

# Comp 790-184: Hardware Security and Side-Channels

## Lecture 2: Practical Cache Attacks

January 16, 2025  
Andrew Kwong



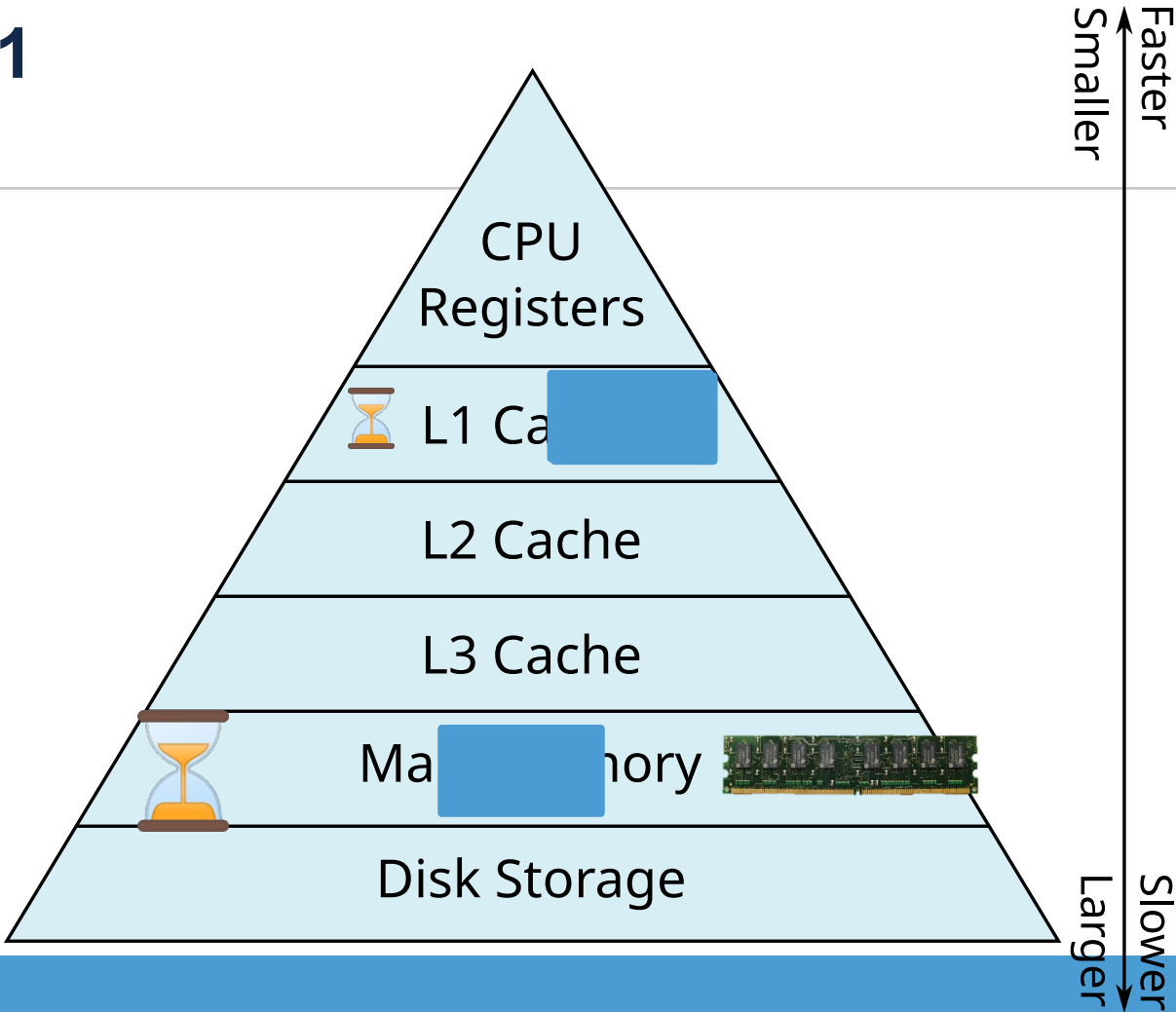
THE UNIVERSITY  
*of* NORTH CAROLINA  
*at* CHAPEL HILL

## Today's Class

---

- How caches work
- How caches can be used for side-channels
  - What cache side-channels can accomplish
- General background on caches

# Caching 101



## 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.

Slides adapted from Mengjia Yan  
([shd.mit.edu](http://shd.mit.edu))

**The Goal:**

**Monitor access patterns at cache line granularity**

# Leak Crypto Library #1: RSA

---

- Square-and-Multiply Exponentiation

## Input :

base  $b$

modulo  $m$

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

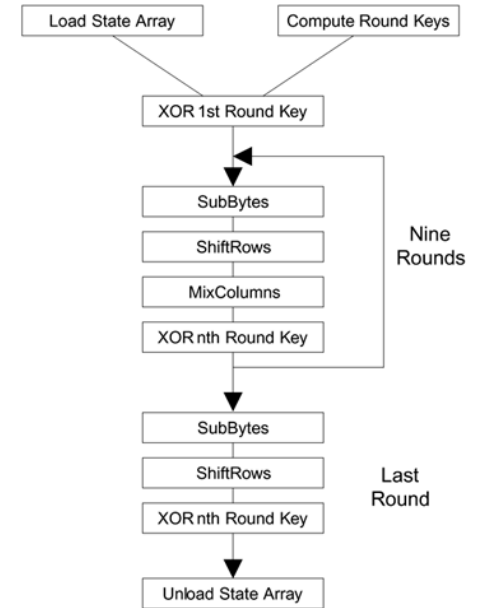
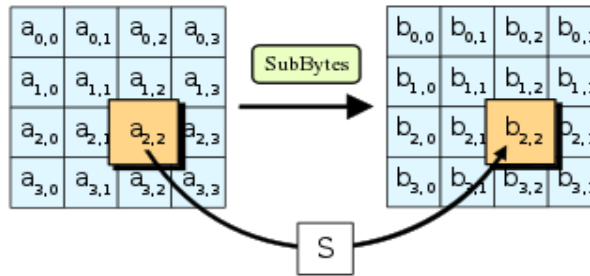
## Output:

$b^e \bmod m$

```
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
```

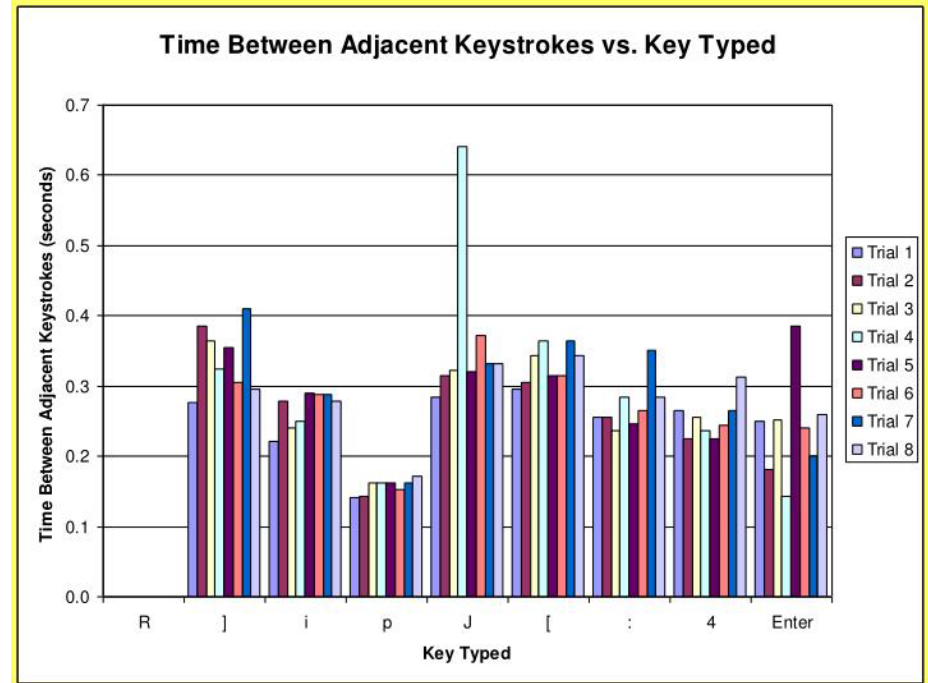
# Leak Crypto Algorithm #2: AES

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



# Keystroke Extraction

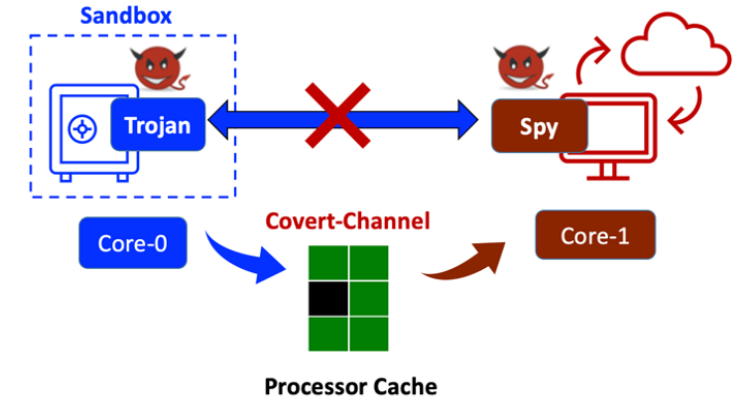
- Keystroke cadence yields keystroke extraction
- Monitor cacheline that updates the output display
  - Victim will access that cacheline every time he types a character





# Covert Channel

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



**How can attacker monitor cacheline usage?**

# Attack Strategy #1: Flush+Reload

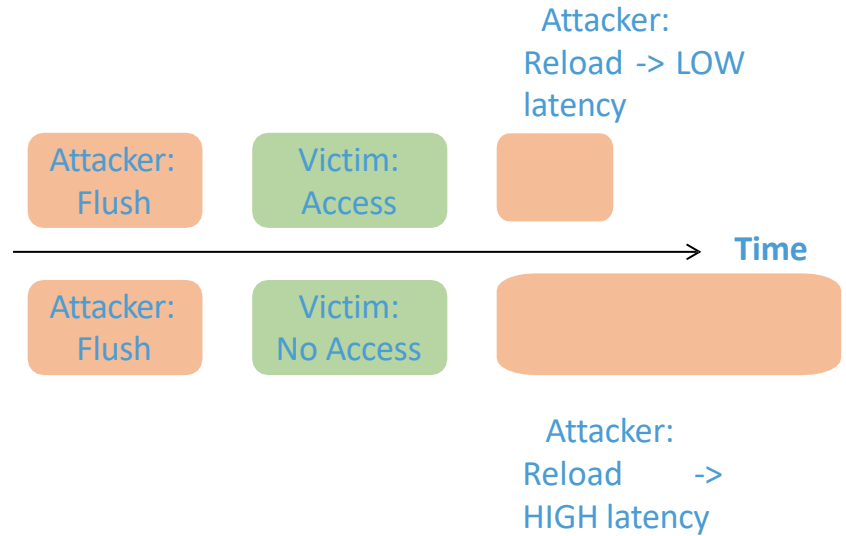
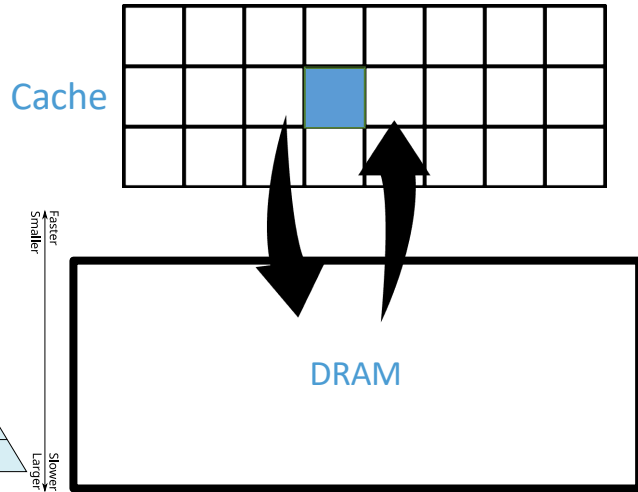
---

- 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.

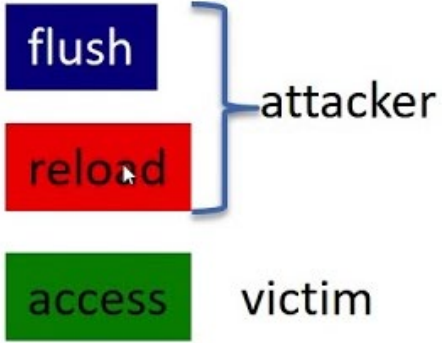
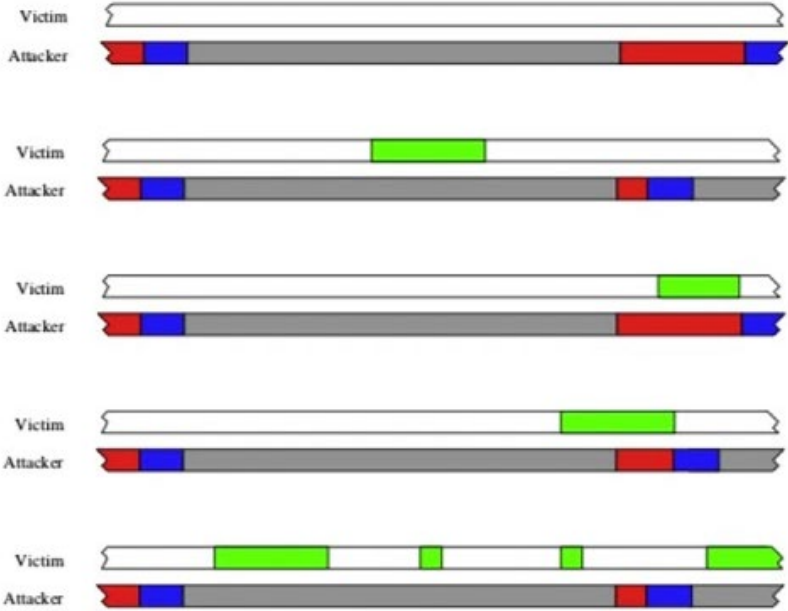
# Flush+Reload



■ A shared cache line  
(latency reveals presence in cache)




# Some possible outcomes



# 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

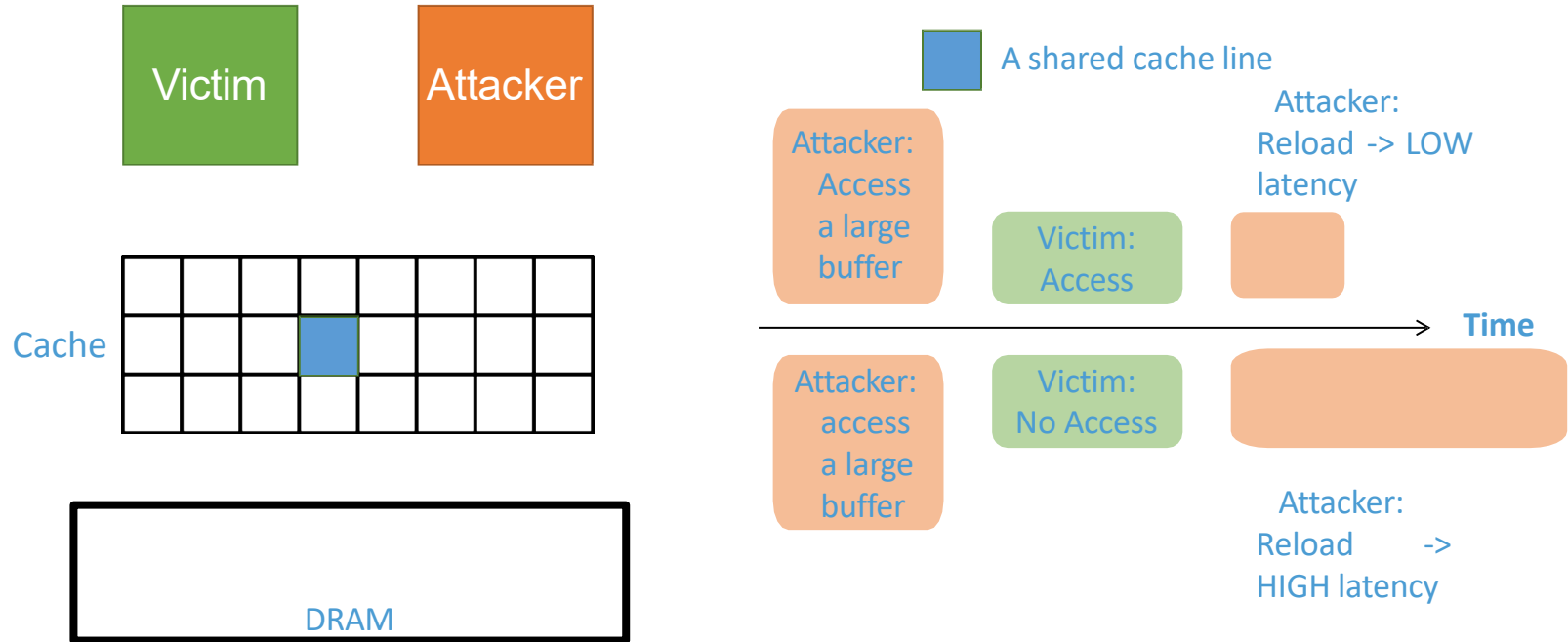
## Attack Strategy #2: ?

---

- 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





# Lessons Learnt So Far

- flush+reload
  - Requires “flush” instruction
- Evict+reload
  - Doesn’t require “flush”
  - Timing required
    - Can’t fully mitigate

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



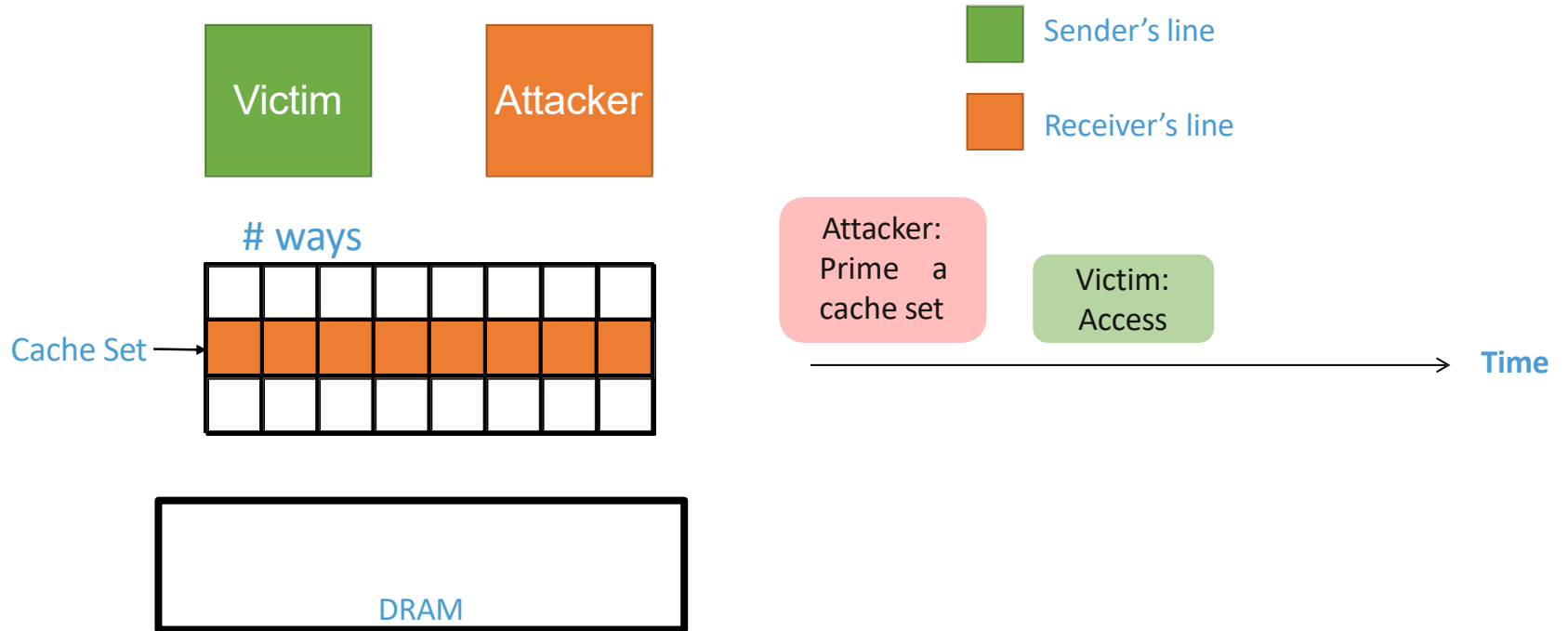
### Details

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.

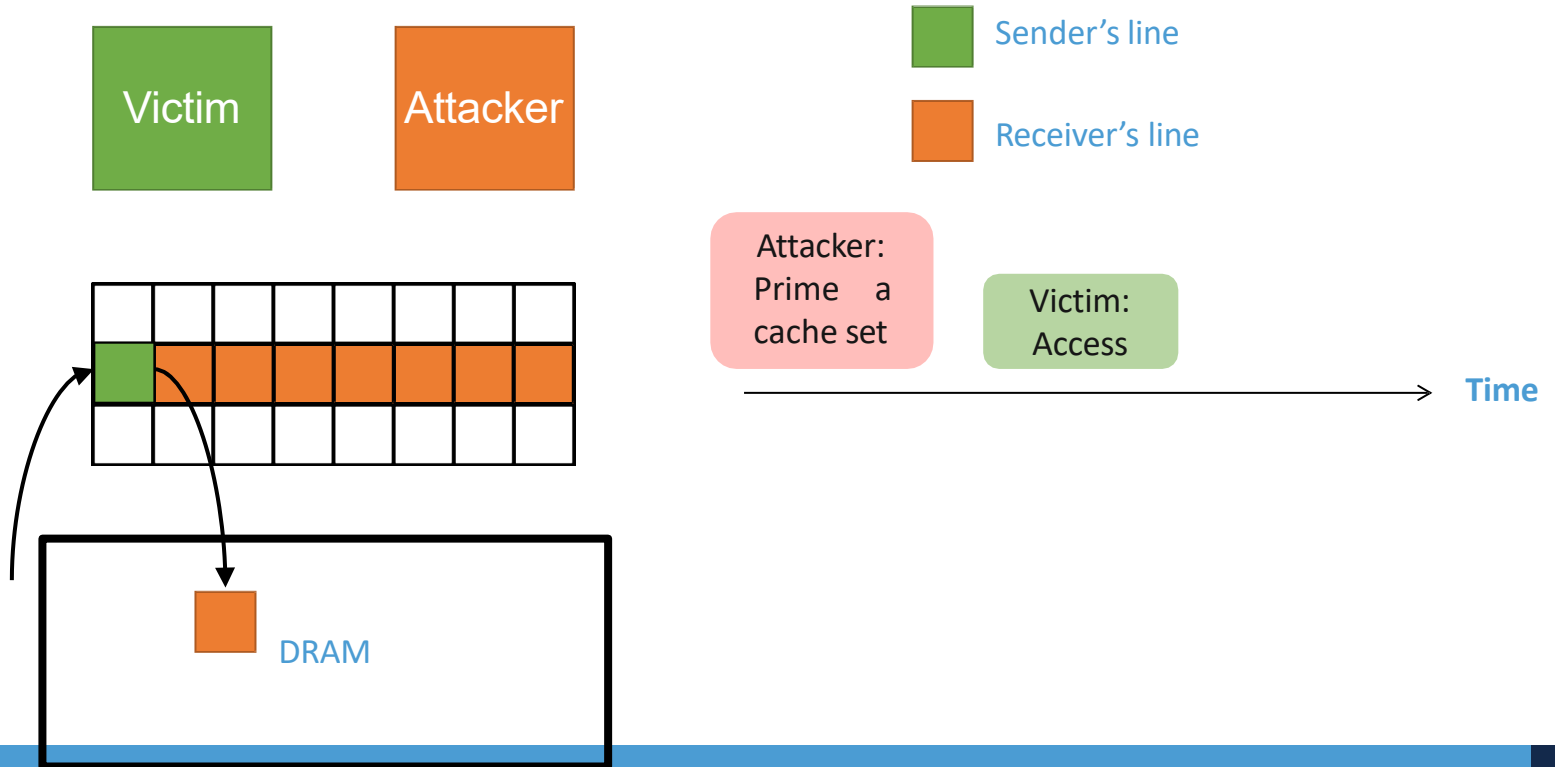
Published academic papers have demonstrated that by forcing a flush and reload of cache memory, it is possible to measure memory timings to try and determine an AES encryption key in use on another virtual machine running on the same physical processor of the host server if Transparent Page Sharing is 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).

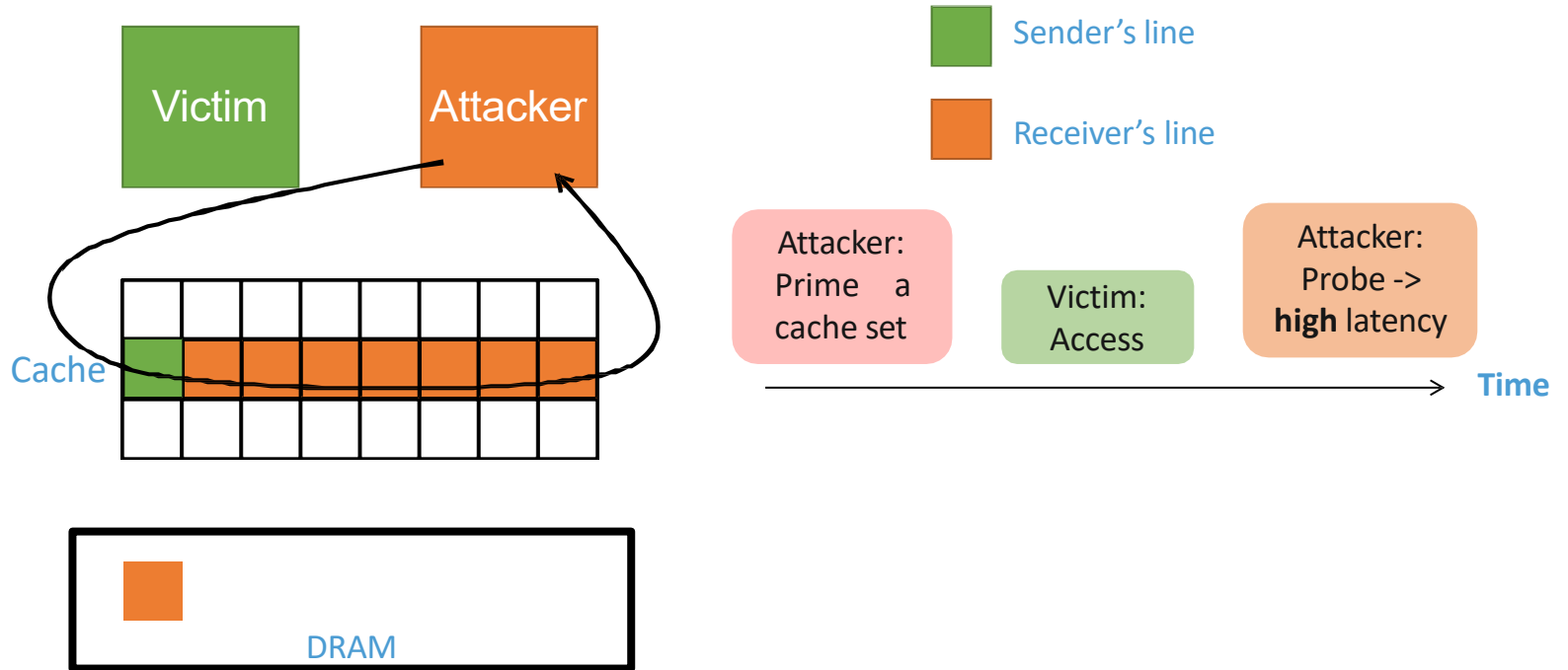
# Attack Strategy #3: Prime+Probe



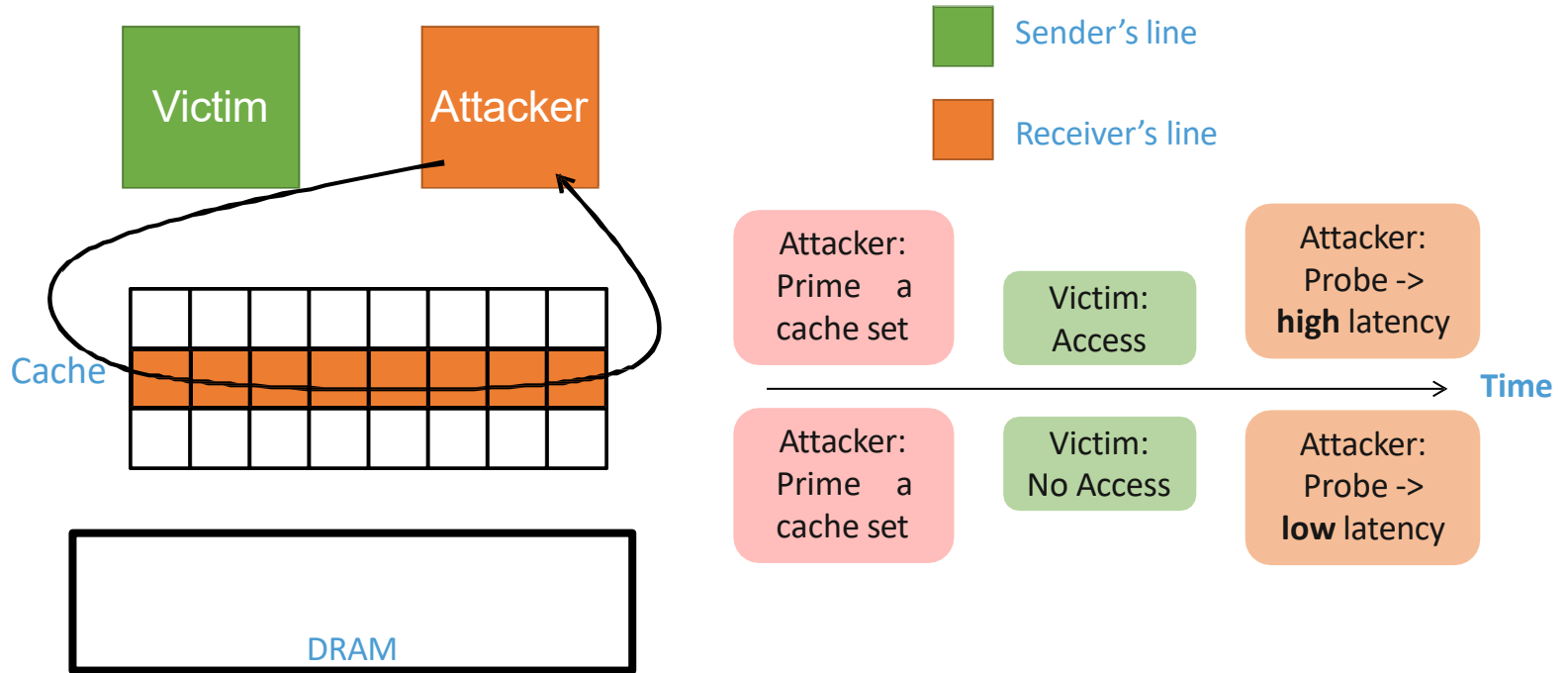
# Attack Strategy #3: Prime+Probe



# Attack Strategy #3: Prime+Probe

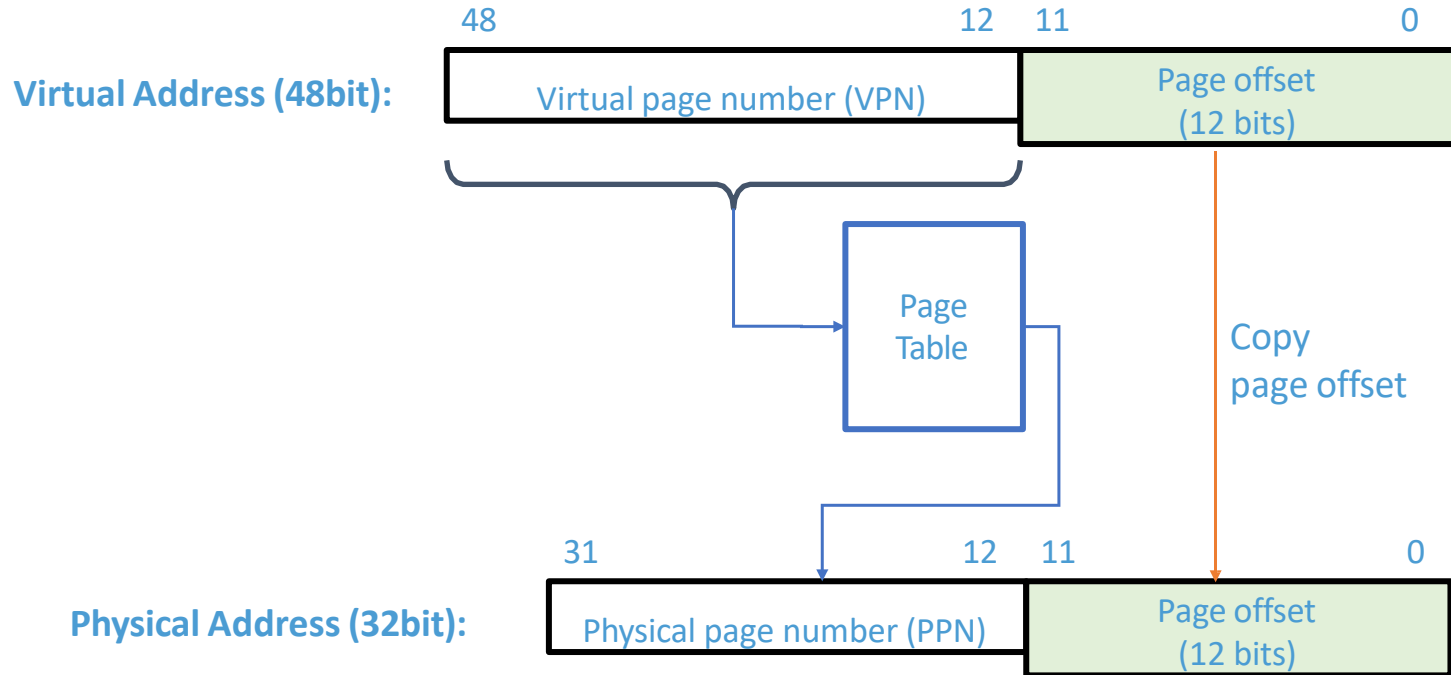


# Attack Strategy #3: Prime+Probe



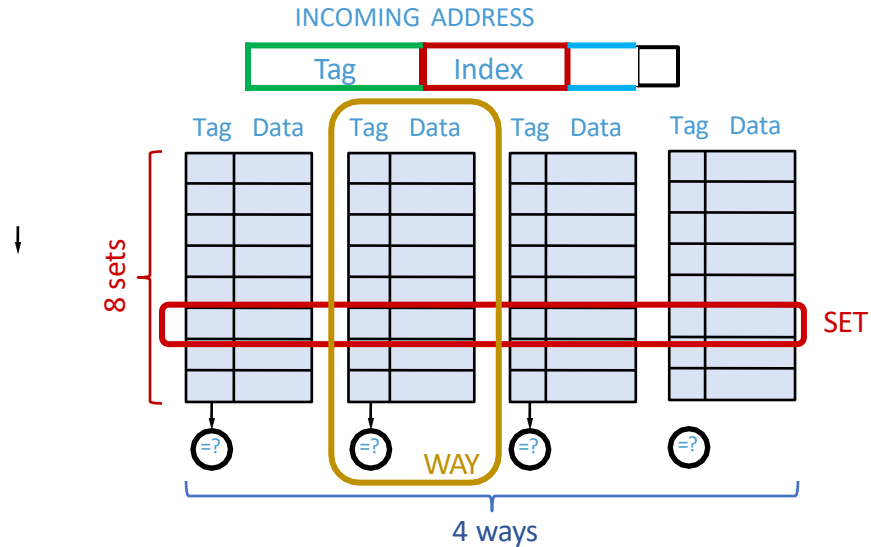
## More Cache Background

# Address Translation (4KB page)



# N-way Set-Associative Cache

- Does cache use virtual address or physical address?

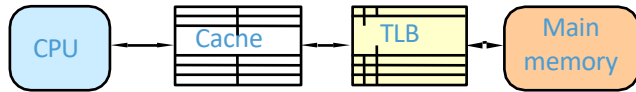




# Using Caches with Virtual Memory

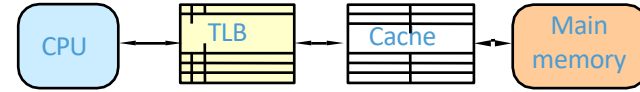
---

## *Virtually-Addressed Cache*



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

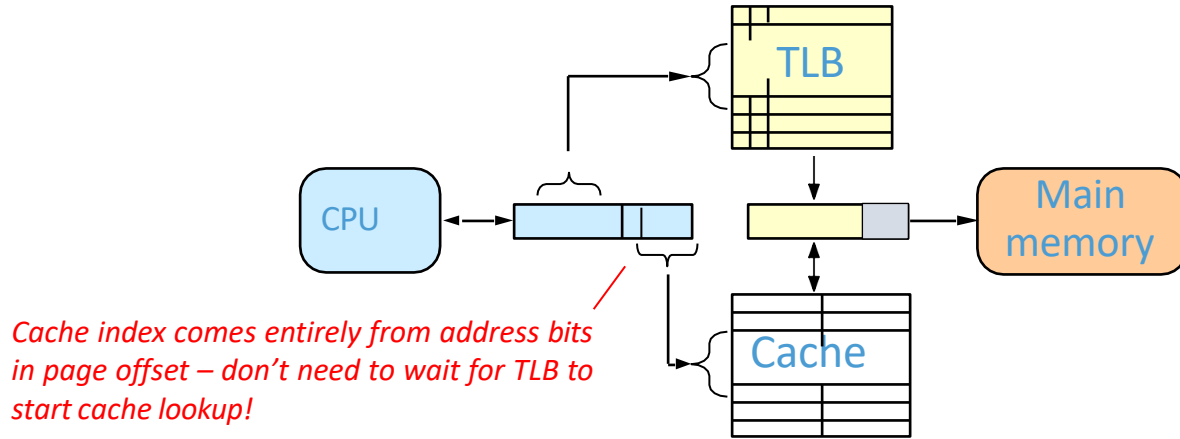
## *Physically-Addressed Cache*



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

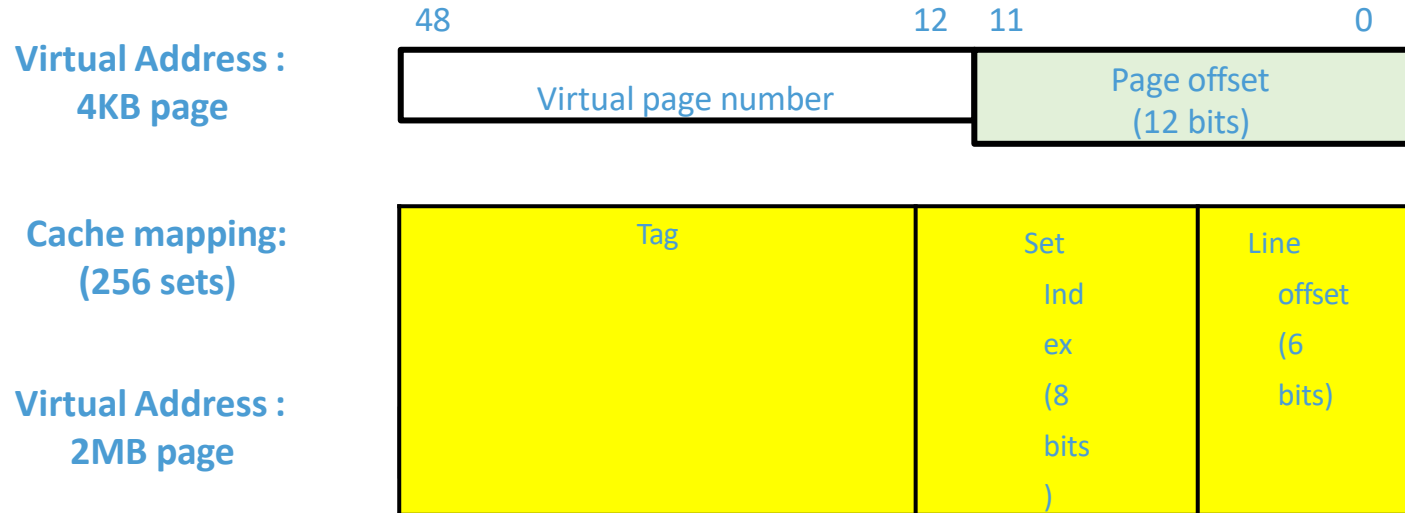
# Best of Both Worlds (L1 Cache): Virtually-Indexed, Physically-Tagged Cache (VIPT)

---



# Using Huge Pages

- Huge page size: 2MB or 1GB



## 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



THE UNIVERSITY  
*of* NORTH CAROLINA  
*at* CHAPEL HILL