

Practical Cache Attacks

Mengjia Yan

Spring 2023



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 Library #2: AES

Why Cache?

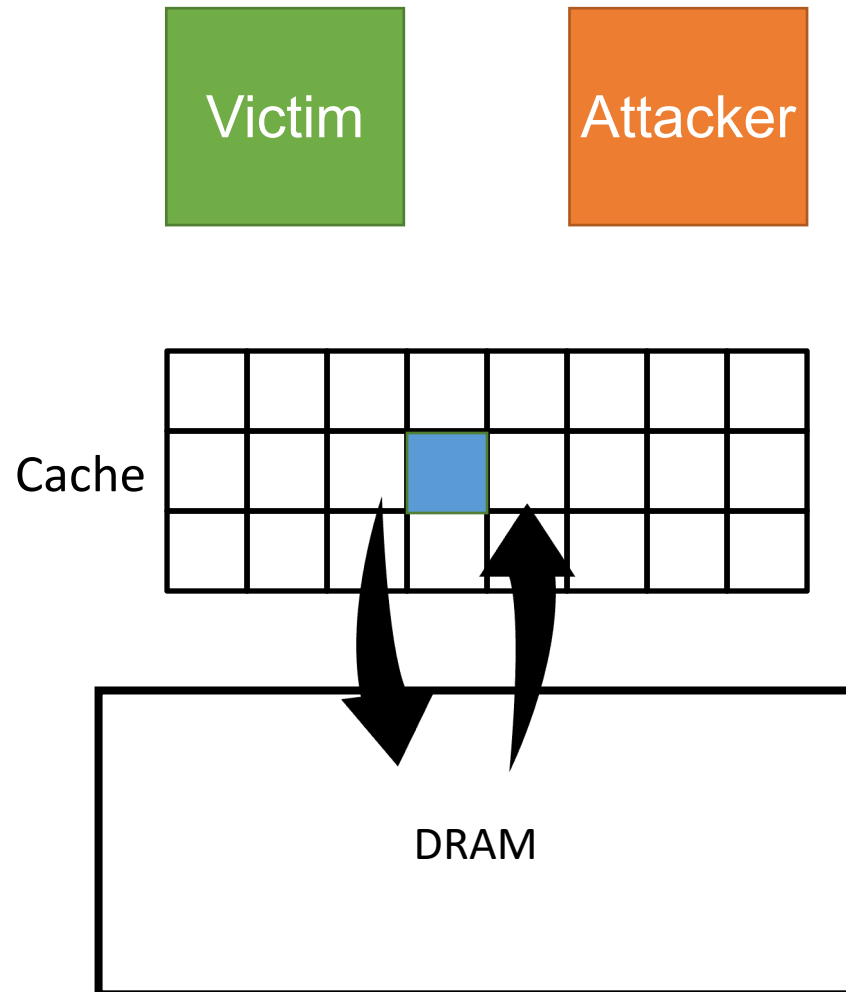
- Large attack surface. Shared across cores/sockets.
- Fast. Can be used to build high-bandwidth channels
- Many states. Can encode secrets spatially to further improve bandwidth and precision.
- There exist many cache-like structures. The same attack concepts and tricks will apply.


**The Goal:
Monitor access patterns at
cache line granularity**

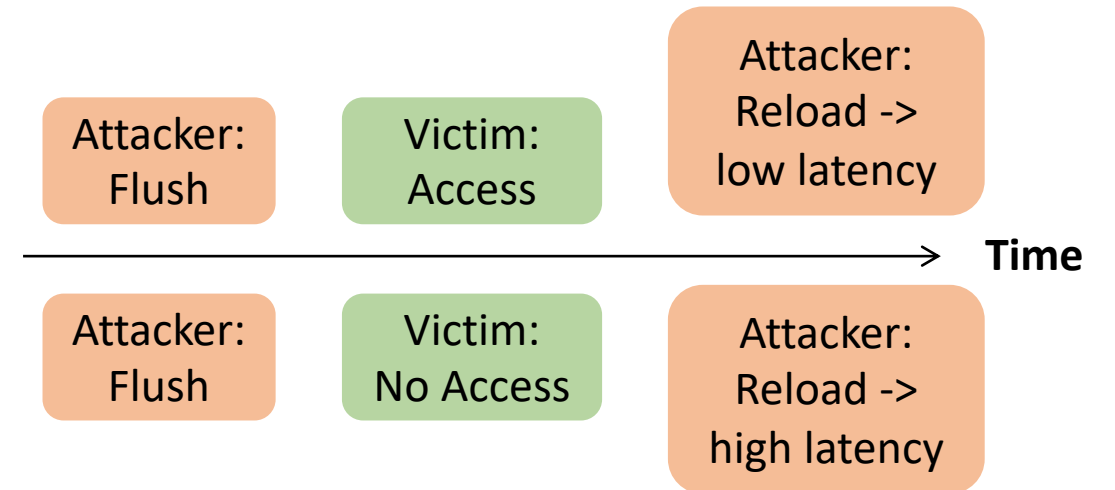
Attack Strategy #1: Flush+Reload

- 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

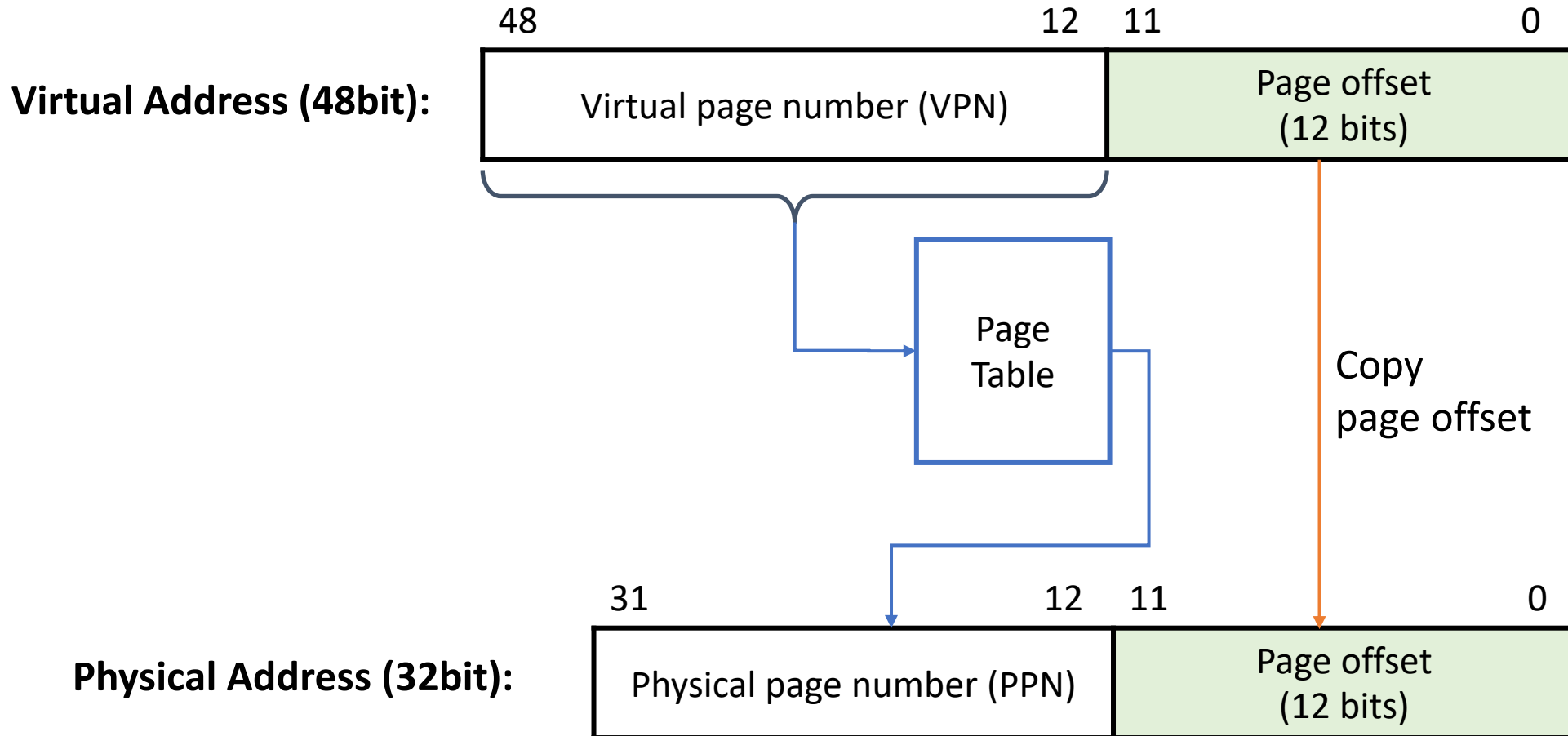
Flush+Reload



 A shared cache line



Address Translation (4KB page)




Shared Memory Between Untrusted Domains?

		Process-1	Process-2	Same?
PID				
PPN of "printf"				
PPN of "a stack variable"				
PPN of "a heap variable"	After allocated			
	After read access			
	After write access			

The Attack 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



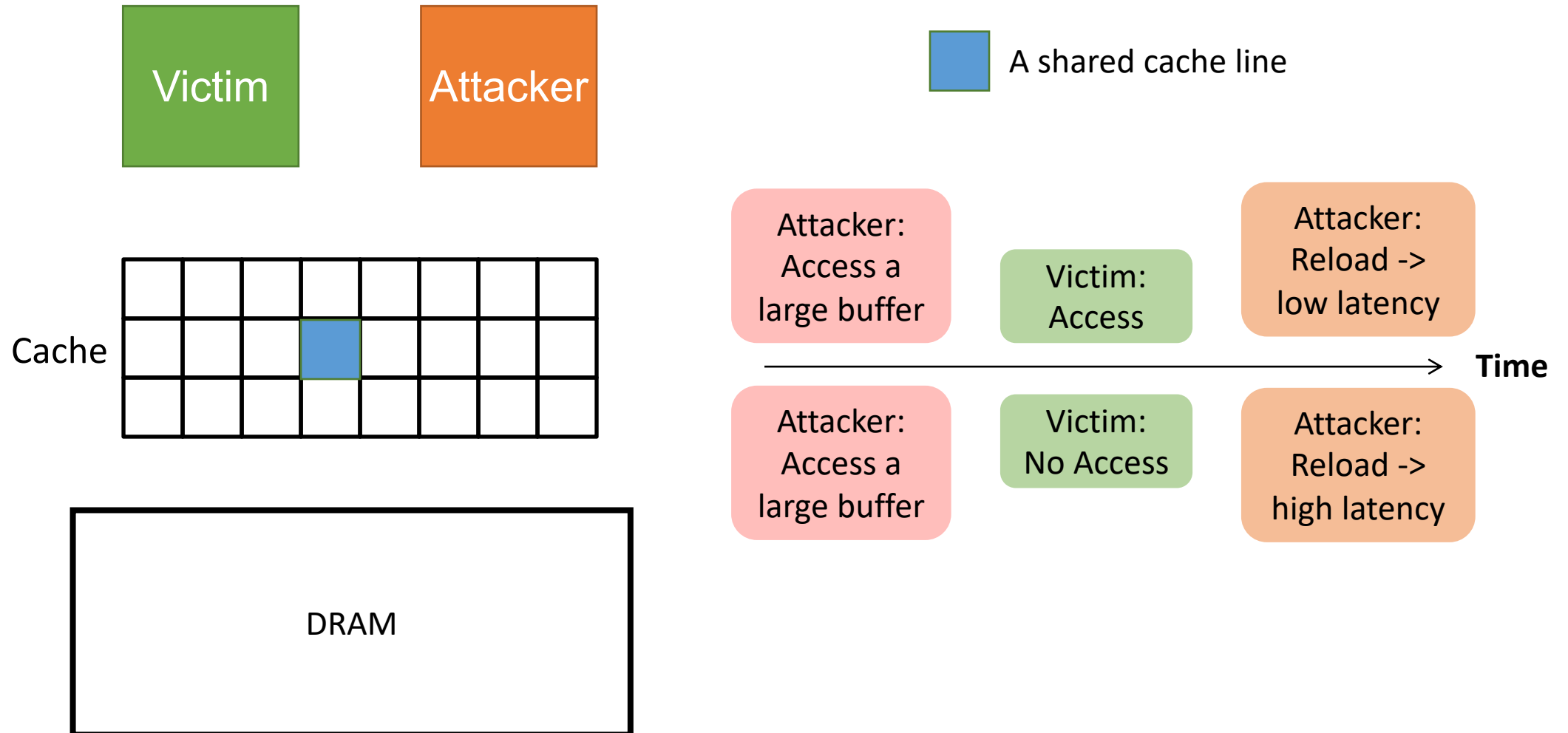
A Demo

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”*

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

Attack Strategy #2: **Evict+Reload**



Lessons Learnt So Far

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



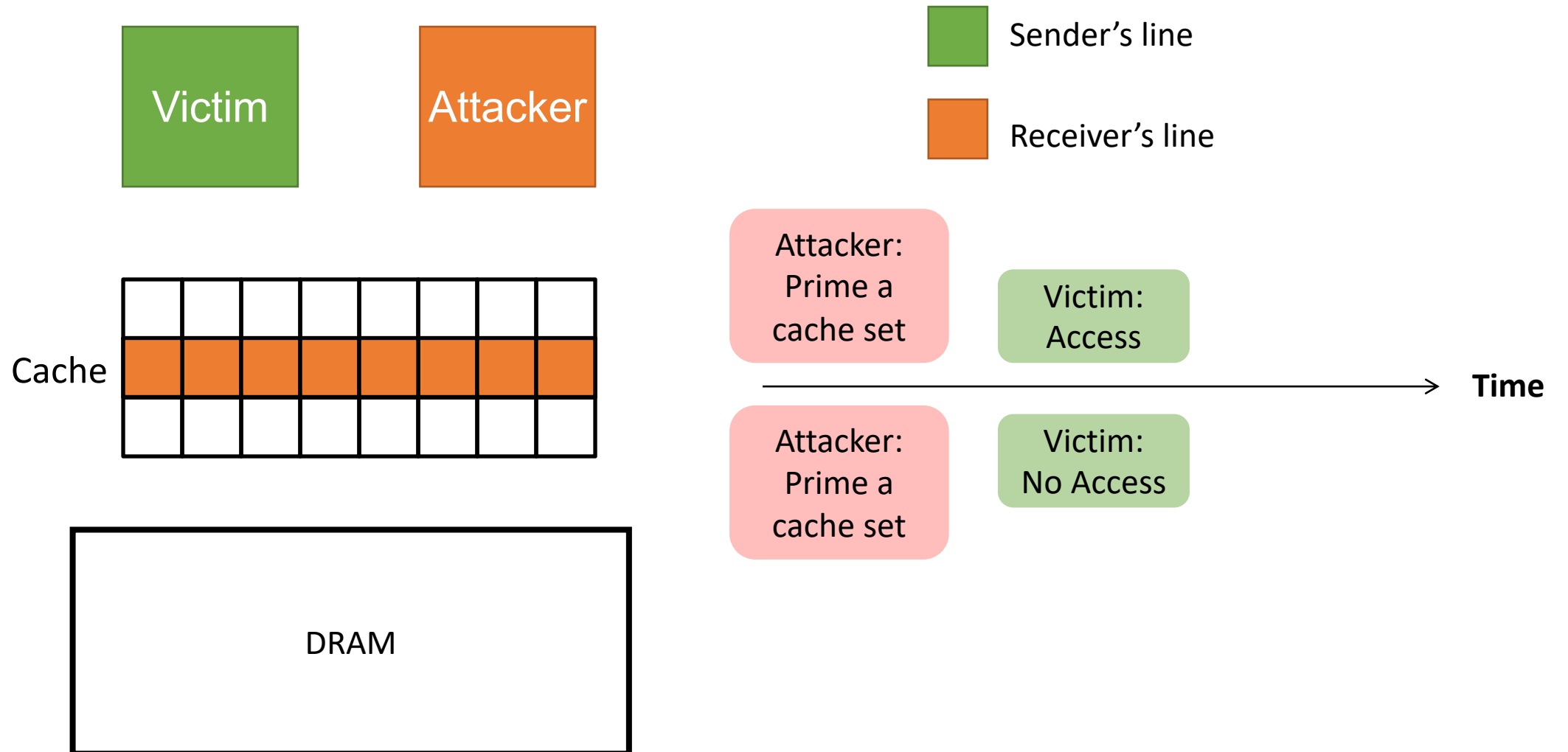
Details

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.

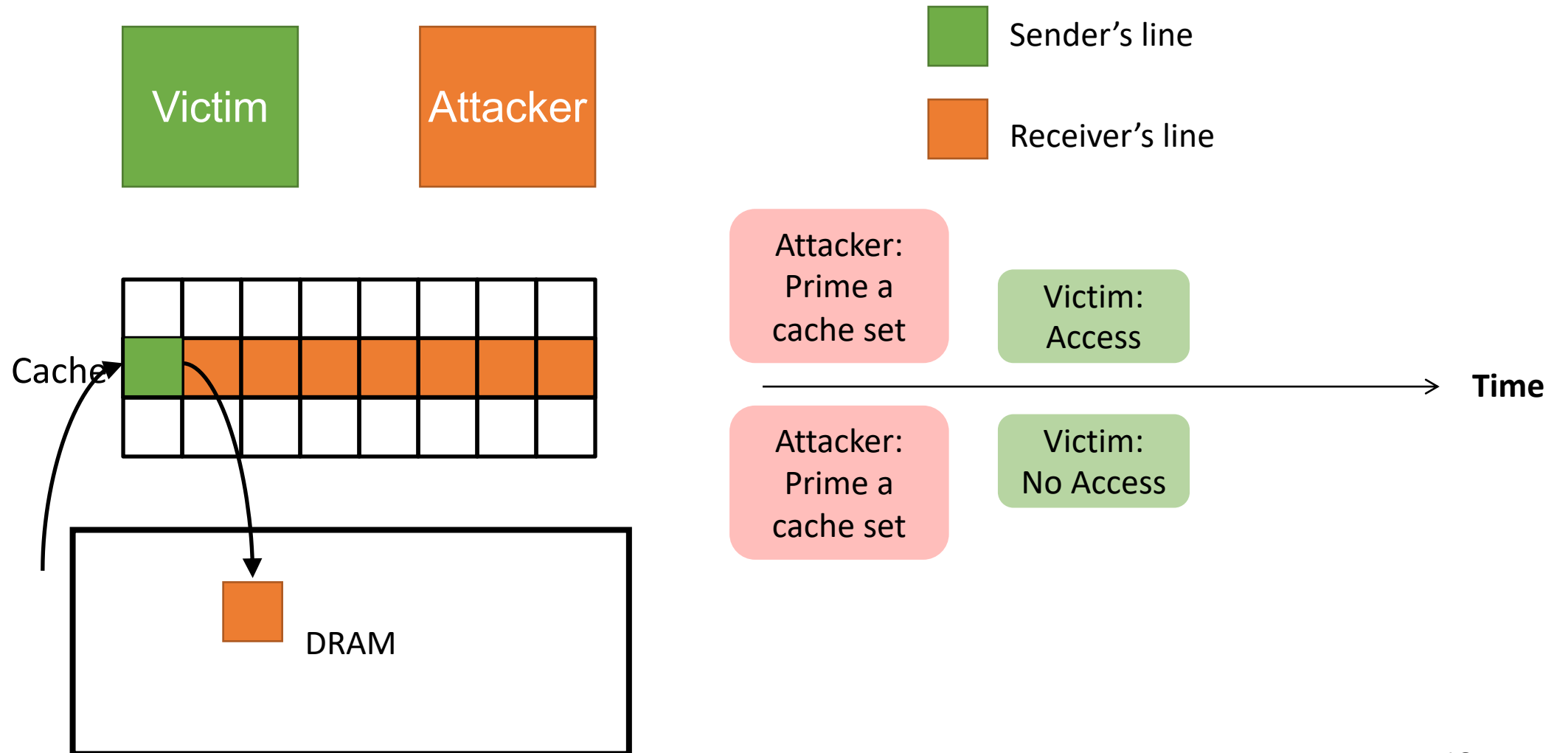
Published academic papers have demonstrated that by forcing a flush and reload of cache memory, it is possible to measure memory timings to try and determine an AES encryption key in use on another virtual machine running on the same physical processor of the host server if Transparent Page Sharing is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

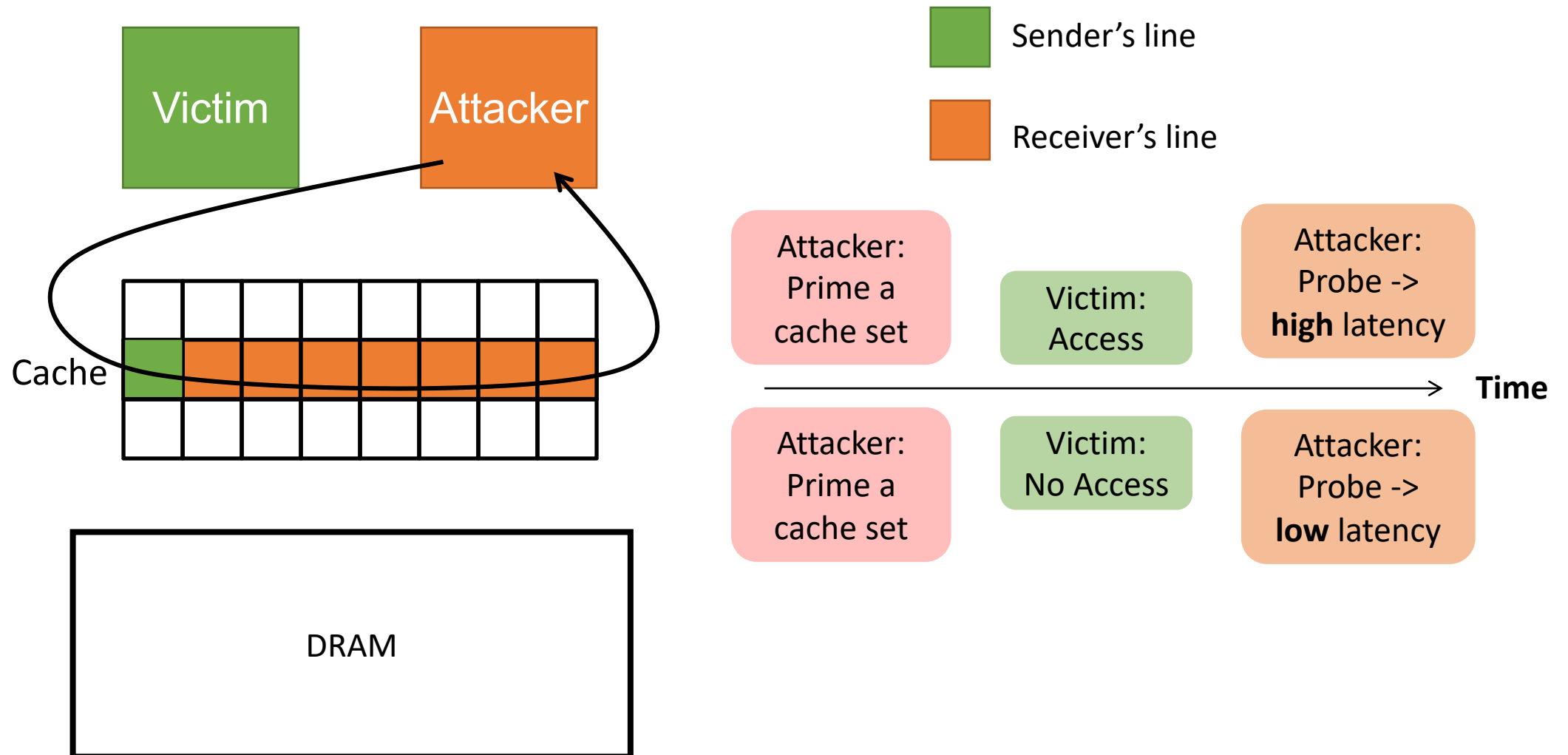
Attack Strategy #3: Prime+Probe



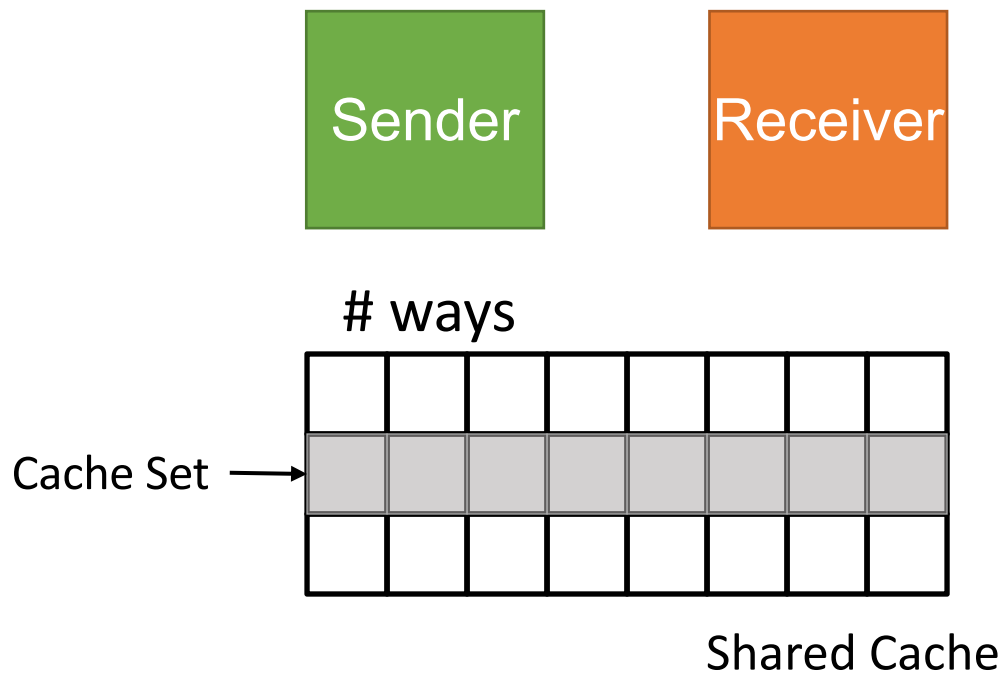
Attack Strategy #3: Prime+Probe



Attack Strategy #3: Prime+Probe



Analogy: Bucket/Ball



Sender's address



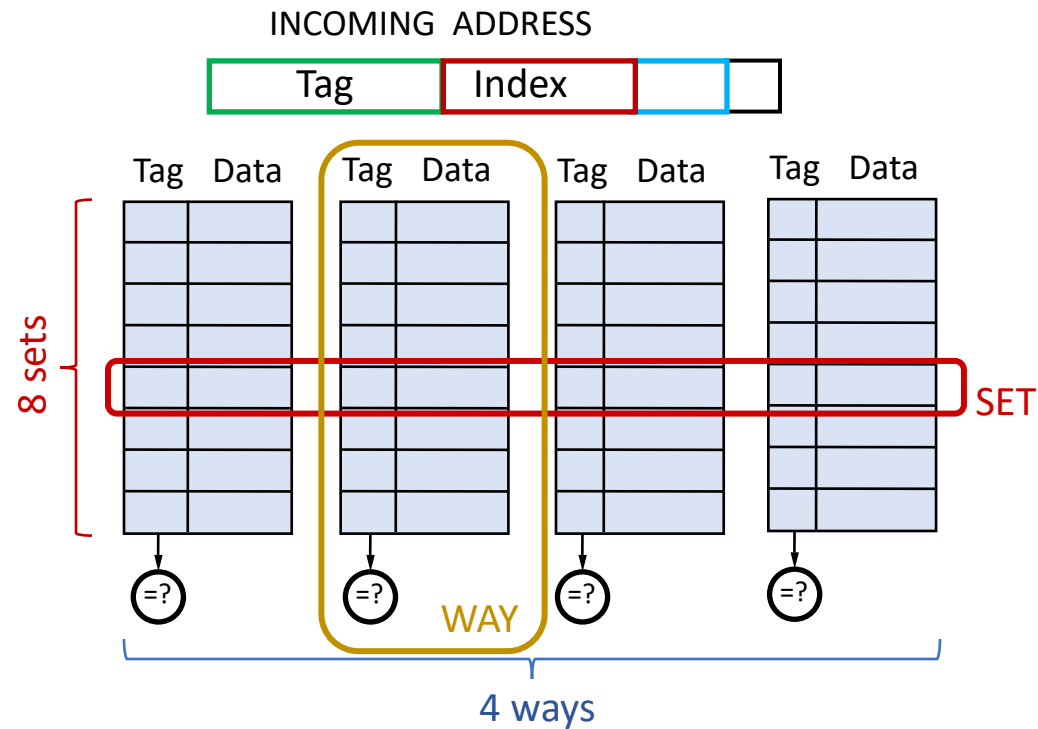
Receiver's address



Each cache set is a bucket that can hold 8 balls

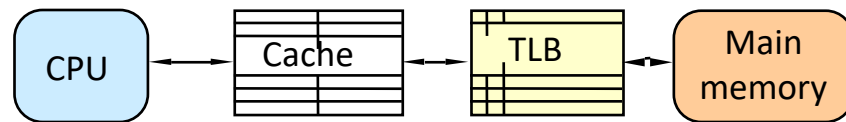
N-way Set-Associative Cache

- Does cache use virtual address or physical address?



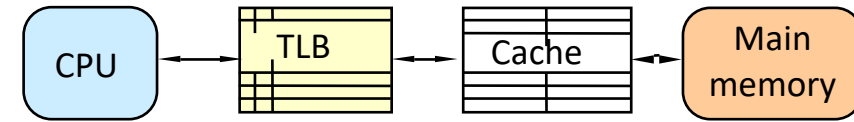
Using Caches with Virtual Memory

Virtually-Addressed Cache



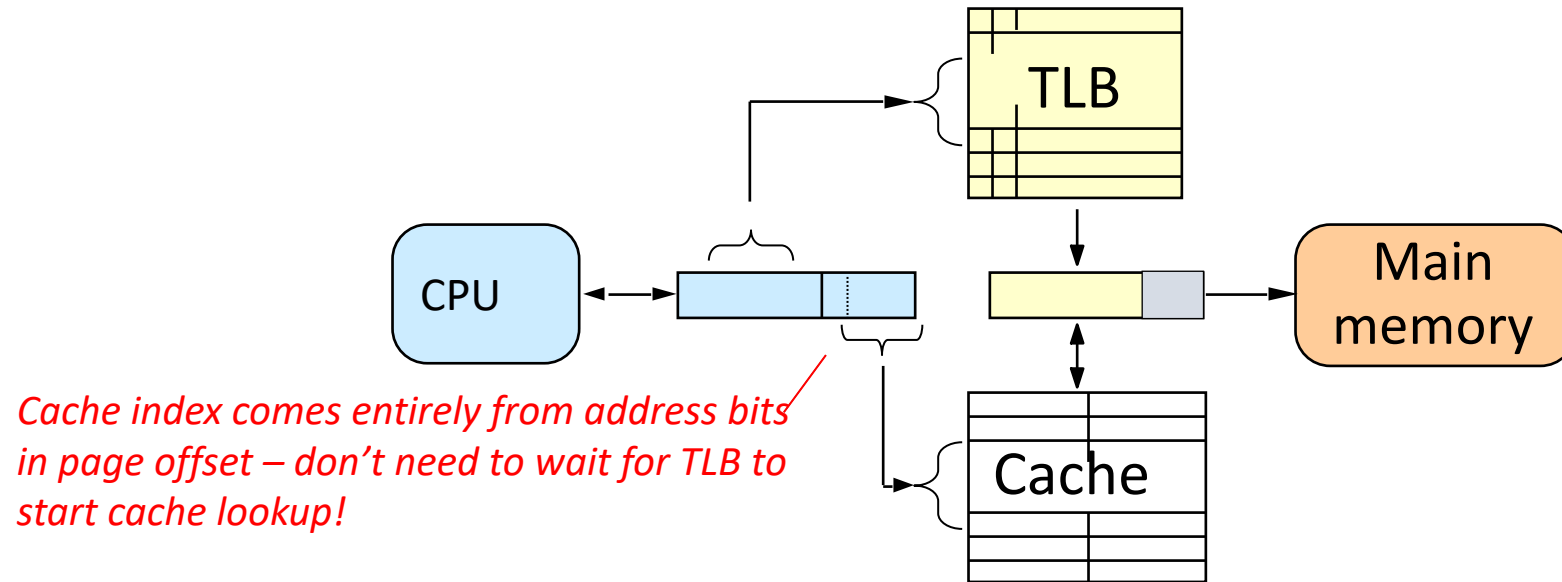
- FAST: No virtual \rightarrow physical translation on cache hits
- Problem: Must flush cache after context switch

Physically-Addressed Cache



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

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

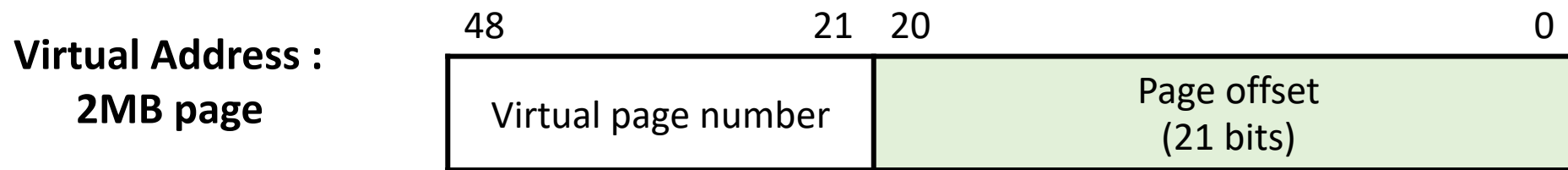
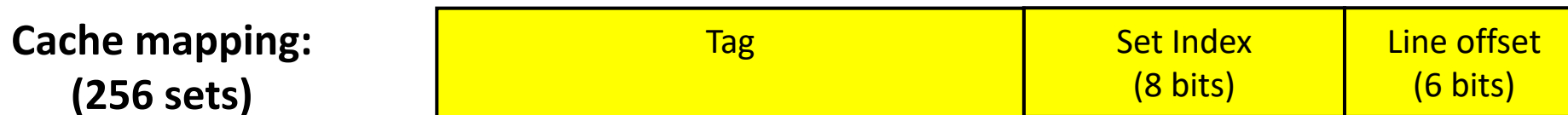
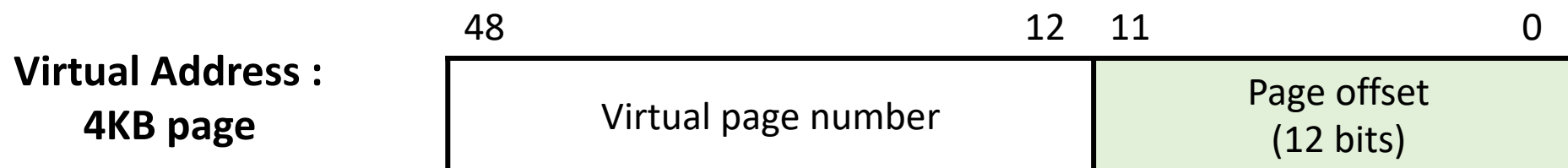


Cache and Address Translation

		Cache L1-D			Cache L3		
		size=32KB (2^{15} Bytes) linesize=64B (2^8 Bytes) associativity=8			size=32MB (2^{25} Bytes) linesize=64B (2^8 Bytes) associativity=16		
		Addr1	Addr2	Conflict?	Addr1	Addr3	Conflict?
Virtual	Address			--			--
	Set Index			Yes			Conflict
Physical	Address			--			--
	Set Index			?			?

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

Next:
Transient Execution Attacks