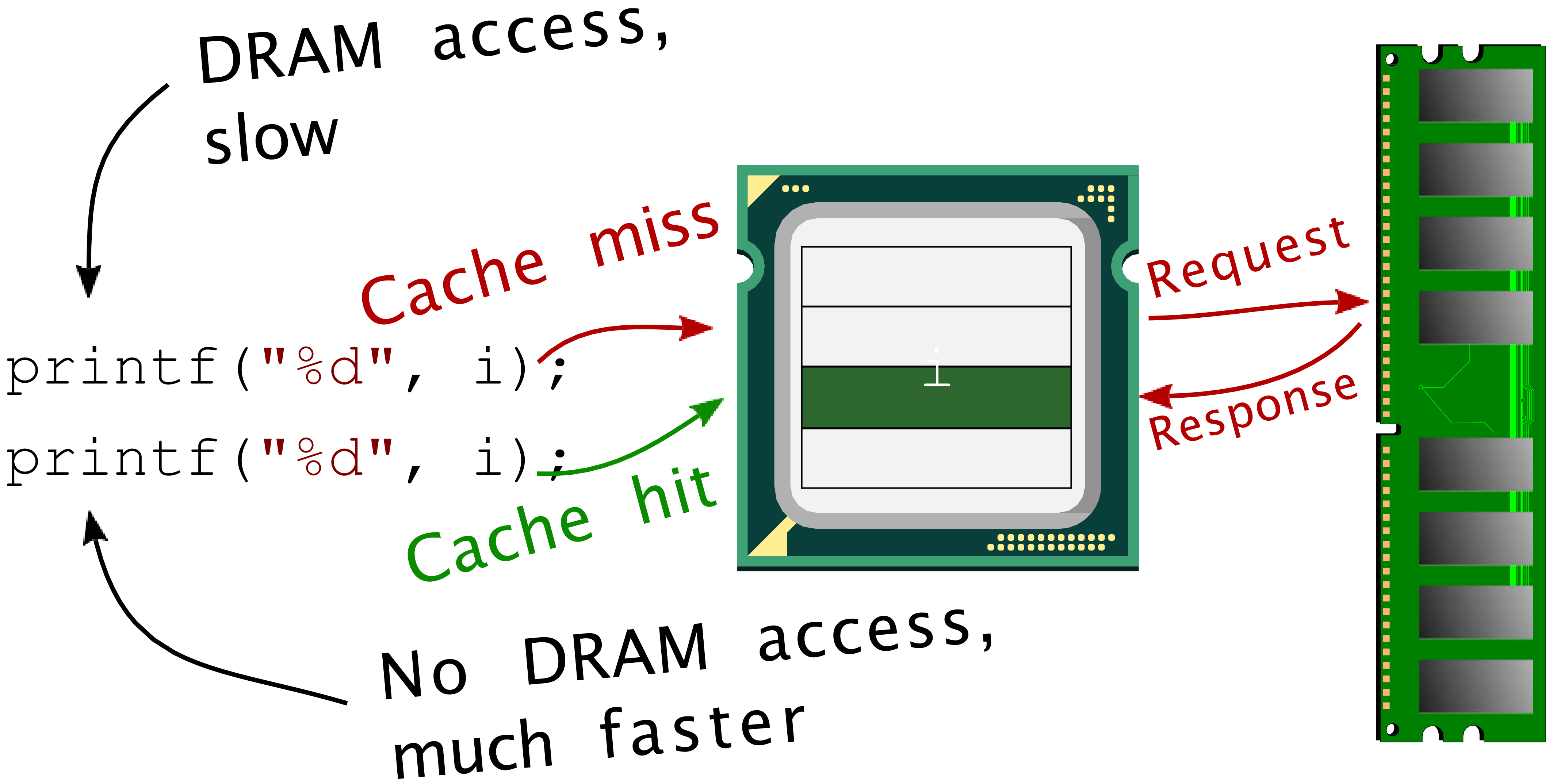
```
printf("%d", i);
printf("%d", i);
```

Cache miss

Cache hit

Request

Response



DRAM access, slow

```
printf("%d", i);
printf("%d", i);
```

Cache miss

Cache hit

Request

Response

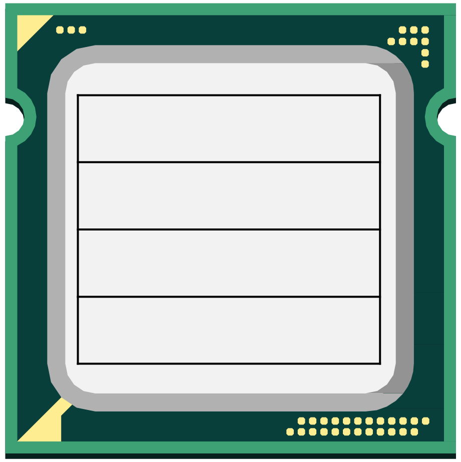
No DRAM access, much faster

Shared Memory

ATTACKER

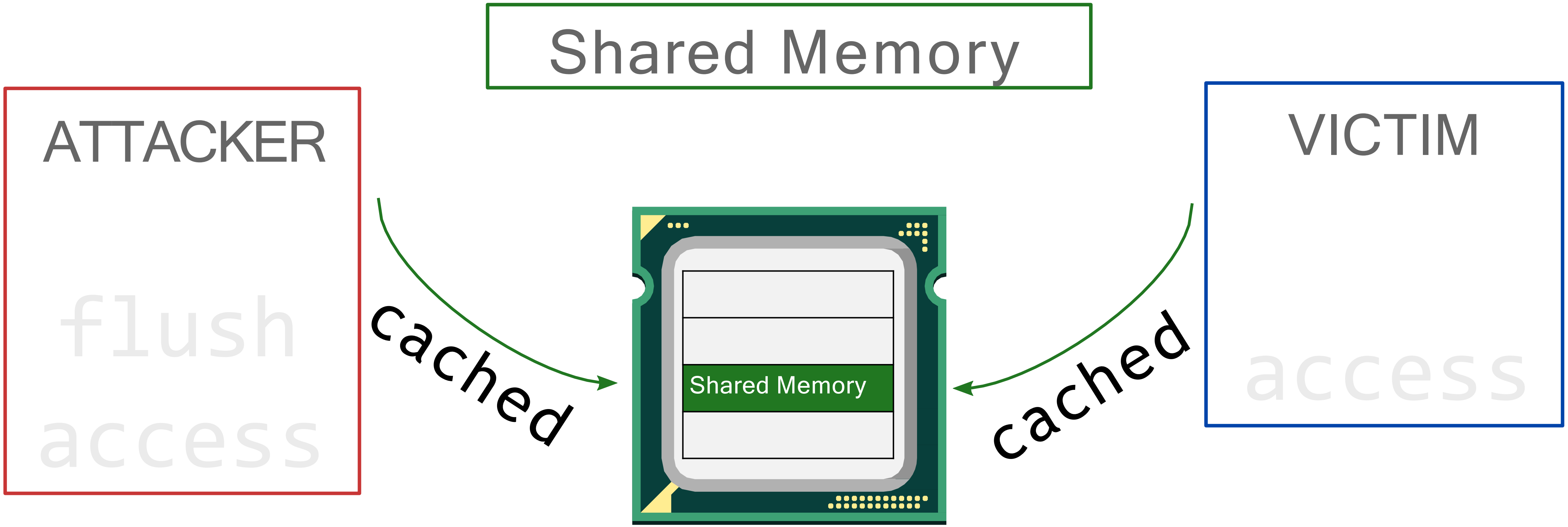
flush

access



VICTIM

access



Shared Memory

ATTACKER

flush

access

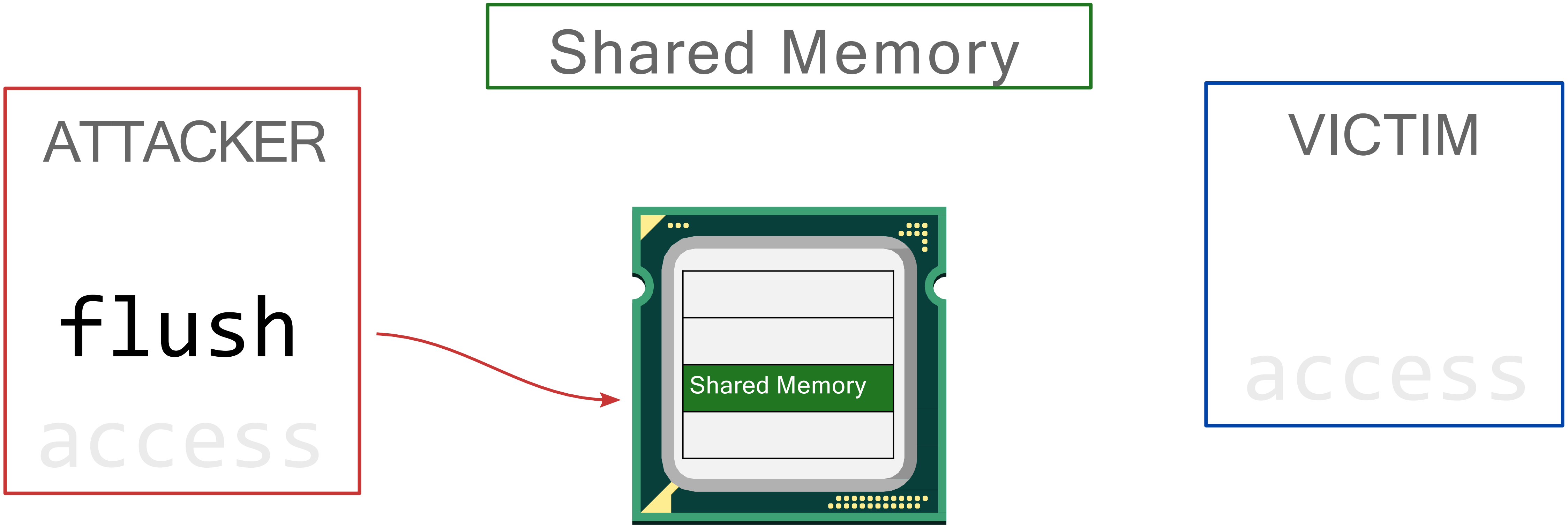
cached

Shared Memory

cached

VICTIM

access



Shared Memory

ATTACKER

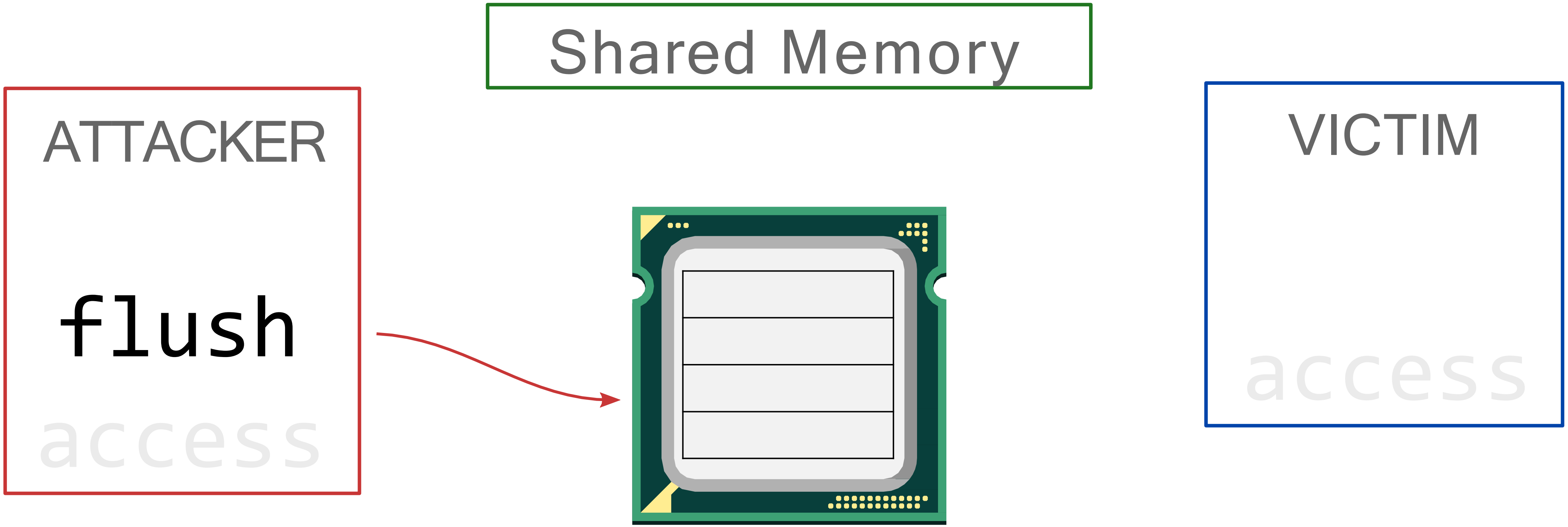
flush

access

Shared Memory

VICTIM

access



Shared Memory

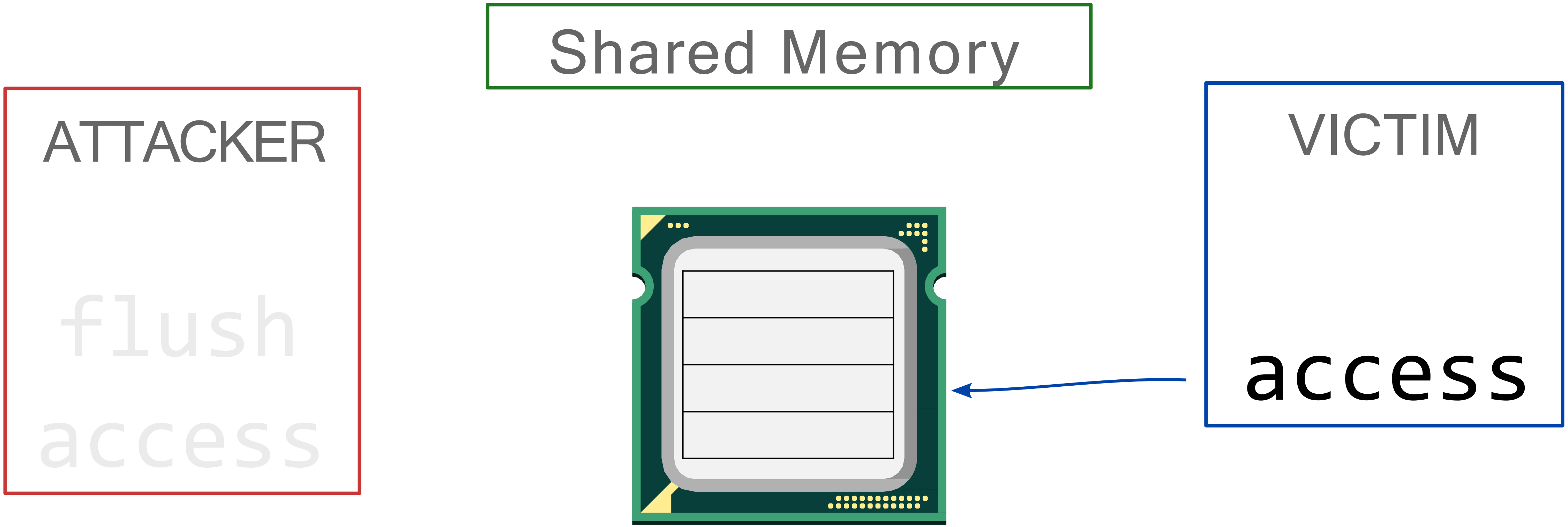
ATTACKER

flush

access

VICTIM

access



Shared Memory

ATTACKER

flush

access

VICTIM

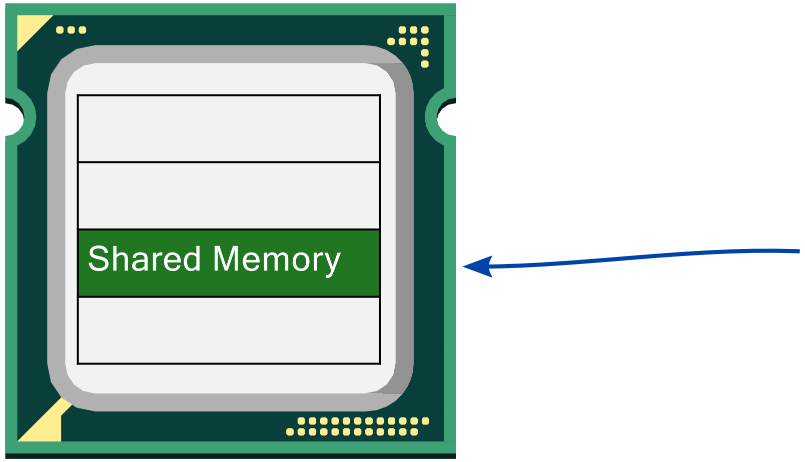
access

Shared Memory

ATTACKER

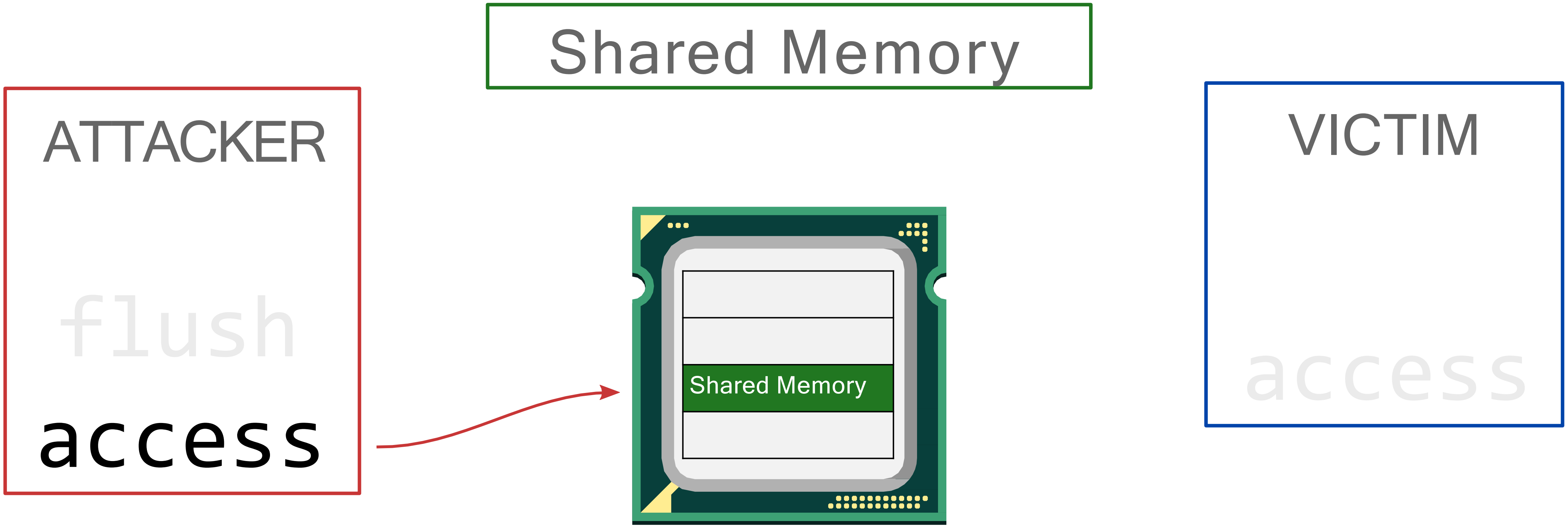
flush

access



VICTIM

access



Shared Memory

ATTACKER

flush

access

Shared Memory

VICTIM

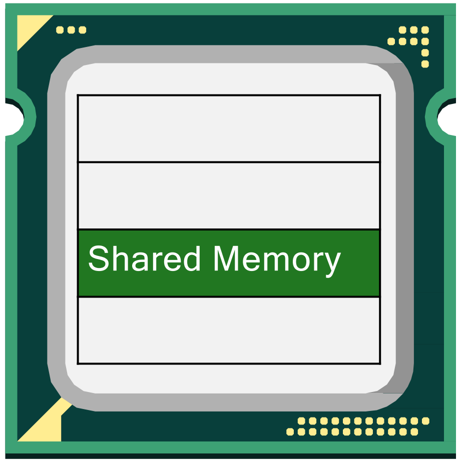
access

Shared Memory

ATTACKER

flush

access

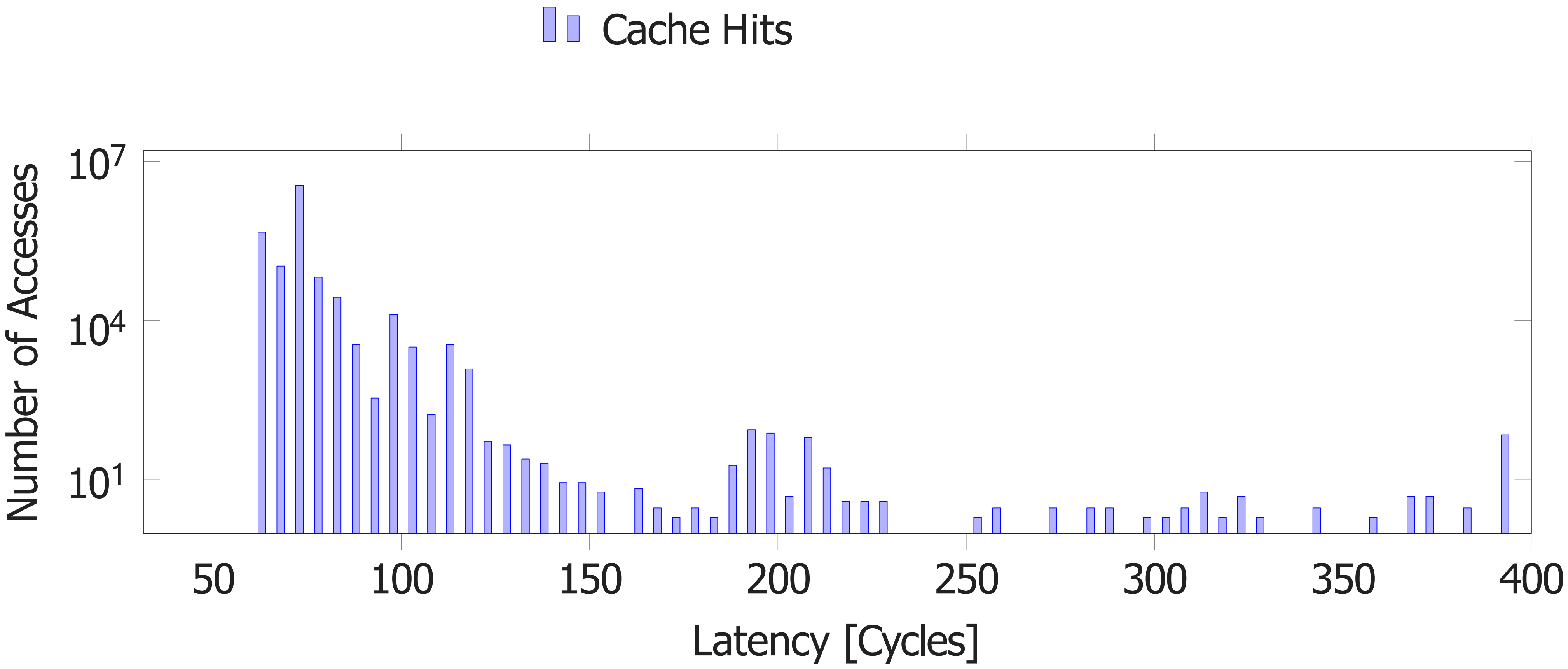


VICTIM

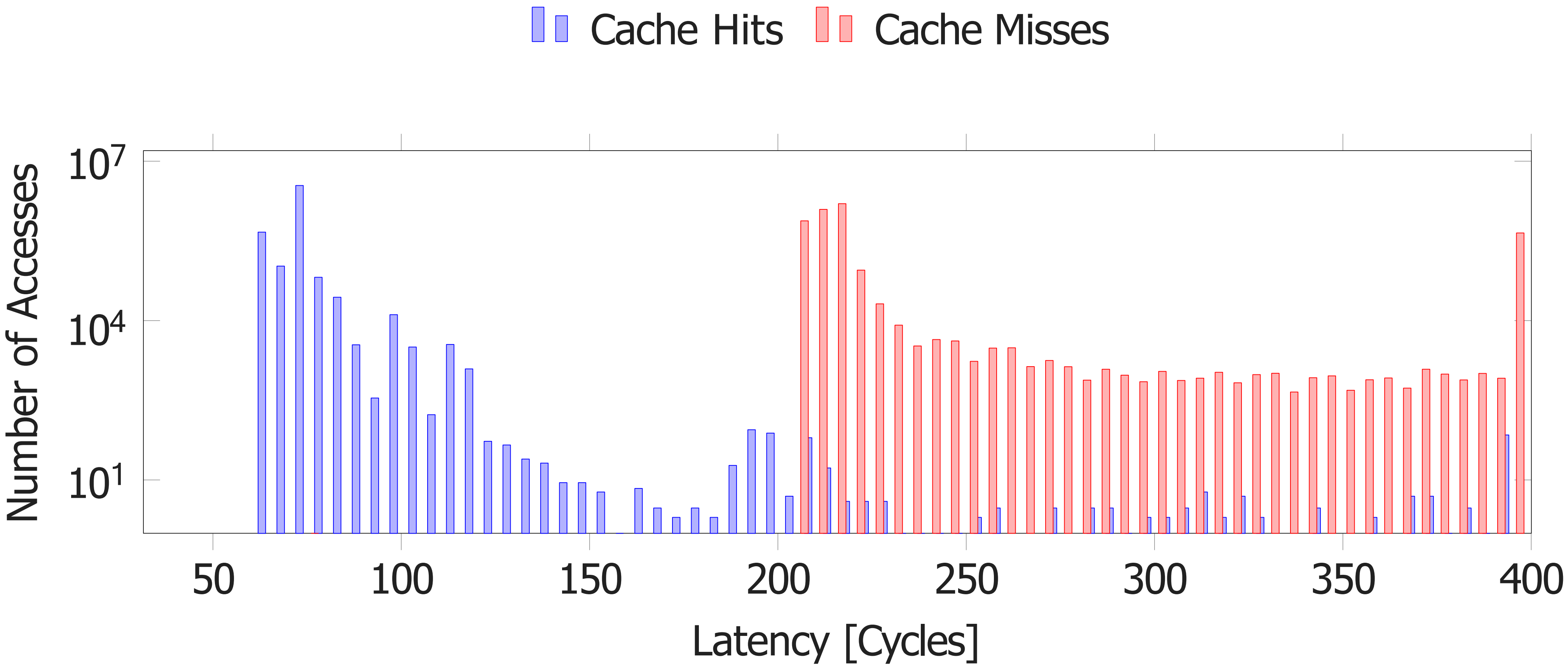
access

fast if victim accessed data,
slow otherwise

Cache Hits


Cache Hits
Number of Accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Latency [Cycles]

Cache Hits and Misses



How to measure time?

rdtsc instruction : (Read Time-Stamp Counter) instruction is used to determine how many CPU ticks took place since the processor was reset.

Questions of interest

What is the use of clflush from an OS point of view?

What is the use of clflush from an end-user point of view?

Demo Time: Let’s see it working