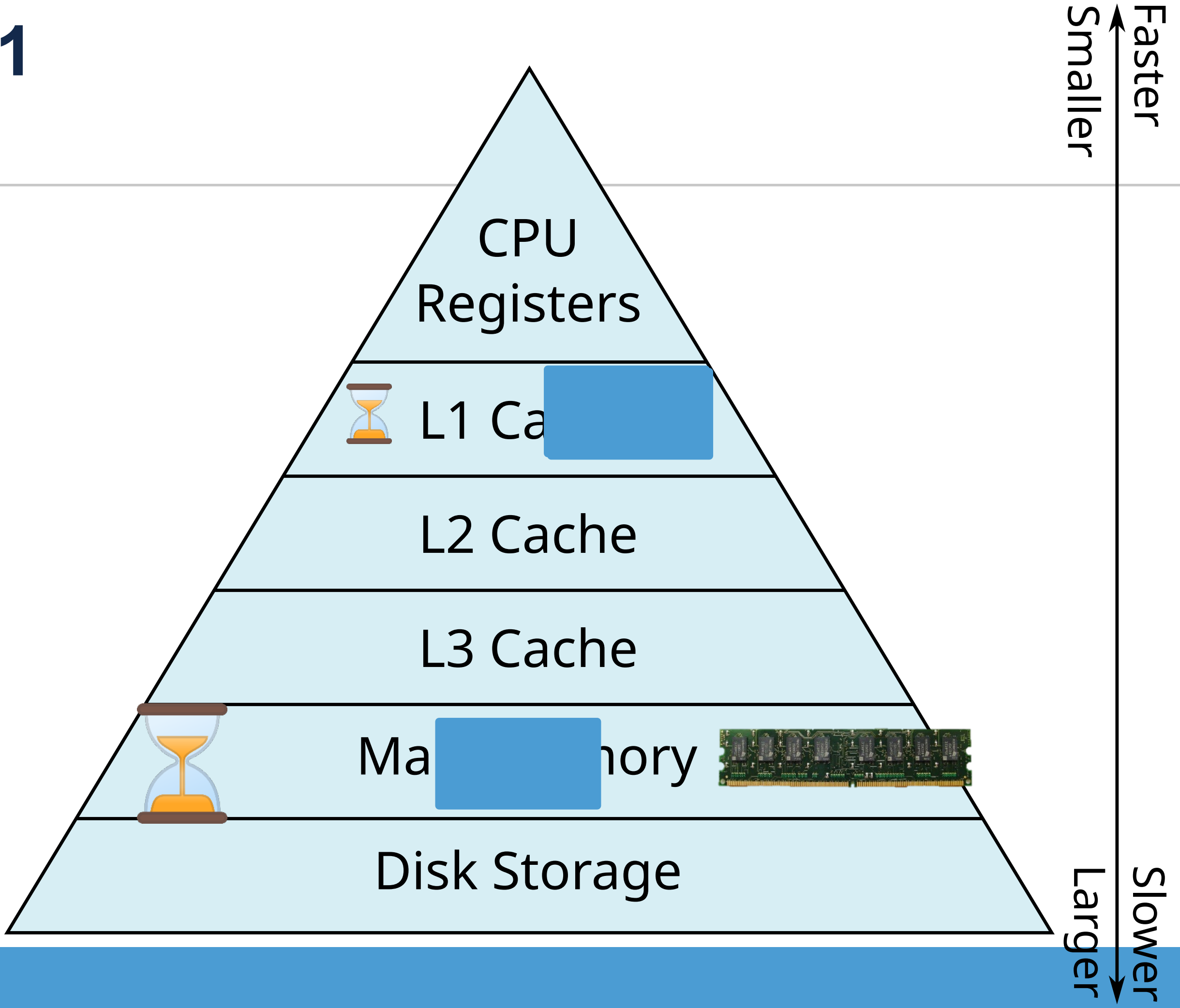
Comp 590-184: Hardware Security and Side-Channels

Lecture 4: Practical Cache Attacks

January 20, 2026
Andrew Kwong

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

Caching 101


Faster
Smaller
CPU
Registers
L1 Ca
L2 Cache
L3 Cache
Ma
ory
Disk Storage
Slower
Larger

Why Cache?

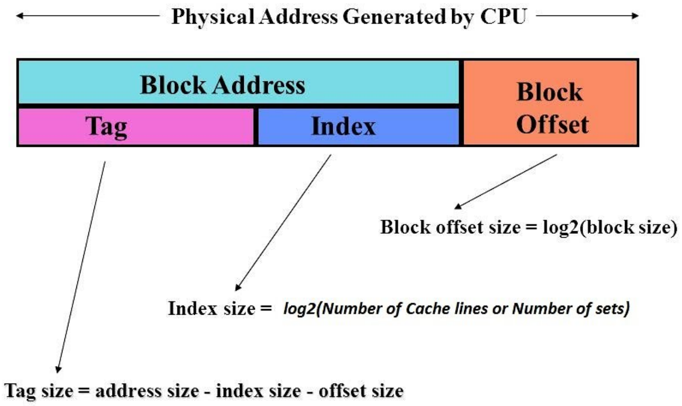
- Large attack surface. Shared across cores/sockets.
- Fast. Can be used to build high-bandwidth channels
- Many states. Can encode secrets spatially to further improve bandwidth and precision.
- There exist many cache-like structures. The same attack concepts and tricks will apply.

The Goal:
Monitor access patterns at cache set/line granularity

Goal

- Attacker wants to learn which *cache sets/lines the victim* accessed

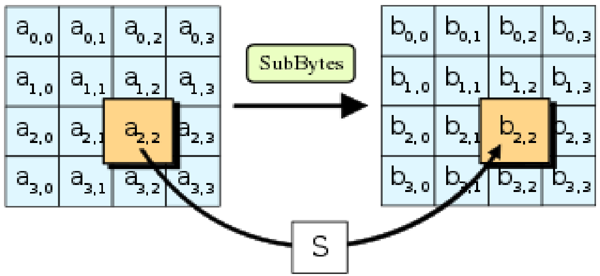
int secret=0xDEADBEEF
*int *ptr=&secret;*
*int x=*ptr;*

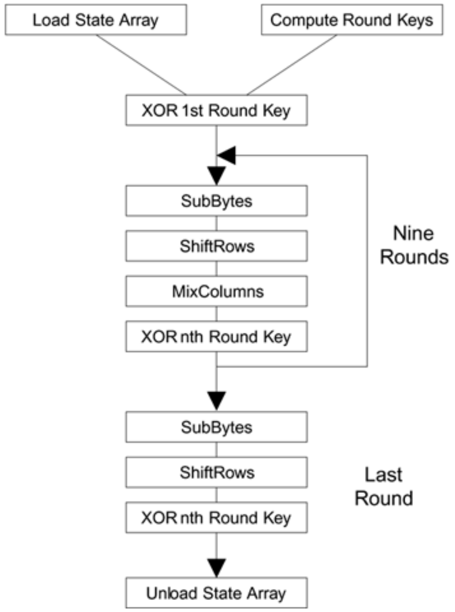


- Can reveal sensitive information

Leaking Crypto Algorithm : AES

- AES implementations can use table lookups
  - S-Box substitutions
  - T-tables





Leaking Crypto Algorithm: RSA

- Square-and-Multiply Exponentiation

Input :

base b

modulo m

exponent $d = (d_{n-1} \ldots d_0)_2$

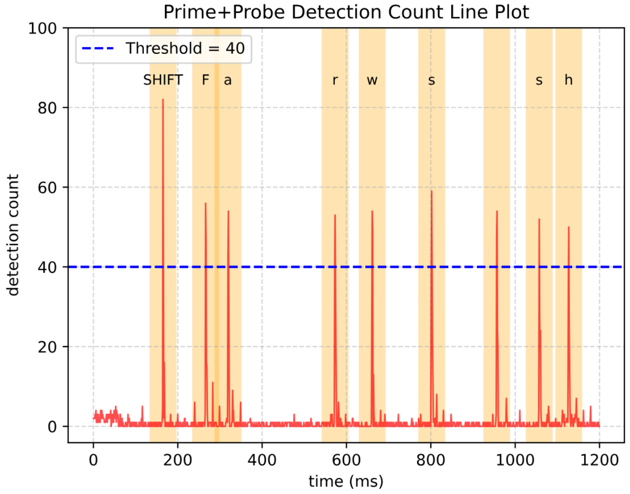
Output:

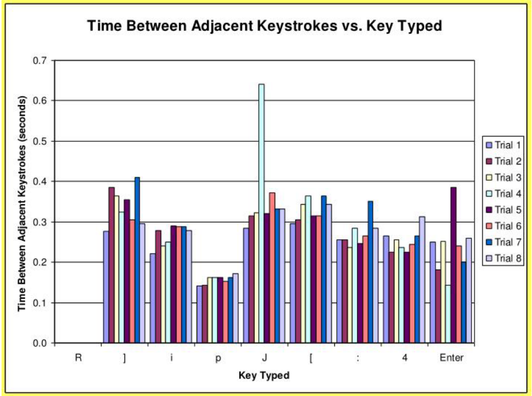
$b^d \bmod m$

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

Keystroke Extraction

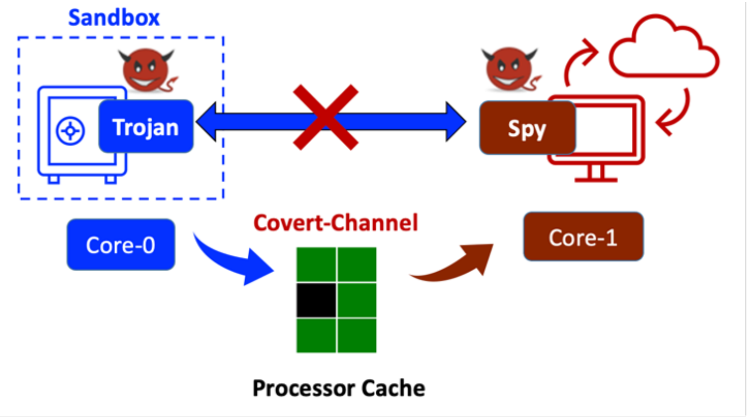
- Keystroke cadence yields keystroke extraction
- Monitor cacheline that registers keystrokes
  - Victim will access that cacheline every time he types a character





Covert Channel

- Two processes can communicate over cache covert channel
- Useful for Spectre!

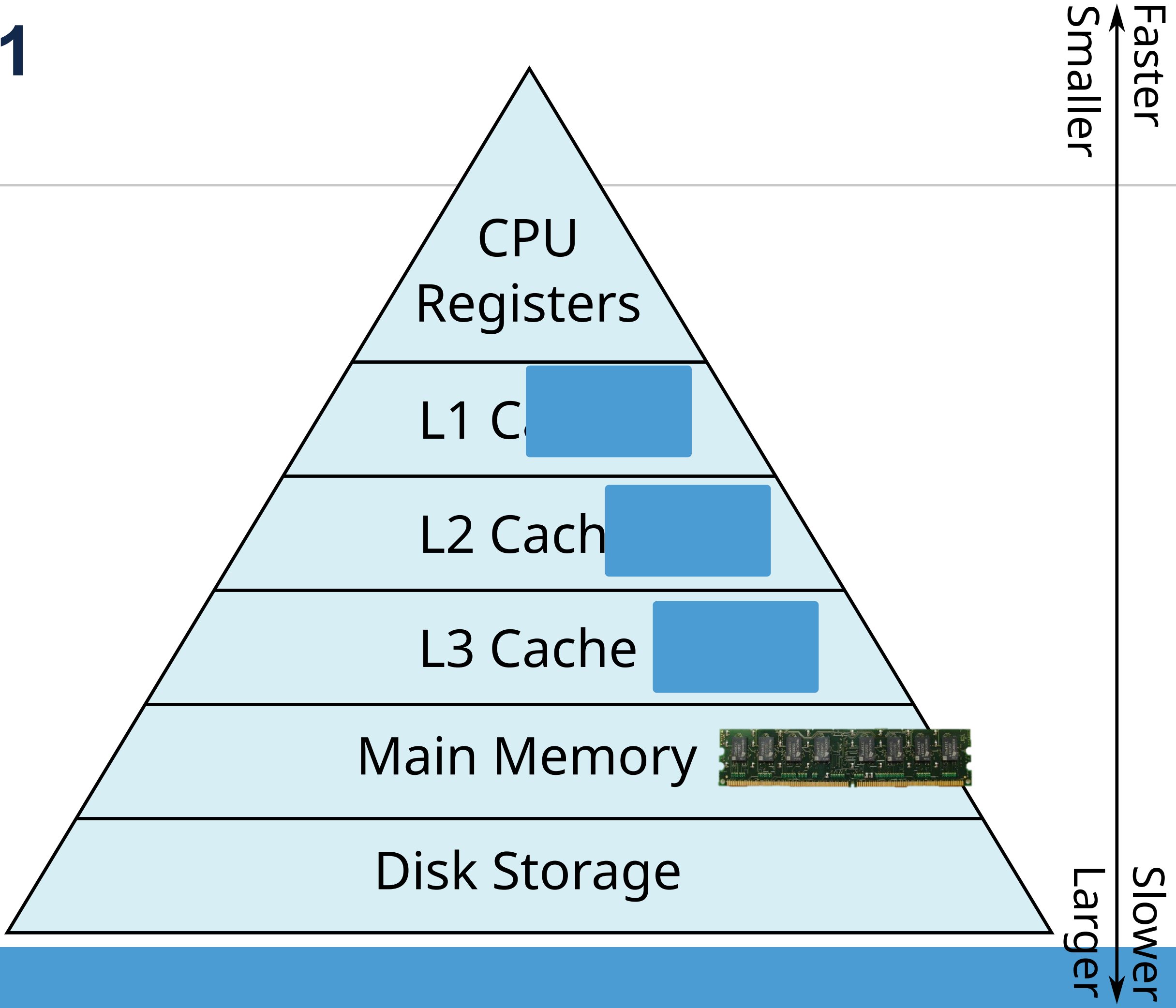


How can attacker monitor cacheline usage?

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.

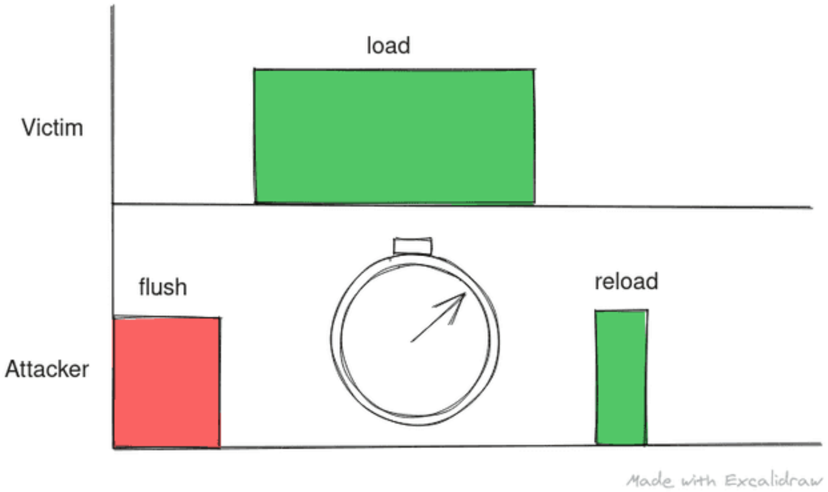
Caching 101



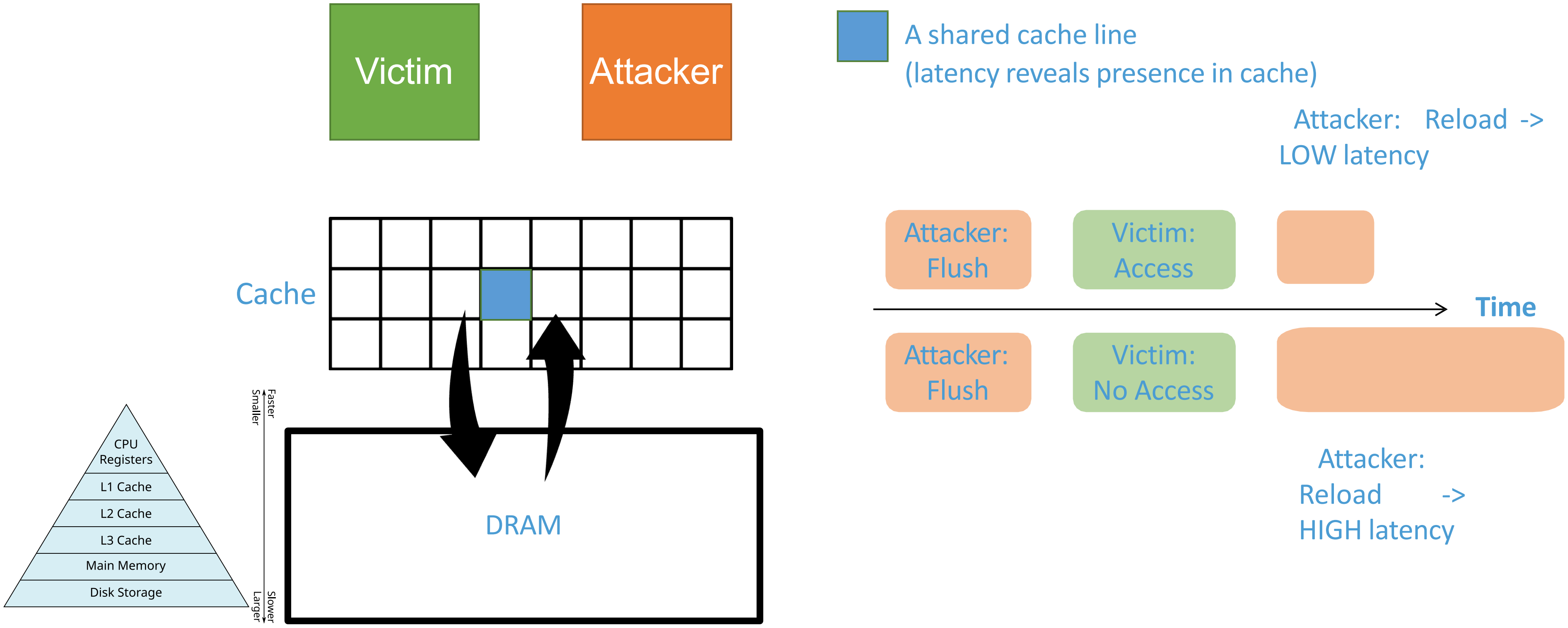
Faster
Smaller
CPU Registers
L1 C
L2 Cach
L3 Cache
Main Memory
Disk Storage
Slower
Larger

Flush+Reload steps

1. Attacker flushes shared memory
2. Attacker waits for victim to access (or not) the memory
3. Attacker reloads the cache line
4. Latency reveals whether the victim accessed that cache line



Flush+Reload



Flush+Reload

Cache Lines

Attacker

Victim

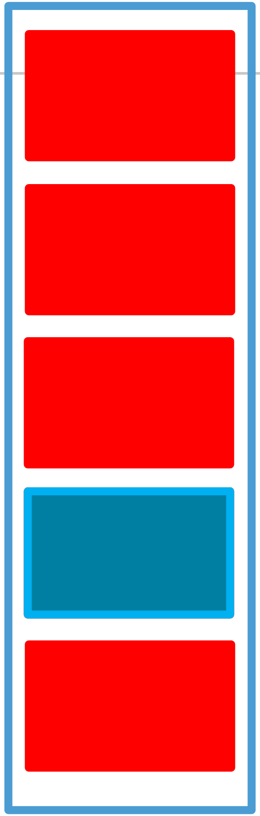
Reload (slow)

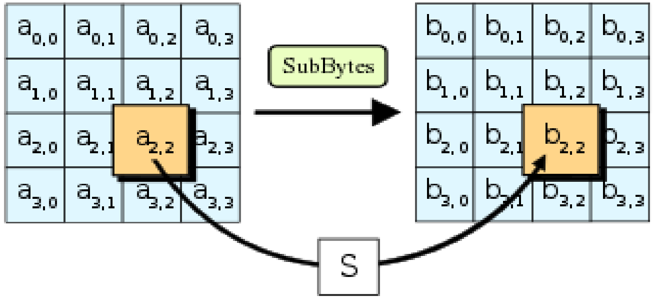
Reload (slow)

Reload (slow)

Reload (fast)

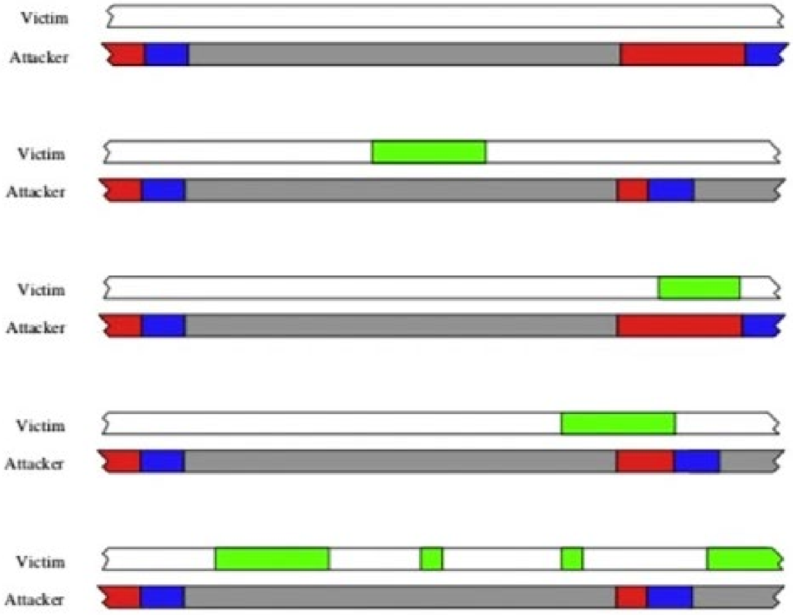
Reload (slow)

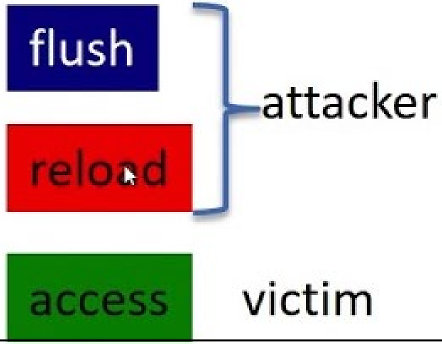




Attacker learns secret byte=3

Some possible outcomes

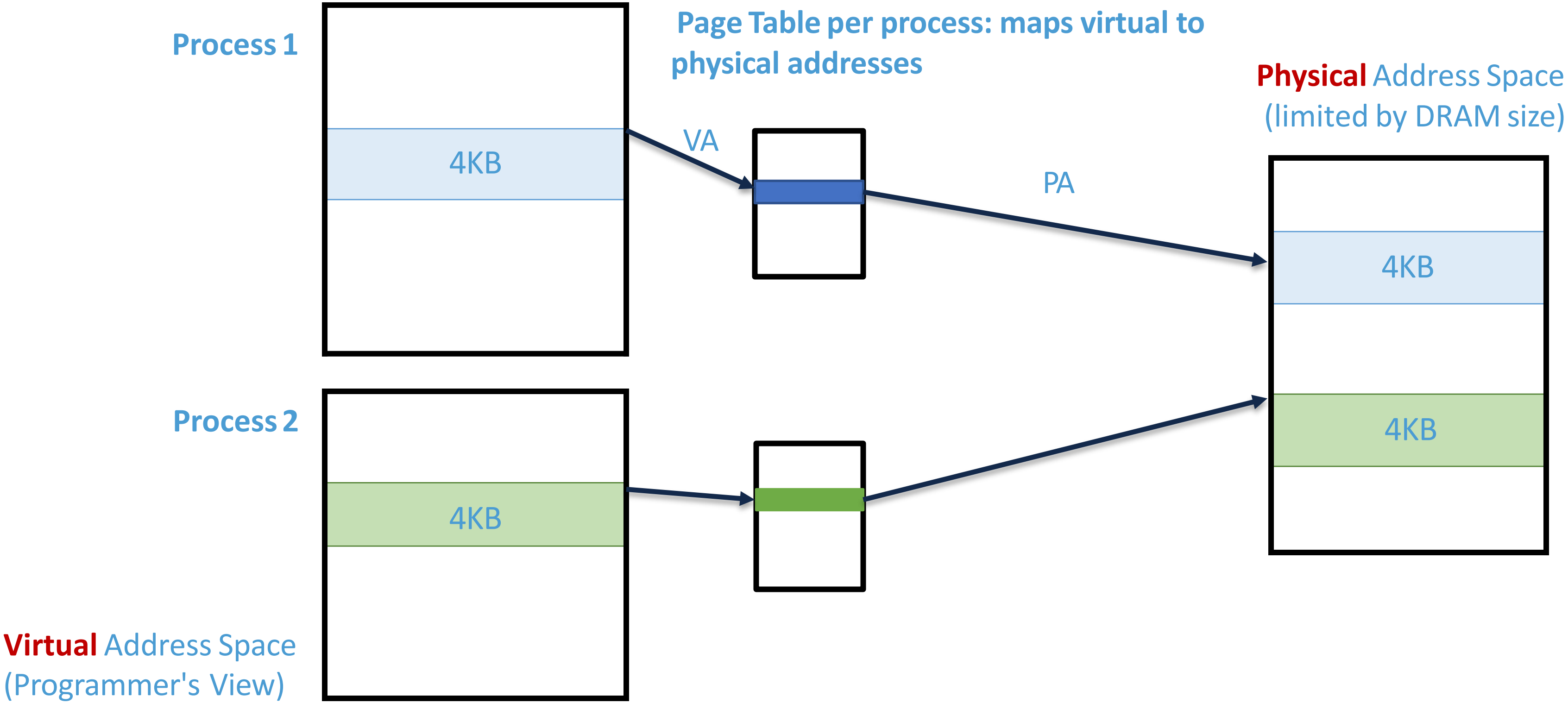




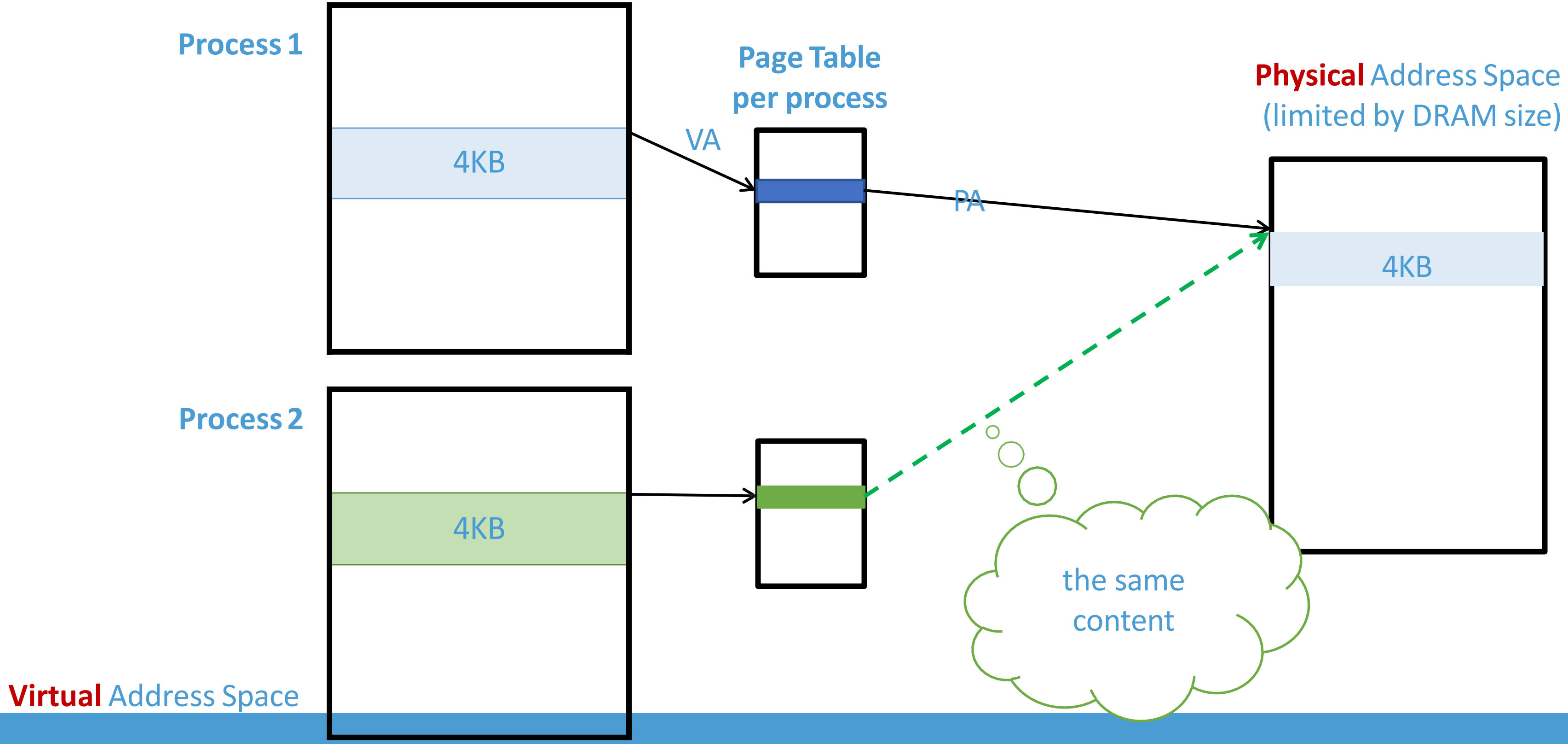

```
r = 1
for i = n-1 to 0 do
        r = sqr(r)
        r     = mod(r, m)

        if d_i == 1 then
                r = mul(r, b)
            r = mod(r, m)
        end
end
```

Shared Memory in Practice

Page Mapping



Transparent Page Deduplication



Attack Strategy #2: ?

- Cache state manipulation instructions
  - In X86, `clflush vaddr`
  - In ARM, `DC CIVAC vaddr`

- What if these instructions are not available in user space?
  - Apple devices
  - *"Except ARMv8-A CPUs, ARM processors do not support a flush instruction"*
  - Flush instructions removed from Chrome's NaCL after Rowhammer

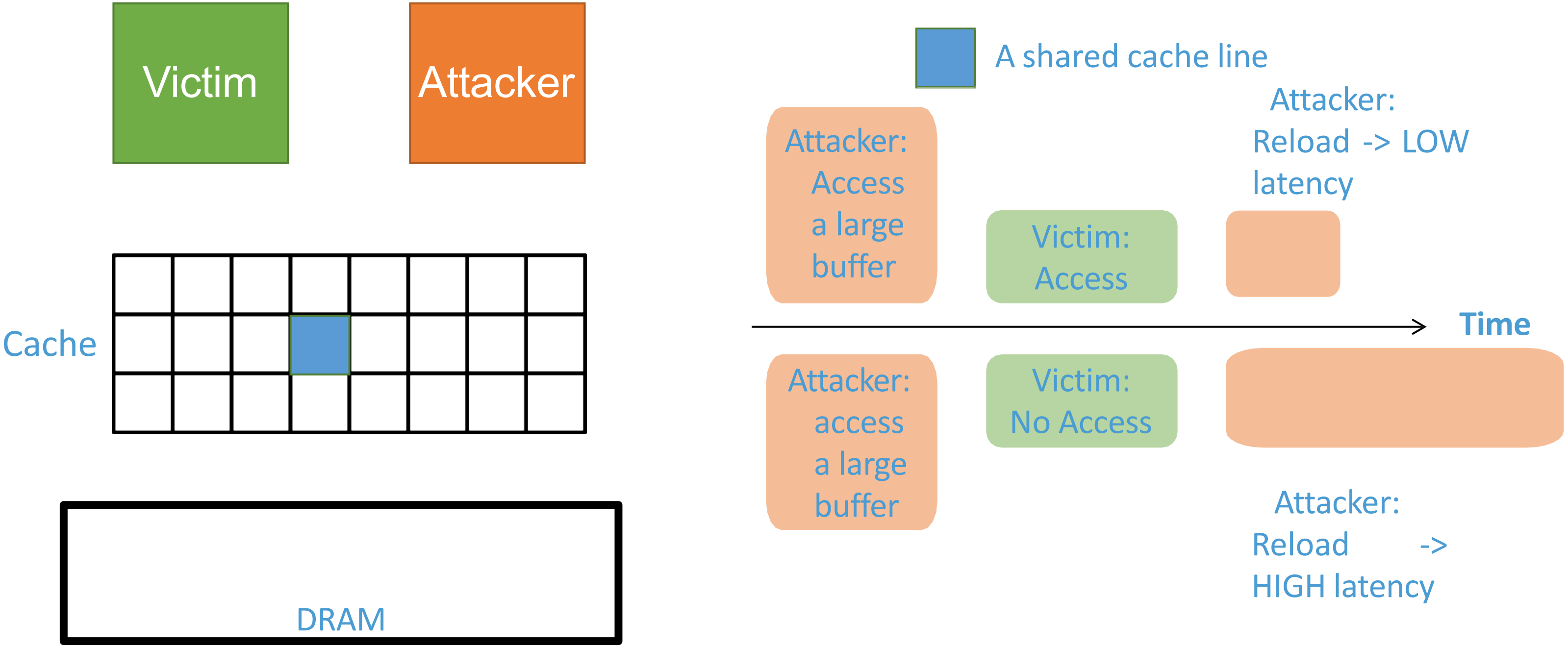
from ARMageddon: Cache Attacks on Mobile Devices (USENIX'16)

Attack Strategy #2: Evict+Reload

- If we can’ use a flush instruction, force eviction instead

1. Attacker accesses large amount of memory to evict shared memory
2. Attacker waits for victim to access (or not) the memory
3. Attacker reloads the cache line
4. Latency reveals whether the victim accessed that cache line

Attack Strategy #2: Evict+Reload



Lessons Learnt So Far

- flush+reload
    - Requires “flush” instruction
- Evict+reload
    - Doesn’t require “flush”

The fundamental problem: **shared memory** between different security domains.

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).

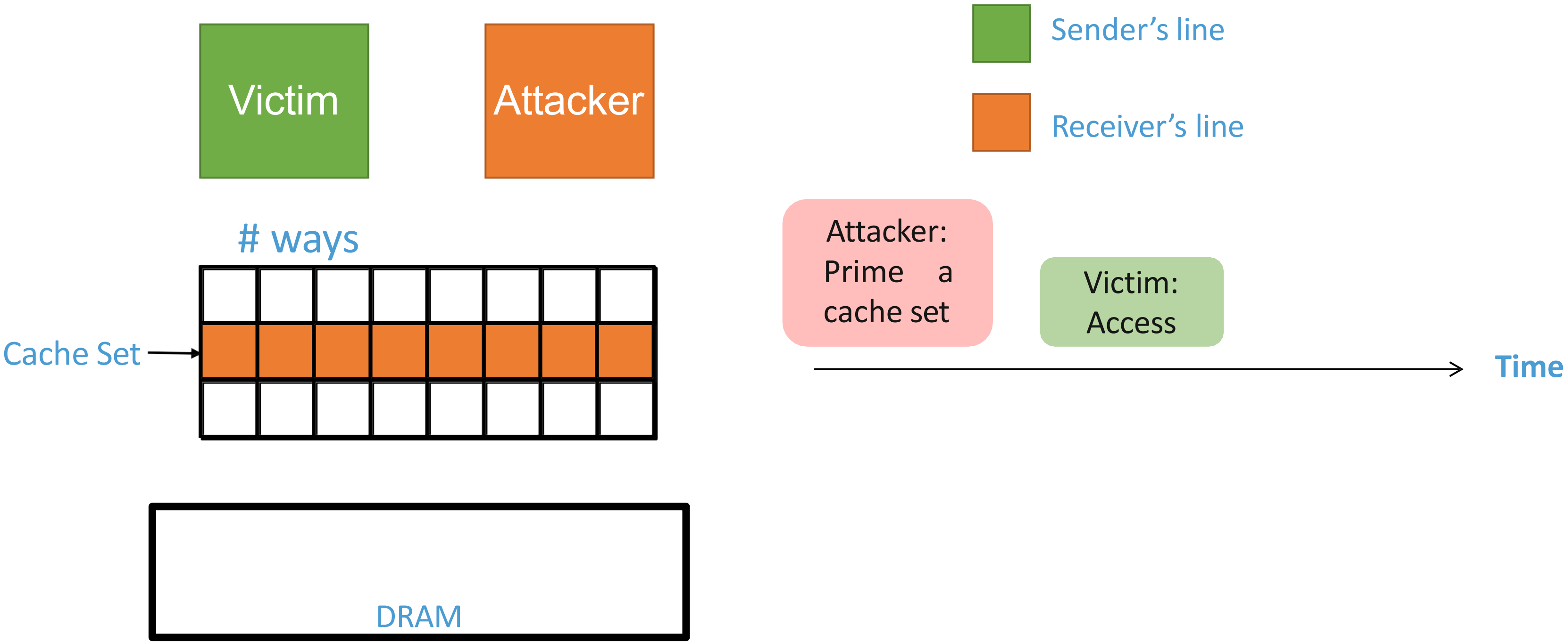
Source: https://kb.vmware.com/s/article/2080735

Attack Strategy #3: Prime+Probe

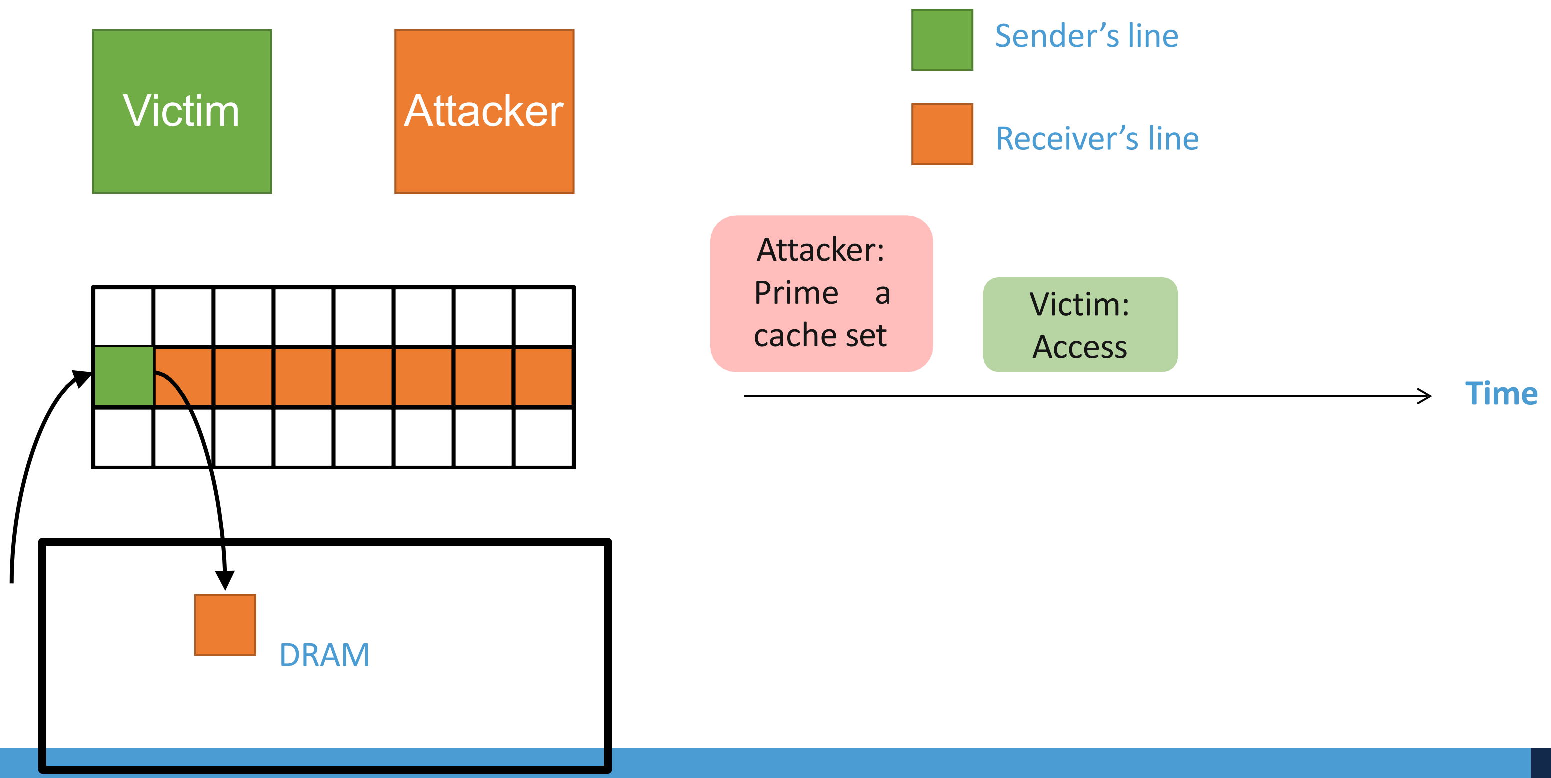
- Removes requirement of shared memory

1. Attacker “primes” a cache-set by accessing A elements in the cache-set
   - Called an eviction set
2. Attacker waits for victim to access (or not) the memory
3. Attacker reloads each cache line
4. Latency reveals whether the victim accessed that cache line

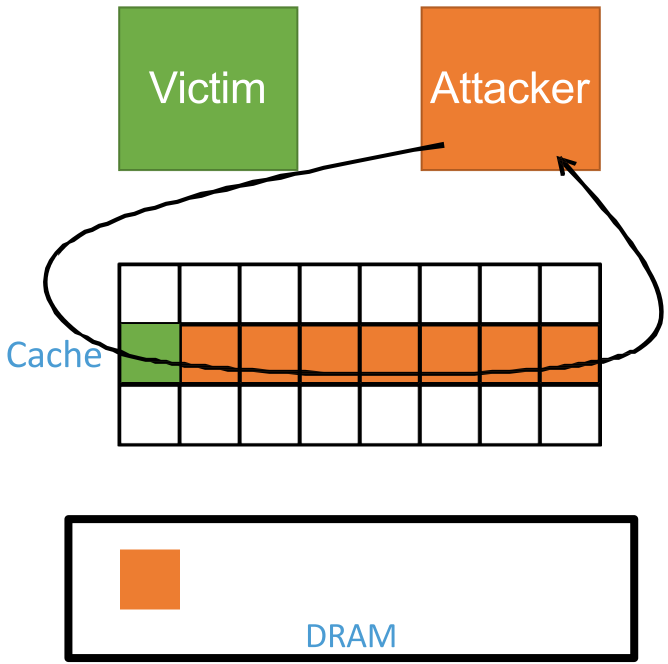
Attack Strategy #3: Prime+Probe

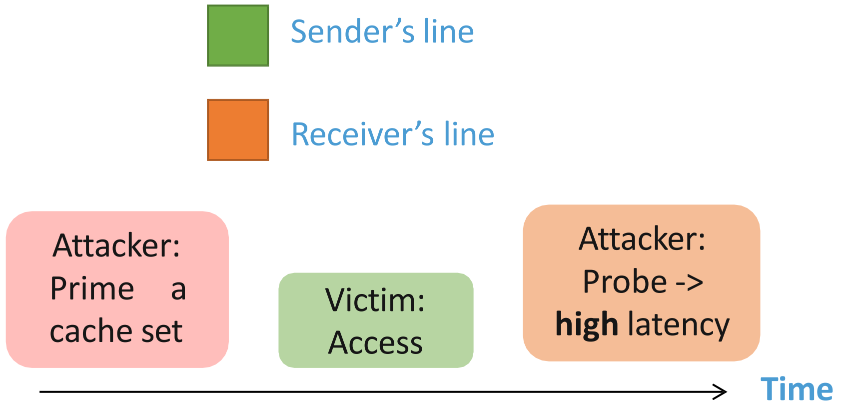


Attack Strategy #3: Prime+Probe

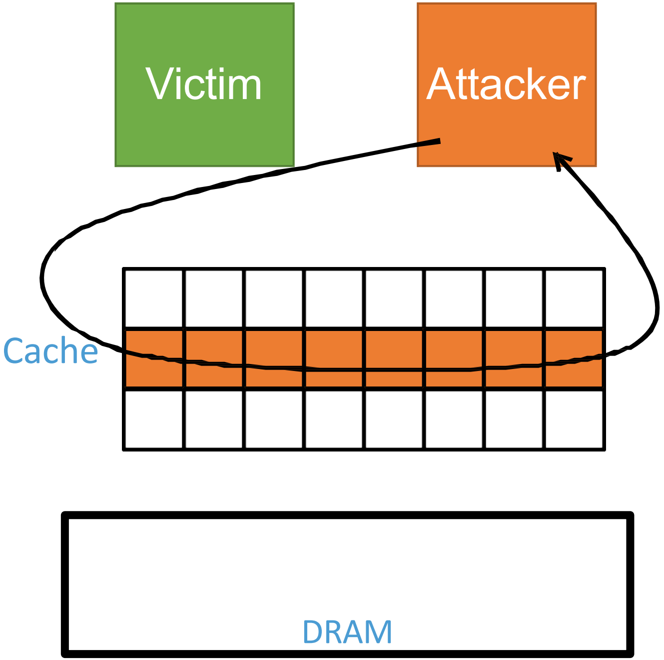


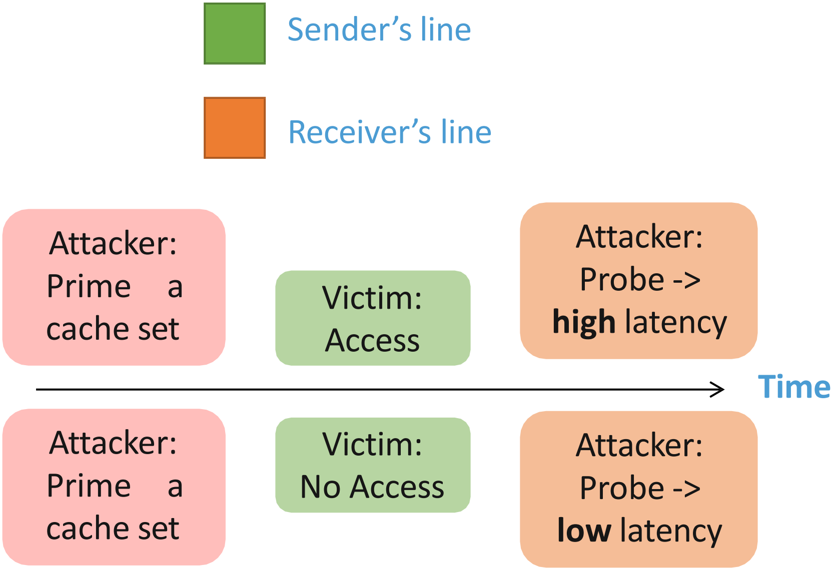
Attack Strategy #3: Prime+Probe





Attack Strategy #3: Prime+Probe





Timing Code

```
lfence
mfence
rdtsc
mov %eax, %edi
mov (<vaddr>), %rsi
lfence
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`:

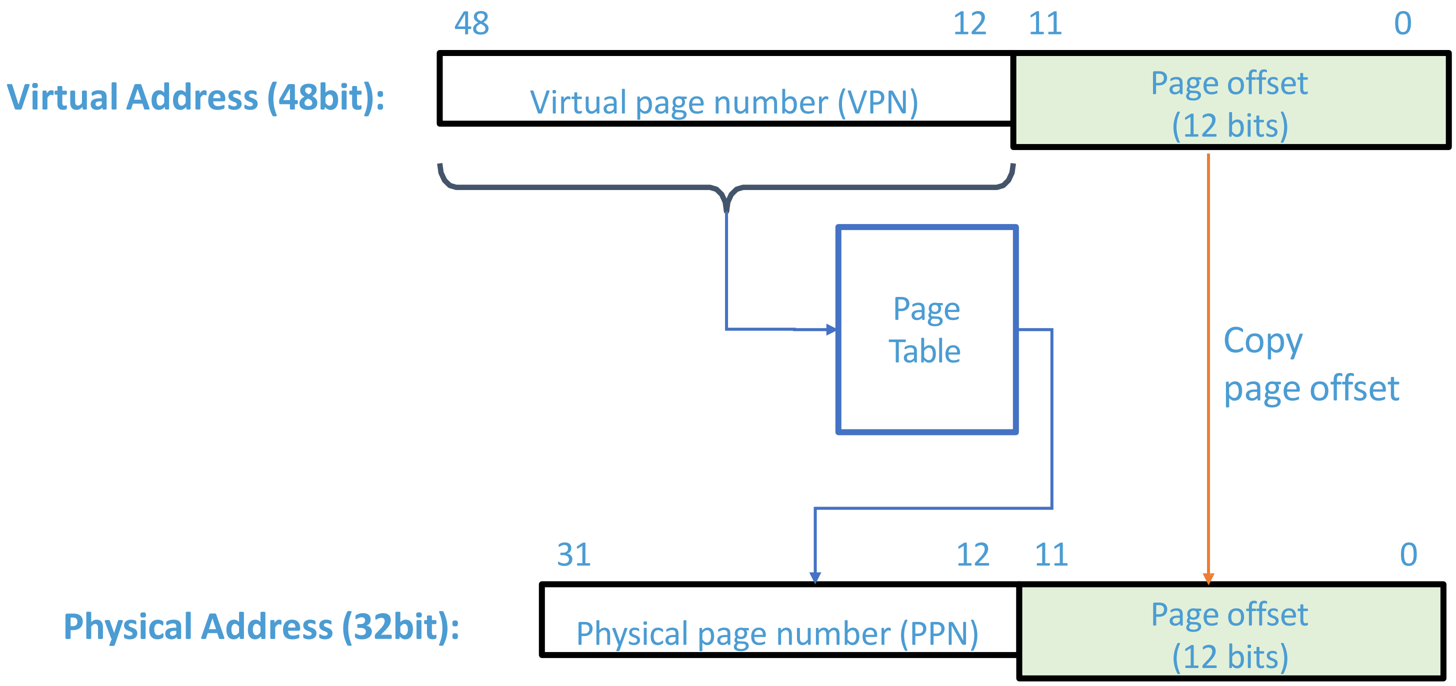
- Read Time-Stamp Counter
- `edx:eax` := TimeStampCounter;

`lfence`:

- Load Fence
- Performs a serializing operation on all load instructions

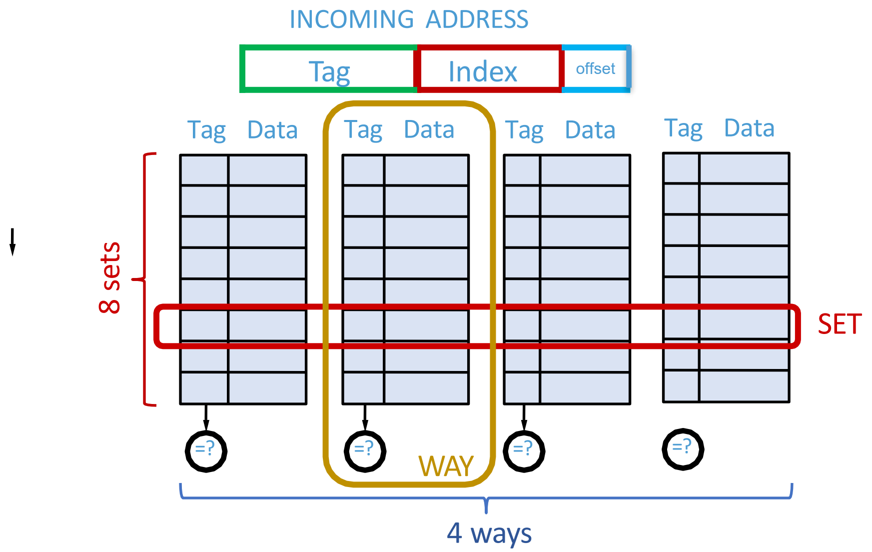
More Background

Address Translation (4KB page)



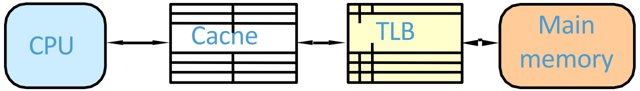
N-way Set-Associative Cache

- Does cache use virtual address or physical address?



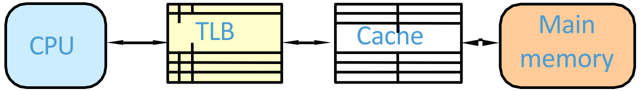
Using Caches with Virtual Memory

Virtually-Addressed Cache



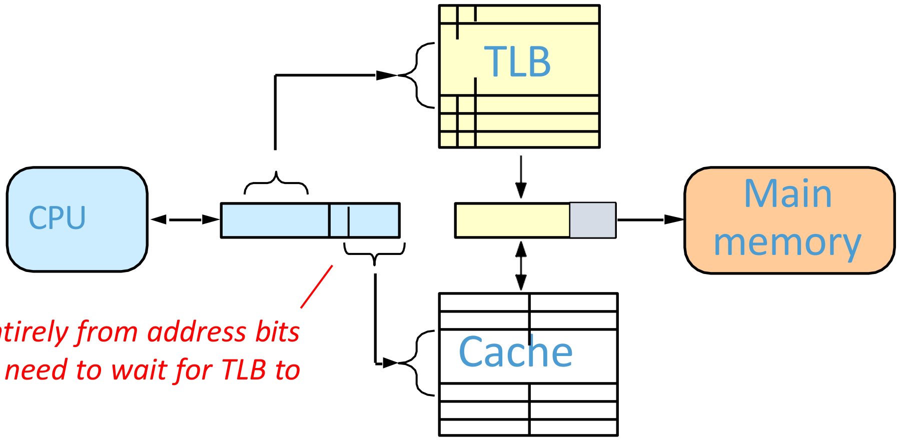
- FAST: No virtual➔physical translation on cache hits
- Problem: Must flush cache after context switch

Physically-Addressed Cache



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

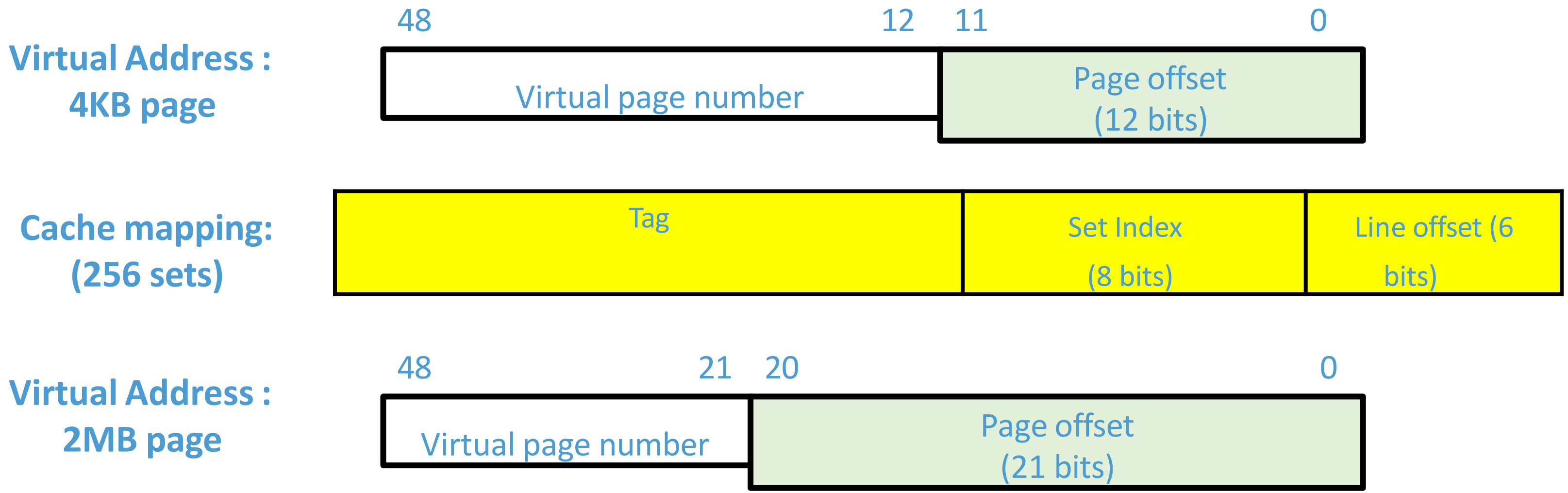
Best of Both Worlds (L1 Cache): Virtually-Indexed, Physically-Tagged Cache (VIPT)



Cache index comes entirely from address bits in page offset – don't need to wait for TLB to start cache lookup!

Using Huge Pages

- Huge page size: 2MB or 1GB



Quiz!

- I have a virtual address: 0xAAAA
- The cache parameters are as below
  - Cache size: 32KB
  - Line size/Block size: 64B
  - Associativity: 8

Question 1:
What is the cache set index?

Question 2:
What is the next address that map to **the same cache set** as this one but not the same cache line?

Takeaways

- Practical challenges in implementing a reliable cache attack
  - Page sharing
  - 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

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL