CASPER

CS773: Computer Architecture for Performance and Security

Lecture 3: Timing Channel Attacks

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

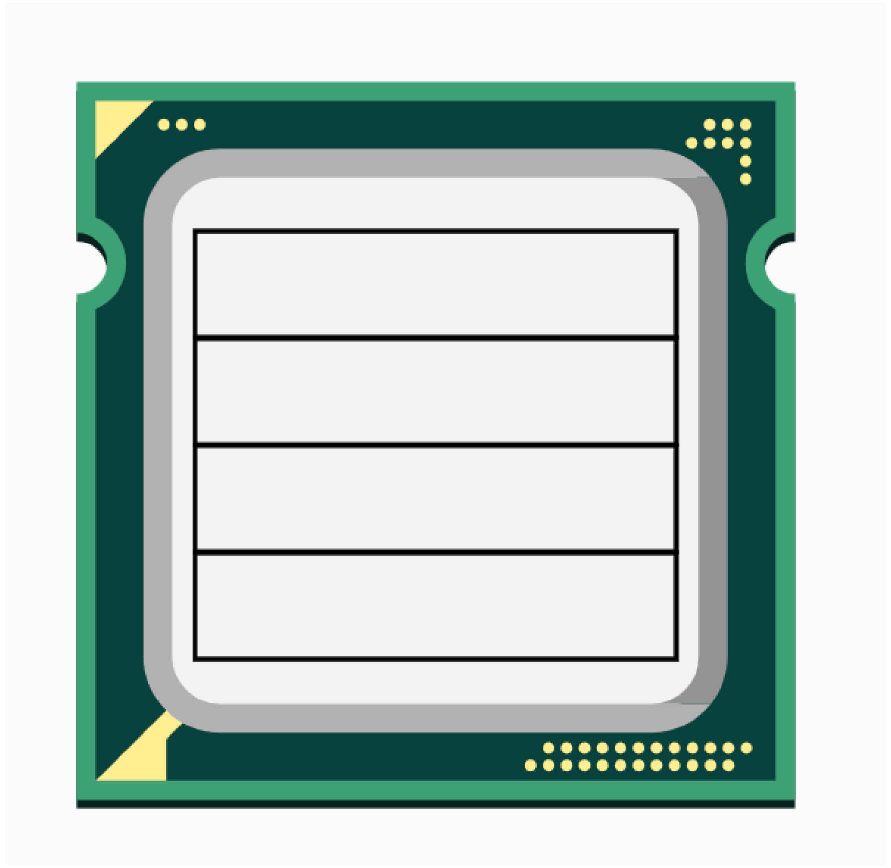
Logistics

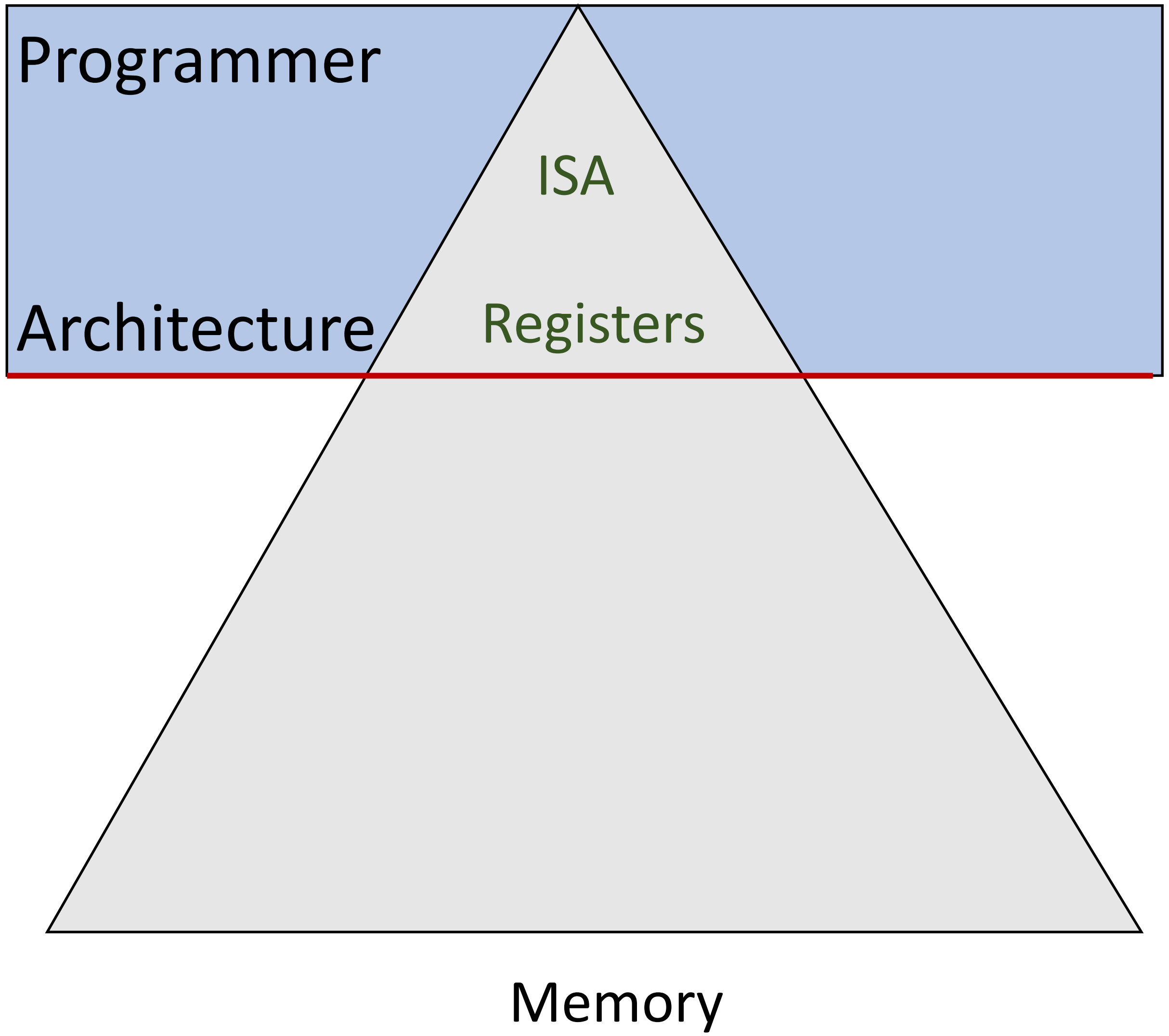
Paper review/presentation from January 31.

We will float a link soon.

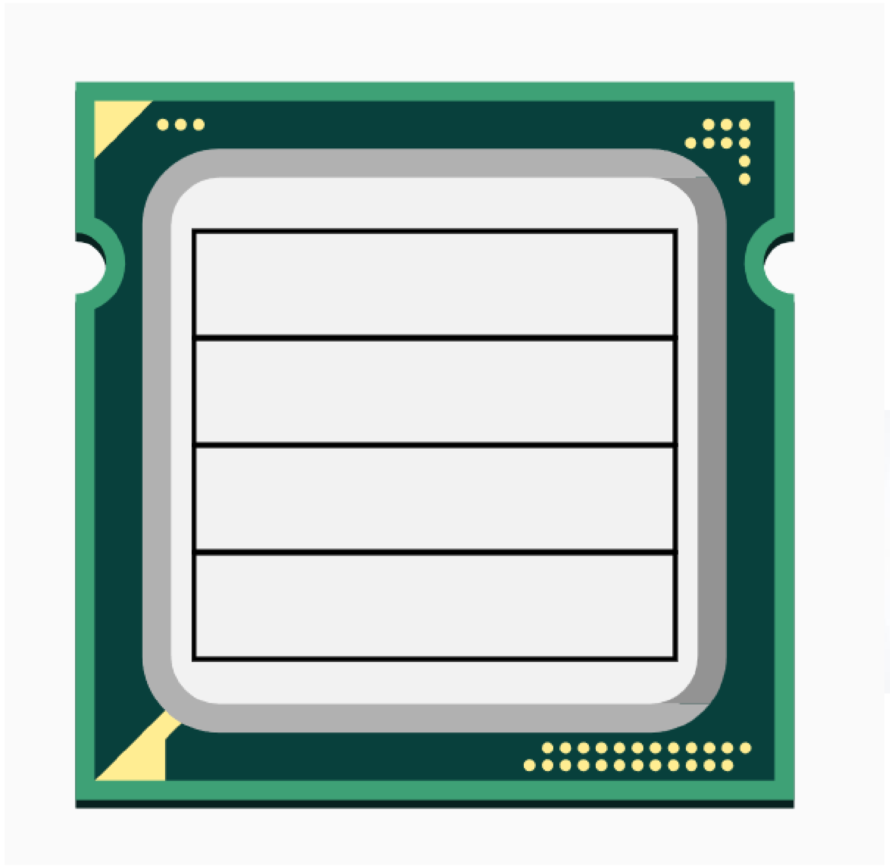
We will float papers of interest by the end of this week.

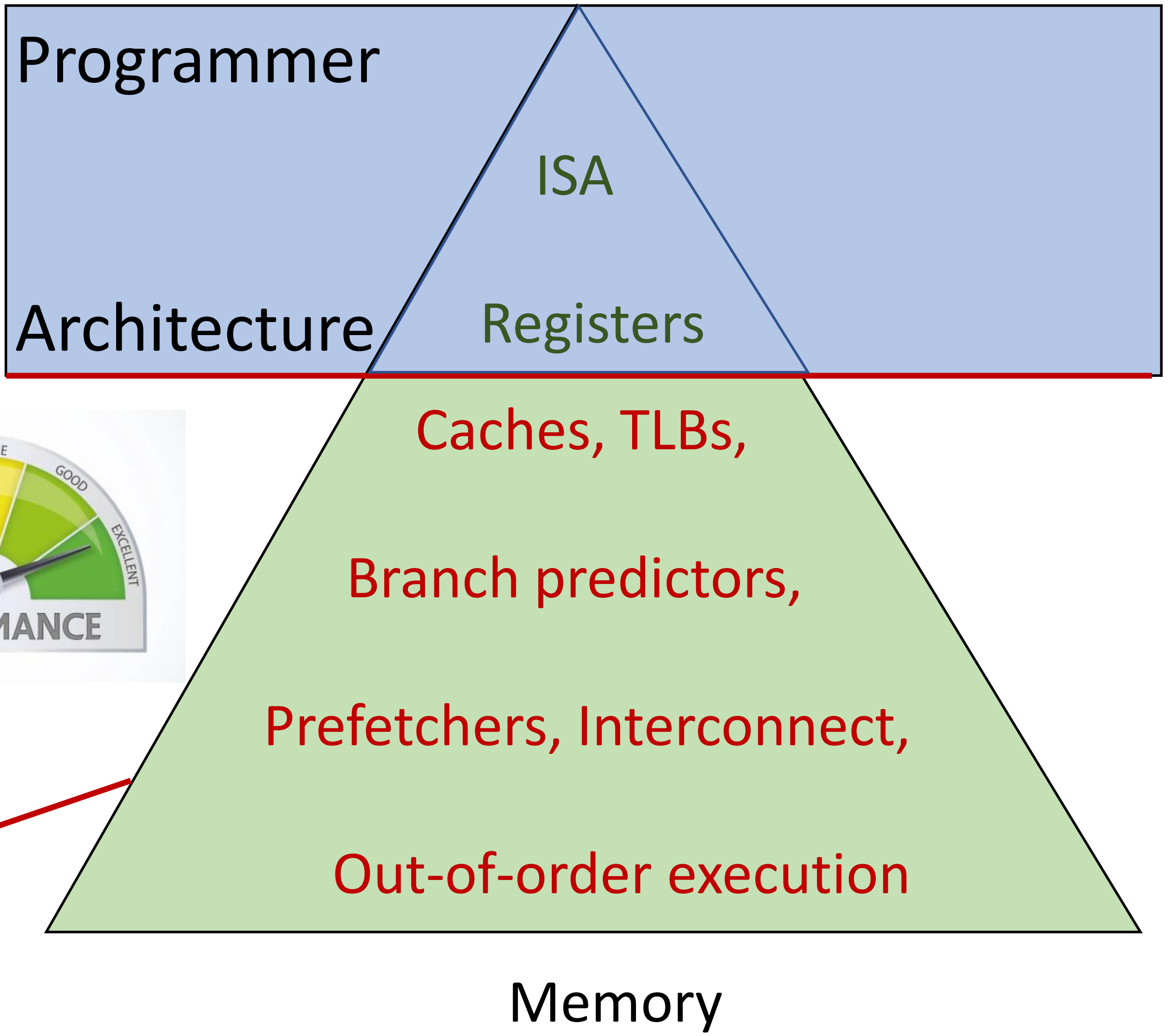
Architecture:101





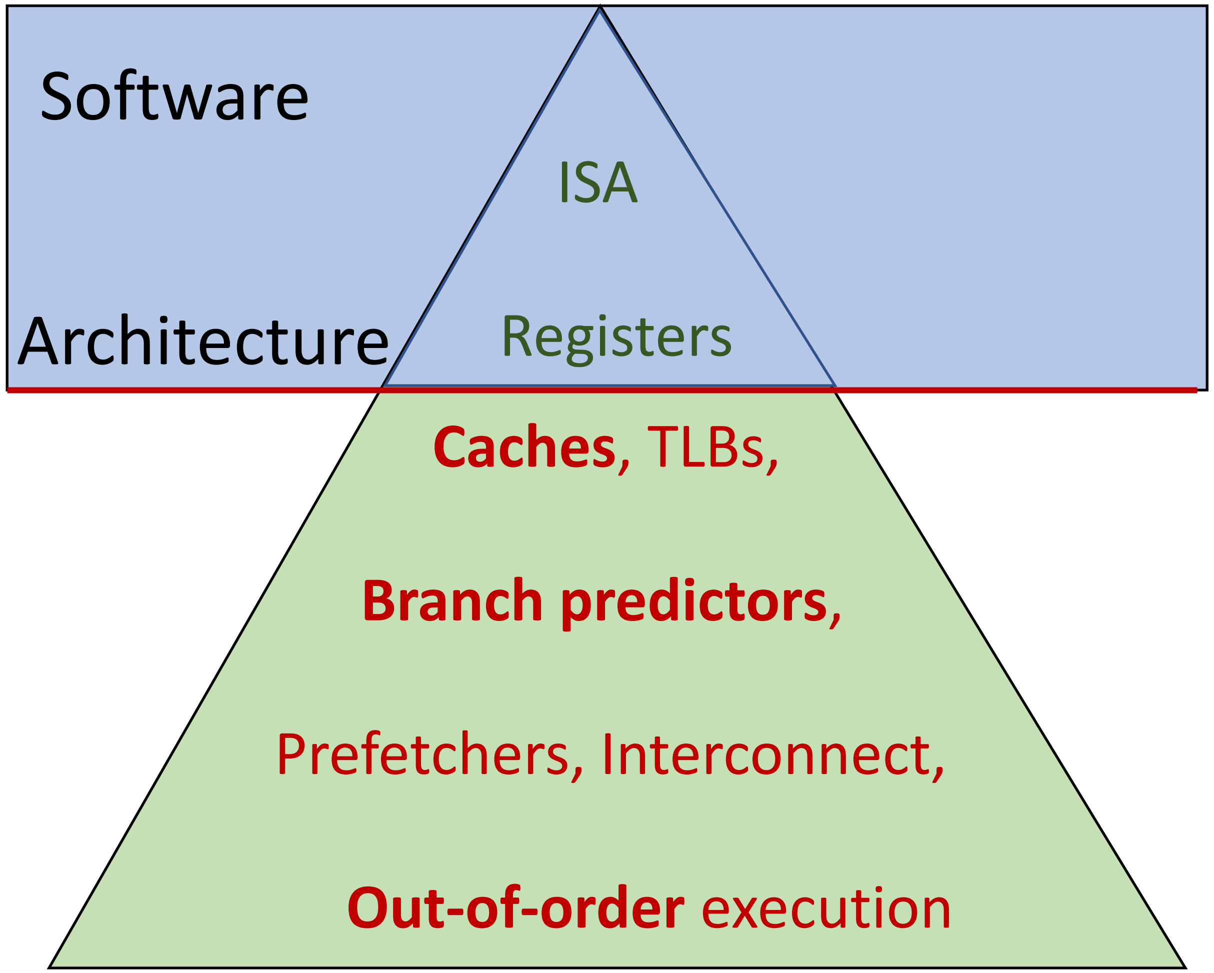
Microarchitecture:101





Not exposed to programmer

From Performance to Security: 10K Feet View



Security: A bit Subtle

Confidentiality	*You do not **see** **(READ)** what you are not supposed to see*
Integrity	*You do not **change** **(WRITE)** what you are not supposed to see*
Availability	*You do not **affect** **(DELAY)** others (un)intentionally*

Brushing-up: Information Leakage

```
$x \leftarrow 1$
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

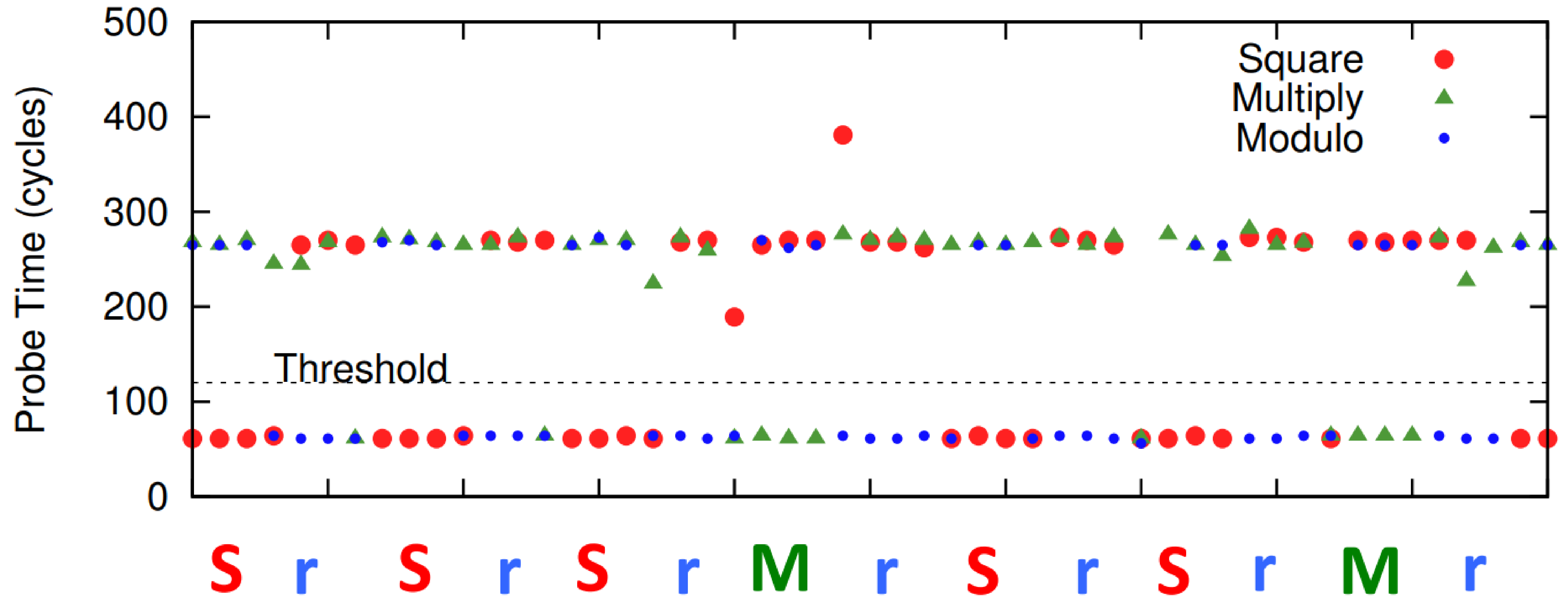
Modular exponentiation, $b^e \bmod n$

Exponent e is used for decryption

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

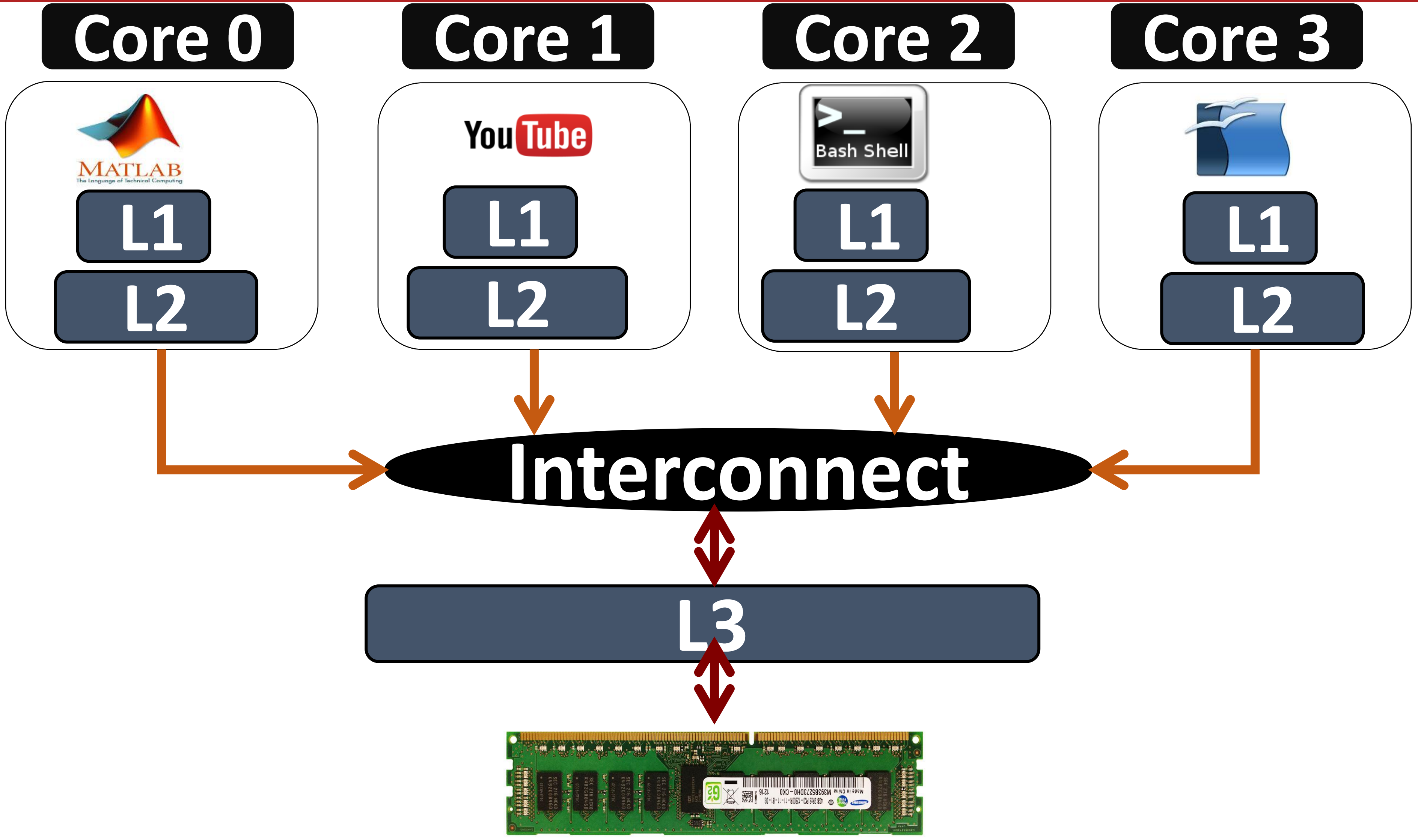
Attacker tries to get the e

Timing Channel



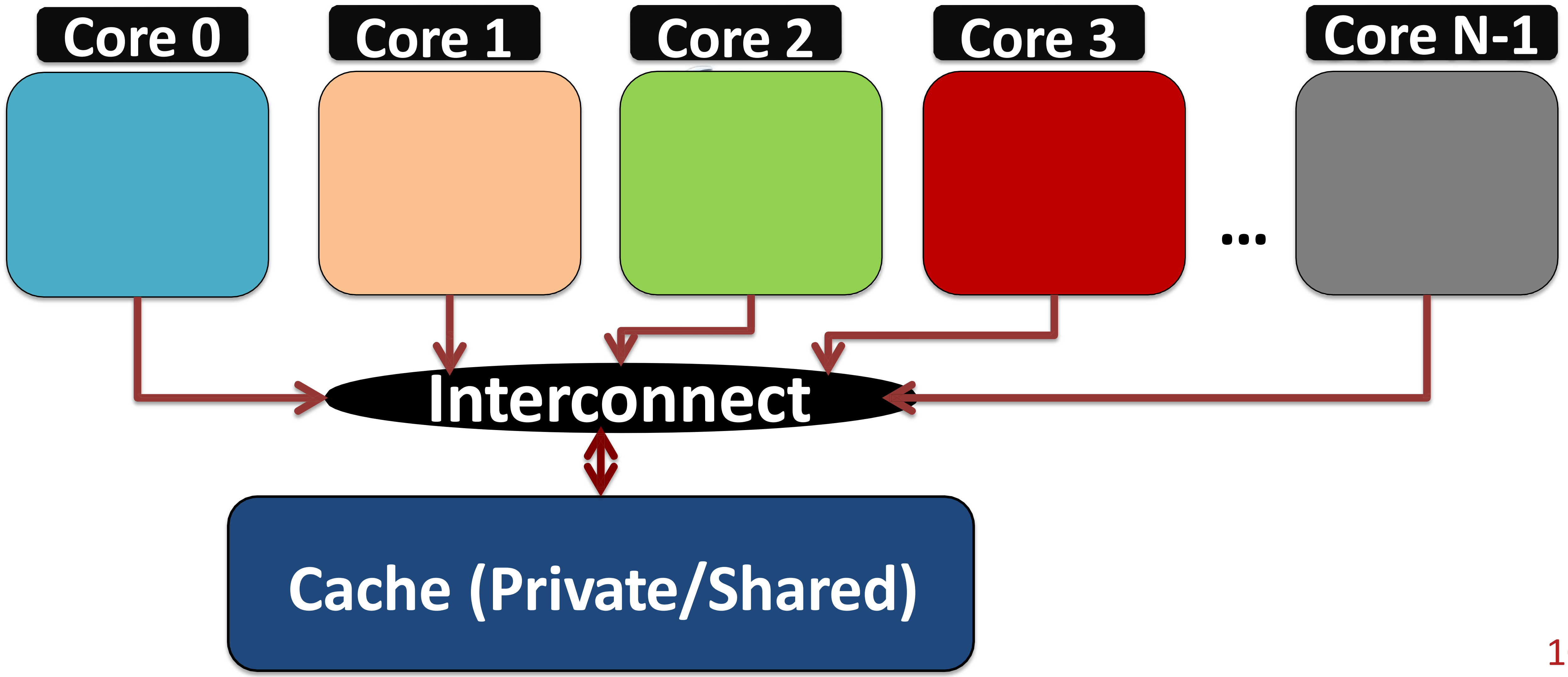
Adpated from flush+reload attack [Usenix Security ‘14]

Multi-core

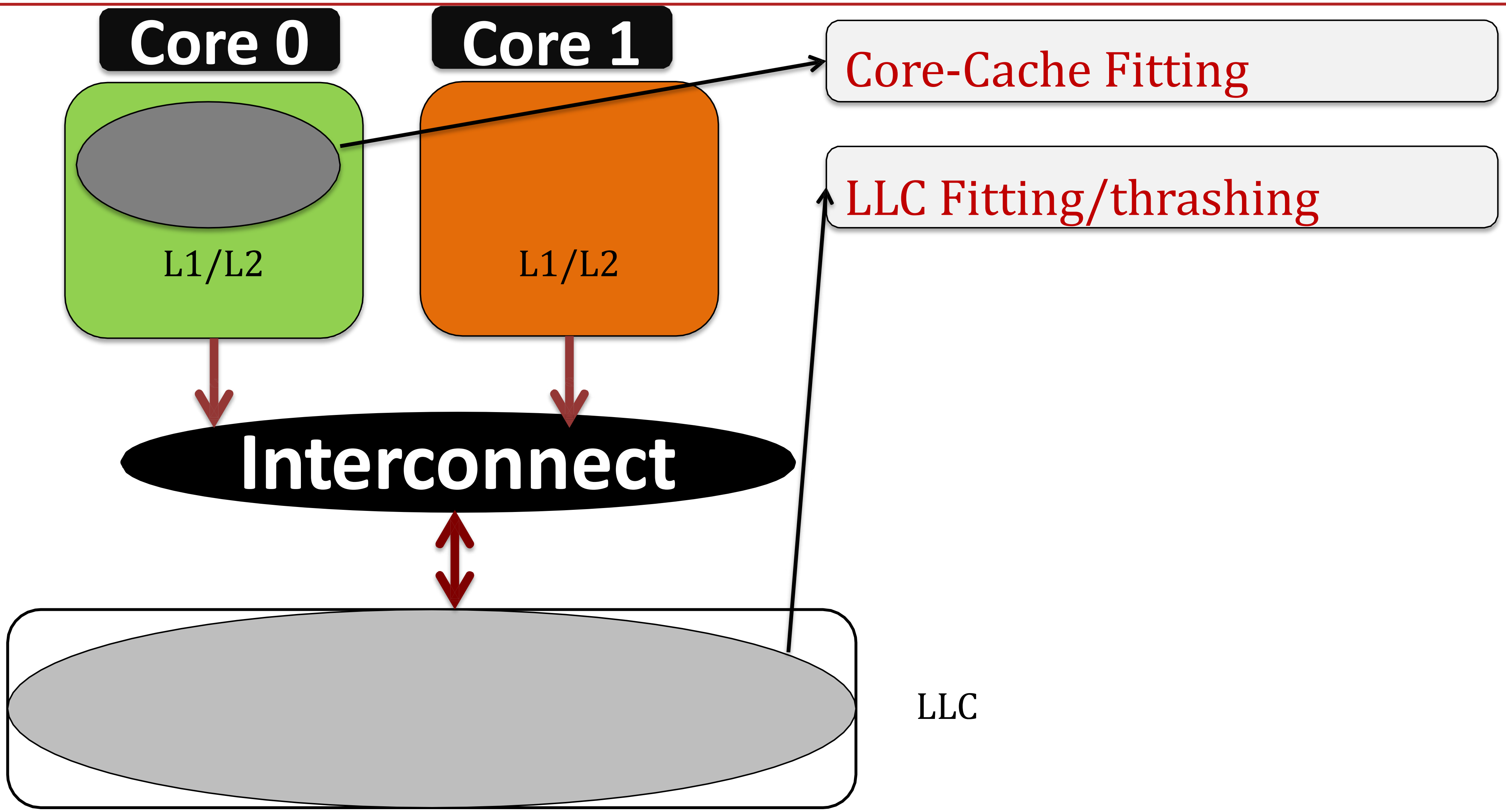


Core 0
Core 1
Core 2
Core 3
MATLAB
YouTube
Bash Shell
L1
L2
L1
L2
L1
L2
L1
L2
Interconnect
L3

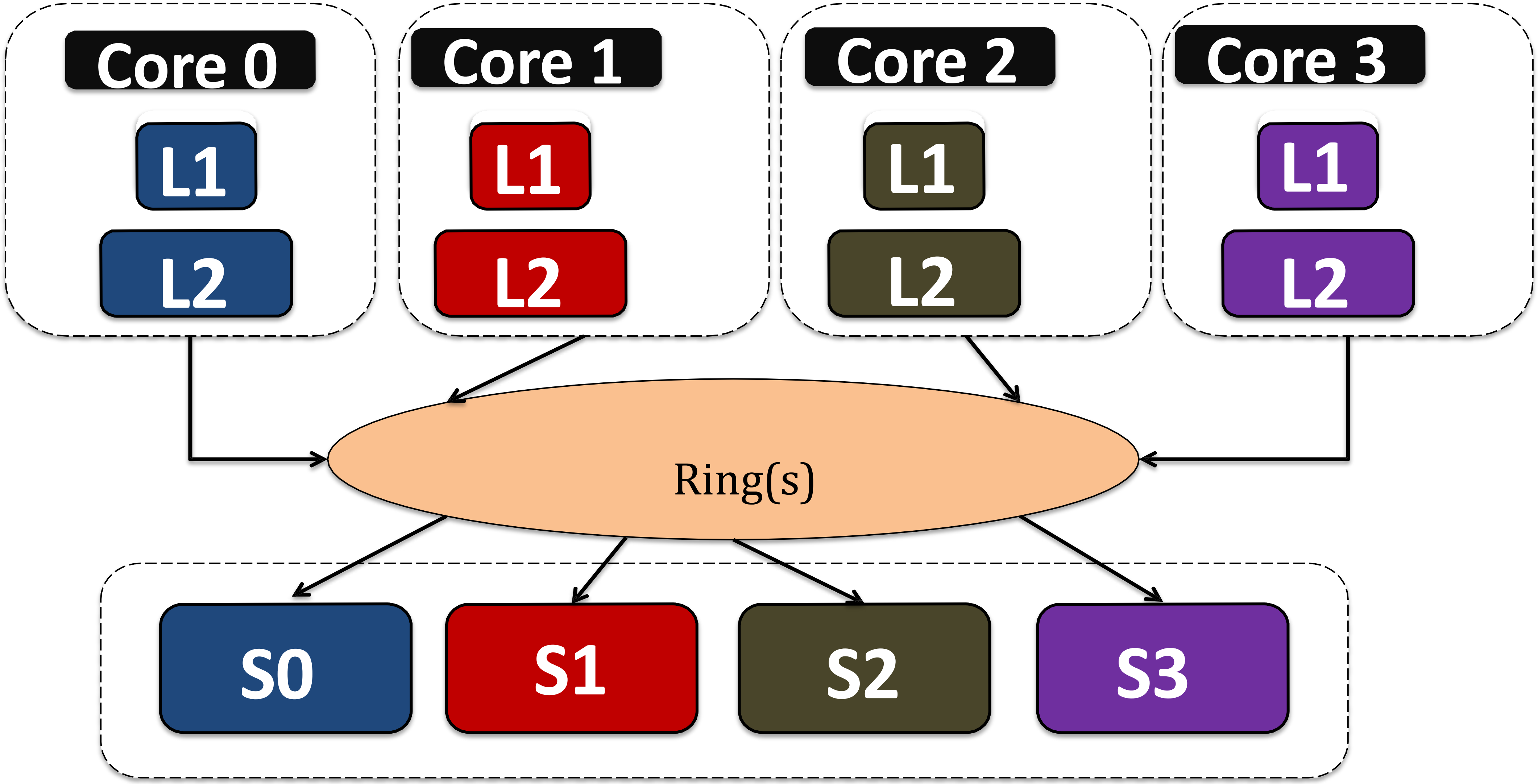
Private vs Shared?



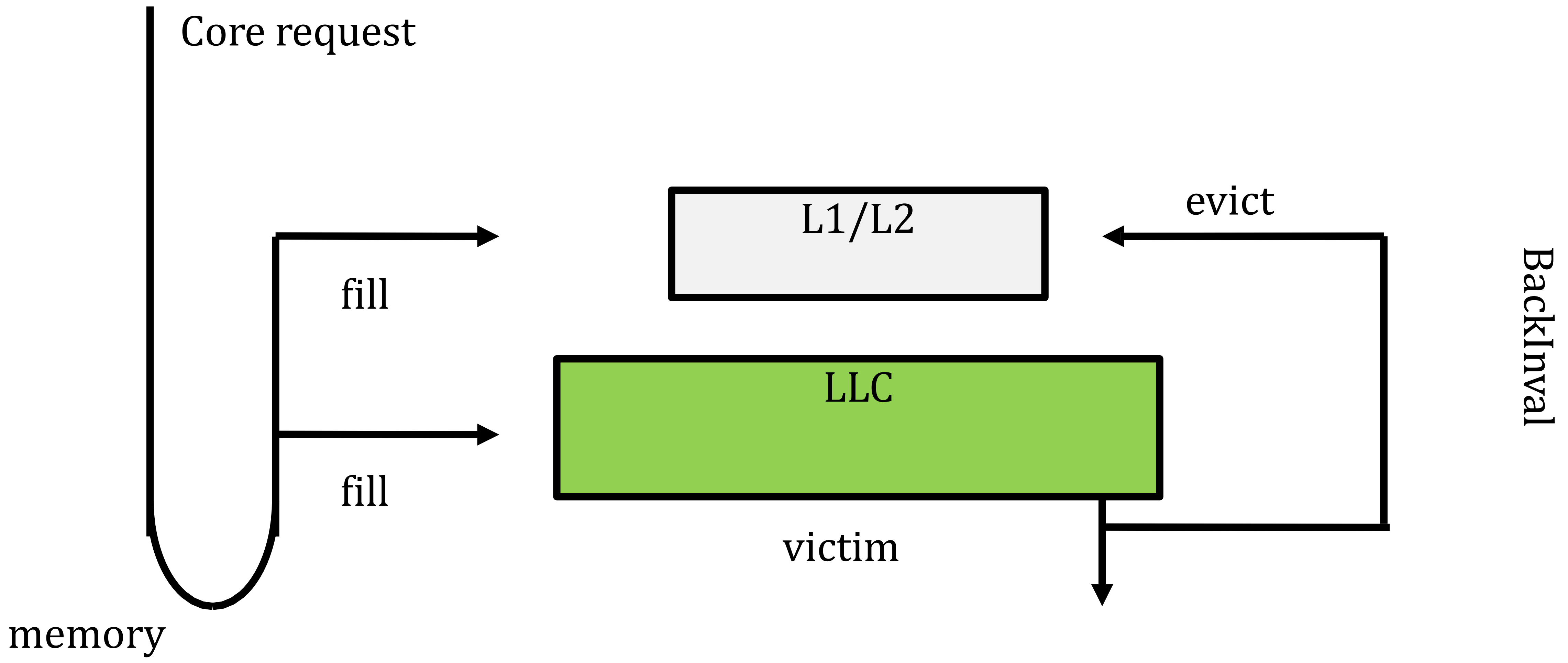
Application Behavior



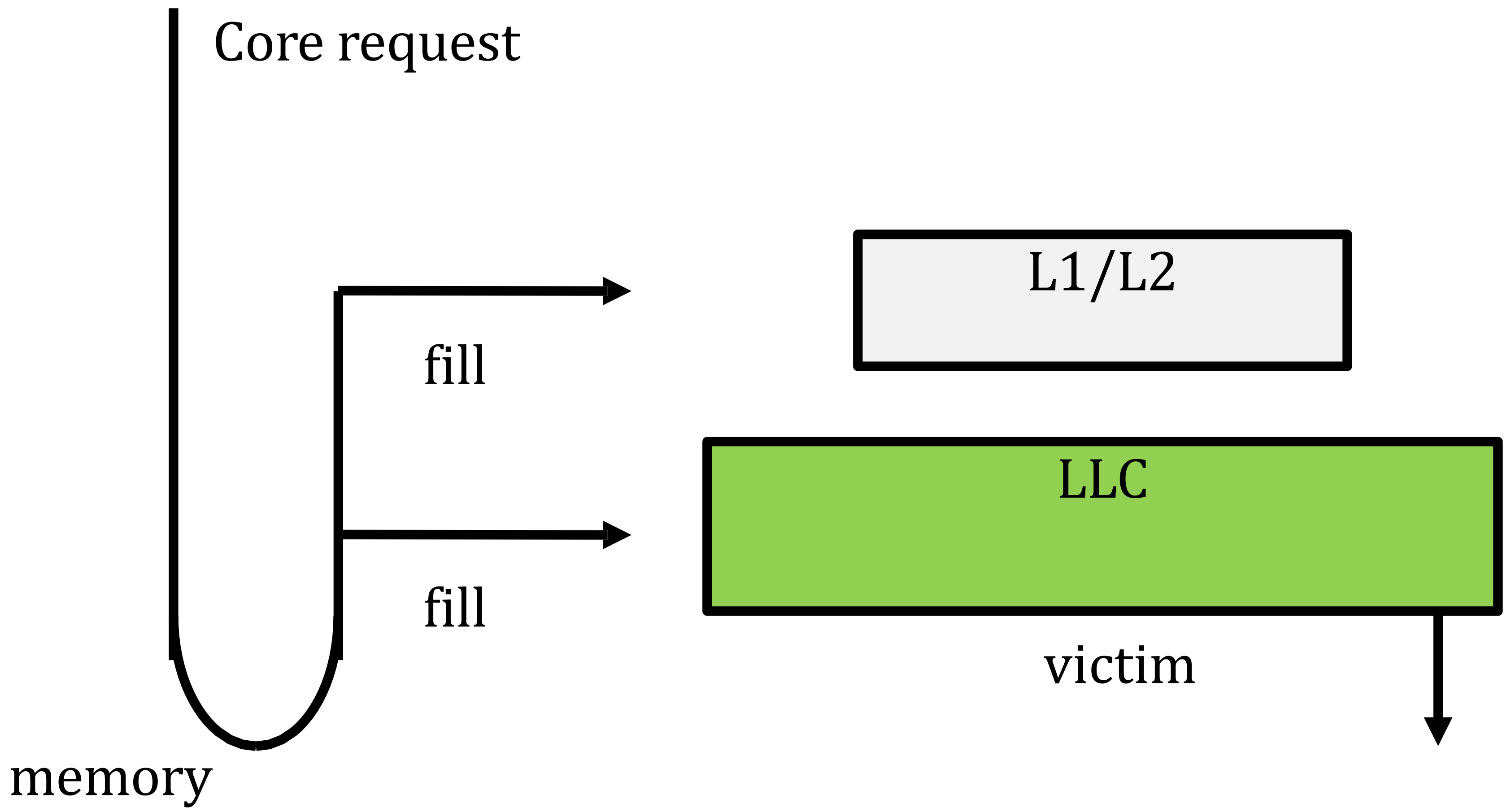
Shared Last-level Cache (LLC): Banked or Sliced



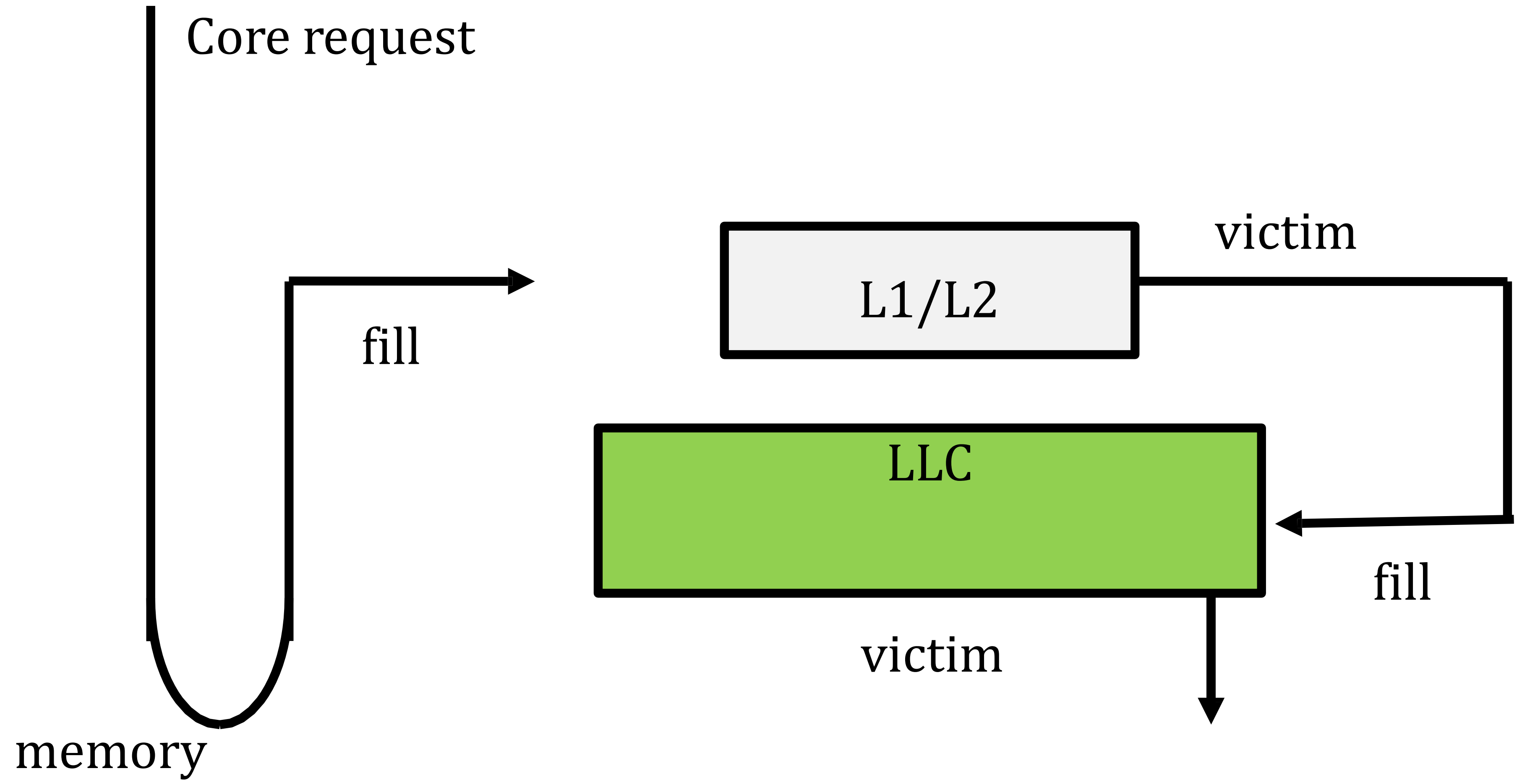
Inclusiveness



Non-inclusive (Commercial machines)



Exclusive



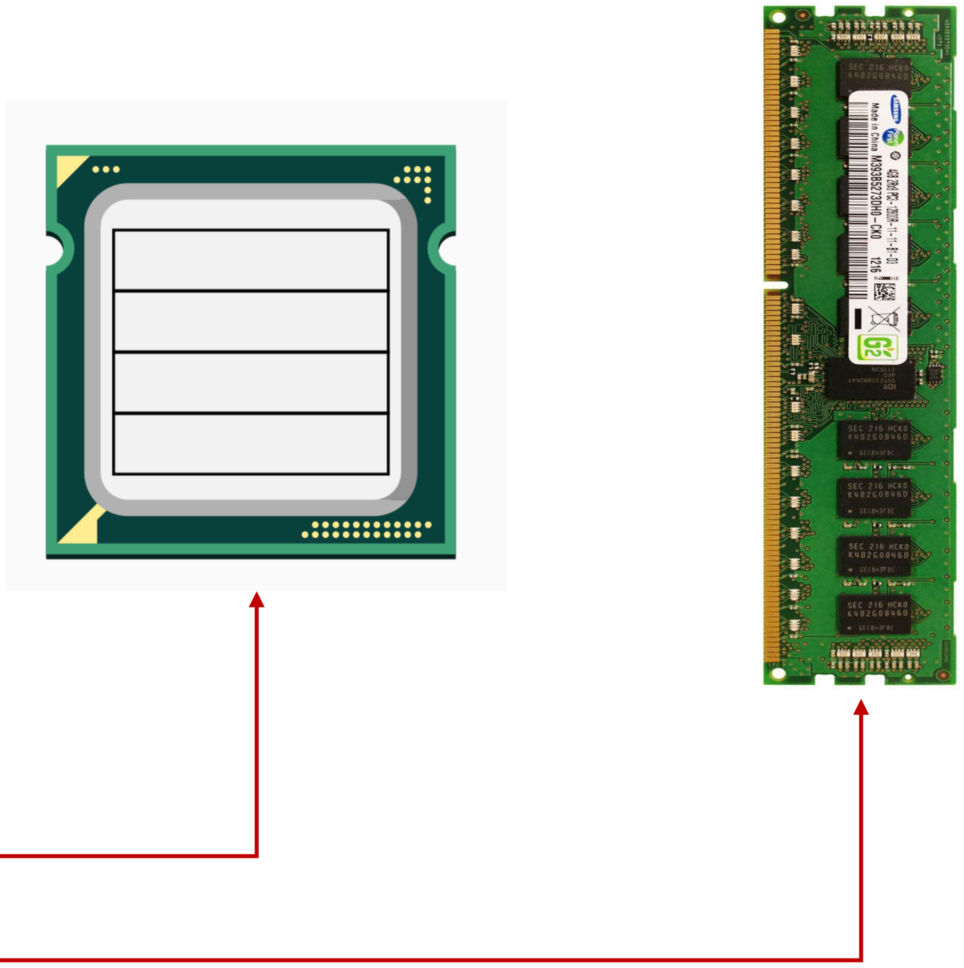
Toy Example: Flush Based Attacks

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

Victim

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

Attacker

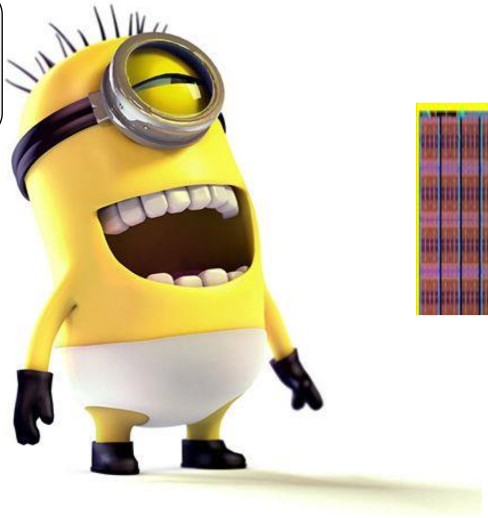


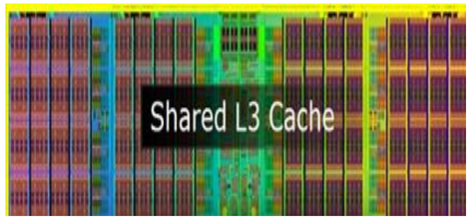
Fast – 1

Slow – 0

Side and Covert Channels

Spy

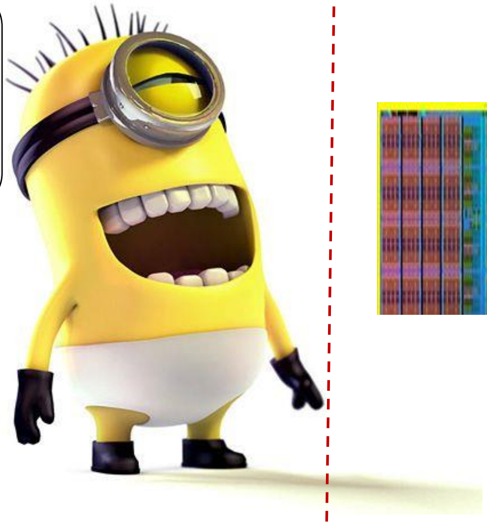


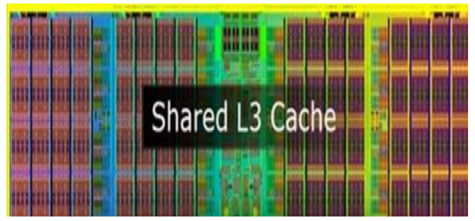


Victim

Side-channel attacks

Let's play



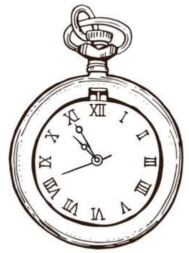


Oh Yes!!

Covert-channel attacks

Shared LLC Attacks

Attacks at the LLC exploit timing channels:
LLC miss > LLC hit



Flush + Reload

Evict + Reload

Prime + Probe

clflush

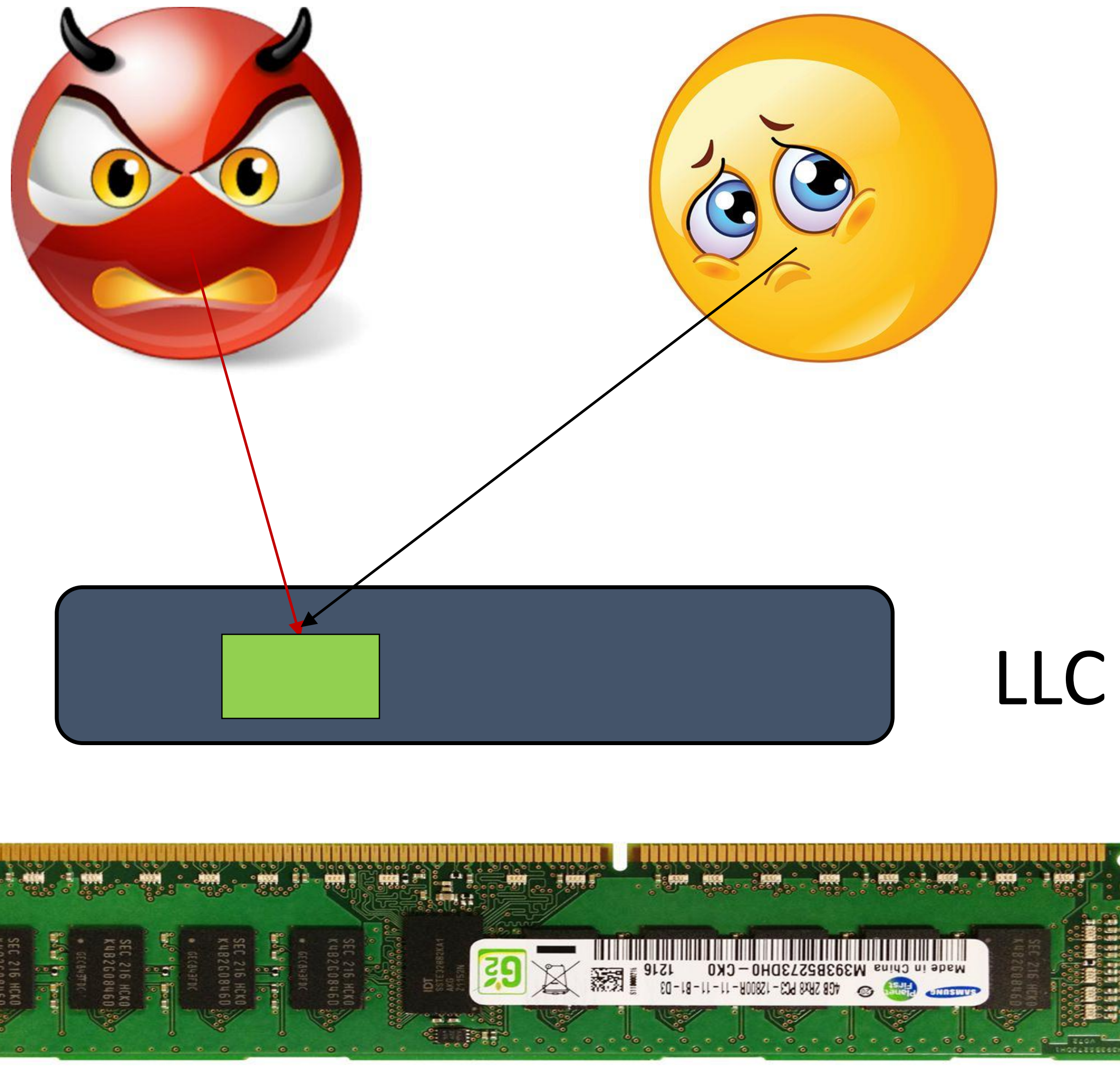
Eviction based attacks

Threat Model

Knowing the victim *has accessed a cache set (line)* can be considered as a *successful* attack

Pause

Flush+Reload Attack (Shared Memory Attack)



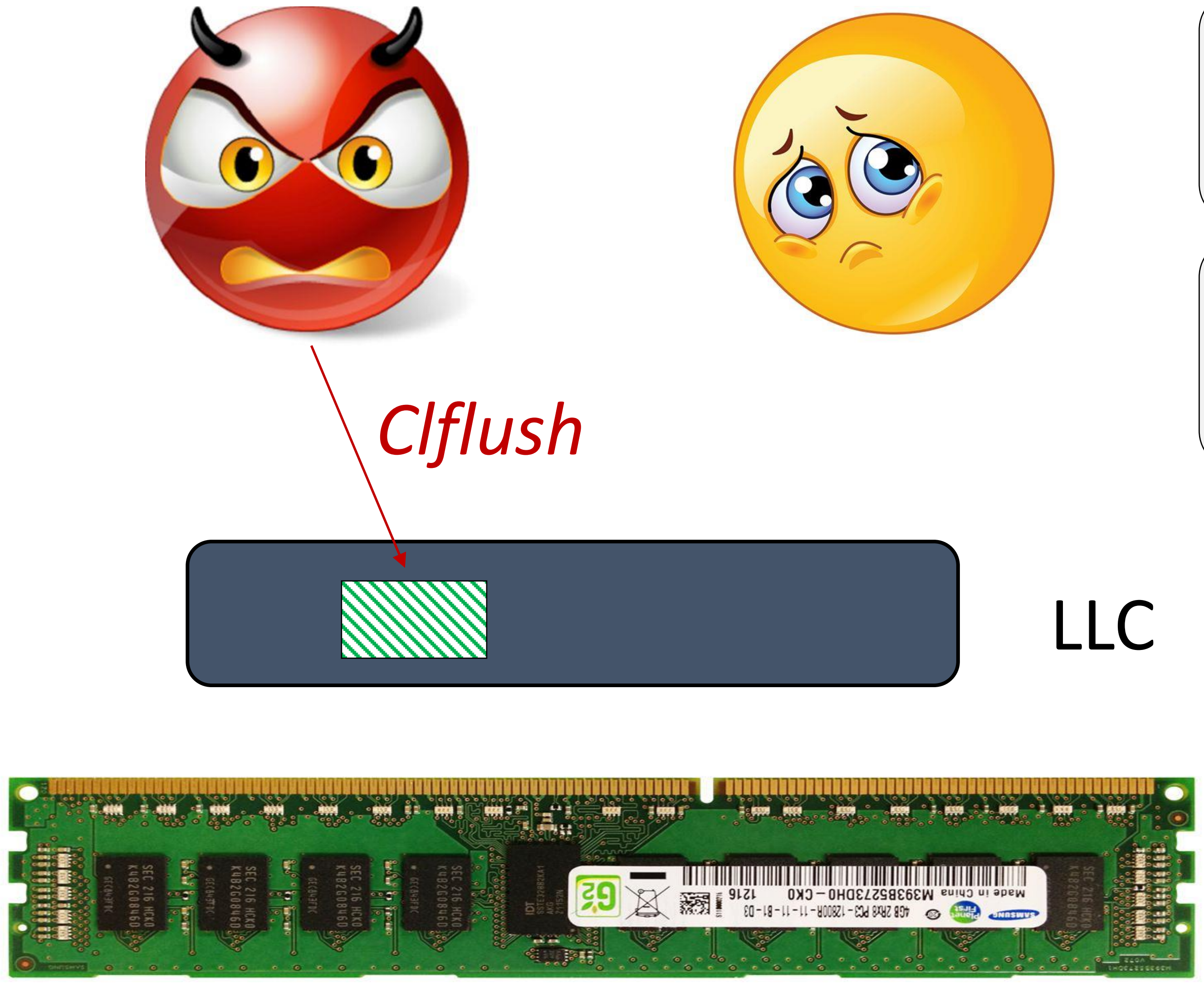
Step 0:Spy *maps* the shared library, shared in the cache

Shared library: Shared Address(es)

Usage of clflush instruction (Flush Cache Line)

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.

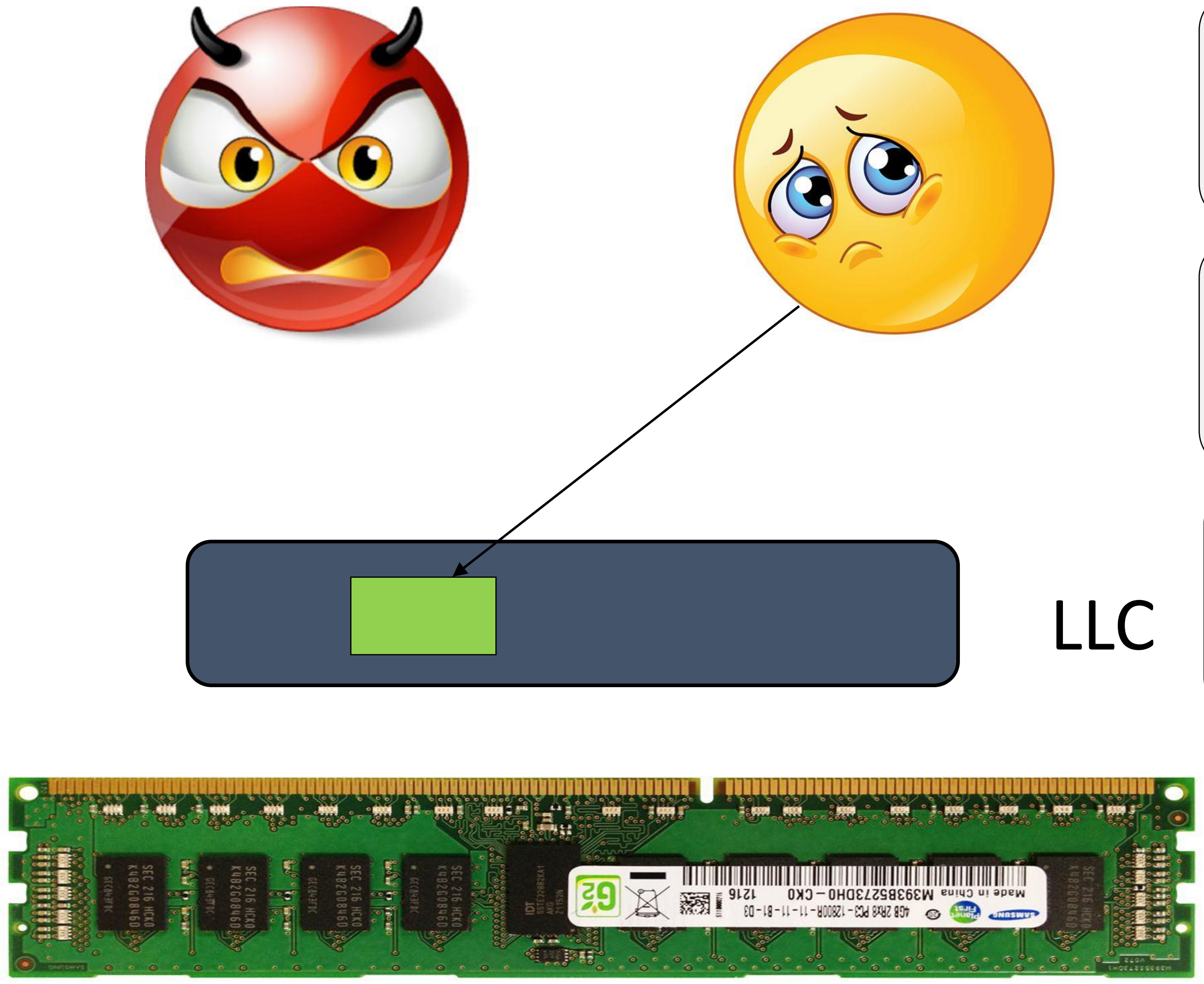
Flush+Reload Attack



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

Step 1:Spy *flushes* the cache block

Flush+Reload Attack

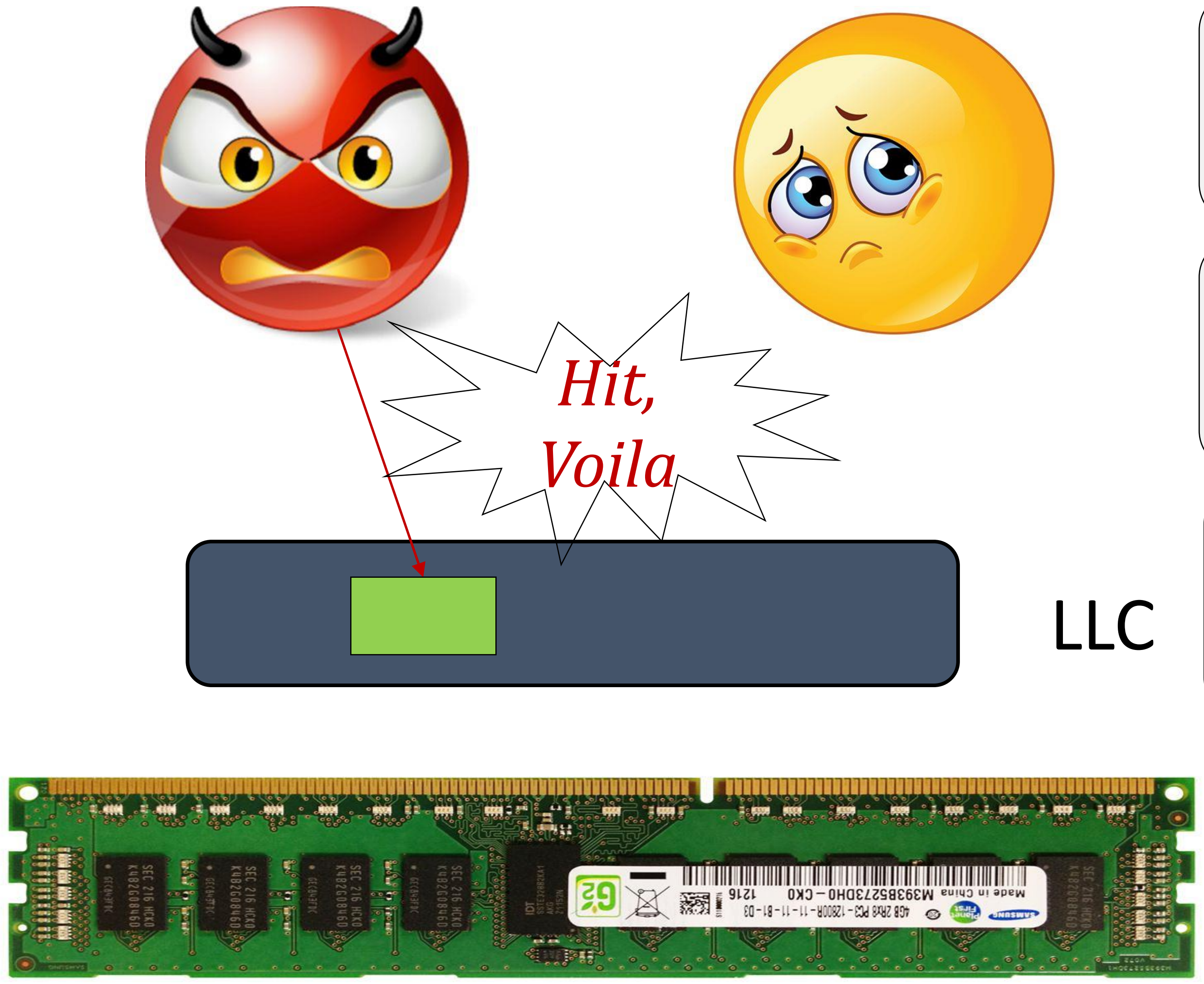


Step 0:Spy *maps* the shared library, shared in the cache

Step 1:Spy *flushes* the cache block

Step 2: Victim *reloads* the cache block

Flush+Reload Attack



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

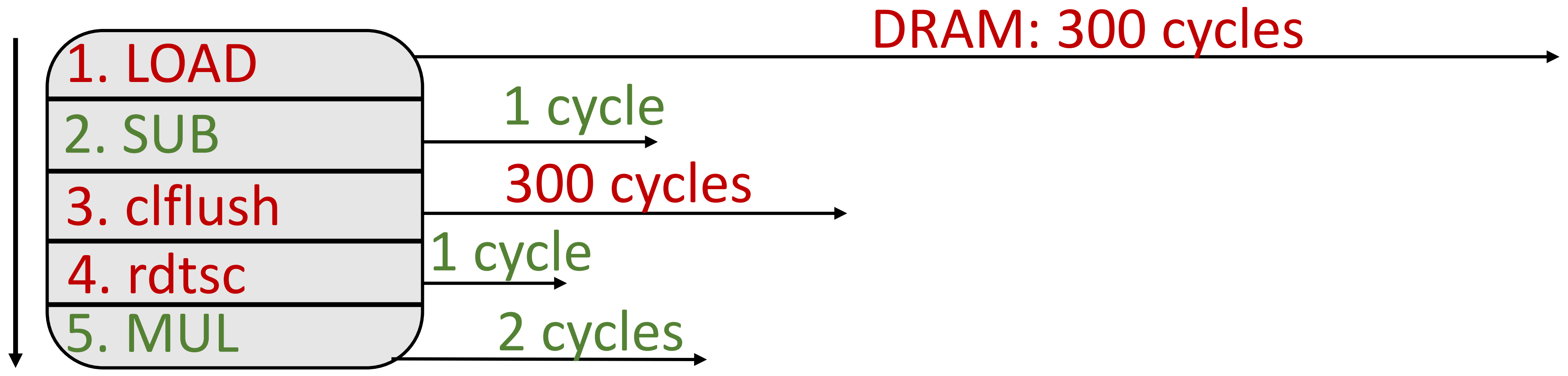
Step 3: Spy *reloads* the cache block (hit/miss)

Hit/Miss; Faster/Slower access

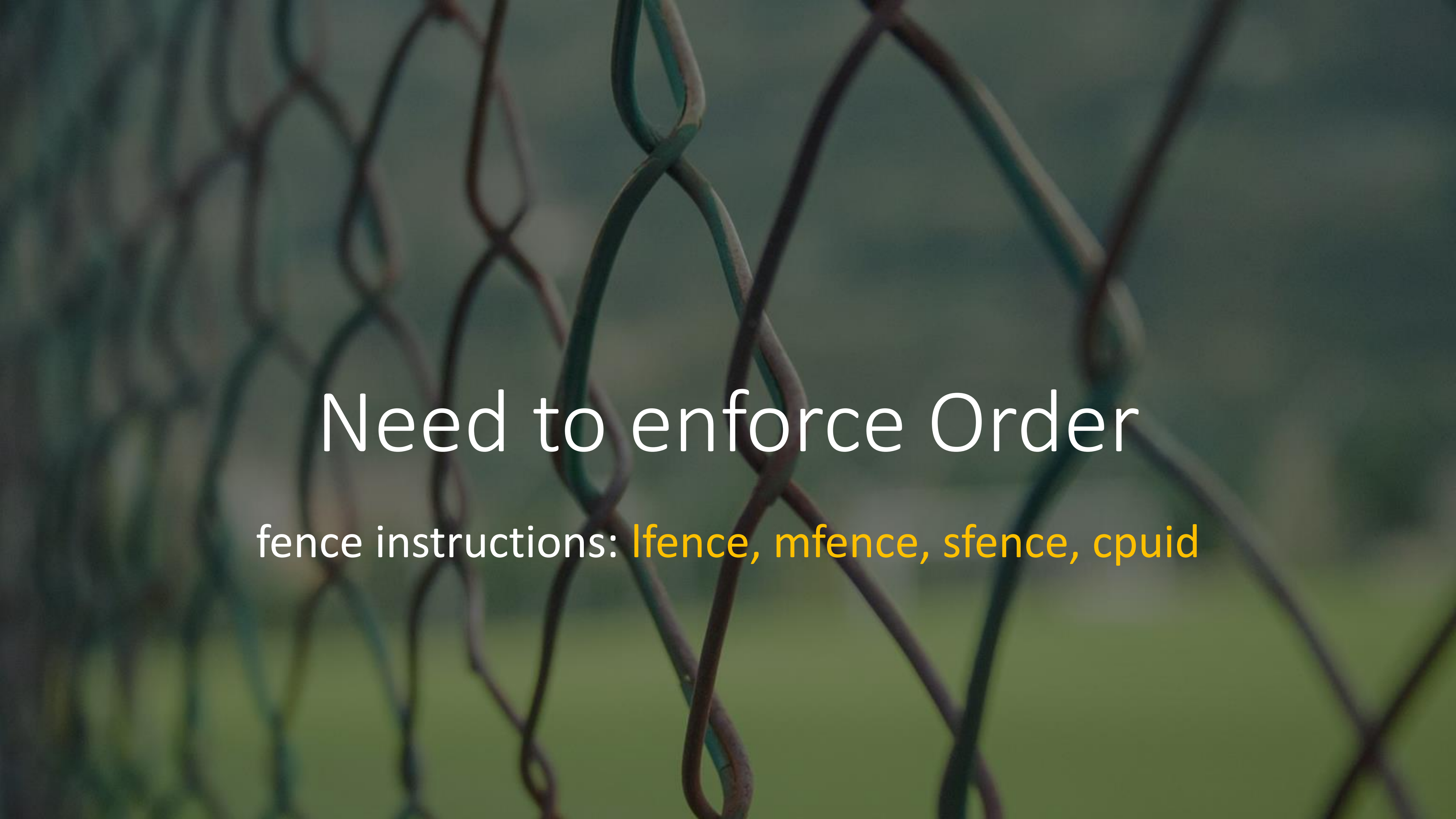
How?

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

Out of order processors



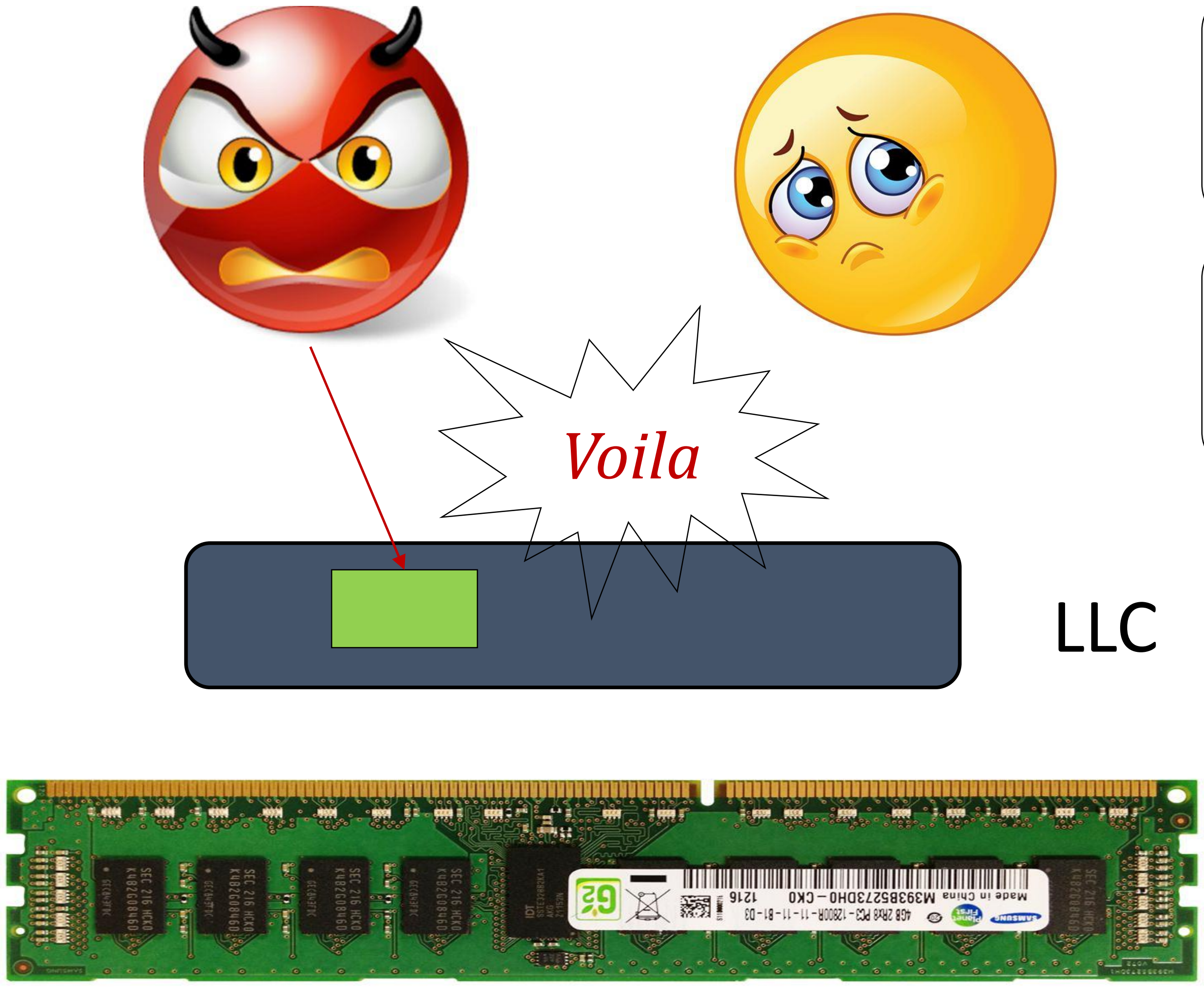
Out-of-order execution ☹
(Multiple fetch in one cycle)



Need to enforce Order

fence instructions: lfence, mfence, sfence, cpuid

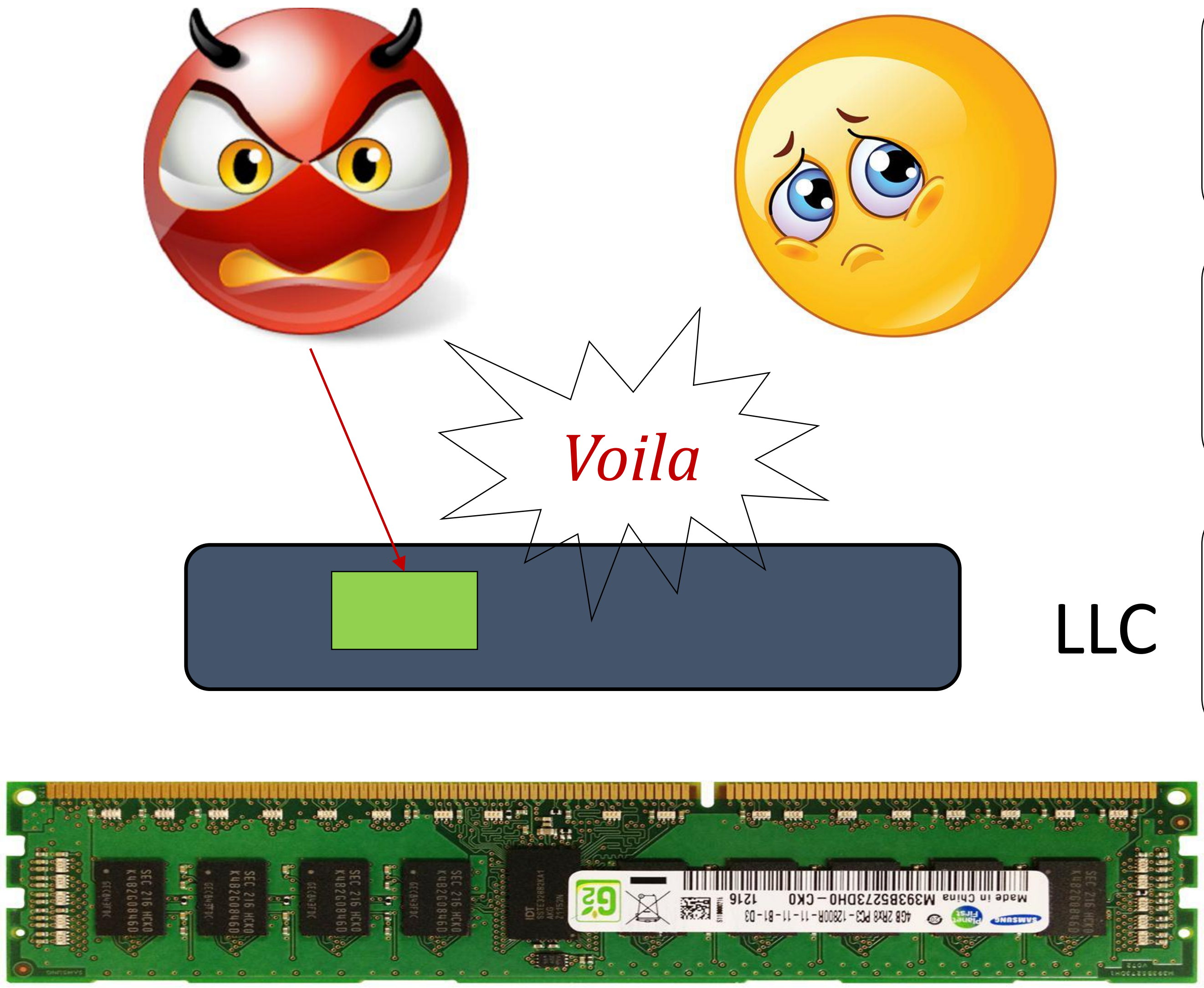
Flush + Flush



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

Step 1:Spy *flushes* the cache block

Flush + Flush

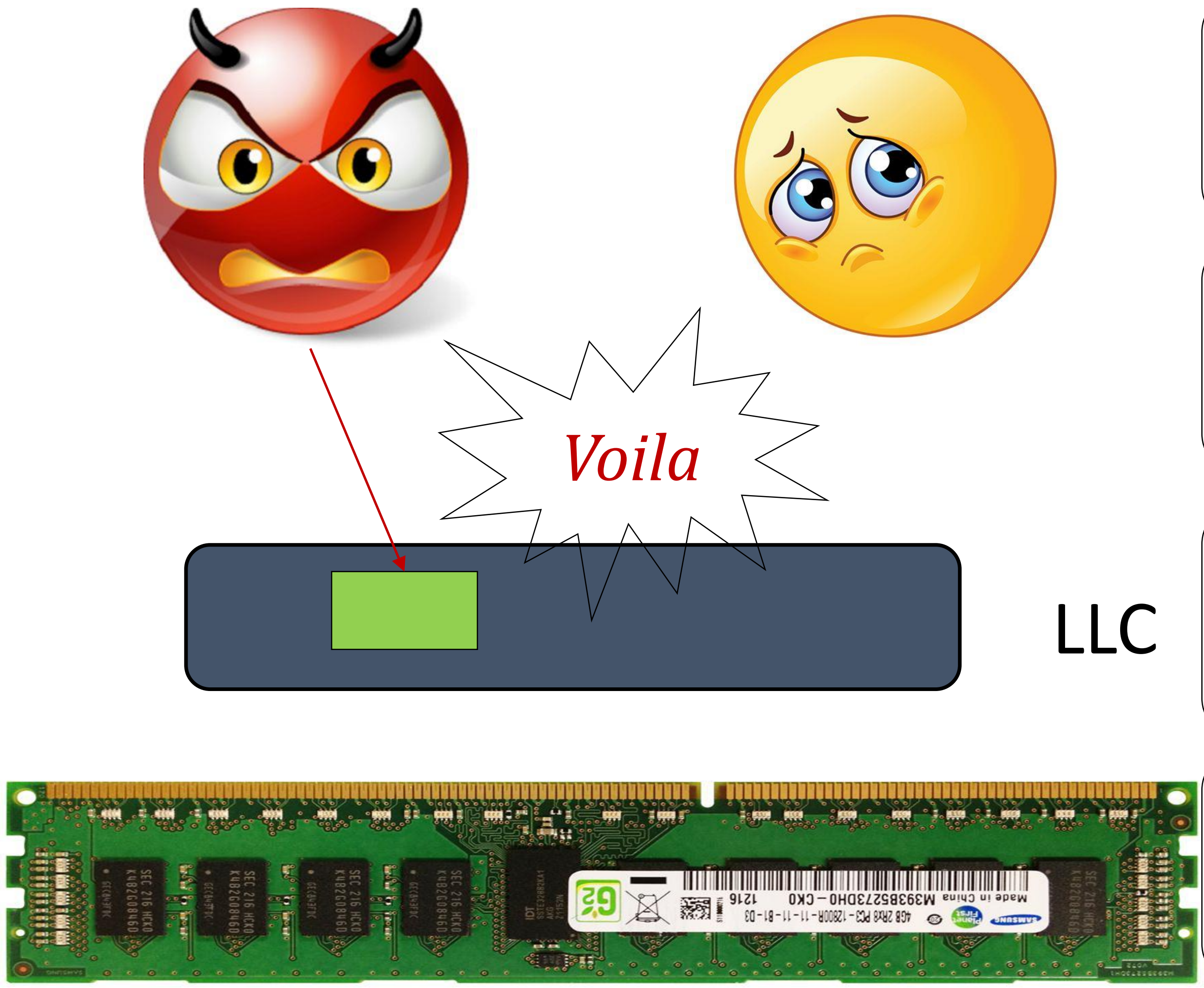


Step 0:Spy *maps* the shared library, shared in the cache

Step 1:Spy *flushes* the cache block

Step 2: Victim *reloads* the cache block

Flush + Flush



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 *flushes* the cache block again

No sharing?

What If I do not share anything with you ??

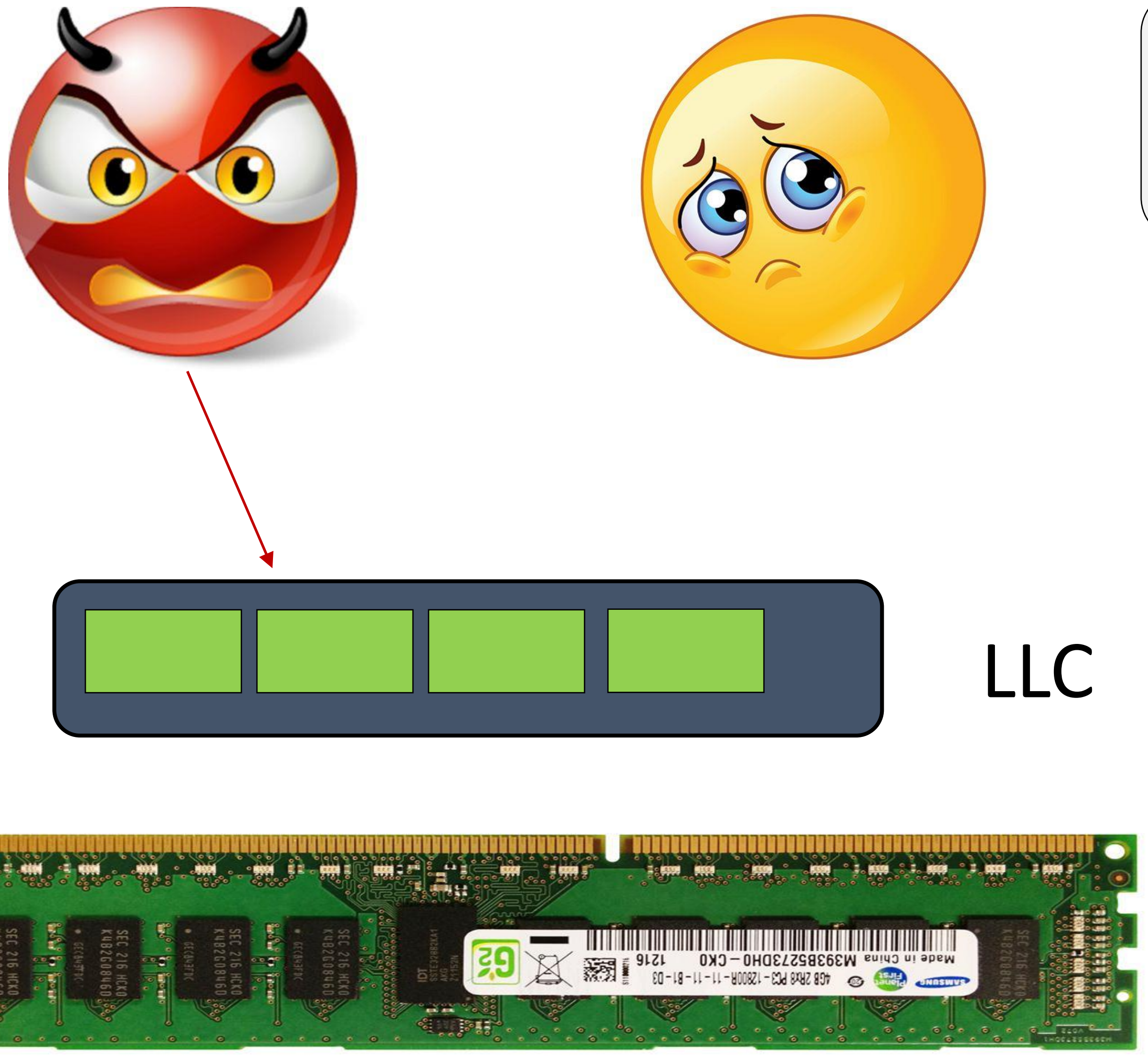
Do not worry, I have Amazon Prime

amazon.com Prime

Sorry:

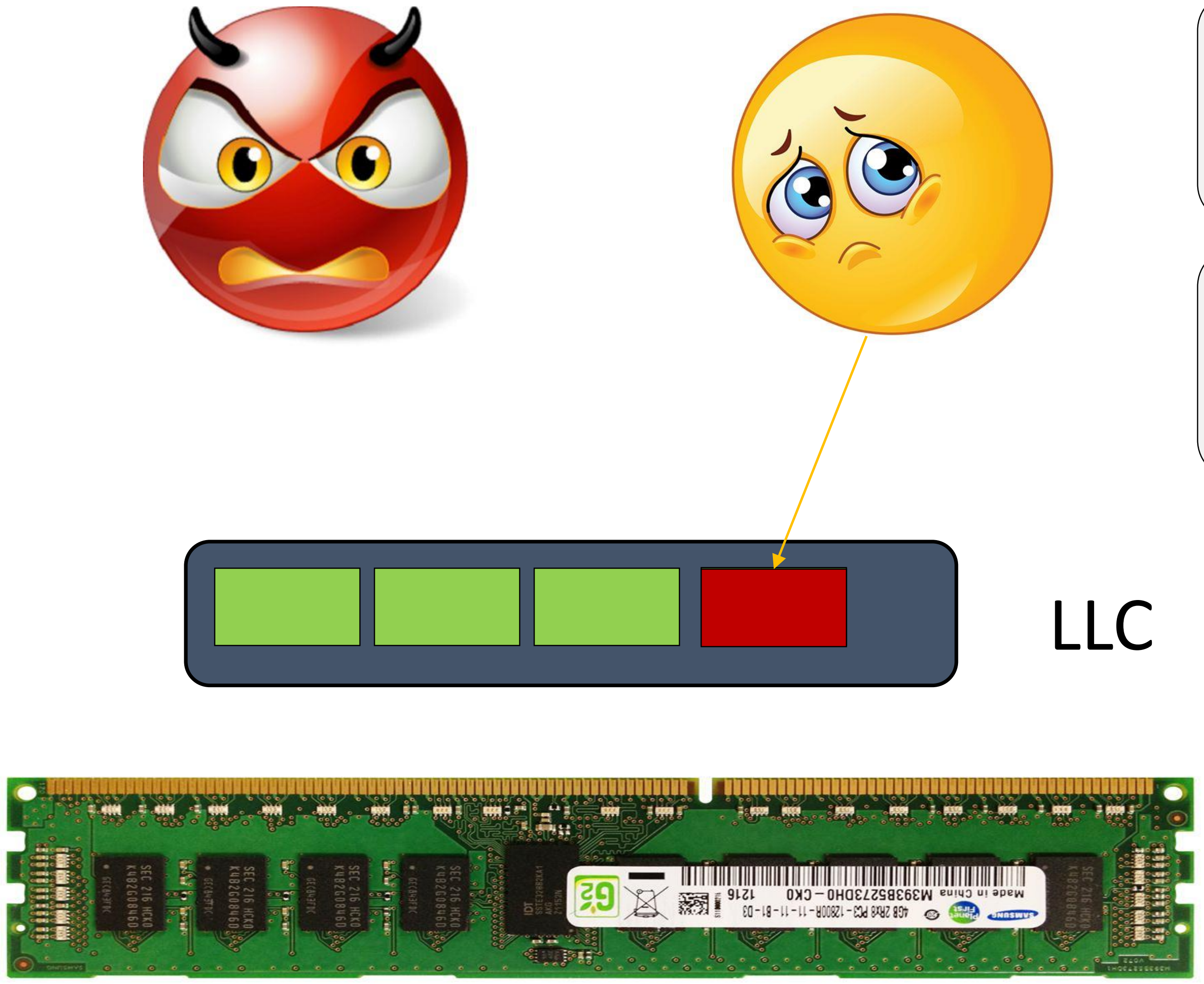
amazon.com Prime+Probe

Prime+Probe



Step 0:Spy *fills* the entire shared cache

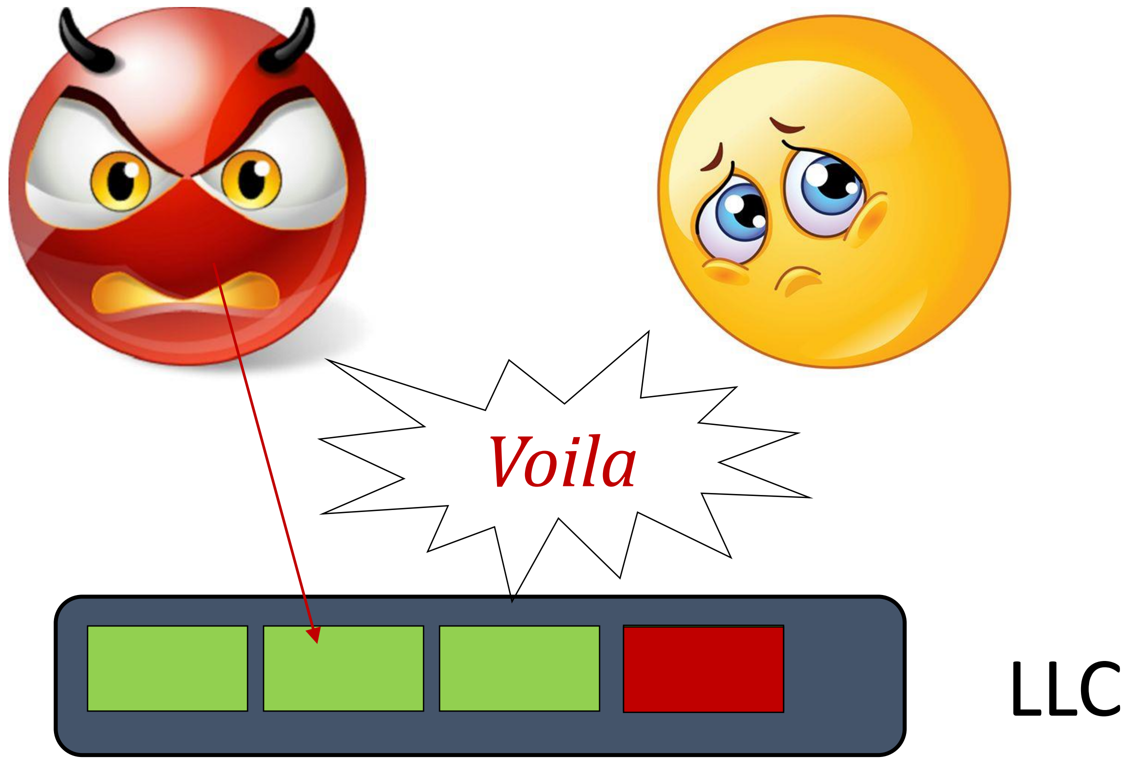
Prime+Probe



Step 0:Spy *fills* the entire shared cache

Step 1: Victim *evicts* cache blocks while running

Prime+Probe



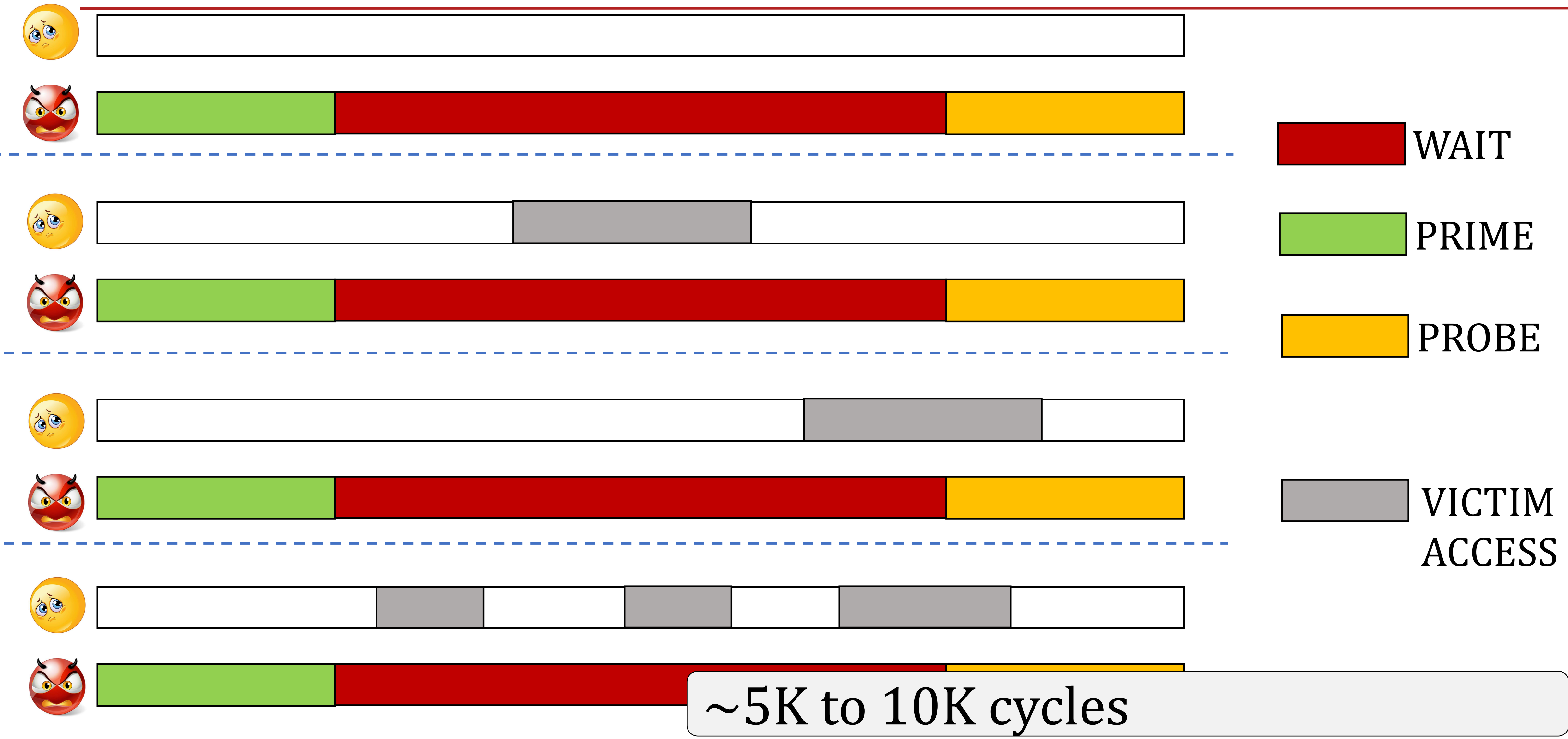
Step 0:Spy *fills* the entire shared cache

Step 1: Victim *evicts* cache blocks while running

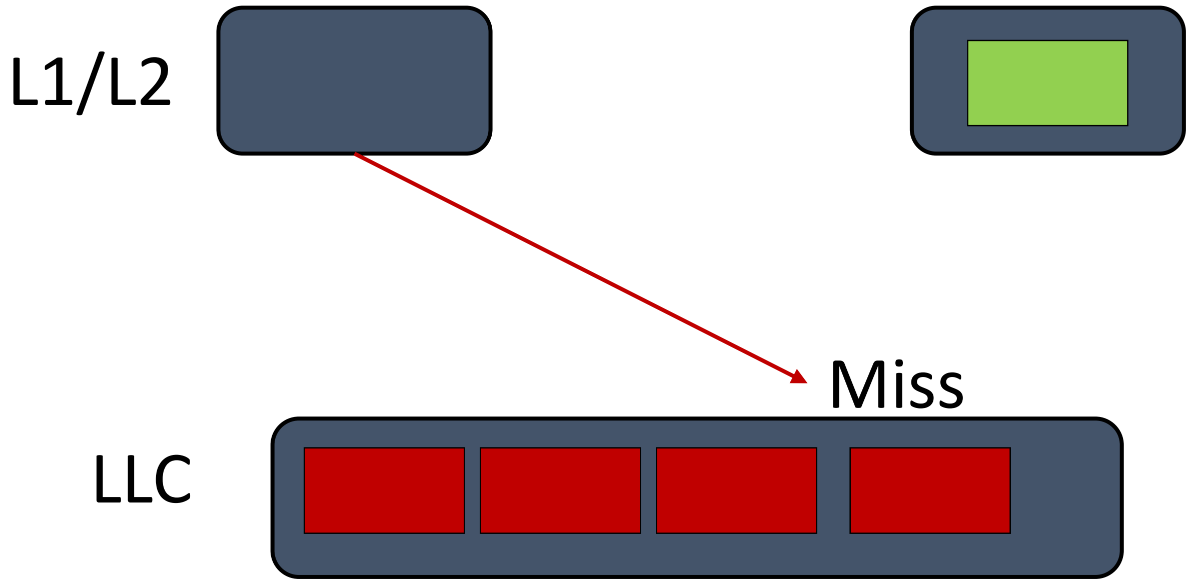
Step 2: Spy *probes* the cache set

If misses then victim has accessed the set

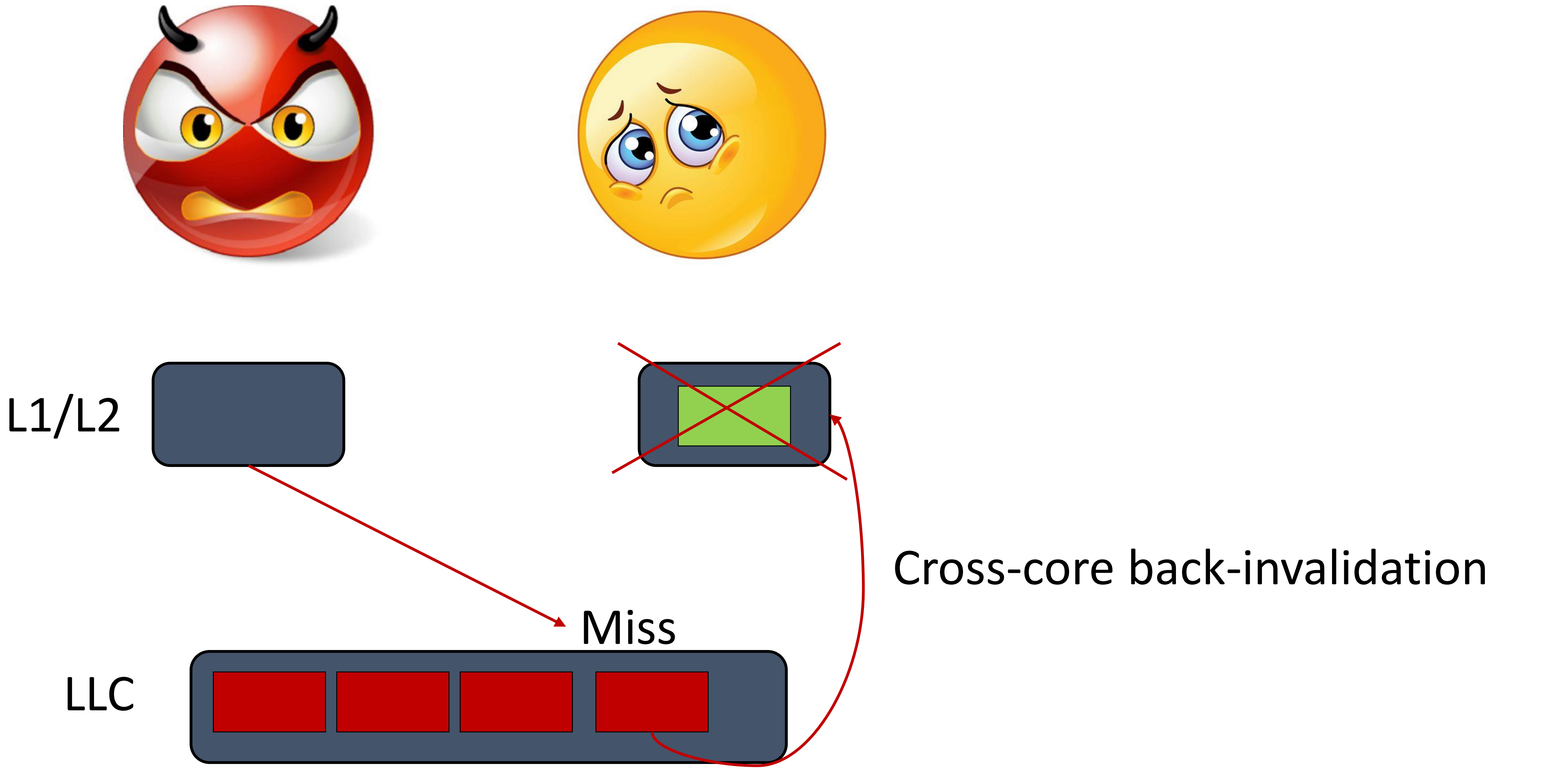
Notion of Time Gap



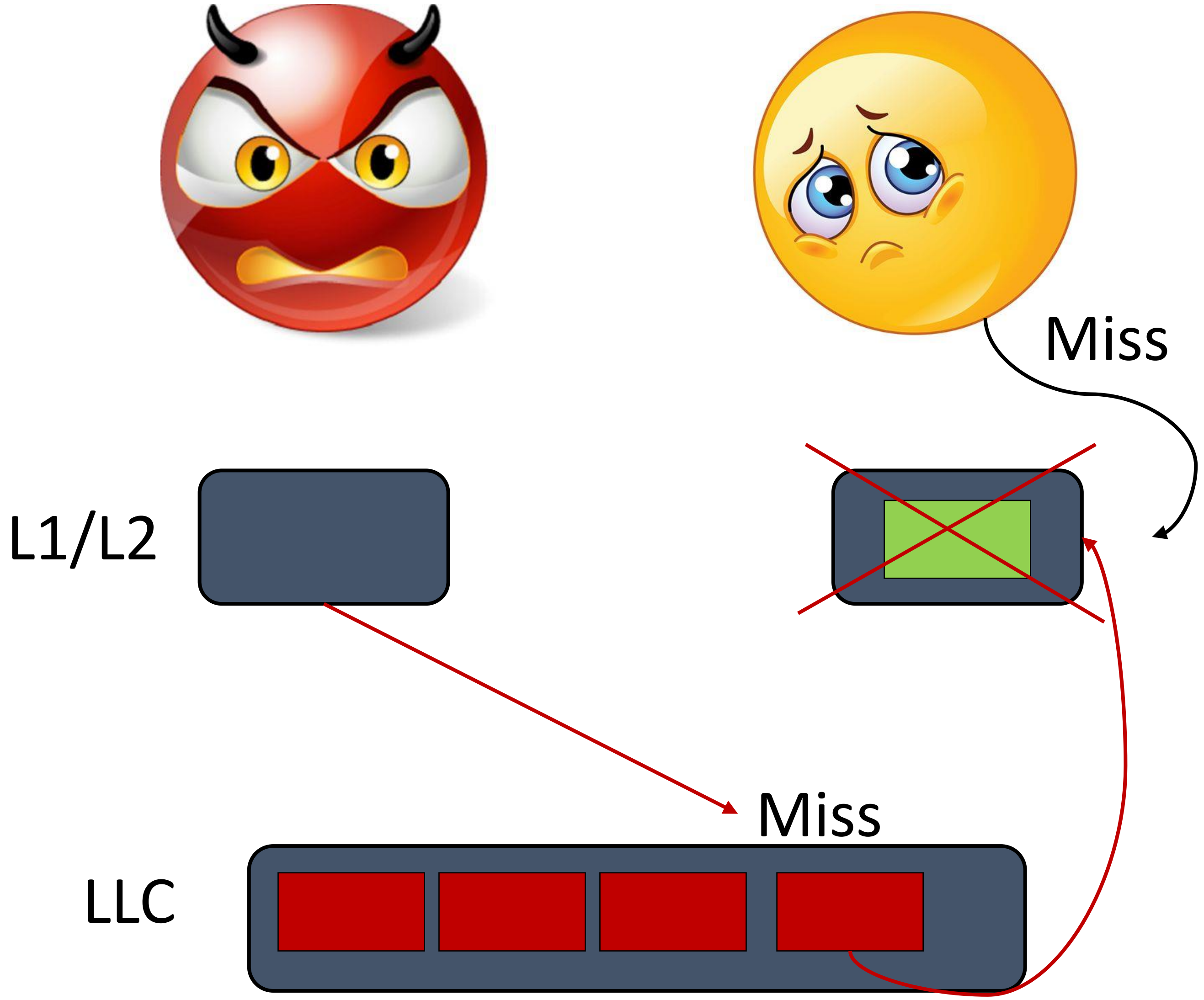
Inclusiveness



Inclusiveness



Inclusiveness



Attacker knows whether victim has accessed a set or not

Job of an attacker

CALIBRATION; FOR LATENCY THRESHOLD

FIND OUT ADDRESSES OF INTEREST

BITS OF INTEREST

Pause

Readings

- Flush+Reload: https://www.usenix.org/node/184416.
- Flush+Flush: https://arxiv.org/abs/1511.04594

Source Code

https://github.com/0xd3ba/Flush-Reload

Thanks