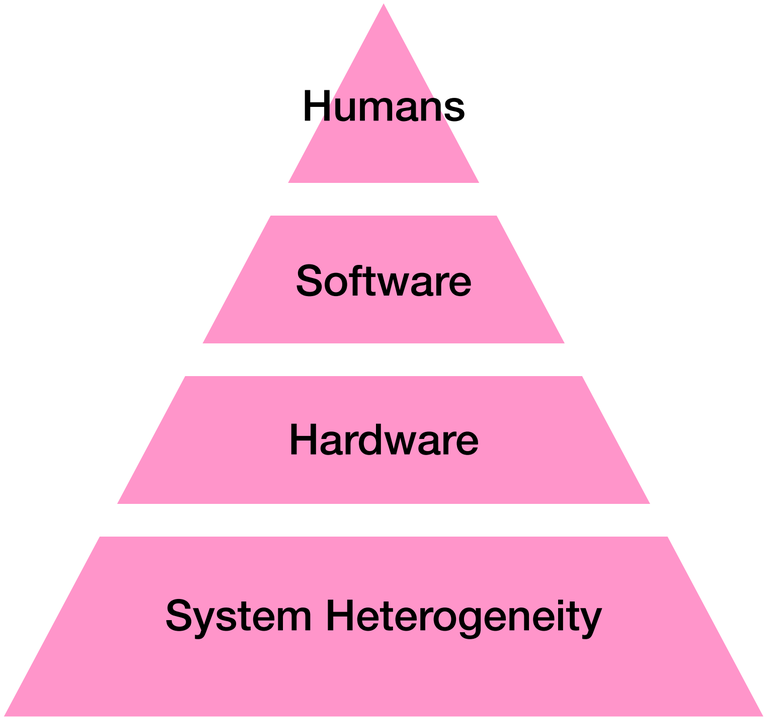
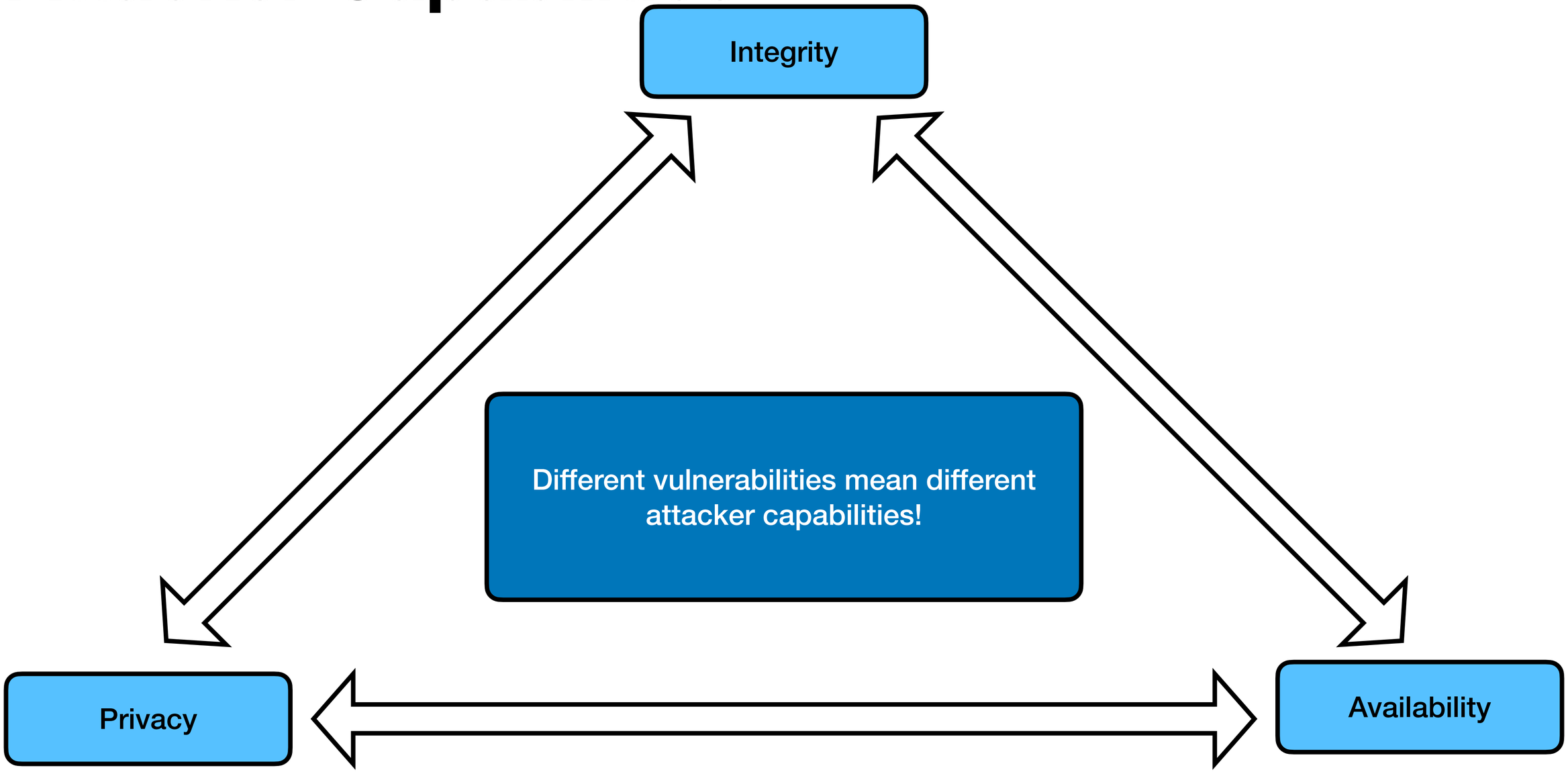
Hardware Security

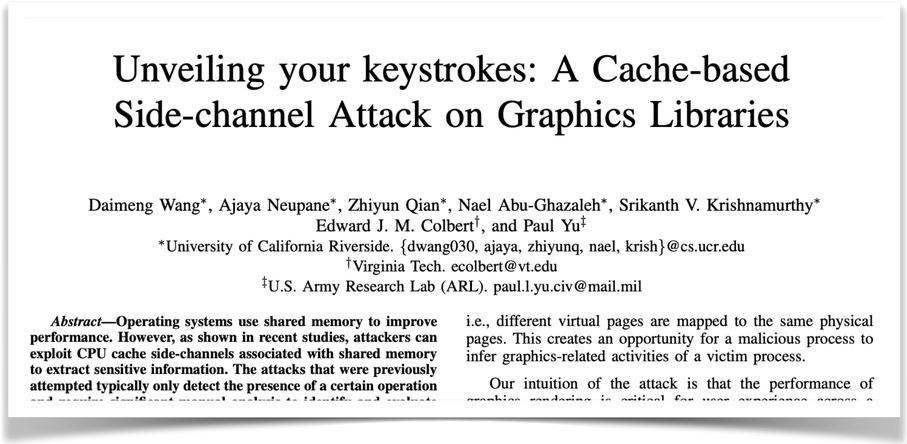
The Security Stack

- In order to understand how to build secure systems, we should think about ways in which they are vulnerable
- Different levels of threat imply different degrees of defenses
- In general, defenses at one level of abstraction do not apply to the next level of vulnerability
- Different degrees of threat may apply to different computing contexts



Attacker Capabilities





Unveiling your keystrokes: A Cache-based Side-channel Attack on Graphics Libraries

Daimeng Wang*, Ajaya Neupane*, Zhiyun Qian*, Nael Abu-Ghazaleh*, Srikanth V. Krishnamurthy*
Edward J. M. Colbert†, and Paul Yu‡
*University of California Riverside. {dwang030, ajaya, zhiyunq, nael, krish}@cs.ucr.edu
†Virginia Tech. ecolbert@vt.edu
‡U.S. Army Research Lab (ARL). paul.l.yu.civ@mail.mil

***Abstract*—Operating systems use shared memory to improve performance. However, as shown in recent studies, attackers can exploit CPU cache side-channels associated with shared memory to extract sensitive information. The attacks that were previously attempted typically only detect the presence of a certain operation**

i.e., different virtual pages are mapped to the same physical pages. This creates an opportunity for a malicious process to infer graphics-related activities of a victim process.

Our intuition of the attack is that the performance of

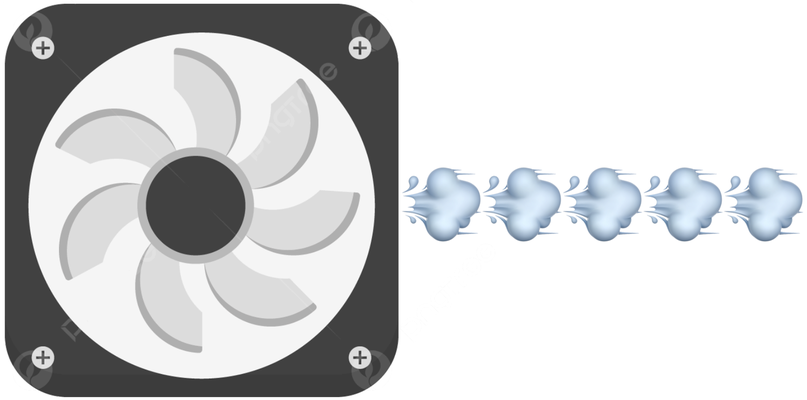
Leaking Information via Side Channels

Privacy

Defining Side Channels

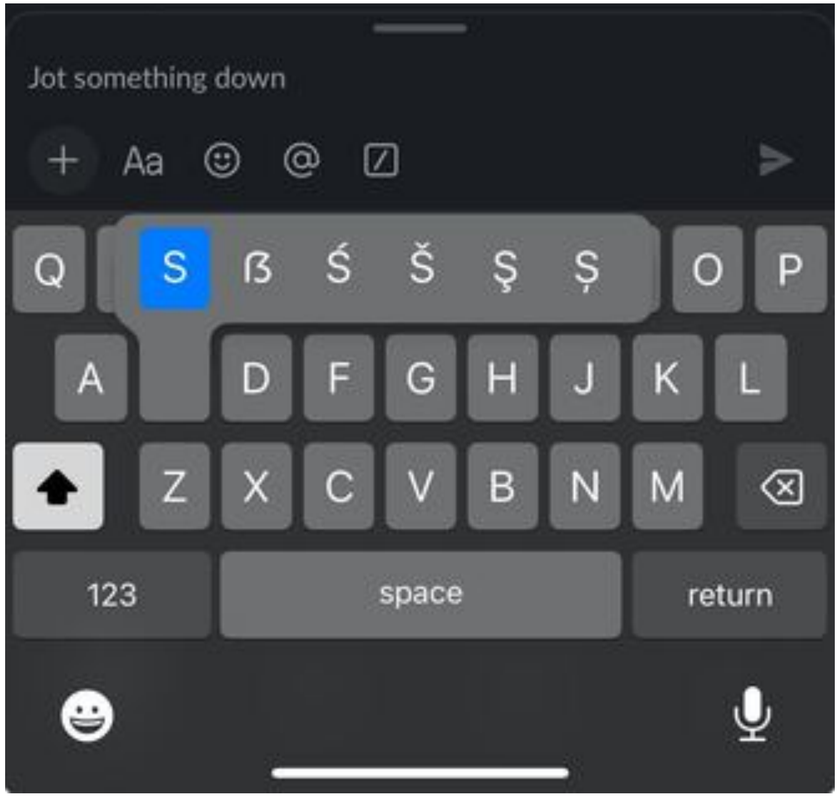
- A *side channel* describes incidental information leakage that can be inferred from observing normal execution
- How does hardware leak information?
  - Noise! 🐓🕺
  - Heat (and power dissipation)! ☀️
  - Timing! ⏱️
- Adversaries need to have some notion of meaning associated with the information that is leaked by the behavior

```
for (;;) {
   // super intense computation!
}
```



Privacy Violation with Hardware

- If an adversary can learn your keystrokes, then they can know anything that you type (e.g., passwords, unsubmitted searches, etc)!
- When pressing keys on a virtual keyboard (i.e., on a touchscreen), typically the pressed key is highlighted
- To perform the “highlight” operation, the keyboard includes and executes code from a graphics library that will update the display



Identifying Vulnerable Code

Source from libcairo.so which is used in Ubuntu Linux at the beginning and end of calling "renderStart" and "renderEnd"!

```
static void D32_LCD32_Opaque(...) {
    ...
    do {
        blit_lcd32_opaque_row(dstRow, srcRow, color, width);
        dstRow = (SkPMColor*)((char*)dstRow + dstRB);
        srcRow = (const SkPMColor*)((const char*)srcRow + maskRB);
    } while (--height != 0);
}
```

```
static void blit_lcd32_opaque_row(dst, src, color, width) {
    ...
    for (int i = 0; i < width; i++) {
        if (0 == src[i]) {
            continue;
        }
        ...
    }
}
```

Image credit: https://www.ndss-symposium.org/ndss-paper/unveiling-your-keystrokes-a-cache-based-side-channel-attack-on-graphics-libraries/

Memory access by calling src[i]

Asymmetric timing depending on the state of the data!

Identifying Vulnerable Code

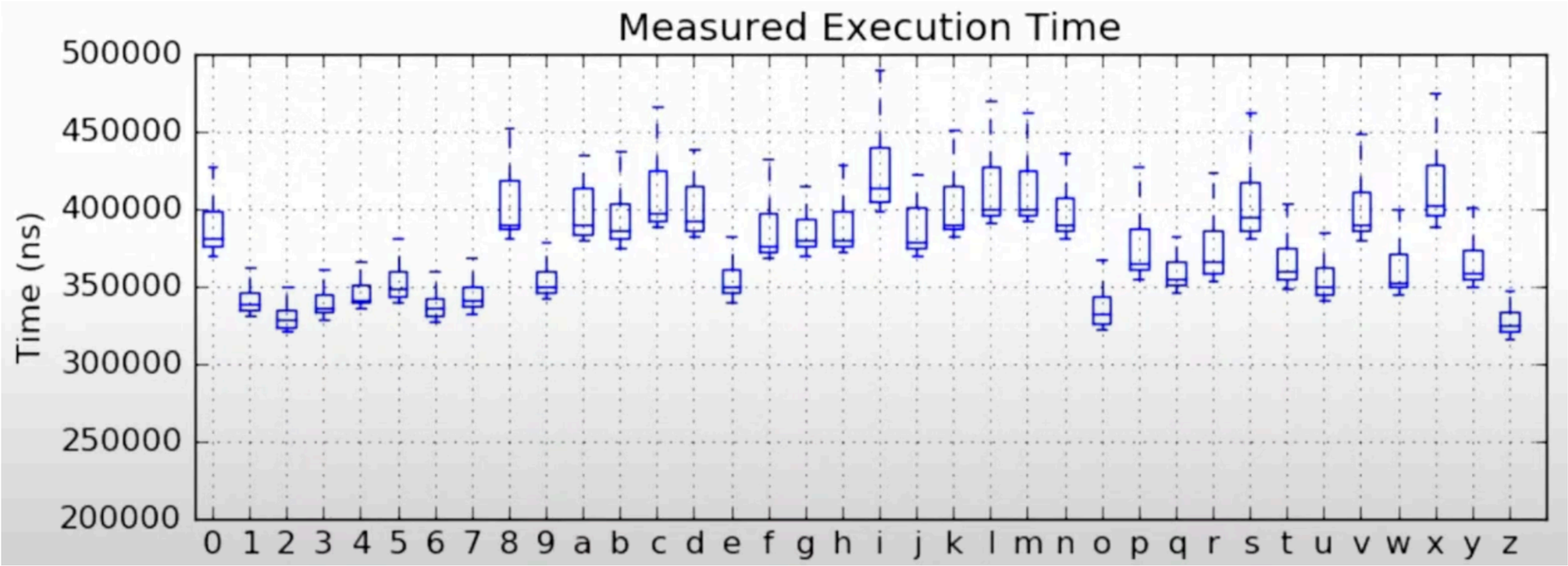
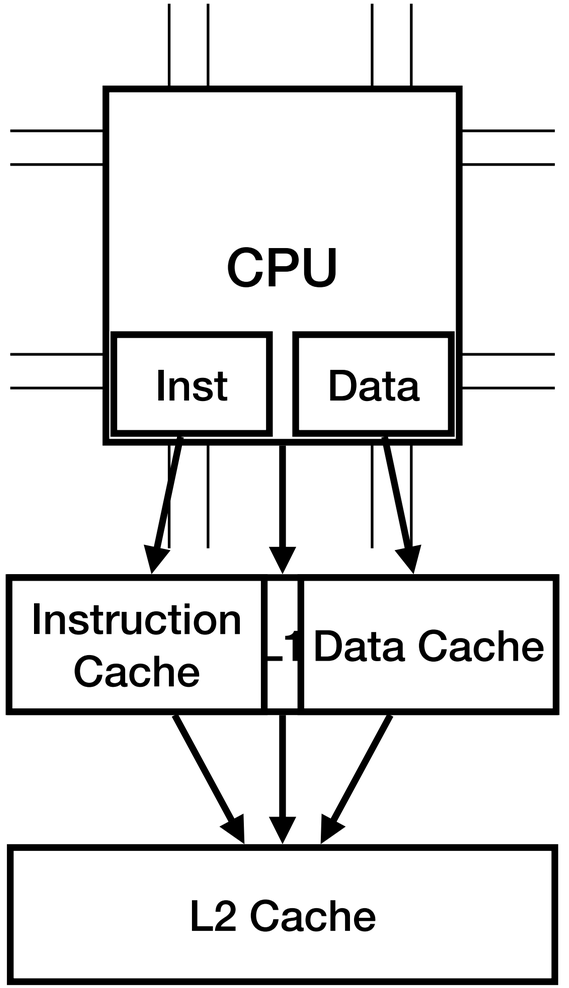


Image credit: https://www.ndss-symposium.org/ndss-paper/unveiling-your-keystrokes-a-cache-based-side-channel-attack-on-graphics-libraries/

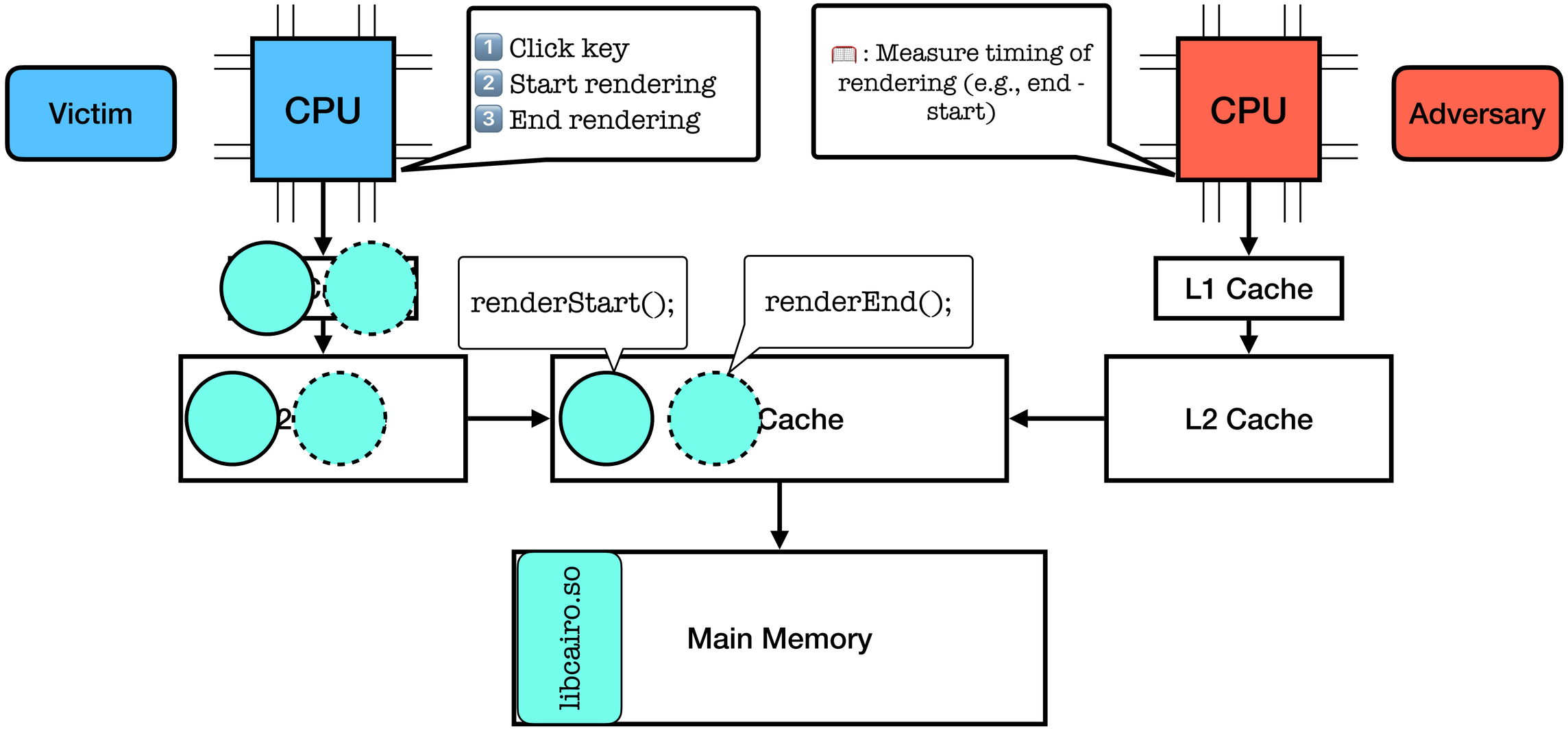
Different characters to render means that there are different amounts of whitespace, so the timing to render different characters will be distinct

Review: Cache Hierarchy

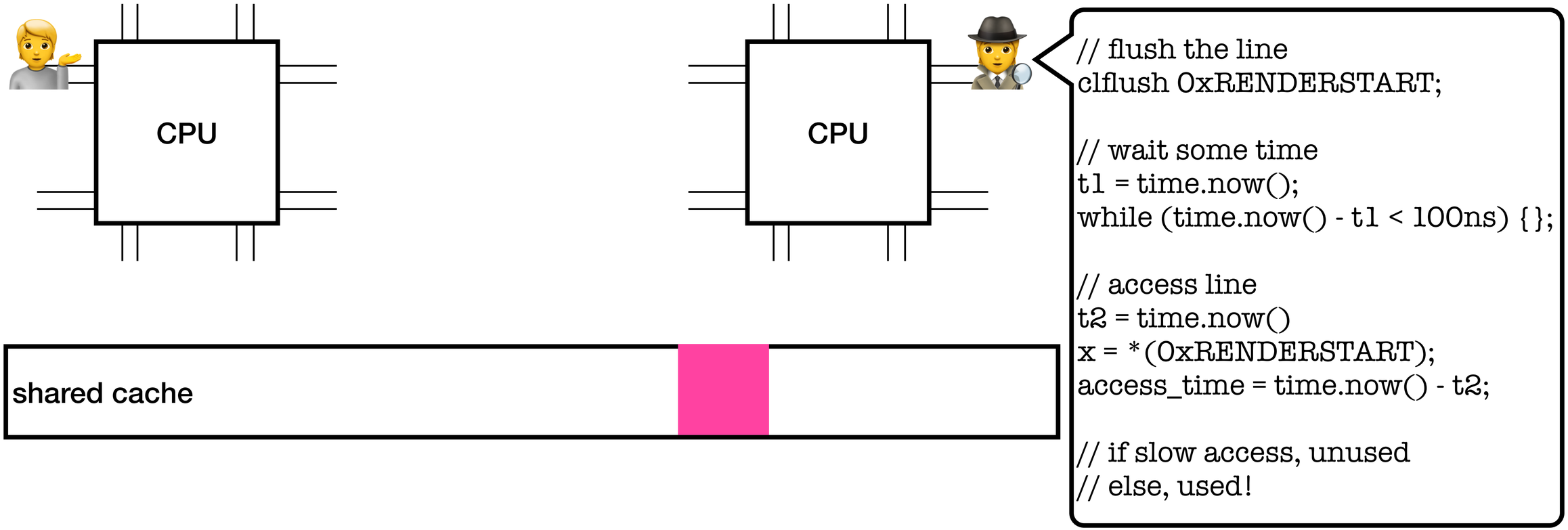
- Hardware uses caches to maintain recently accessed data
- Data means both application data *and* instructions!
- If something was used recently, then it will appear in the caches… can this be exploited?



Shared Memory Model



Flush + Reload Attack



So, how effective was the attack?

TABLE III: Example dictionary-assisted password guessing attack for password “hello”.

Input	Confidence Vector (Partial)							
	e	h	i	j	l	o	s	y
h	0.0	0.39	0.0	0.23	0.0	0.0	0.0	0.03
e	0.21	0.0	0.0	0.0	0.0	0.0	0.0	0.03
l	0.0	0.0	0.05	0.0	0.37	0.0	0.07	0.0
l	0.0	0.0	0.05	0.0	0.37	0.0	0.07	0.06
o	0.0	0.0	0.0	0.0	0.0	0.15	0.0	0.0

Rank	Dictionary Words	Confidence Value
1	hello	$0.39 + 0.21 + 0.37 + 0.37 + 0.15 = 1.49$
2	jelly	$0.23 + 0.21 + 0.37 + 0.37 + 0.0 = 1.18$
3	hills	$0.39 + 0.0 + 0.37 + 0.37 + 0.0 = 1.13$
4	holly	$0.39 + 0.0 + 0.37 + 0.37 + 0.0 = 1.13$

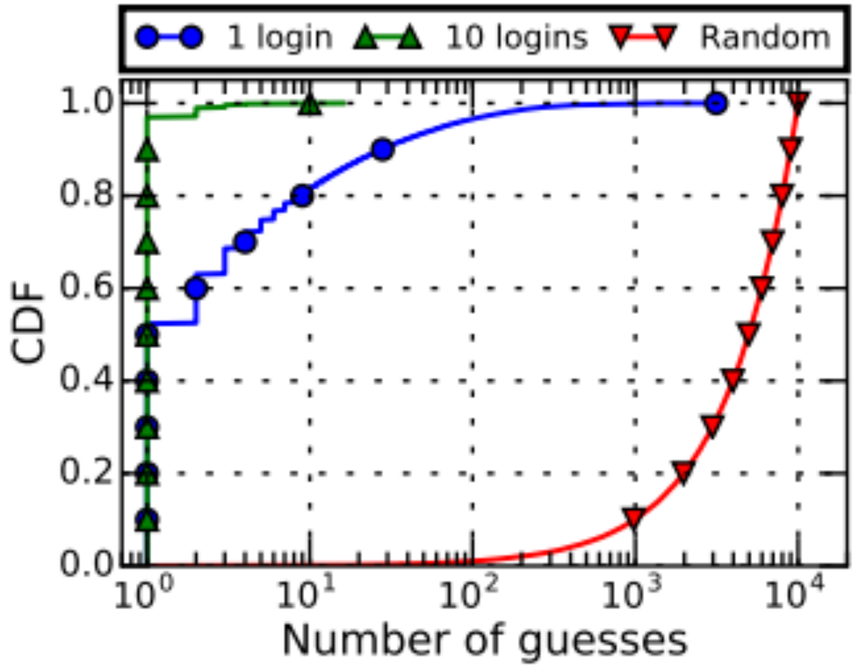


Image credit: https://www.ndss-symposium.org/ndss-paper/unveiling-your-keystrokes-a-cache-based-side-channel-attack-on-graphics-libraries/

Specu tacks: The Fu annel

DOI:10.1145/3399742

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, and Yuval Yarom

Abstract

Modern processors use branch prediction and speculative execution to maximize performance. For example, if the destination of a branch depends on a memory value that is in the process of being read, CPUs will try to guess the destination and attempt to execute ahead. When the memory value finally arrives, the CPU either discards or commits the speculative computation. Speculative logic is unfaithful in how it executes, can access the victim's memory and registers, and can perform operations with measurable side effects.

Spectre attacks involve inducing a victim to speculatively perform operations that would not occur during correct

attacks, which do not require external measurement equipment. Although some attacks exploit software logic errors, other software attacks leverage hardware properties to infer sensitive information. Attacks of the latter type include microarchitectural attacks exploiting cache timing[3, 6, 17] and branch prediction history.[1] Software-based techniques have also been used to induce computation errors, such as fault attacks that alter physical memory[11] or internal CPU values.[25]

Several microarchitectural design techniques have facilitated the increase in processor speed over the past decades. One such advancement is speculative execution, which is widely used to increase performance and involves having

About speculative executio vulnerabilities in ARM-based and Intel CPUs

- Apple has released security updates for macOS Sierra and El Capitan with mitigations for Meltdown.
- Apple has released updates for iOS, macOS High Sierra, and Safari on Sierra and El Capitan to help defend against Spectre.
- Apple Watch is unaffected by both Meltdown and Spectre.

Security researchers have recently uncovered security issues known by two names, Meltdown and Spectre. These issues apply to all modern processors and affect nearly all computing devices and operating systems. All Mac systems and iOS devices are affected, but there are no known exploits

Privacy

Image Credit: https://support.apple.com/en-us/101886

(Way too fast) Instruction Level Parallelism

```
if (arr[0] == 1) {
    fn1();
} else {
    fn2();
}
```

```
mov     ... // fetch arr[0]
test    ...
jnz     ... // if statement
call    ... // call fn1
jmp     ... // skip else
call    ... // call fn2
```

Chat with your neighbor(s)!

Suppose your processor starts one instruction per cycle. How many instructions are running at clock cycle 5?

For a five-stage pipeline, five instructions in parallel! Why? Take computer architecture

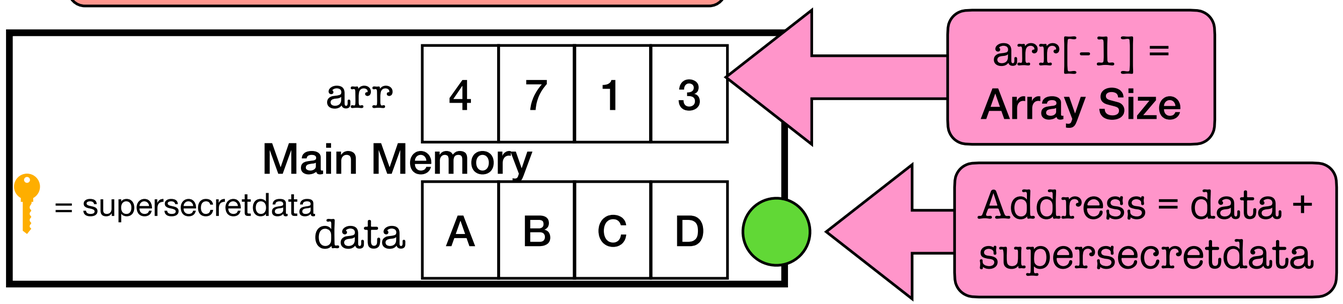
If it takes five cycles to finish executing an instruction, how does our processor know which instruction to execute next when it encounters a branch?

It predicts with >99% accuracy! If wrong, no big deal… just roll back the state

Spectre Variant 1 Vulnerability

```
void fn(x) {
  if (x < size) {
    y = array[x];
  } else ...
}
```

```
void fn(x) {
  if (arr[x] < arr[y]) {
    z = data[arr[x]];
  } else ...
}
```



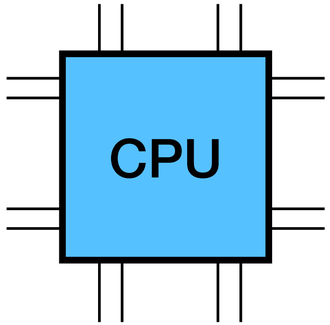
If the branch predictor will guess that the branch is *not taken* then the "if" condition will execute speculatively

An adversary can pass an input value `x` where `x > size` which will lead to a `load` from an arbitrary location in memory… bad!

Is this dangerous? From our cache side channels, all that the adversary can leak is that an address exists in the cache, not its data…

Tying Data Contents to Memory Location

Suppose we wanted to steal the contents of `arr[2]`...




```
void fn(x) {
  if (arr[x] < arr[y]) {
    z = data[arr[x]];
  } else ...
}
```

L3 Cache

arr

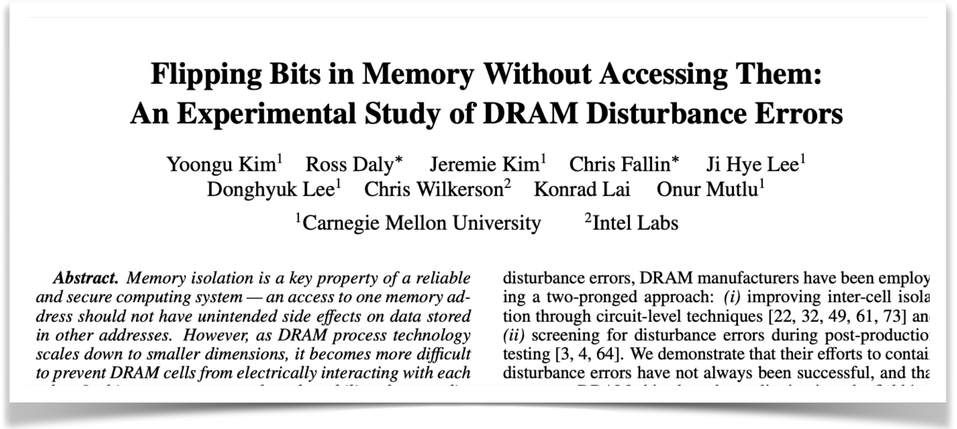
data

Can we read the *contents* of the block? No! We can only read the address of the block...

04	07	01	03	00	00	00	0A	0B	0C	0D	00	00	supersecretdata	
0xff00	0xff01	0xff02	0xff03	0xff04	0xff05	0xff06	0xff07	0xff08	0xff09	0xff0a	0xff0b	0xff0c	0xff0d	

What was the data at `arr[2]`? The contents are address of read block - address of the start of `data`

Integrity



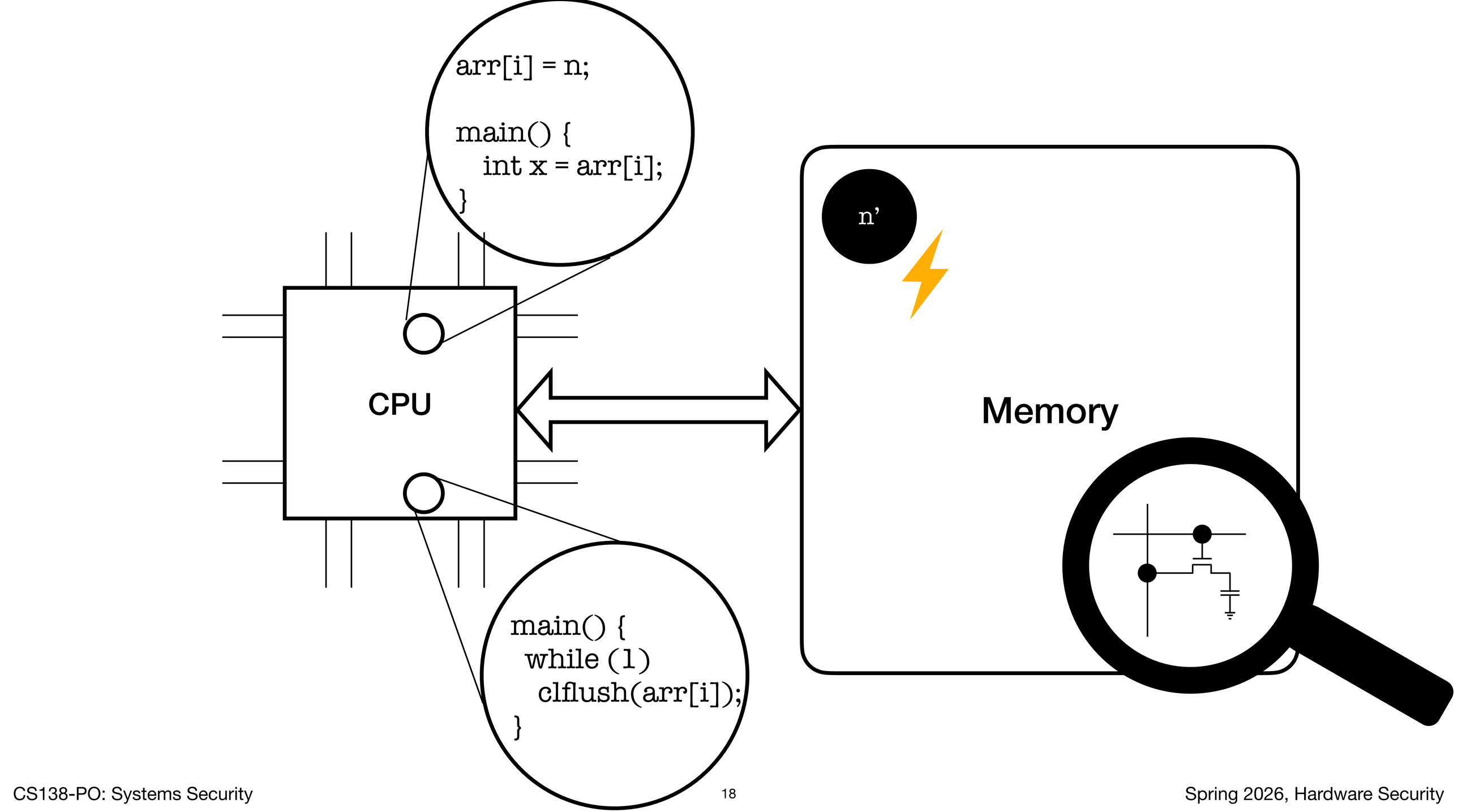
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Yoongu Kim[1] Ross Daly* Jeremie Kim[1] Chris Fallin* Ji Hye Lee[1]
Donghyuk Lee[1] Chris Wilkerson[2] Konrad Lai Onur Mutlu[1]
[1]Carnegie Mellon University [2]Intel Labs

Abstract. *Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology scales down to smaller dimensions, it becomes more difficult to prevent DRAM cells from electrically interacting with each*

disturbance errors, DRAM manufacturers have been employ
ing a two-pronged approach: *(i)* improving inter-cell isola
tion through circuit-level techniques [22, 32, 49, 61, 73] an
(ii) screening for disturbance errors during post-productio
testing [3, 4, 64]. We demonstrate that their efforts to contai
disturbance errors have not always been successful, and tha

Rowhammer Attacks


arr[i] = n;
main() {
int x = arr[i];
}
CPU
n'
Memory
main() {
while (1)
clflush(arr[i]);
}