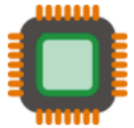


CASPER

# CS773-2025-Spring: Computer Architecture for Performance and Security

## Lecture 3: Catch the Cache-II

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


CASPER

# Recap

- Basics of caches/cache hierarchy
- Security at a high level, what does it mean
- Side/covert channels
- Any interesting side/covert channels in IITB?
- Today: Cache-based side channels

# Information leakage in the real world

$x \leftarrow 1$

**for** $i \leftarrow |e|\text{-}1$ **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

*Attacker tries to get the e*

# Information leakage

$x \leftarrow 1$

**for** $i \leftarrow |e|\text{-}1$ **downto** 0 **do**

$x \leftarrow x^2 \bmod n$

*square*

*reduce*

**if** $(e_i = 1)$ **then**

$x = xb \bmod n$

**endif**

*multiply*

**done**

**return** $x$

Modular exponentiation, $b^e$ mod n

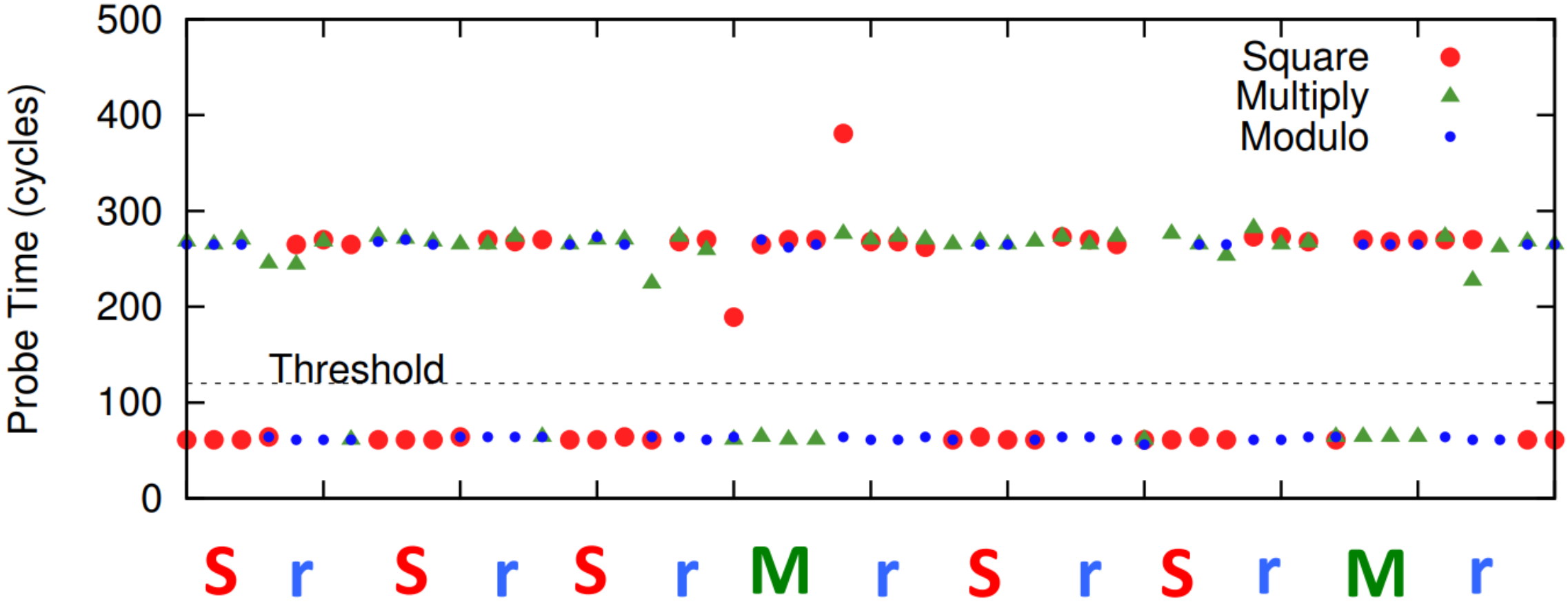
Exponent $e$ is used for decryption

$e_i = 0$, *Square Reduce (SR)*

$e_i = 1$, *SRMR*

*Attacker tries to get the e*

# Timing Channel



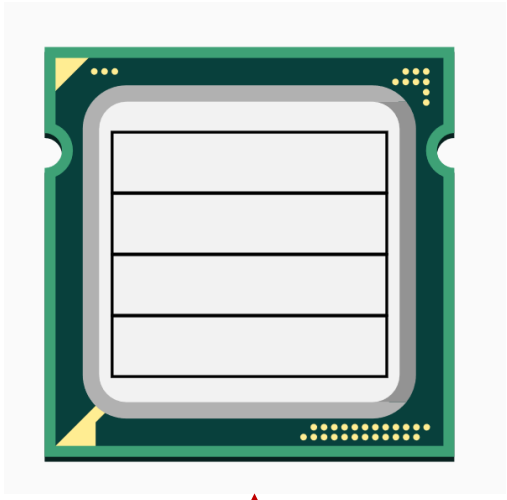
# Flush based attacks

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

Victim

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

Attacker



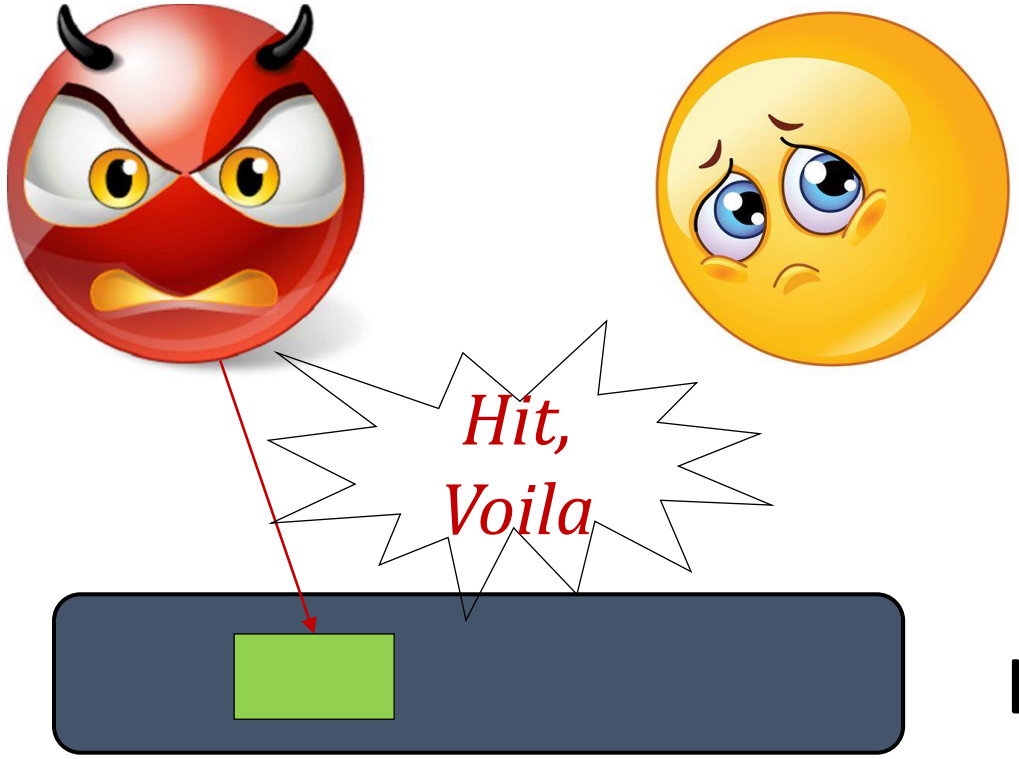
Fast – 1

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

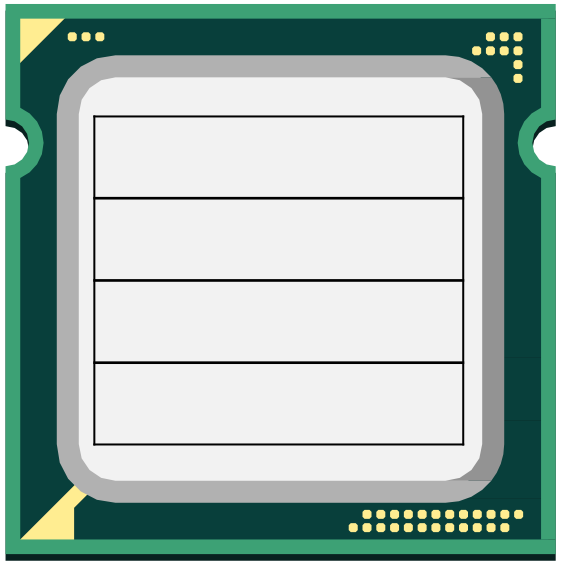
Step 1:Spy *flushes* the cache block

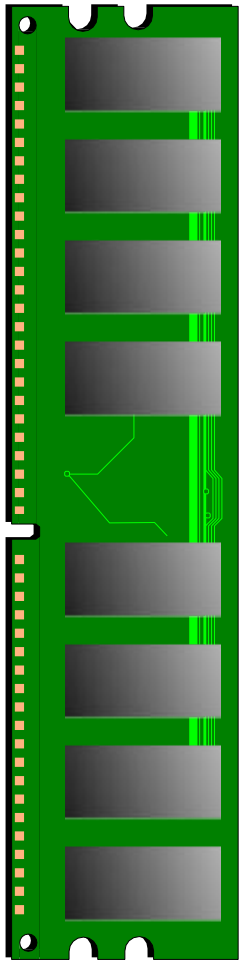
Step 2: Victim *reloads* the cache block

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

# Let’s see step by step

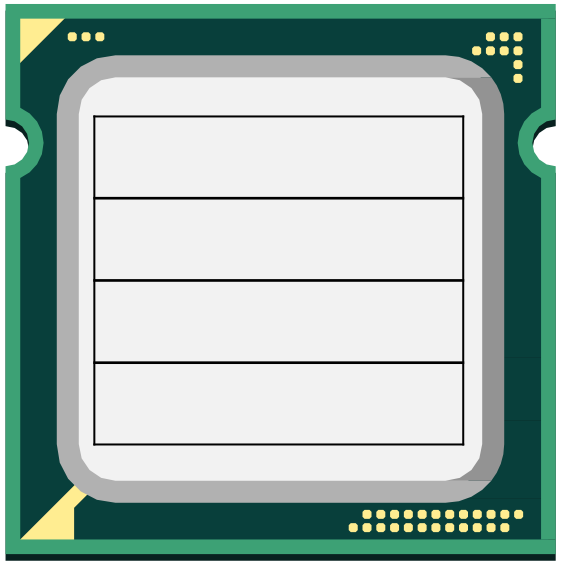
```
printf("%d", i);
printf("%d", i);
```

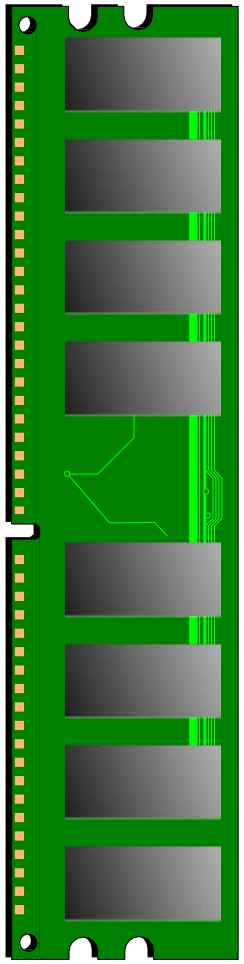




```
printf("%d", i);
printf("%d", i);
```

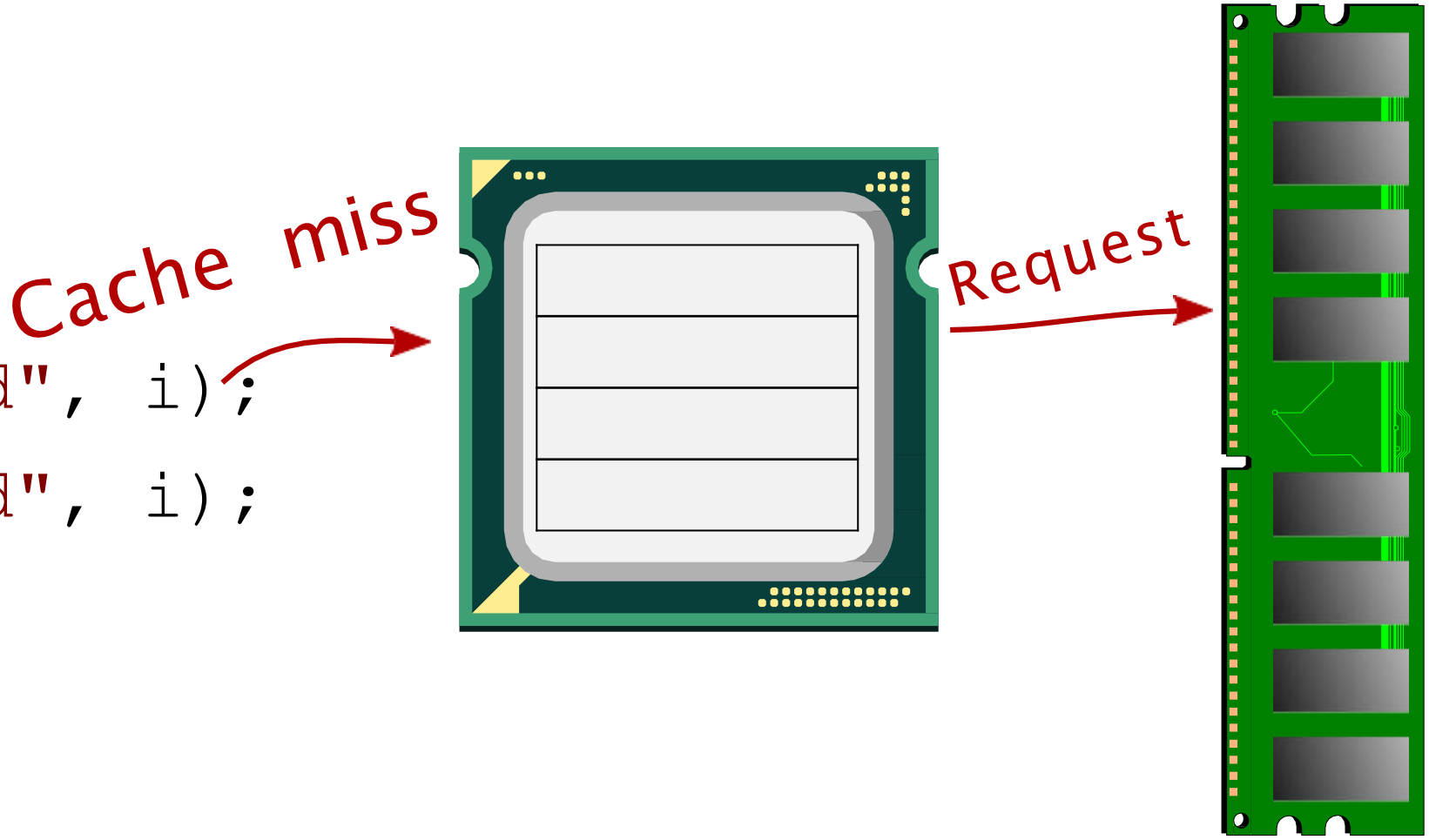
Cache miss



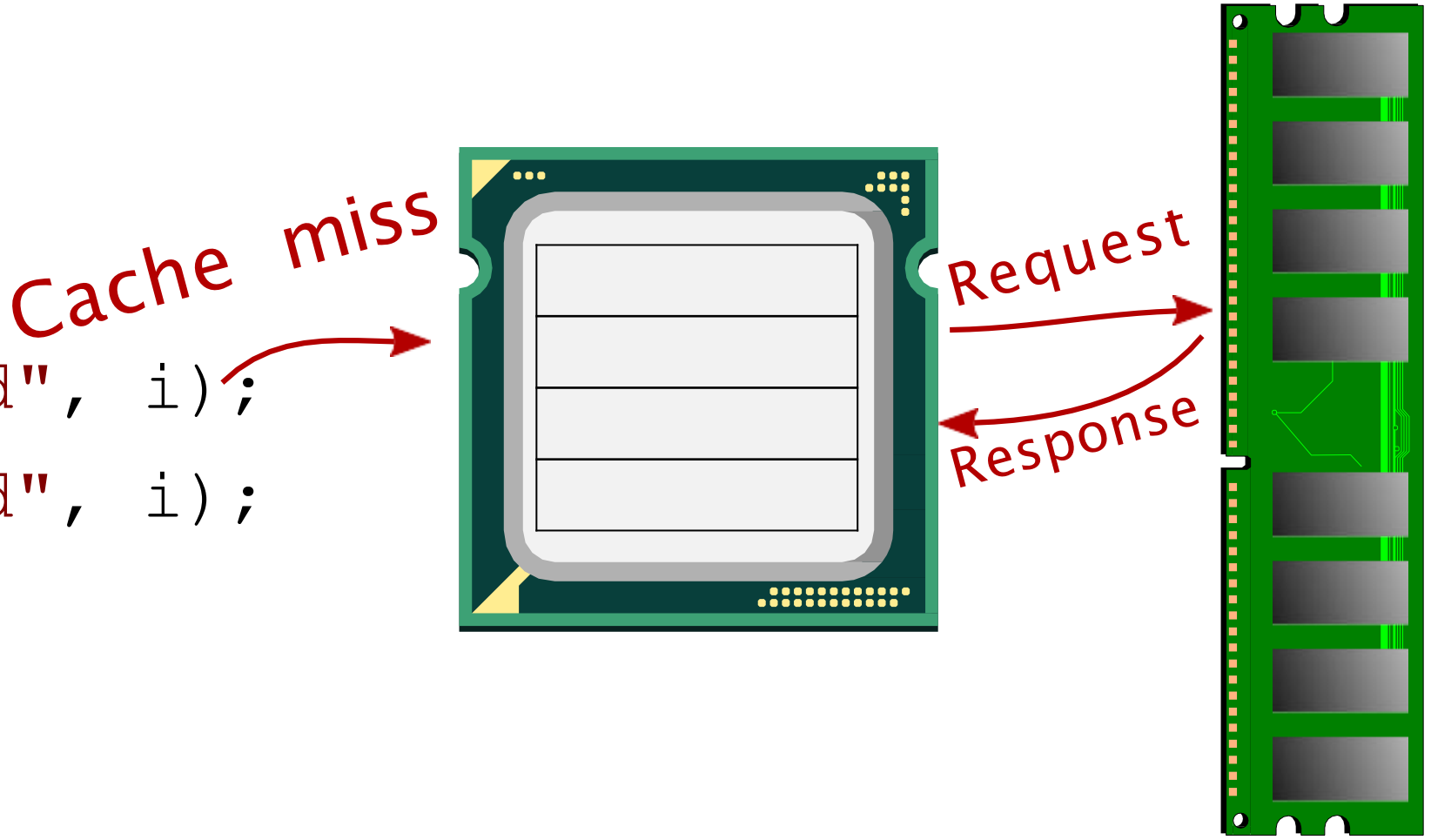


CASPER

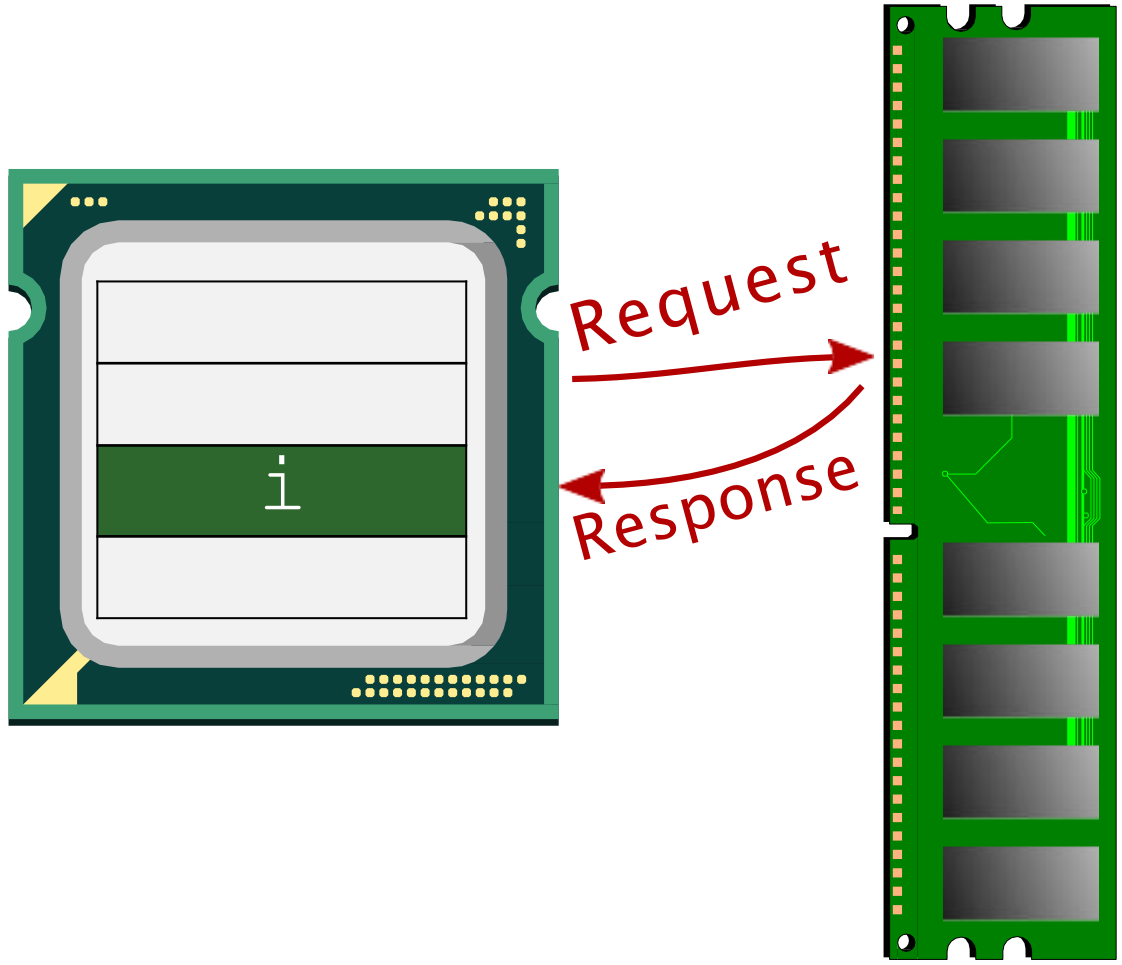
```
printf("%d", i);
printf("%d", i);
```

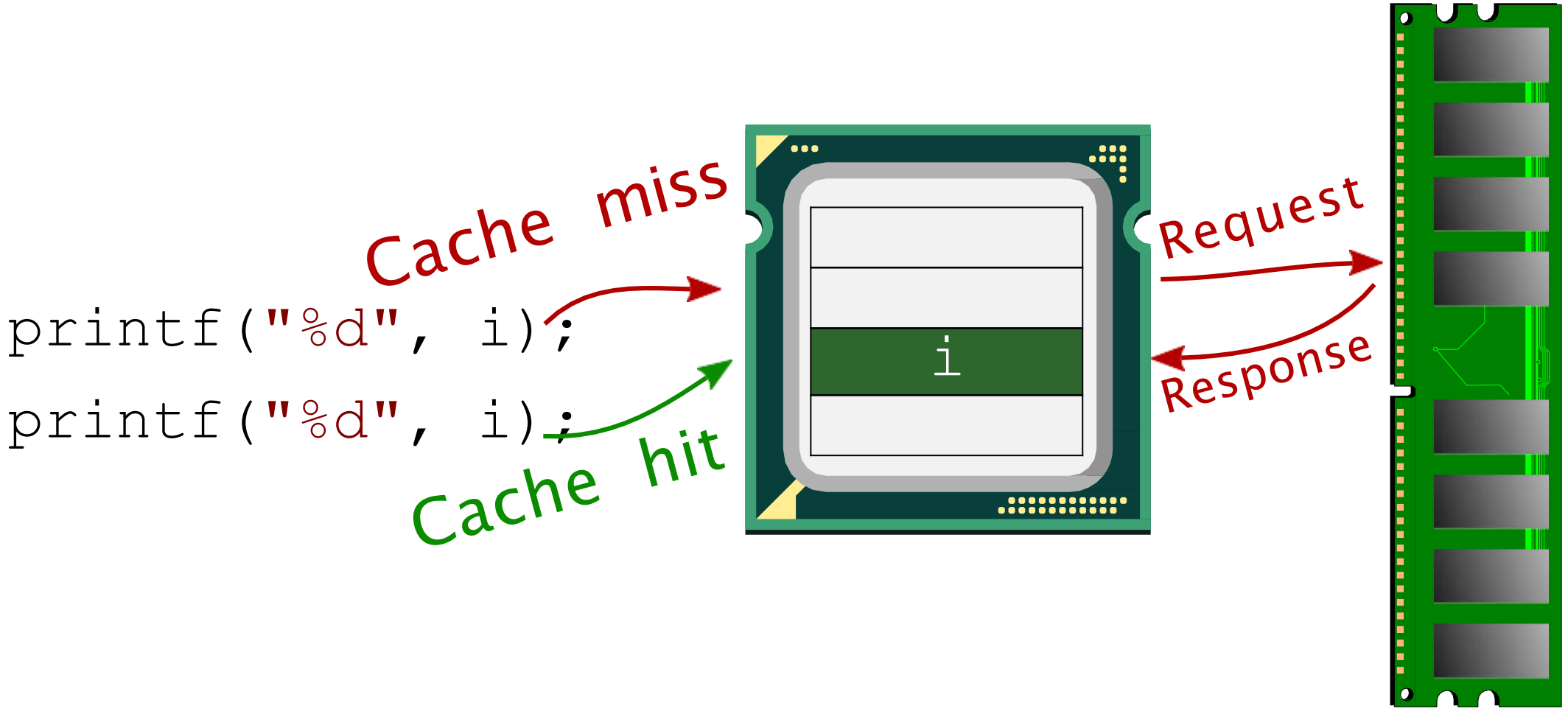


```
printf("%d", i);
printf("%d", i);
```



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



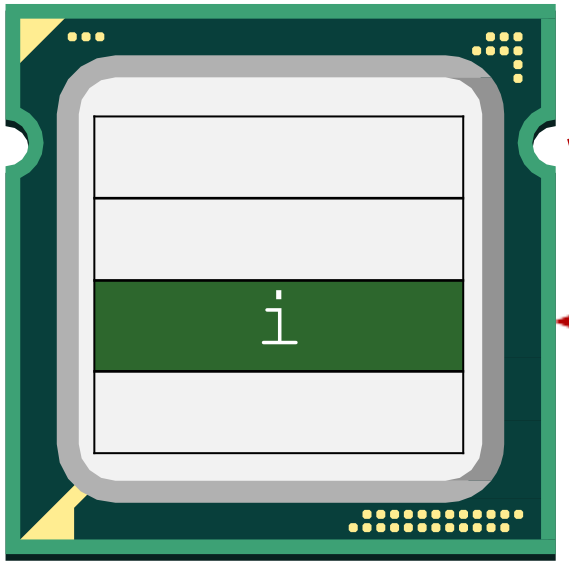

Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
i
Request
Response

DRAM access, slow

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

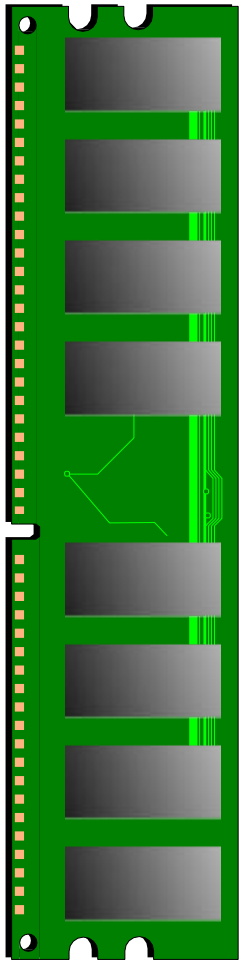
Cache miss

Cache hit




Request

Response

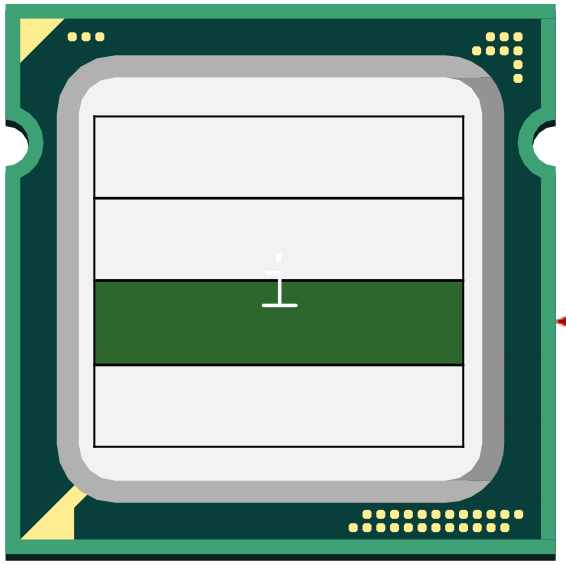


DRAM access, slow

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

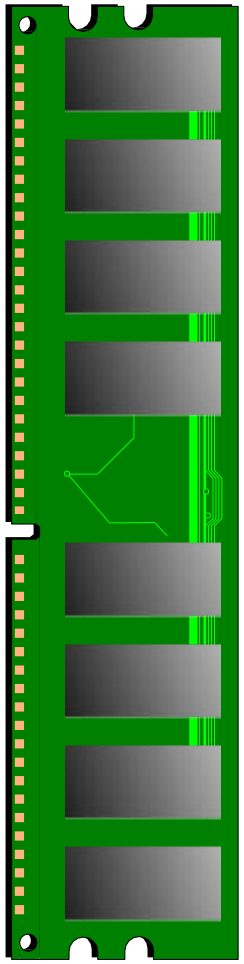
Cache miss

Cache hit



Request

Response



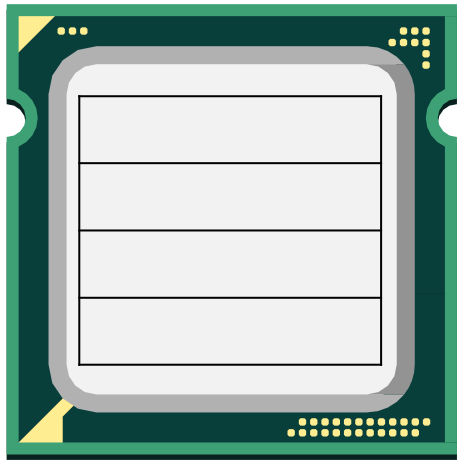
No DRAM access, much faster

Shared Memory

ATTACKER

flush

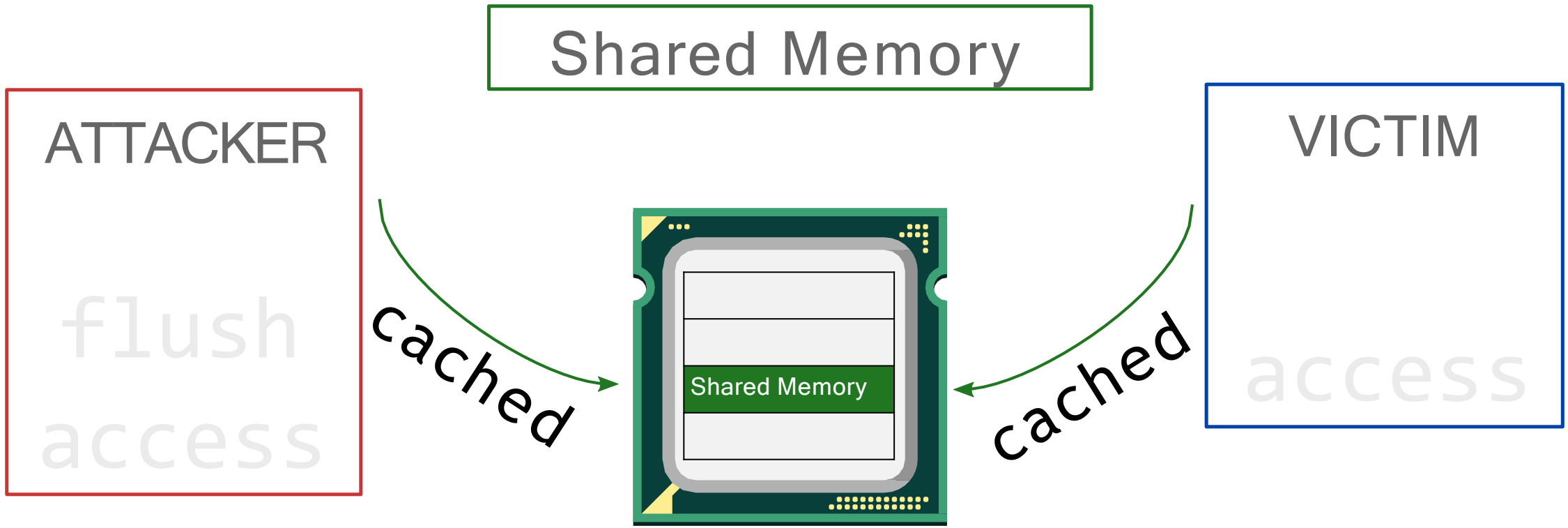
access



VICTIM

access

CASPER


Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

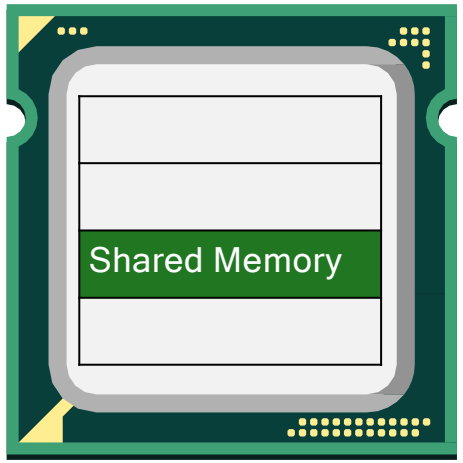
CASPER

Shared Memory

ATTACKER

flush

access




VICTIM

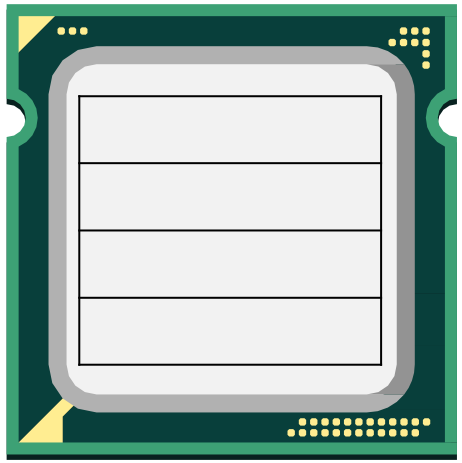
access

Shared Memory

ATTACKER

flush

access



VICTIM

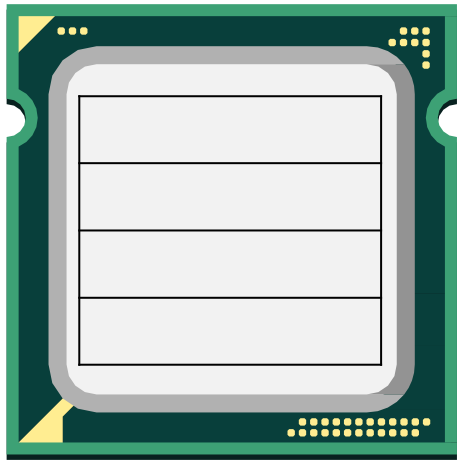
access

Shared Memory

ATTACKER

flush

access



VICTIM

access

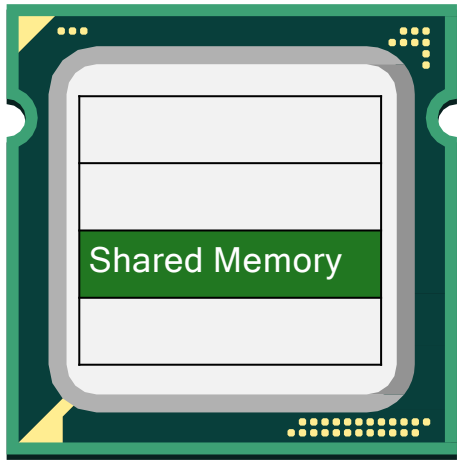
CASPER

Shared Memory

ATTACKER

flush

access




VICTIM

access

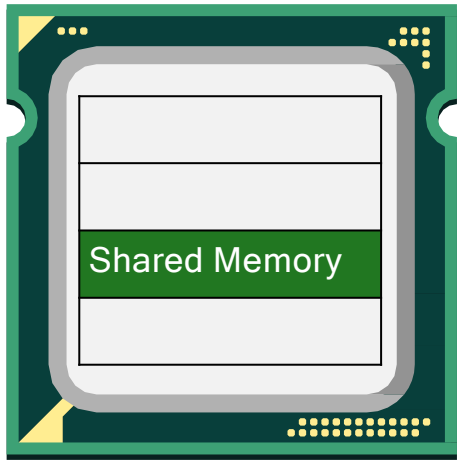
CASPER

Shared Memory

ATTACKER

flush

access




VICTIM

access

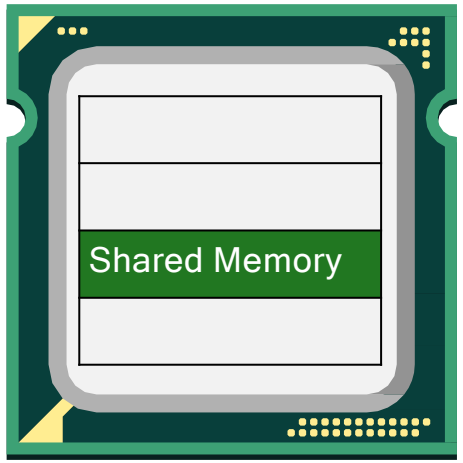
CASPER

Shared Memory

ATTACKER

flush

access

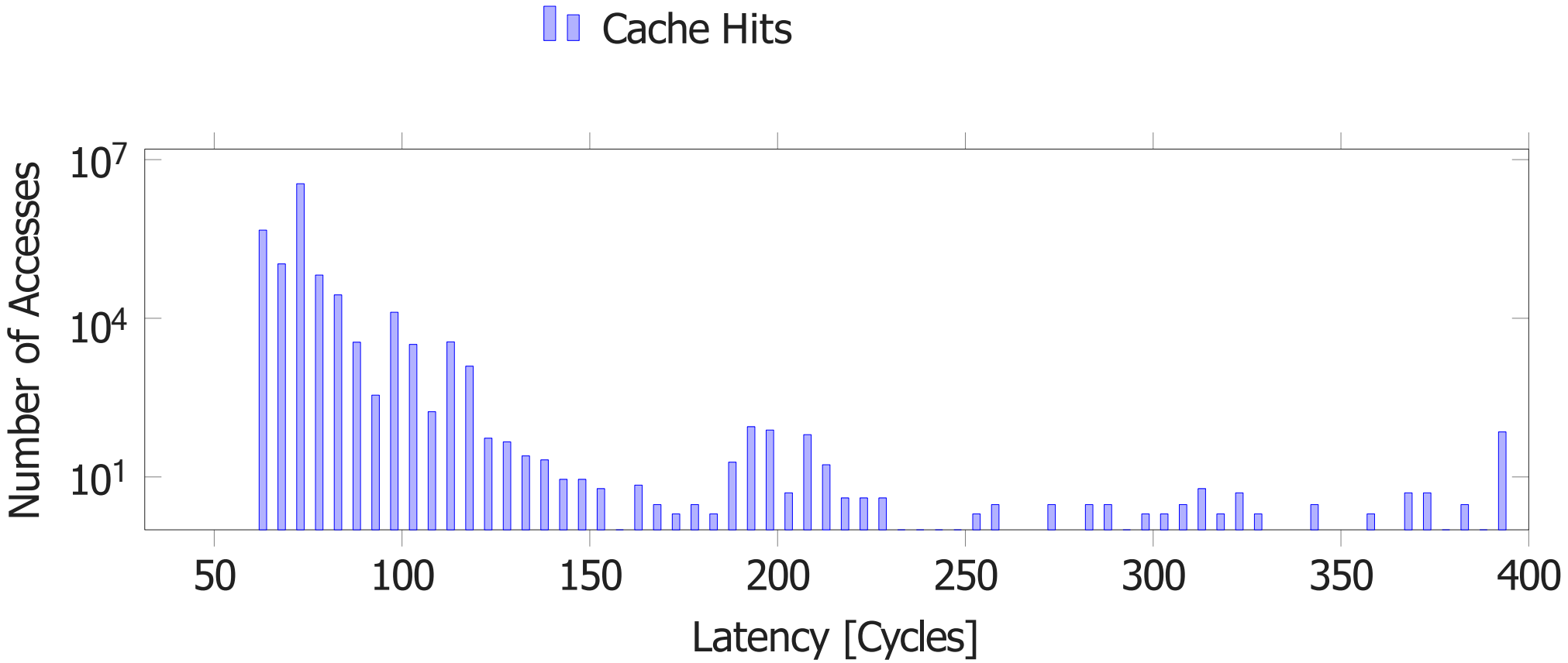



VICTIM

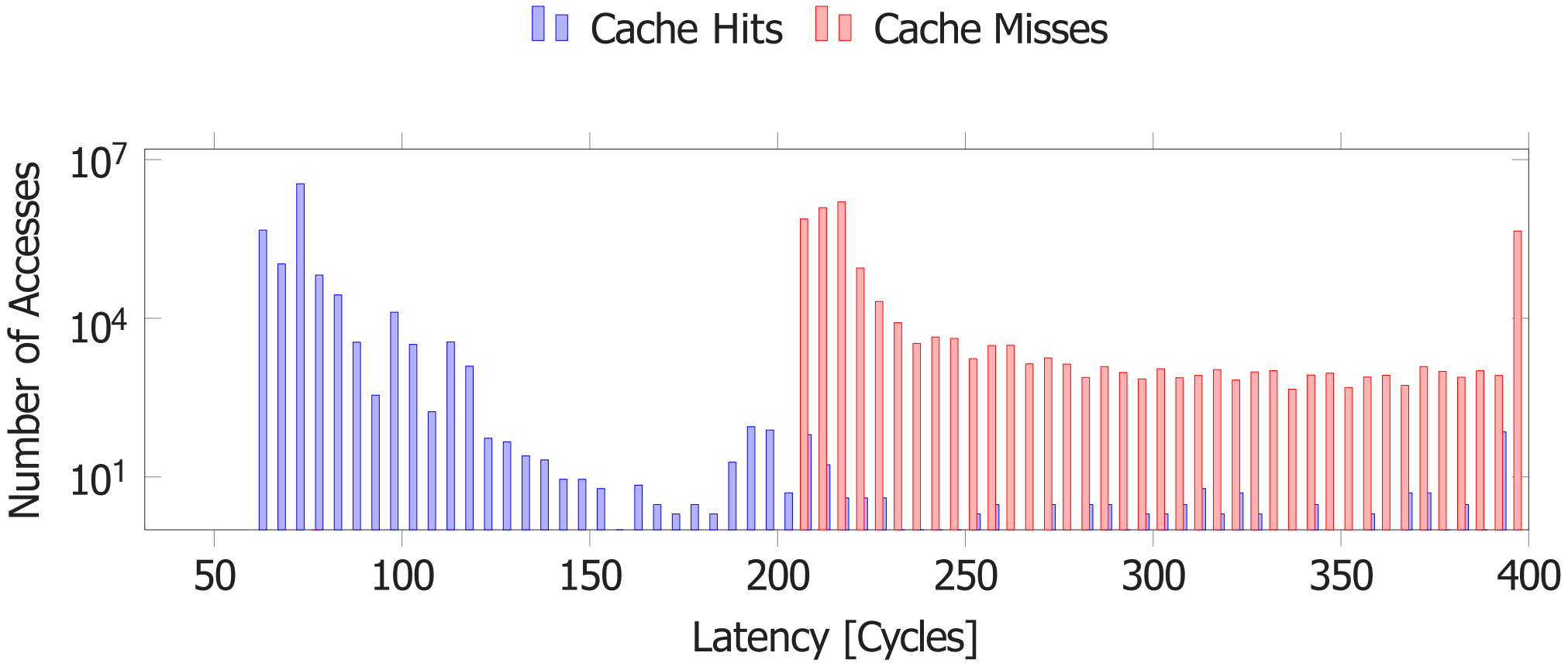
access

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?

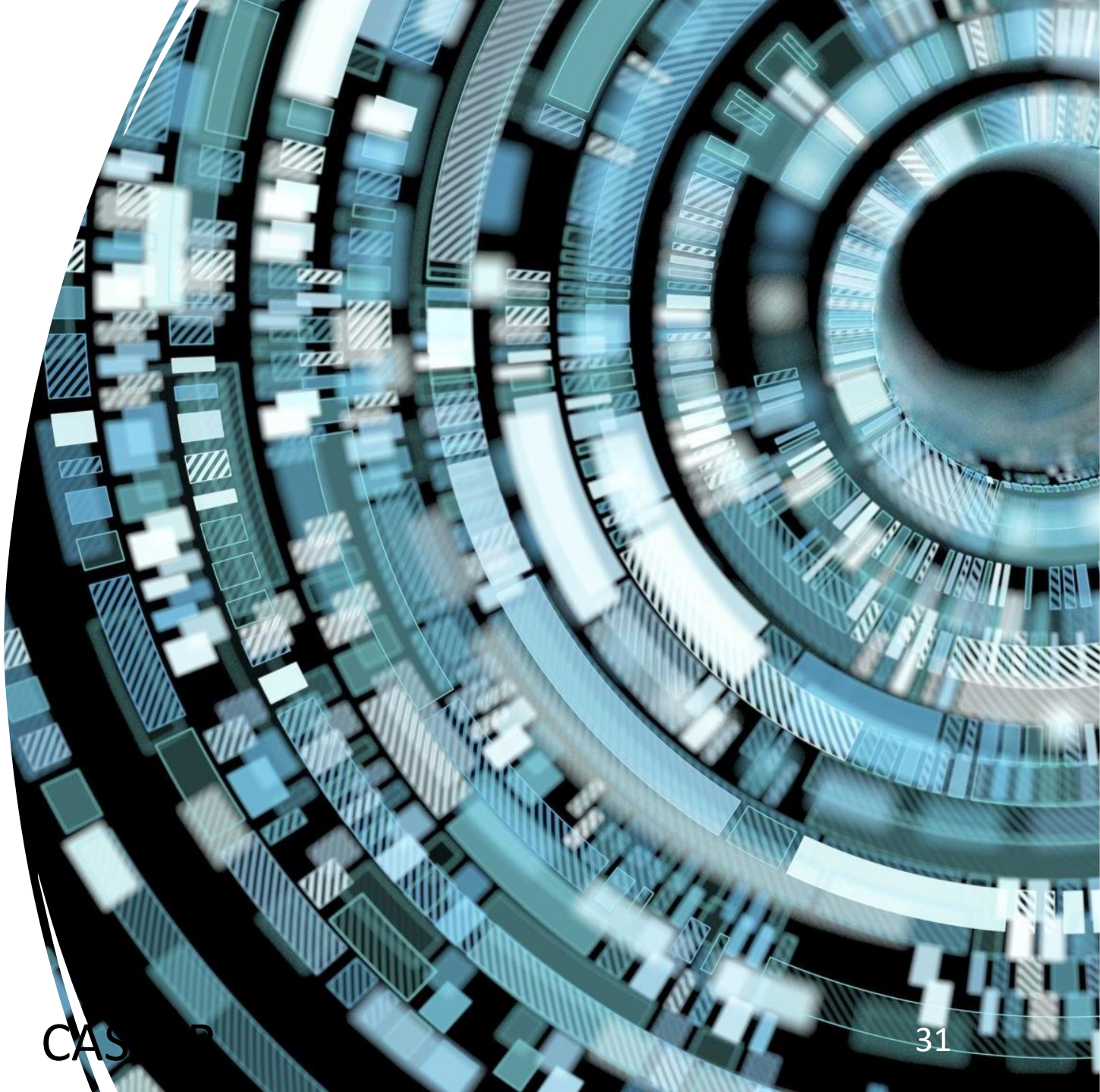
Why share memory?

# Let’s understand it

# OS

- What is it?
- OS: The world of Maya

Virtual world: Virtual memory, Process (virtual CPUs)



# Operating System and Architecture: Bandish 101

Case 1: Programmer wants to run 100 things

CPU says I am alone ☹

OS says I can create an illusion of multiple CPUs ☺

Case 2: Programmer wants 100s of GBs of data

Memory says I am just 10 GB ☹

OS says, never mind, I can create an illusion of TBs ☺

# Operating System and Architecture: Bandish 101

Case 5: OS needs clflush, why?

User needs clflush, why?

CPU says sure, why not? ☺

# From a program to a process

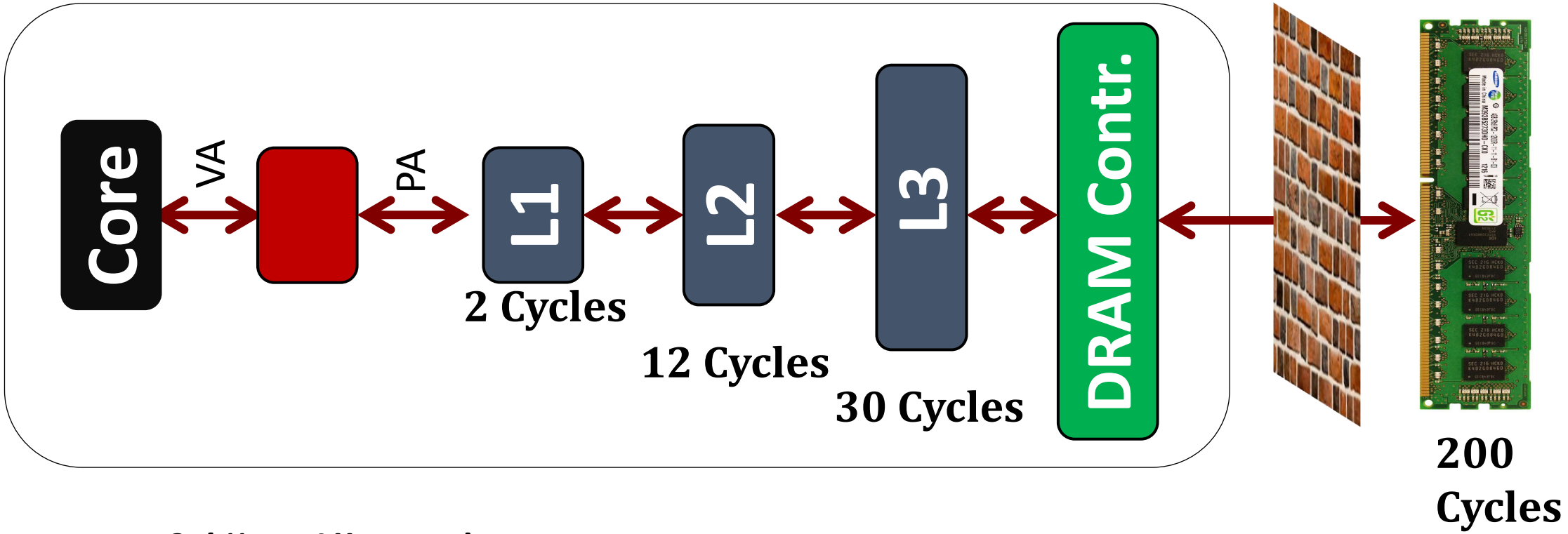
Process: A program that is alive and not-dead (running, waiting ..) ☺

OS creates, manages, schedules them

Allocates memory and initialize CPU state (PC) to kickstart

OS can run multiple processes concurrently even on a single core

# Virtual World




Printf (“%d”, &a);

Virtual address

# Virtual Memory

App. 1

**Virtual address space**

Printf (“%d”, &a);

App. 2

**Virtual address space**

Printf (“%d”, &a);

# A bit of detour towards OS: Paging

Memory space divided into pages.
Typical page size: 4KB
Huge page: 2MB, 1GB pages

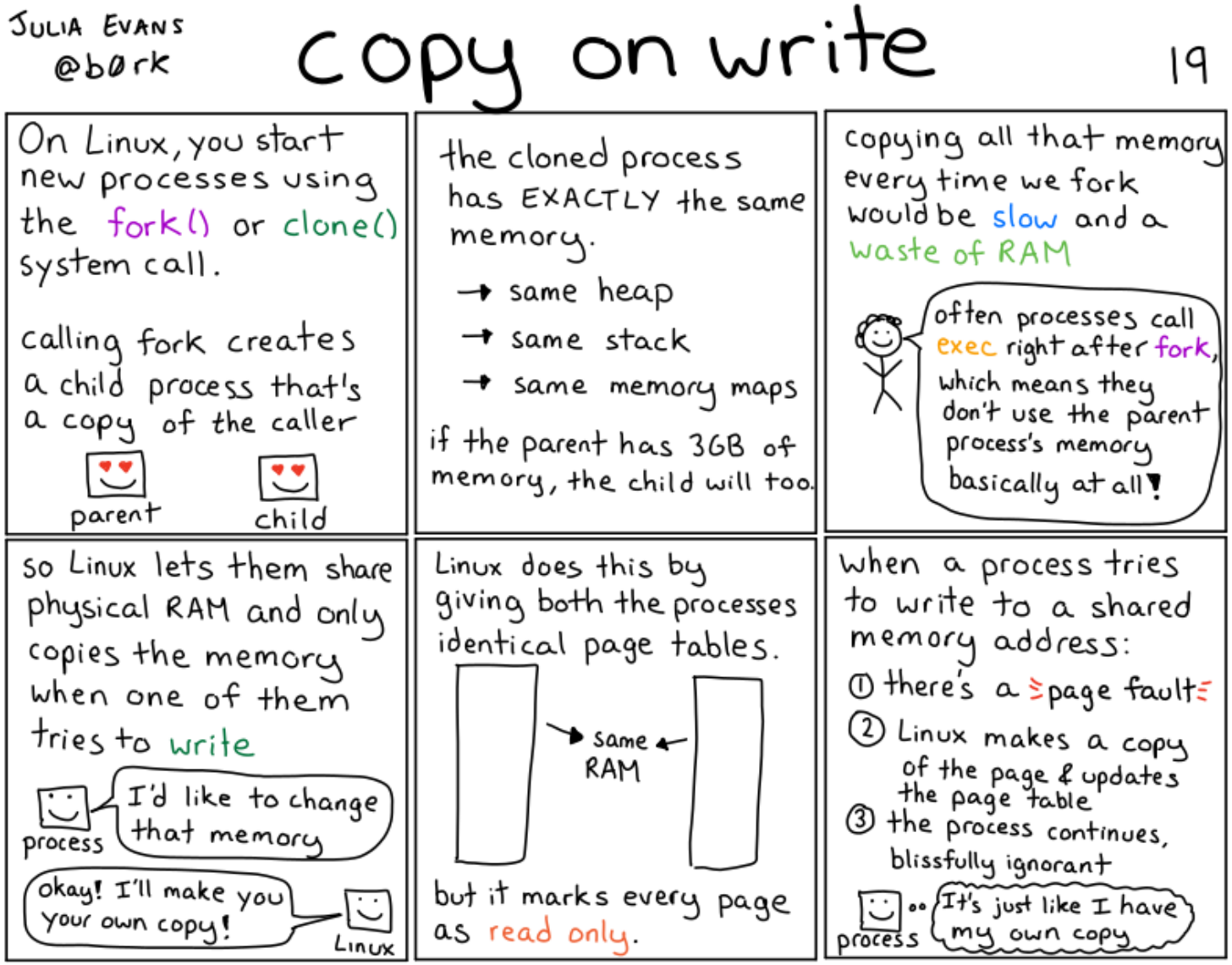
A software table that stores the paging information: Page table

An entry in the page table is known as page-table entry (PTE)

# Per process page table (stored in memory)

Multiple levels to save space

| Virtual page | Physical page |
| --- | --- |
| | |
| | |
| | |
| | |
| | |
| | |


JULIA EVANS
@b0rk
copy on write
19
On Linux, you start new processes using the fork() or clone() system call.
calling fork creates a child process that's a copy of the caller
parent
child
the cloned process has EXACTLY the same memory.
→ same heap
→ same stack
→ same memory maps
if the parent has 3GB of memory, the child will too.
copying all that memory every time we fork would be slow and a waste of RAM
often processes call exec right after fork, which means they don't use the parent process's memory basically at all!
so Linux lets them share physical RAM and only copies the memory when one of them tries to write
process
I'd like to change that memory
okay! I'll make you your own copy!
Linux
Linux does this by giving both the processes identical page tables.
same RAM
but it marks every page as read only.
when a process tries to write to a shared memory address:
① there's a page fault
② Linux makes a copy of the page & updates the page table
③ the process continues, blissfully ignorant
process
It's just like I have my own copy

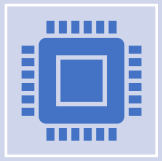
# Think about any shared library ☺ and COW

- Shared library is shared to multiple processes.

- So, all can access it with their virtual addresses.

# What is the use clflush

When OS deallocates a pe from DRAM, it clflushes the corresponding cache lines from the cache hierarchy.



In user space, clflush is used to handle memory inconsistencies such as in memorymapped I/O

# Please refer

Course website for the paper on F+R and more information