

Microarchitectural Security And Attacks

Vincent Ulitzsch <viniul@mit.edu>/Mengjia Yan



What you will learn today: How a CPU's inner workings leak information

1

Programs and users share a CPU

2

This CPU sharing leaks information!

3

And you will learn how to exploit that

Where do we need security?

A wide range of hardware...



running a diverse
set of sensitive
workloads!



What is a side channel?

- By making *indirect* observations (the number of pizzas ordered), one is able to infer partial information

And Bomb The Anchovies

By Paul Gray | Monday, Aug. 13, 1990

 Like 0

 Tweet

 Share

Read Later

Delivery people at various Domino's pizza outlets in and around Washington claim that they have learned to anticipate big news baking at the White House or the Pentagon by the upsurge in takeout orders. Phones usually start ringing some 72 hours before an official announcement. "We know," says one pizza runner. "Absolutely. Pentagon orders doubled up the night before the Panama attack; same thing happened before the Grenada invasion." Last Wednesday, he adds, "we got a lot of orders, starting around midnight. We figured something was up." This time the big news arrived quickly: Iraq's surprise invasion of Kuwait.

Email

Print

Share

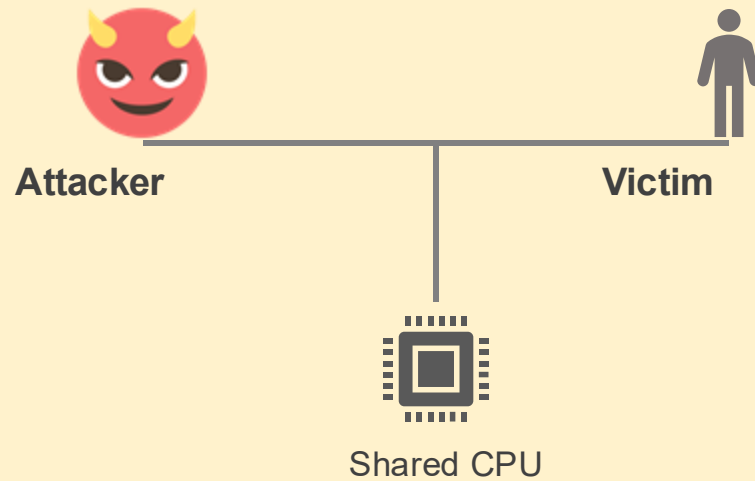
Reprints

Follow @TIME

Threat Model: Sharing the CPU

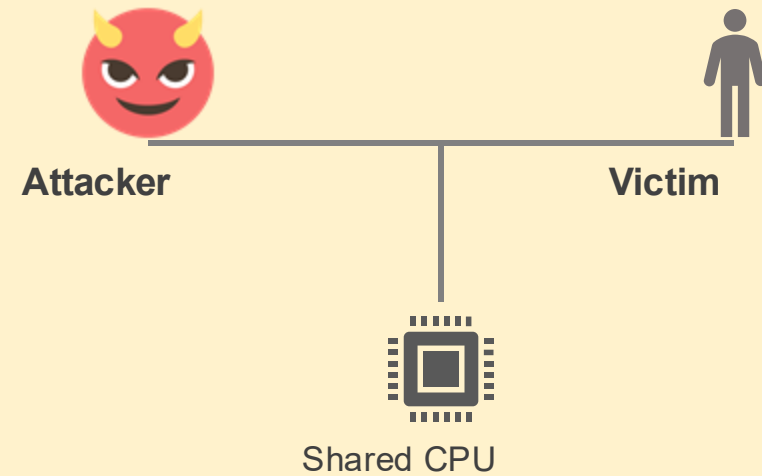
Multi-User System

Many users' programs run on one shared machine



Cloud / Multi-Tenant

Different customers' virtual machines share one physical server under a hypervisor.



What is Covert and Side Channel?

- Gather information by measuring or exploiting **indirect** effects of the system
- Covert channel:
 - **Cooperated/Intended** communication between two or more security parties
- Side channel:
 - **Unintended** communication between two or more security parties
- In both cases:
 - Communication does not follow the system specification
 - The communication medium is not designed to be a communication channel

Side Channels Are Almost Everywhere



Example #1: Acoustic Side Channels

- Monitor keystroke
 - You only need: a cheap microphone + an ML model

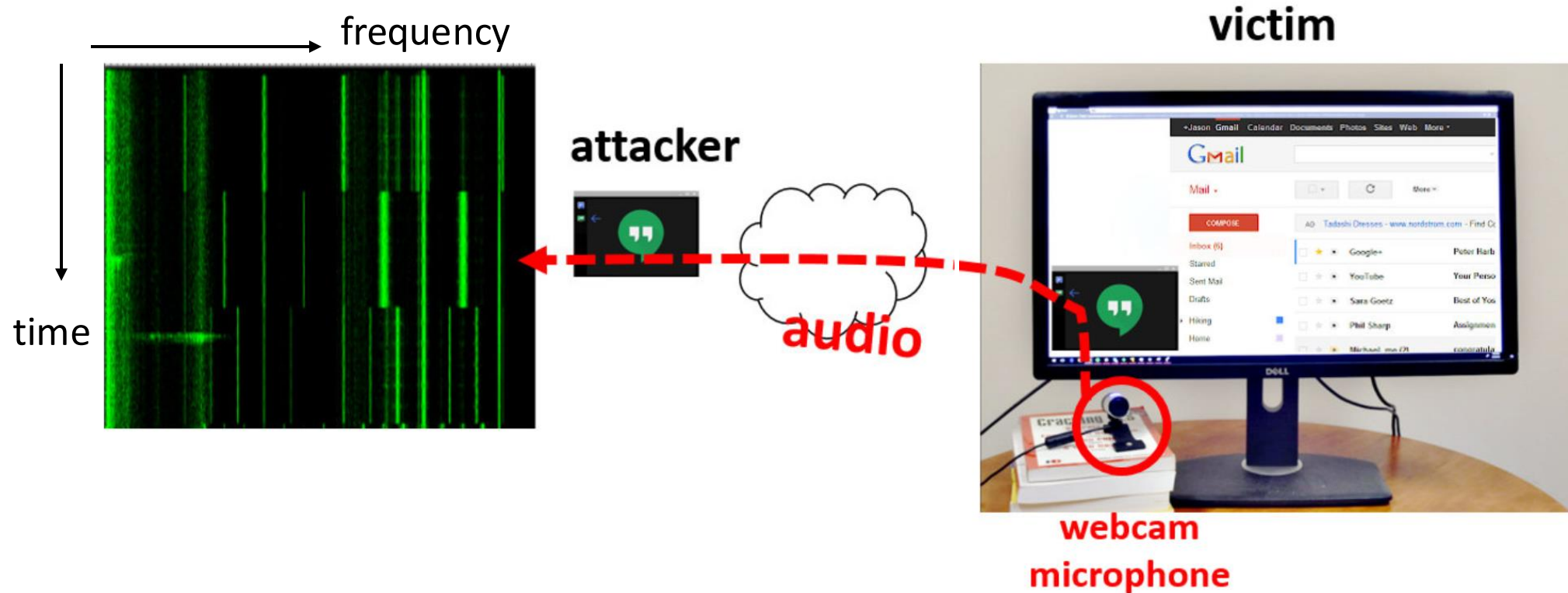
Other sources of acoustic side channels
inside a computer?



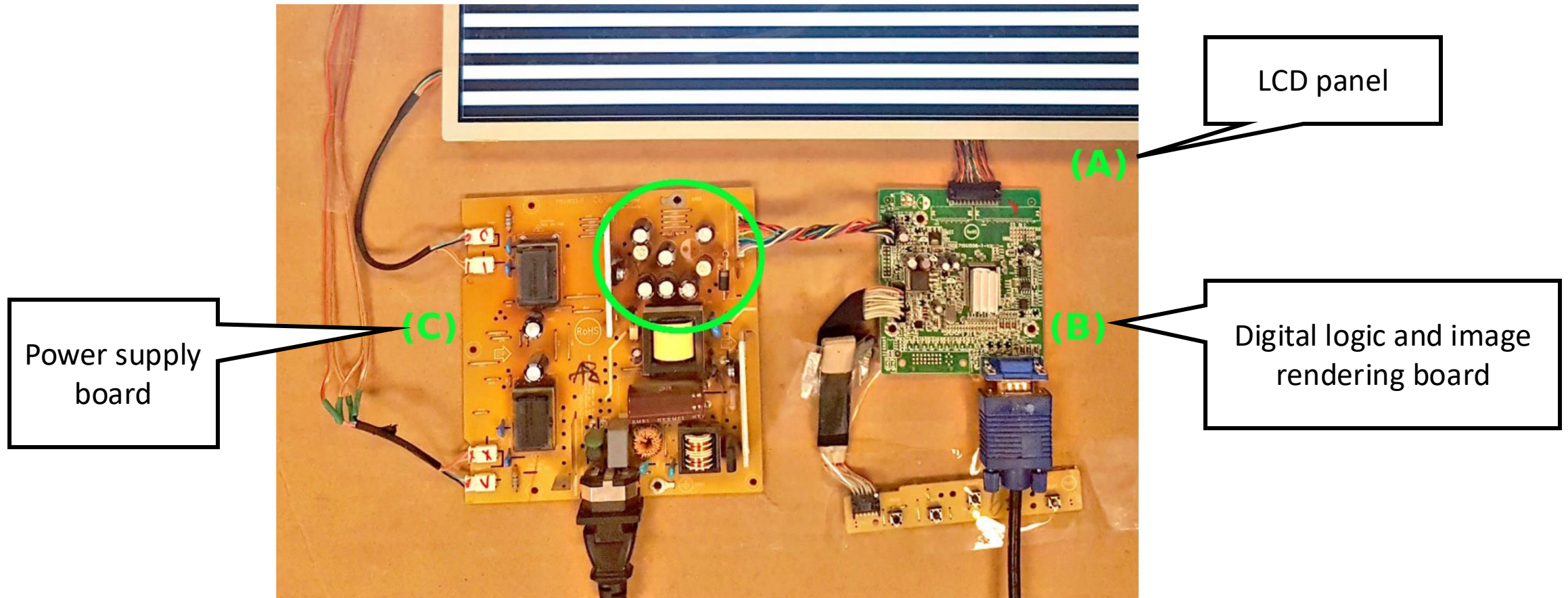
- Another example: “Hear” the screen



“Hear” The Screen

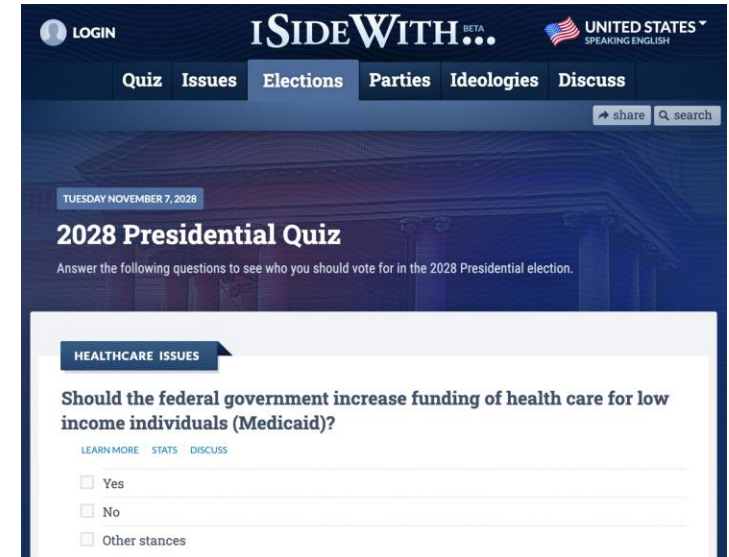


“Hear” The Screen



Example 2: Network Side Channels

- Website Fingerprinting
 - Frequency of packets, size of packets
 - Example: iSideWith.com



- Network traffic contention side channel



Example 3: Timing Side Channel

```
def check_password(input):  
    for i in range(0,128):  
        if (input [i] != password[i]):  
            return ("error");  
  
    return ("success");
```

Early termination

- Password: 128 digits
- How many attempts an attacker needs to brute force a password with **blind guess**?
- Consider the *check_password* program on the left. Can we reduce the number of attempts? How?

Vulnerabilities in Real-world Crypto

- Libgcrypt's Montgomery ladder scalar-by-point multiplication routine

Algorithm 3 Libgcrypt's modular reduction operation (simplified).

Input: Two integers x and m , represented as a sequence of limbs $x_0 \dots x_{l-1}$ and $m_0 \dots m_{k-1}$.

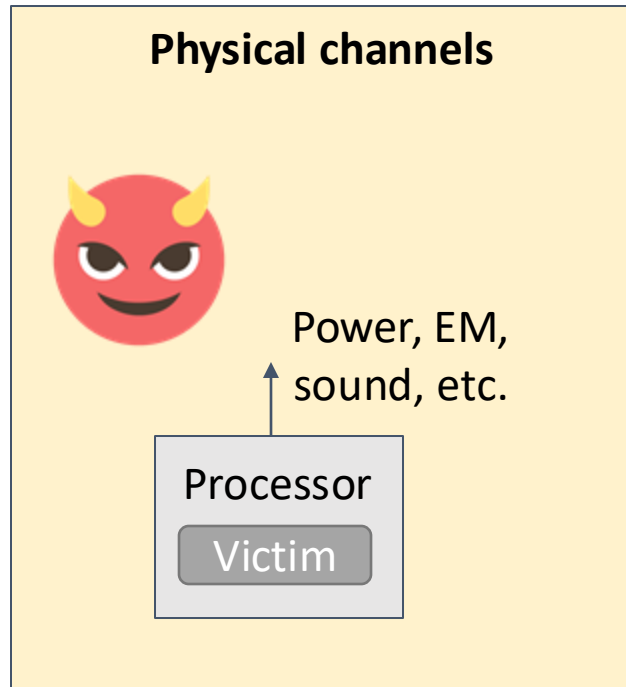
Output: $x \bmod m$.

```
1: procedure MODULAR_REDUCTION( $x, m$ )
2:    $l \leftarrow \text{SIZE\_IN\_LIMBS}(x)$ 
3:    $k \leftarrow \text{SIZE\_IN\_LIMBS}(m)$ 
4:   if  $l < k$  then
5:     return  $x$             $\triangleright$  Early exit if  $x$  is smaller than  $m$ 
6:   for  $i \leftarrow l - 1$  downto  $k - 1$  do
7:      $q \leftarrow (x_i \cdot 2^{64} + x_{i-1}) / m_{k-1}$     $\triangleright$  Estimate quotient  $q$ 
8:     if  $q(m_{k-1} \cdot 2^{128} + m_{k-2}) > x_i \cdot 2^{128} + x_{i-1} \cdot 2^{64} + x_{i-2}$ 
       then
9:        $q \leftarrow q - 1$             $\triangleright$  If  $q$  is too large, adjust estimate
10:       $x \leftarrow x - q \cdot m \cdot 2^{64(i-k)}$     $\triangleright$  Subtract from  $x$ 
11:  return  $x$             $\triangleright x$  holds the remainder
```

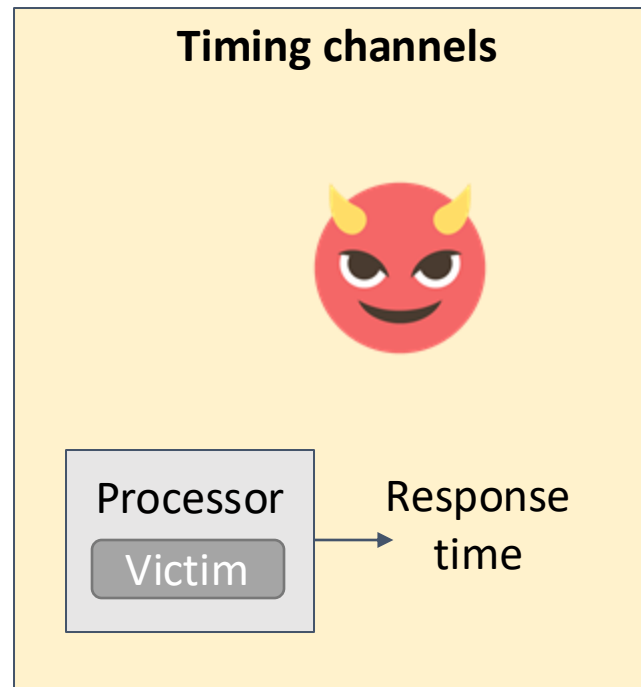
Vulnerability exists in a real-world implementation of Curve25519.

Genkin et al. May the Fourth Be With You: A Microarchitectural Side Channel Attack on Several Real-World Applications of Curve25519; CCS'17

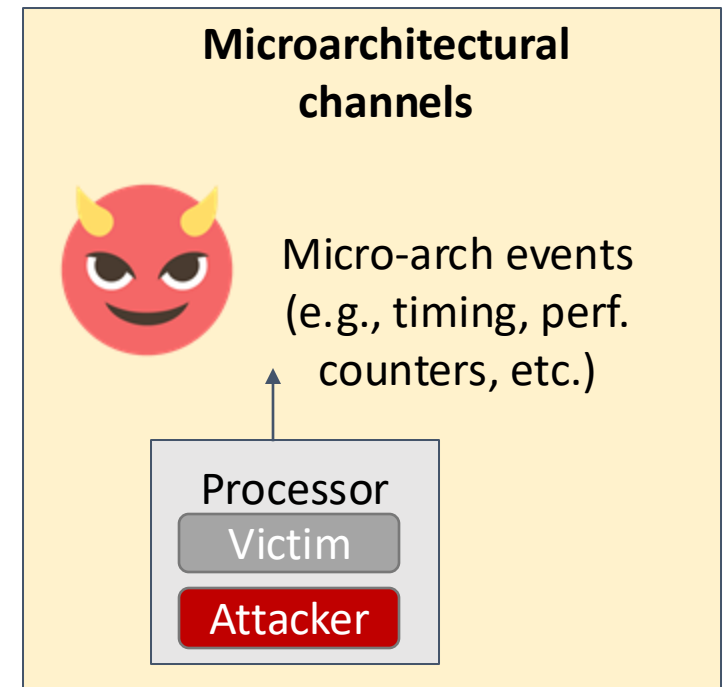
A Rough Classification based on What Attackers Can Observe



Attacker requires measurement equipment → physical access



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

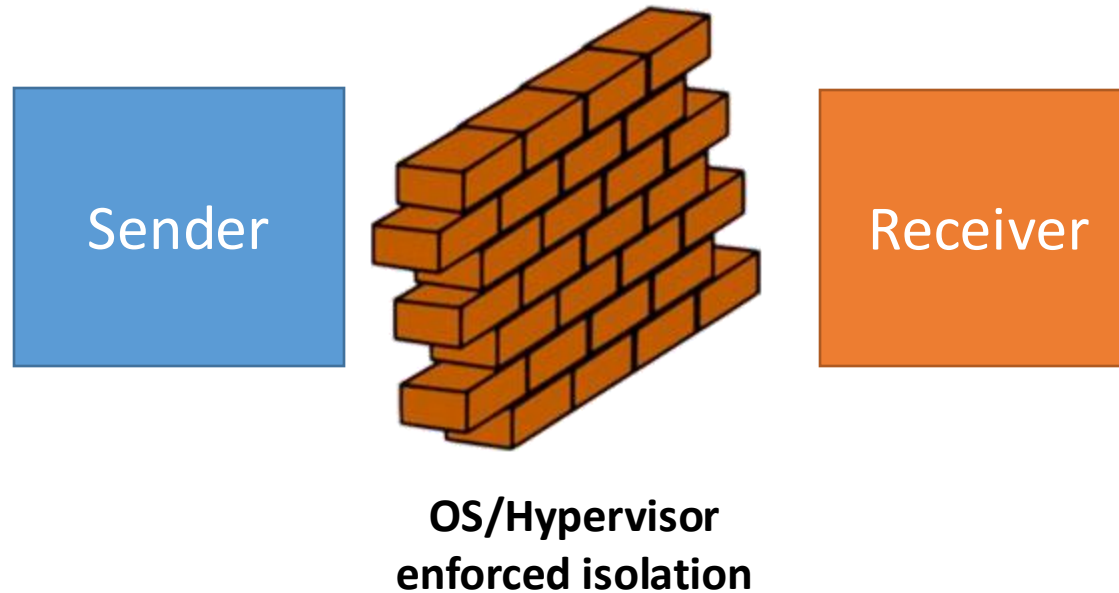


Attacker may be remote, or be co-located

Microarchitecture (uArch) Side Channel



Side Channel Threat Model



File, Socket, Pipe, Shared memory (shm in Linux) ...

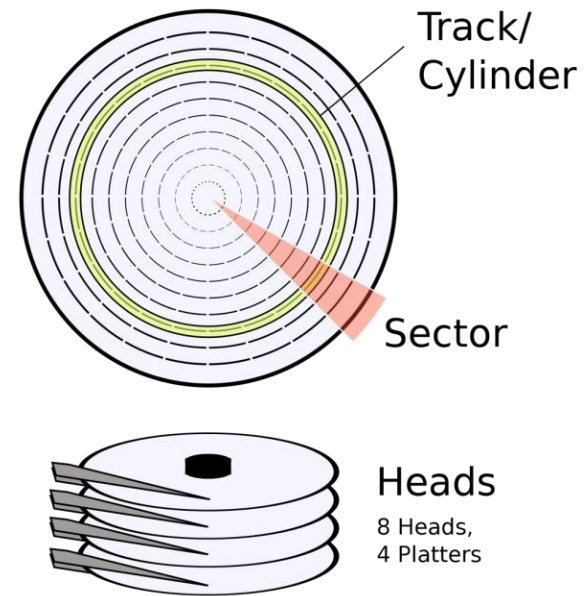
An Example Attack in 1977

- Disk arm optimization
 - Execute requests by the "elevator algorithm."
- Assume attacker's capability:
 - Can issue multiple requests to any tracks
 - Can measure request latency and the order of its own requests get processed by the disk
- Let's attack

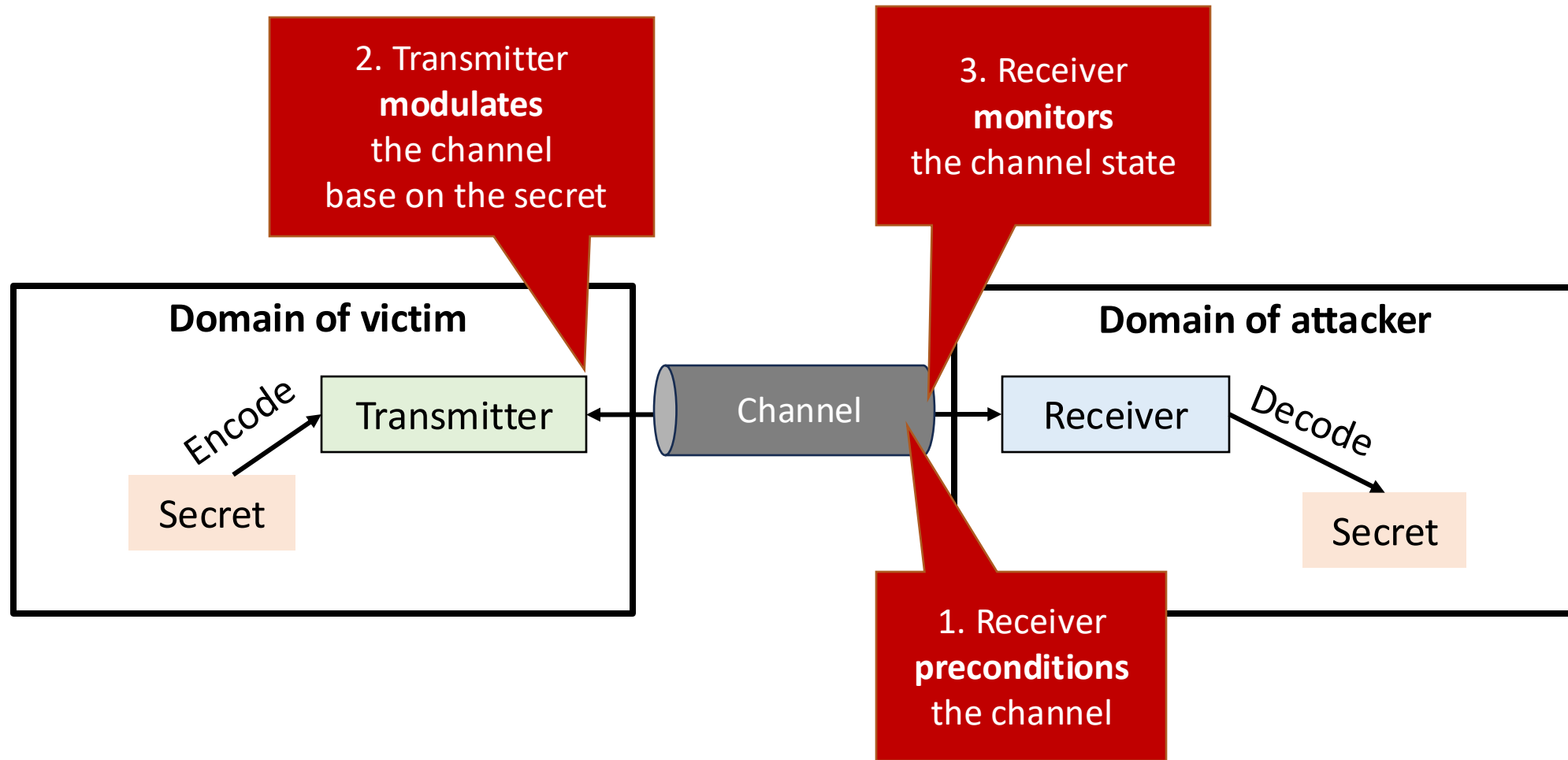


Victim Code:

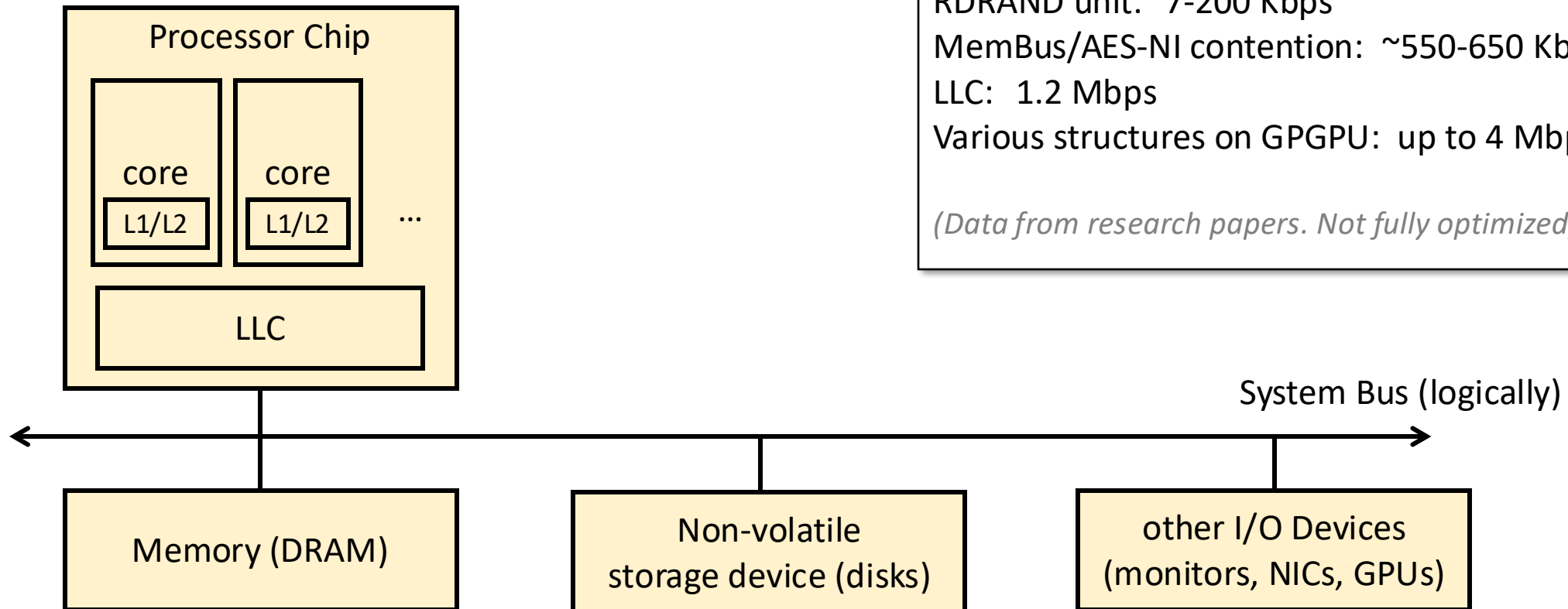
```
if (secret_bit==1)
    access track 2;
else
    access track 8;
```



A Communication Model



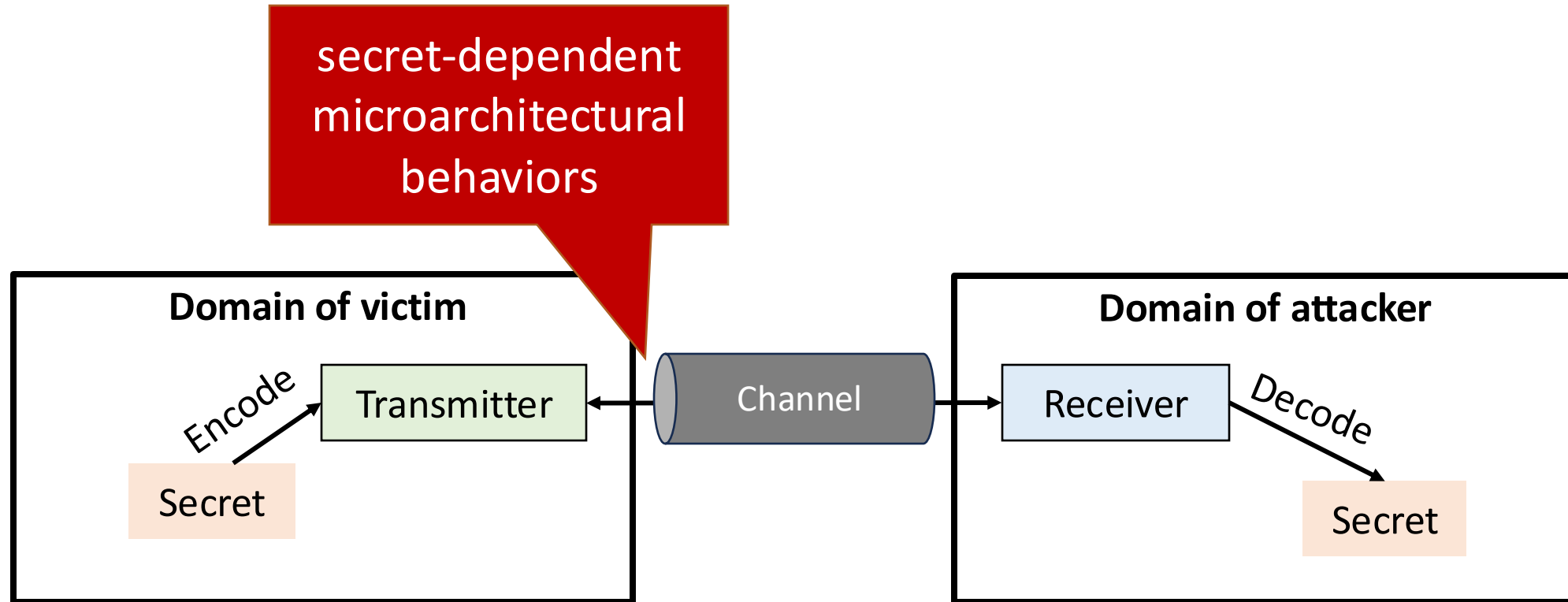
uArch Attacks Generalization



RDRAND unit: 7-200 Kbps
MemBus/AES-NI contention: ~550-650 Kbps
LLC: 1.2 Mbps
Various structures on GPGPU: up to 4 Mbps

(Data from research papers. Not fully optimized)

A Communication Model



Leak Crypto Library #1: RSA

- Square-and-Multiply Exponentiation

Input :

base b

modulo m

exponent $e = (e_{n-1} \dots e_0)_2$

Output:

$b^e \bmod m$

When $b = 3, e = 2^{256}$

How many multiplications do you
need to do to compute:

b^e

Naïve: 2^{256}

Repeated Squaring: **256**

Leak Crypto Library #1: RSA

- Square-and-Multiply Exponentiation

Input :

base b

modulo m

exponent $e = (e_{n-1} \dots e_0)_2$

Output:

$b^e \bmod m$

```
r = 1
for i = n-1 to 0 do
    r = sqr(r)
    r = mod(r, m)
    if ei == 1 then
        r = mul(r, b)
        r = mod(r, m)
    end
end
end
```

Leak Crypto Library #2: AES

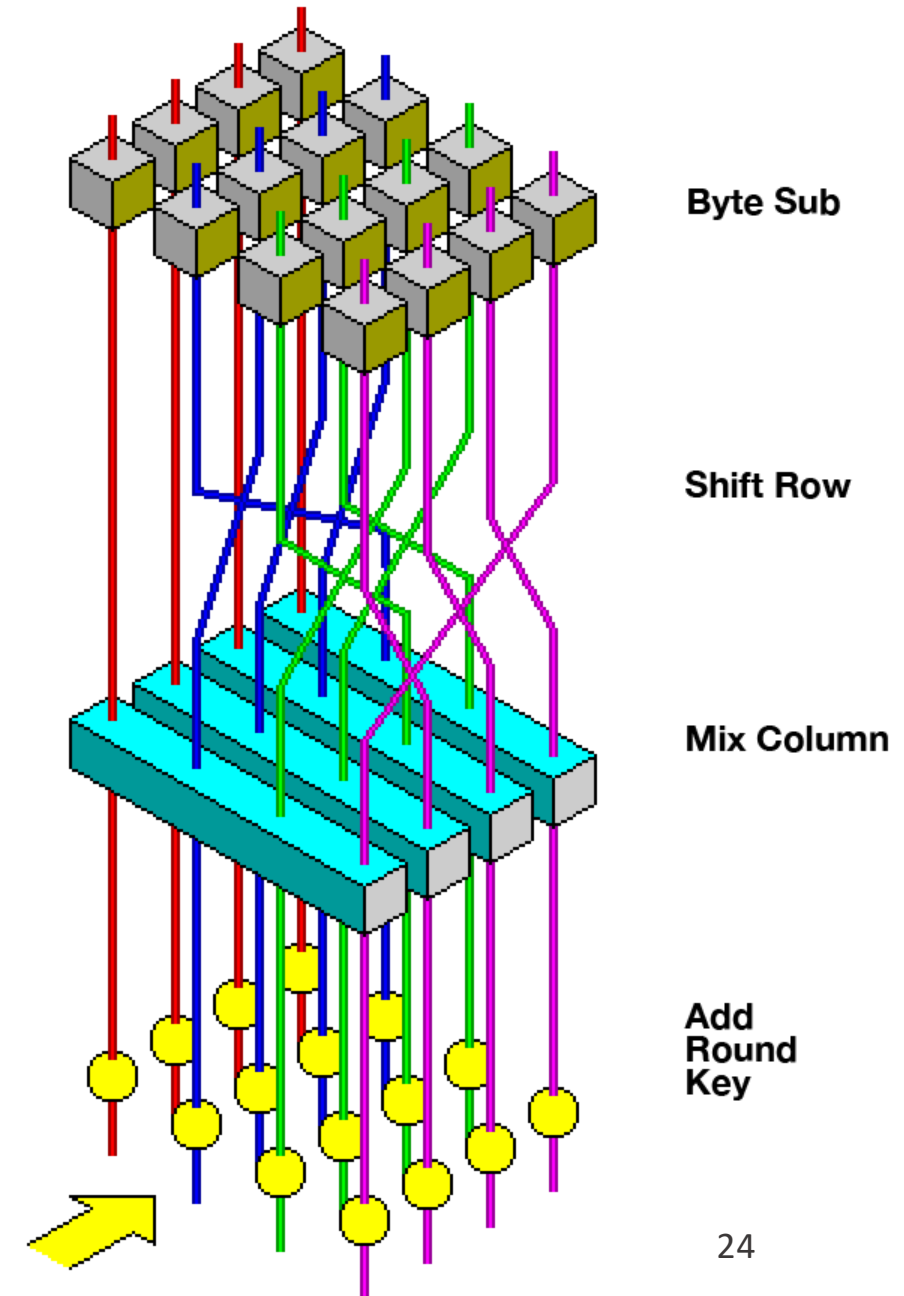
Input :

Plaintext
Secret key
Lookup Tables

Output:

Ciphertext

*Repeat 10, 12, 14 times
depending on key size.*



Leak Crypto Library #2: AES T-Table

```
// Td0, Td1, Td2, Td3 are 4 lookup tables
// s0..s3, t0..t3 are 32-bit integer
for ( ; ; ) {
    t0 = Td0[(s0>>24)] ^ Td1[(s3>>16)&0xff] ^ Td2[(s2>>8)&0xff] ^ Td3[s1&0xff] ^ rk[4];
    t1 = Td0[(s1>>24)] ^ Td1[(s0>>16)&0xff] ^ Td2[(s3>>8)&0xff] ^ Td3[s2&0xff] ^ rk[5];
    t2 = Td0[(s2>>24)] ^ Td1[(s1>>16)&0xff] ^ Td2[(s0>>8)&0xff] ^ Td3[s3&0xff] ^ rk[6];
    t3 = Td0[(s3>>24)] ^ Td1[(s2>>16)&0xff] ^ Td2[(s1>>8)&0xff] ^ Td3[s0&0xff] ^ rk[7];

    rk += 8;
    if (--r == 0) {
        break;
    }
    ...
}
```

Observation: Secret-dependent memory accesses.

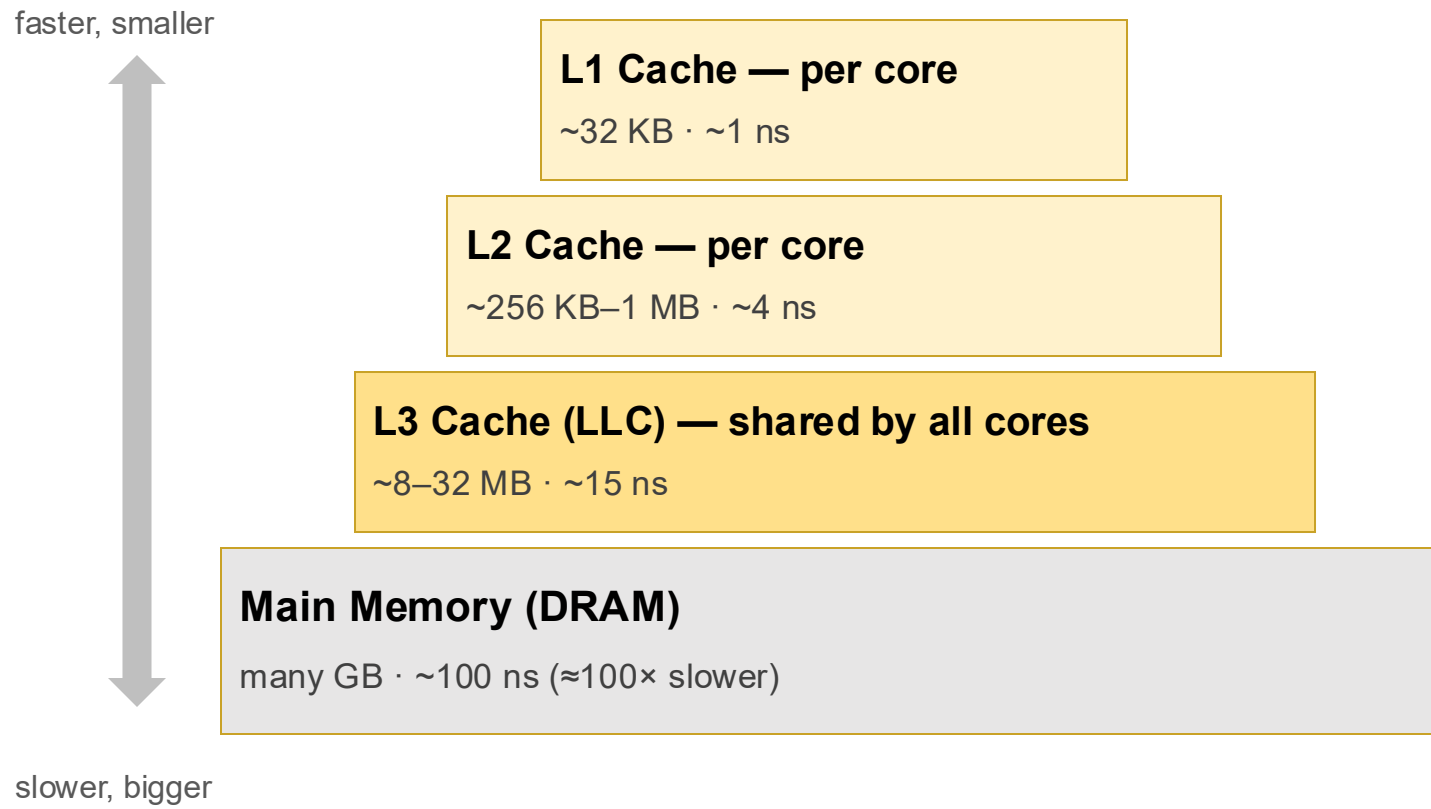
The attacker's goal:

Monitor access patterns at **cache line granularity.**



What is Caching?

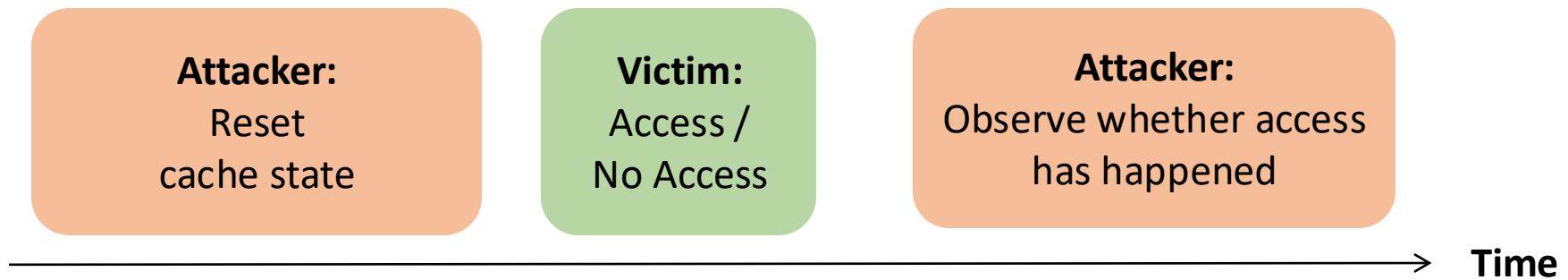
Cache is a small, fast memory that keeps recently-used data close to the CPU.



Why Cache?

- Large attack surface. Shared across cores/sockets.
- Fast. Can be used to build high-bandwidth channels
- Many states. Can encode secrets spatially to further improve bandwidth and precision.
- There exist many cache-like structures. The same attack concepts and tricks will apply.

Cache Attack Plan

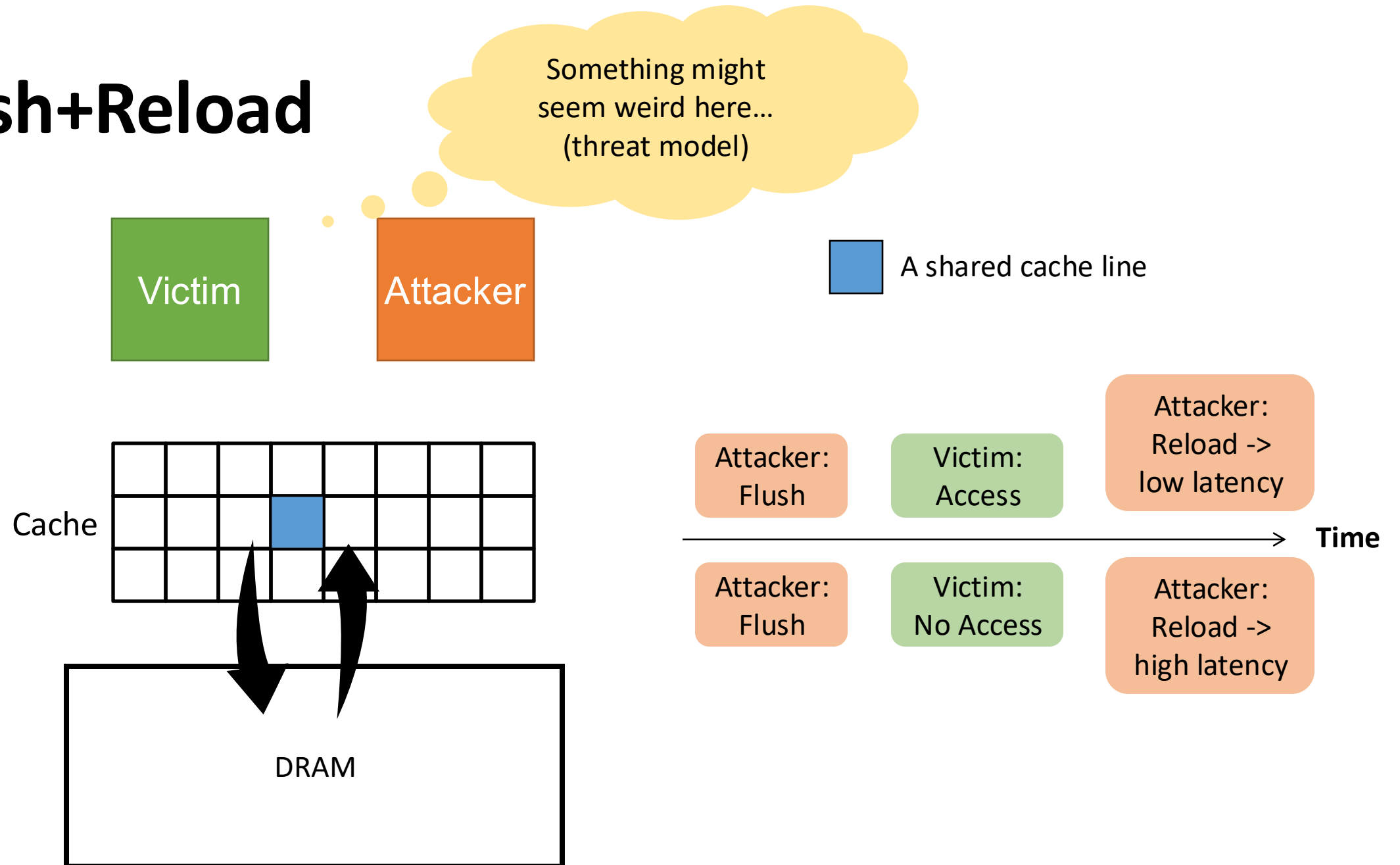


- ① How to reset cache state?
- ② How to monitor cache state?

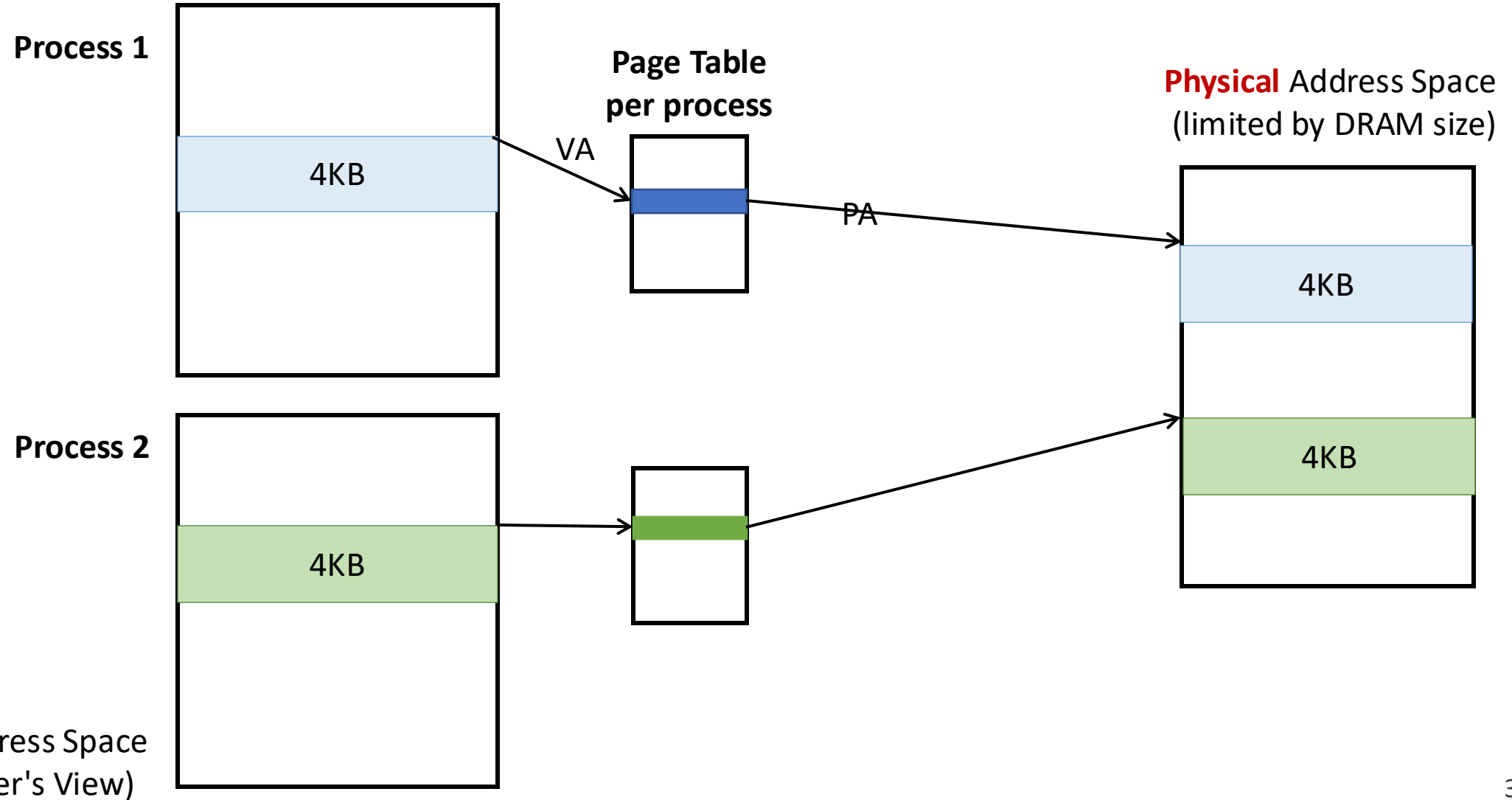
Attack Strategy #1: Flush+Reload

- The flush instructions allow explicit control of cache states
 - In X86, `clflush vaddr`
 - In ARM, `DC CIVAC vaddr`
- What are these flush instructions used for except for attacks?
 - For coherence, in the case when the data in the cache is inconsistent with the data in the DRAM.
 - 1) old time, incoherent DMA
 - 2) nowadays, Non-volatile memory for crash recovery

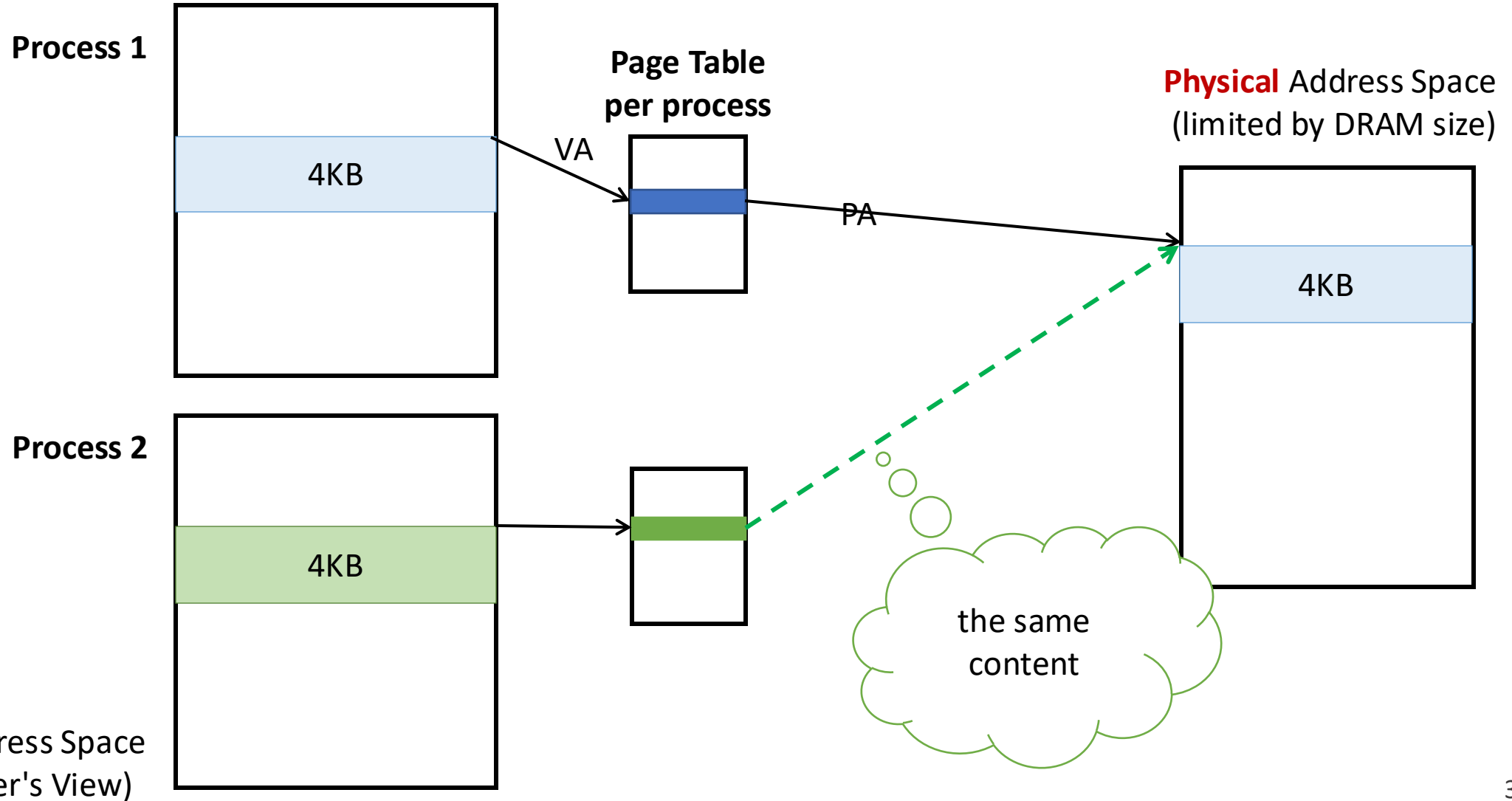
Flush+Reload



Page Mapping



Transparent Page Deduplication



The Attack Monitor Code

```
x1 = read cycle;
```

```
perform the load;
```

```
x2 = read cycle;
```

```
latency = x2 - x1;
```

The Attack Monitor Code

```
mfence
```

```
rdtsc
```

```
mov %eax, %edi
```

```
mov (<vaddr>), %rsi
```

```
mfence
```

```
rdtsc
```

```
sub %edi, %eax
```

In x86, 8 GPR:

- rax, rbx, rcx, rdx
- rsp, rbp
- rsi, rdi

“r” means 64-bit

replacing “r” with “e” means the lower 32 bits.

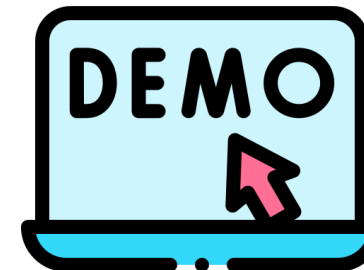
rdtsc:

- Read Time-Stamp Counter
- **edx:eax** := TimeStampCounter;

mfence:

- Memory Fence
- Performs a serializing operation on all memory instructions

A Demo



Sender

Access one element



Figure out which element
has been accessed



Receiver



Lessons Learnt So Far

So the fundamental problem:
shared memory between
different security domains.

Source: <https://kb.vmware.com/s/article/2080735>

Security considerations and disallowing inter-Virtual Machine Transparent Page Sharing (2080735)

Last Updated: 8/25/2021

Categories: Informational

Total Views: 66593



5

Language: 英文



SUBSCRIBE



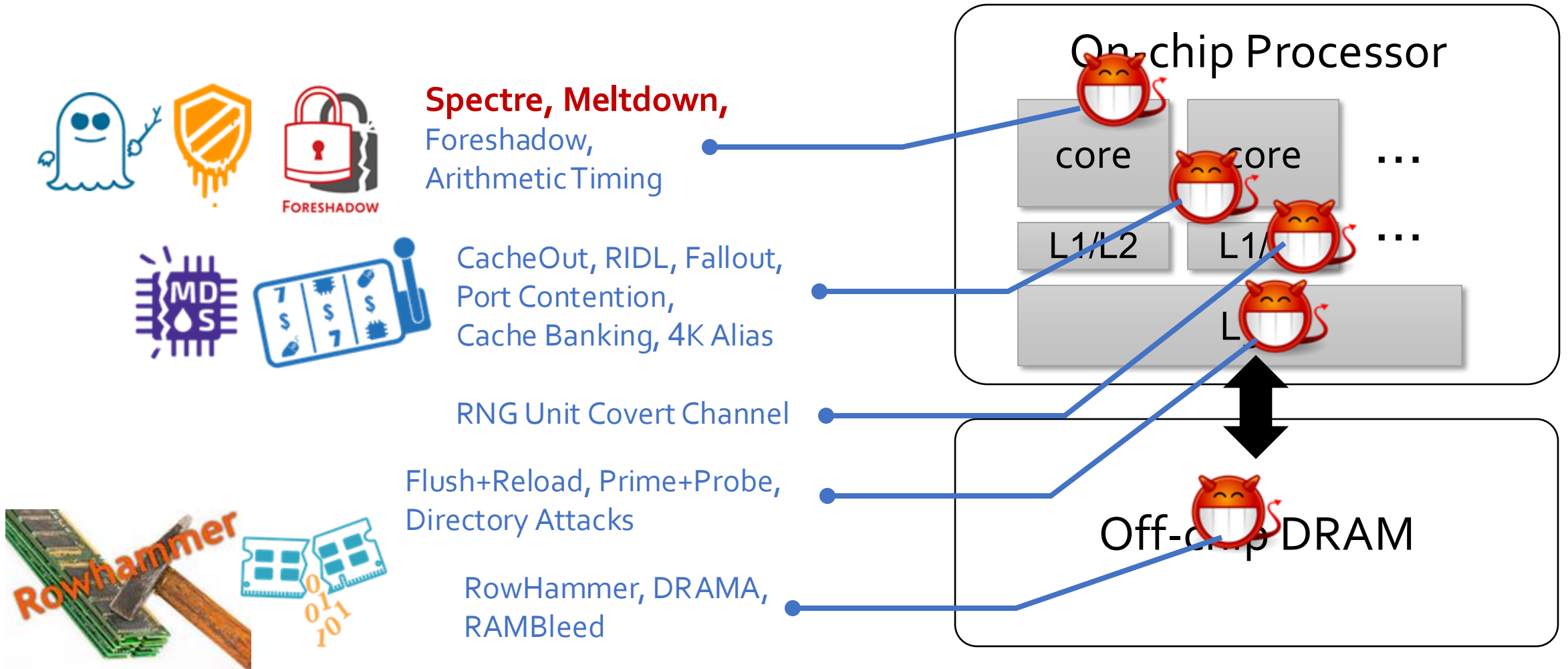
Details

This article acknowledges the recent academic research that leverages Transparent Page Sharing (TPS) to gain unauthorized access to data under certain highly controlled conditions and documents VMware's precautionary measure of restricting TPS to individual virtual machines by default in upcoming ESXi releases. At this time, VMware believes that the published information disclosure due to TPS between virtual machines is impractical in a real world deployment.

Published academic papers have demonstrated that by forcing a flush and reload of cache memory, it is possible to measure memory timings to try and determine an AES encryption key in use on another virtual machine running on the same physical processor of the host server if Transparent Page Sharing is enabled between the two virtual machines. This technique works only in a highly controlled system configured in a non-standard way that VMware believes would not be recreated in a production environment. .

Even though VMware believes information being disclosed in real world conditions is unrealistic, out of an abundance of caution **upcoming ESXi Update releases will no longer enable TPS between Virtual Machines by default** (TPS will still be utilized within individual VMs).

There are many more side-channels possible



Takeaways

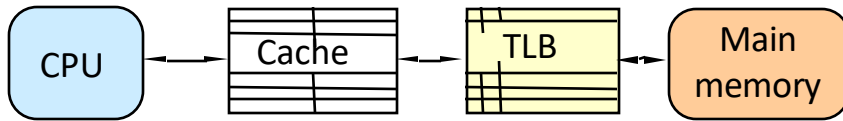
- **Practical** challenges in implementing a reliable cache attack
 - Page sharing
 - Noise due to prefetchers
 - Uncertainty due to page mapping
 - Replacement policy
 - etc.
- **Hardware and software optimizations** make attacks easier
 - Transparent page sharing
 - Copy-on-write
 - Huge pages
 - Virtually-indexed and physically-tagged caches

Questions/Comments?
viniul@mit.edu



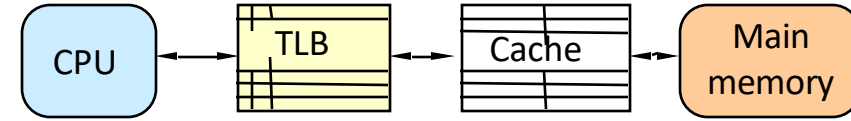
Using Caches with Virtual Memory

Virtually-Addressed Cache



- FAST: No virtual → physical translation on cache hits
- Problem: Must flush cache after context switch

Physically-Addressed Cache



- Avoids stale cache data after context switch
- SLOW: virtual → physical translation before every cache access

Best of Both Worlds (L1 Cache): Virtually-Indexed, Physically-Tagged Cache (VIPT)

