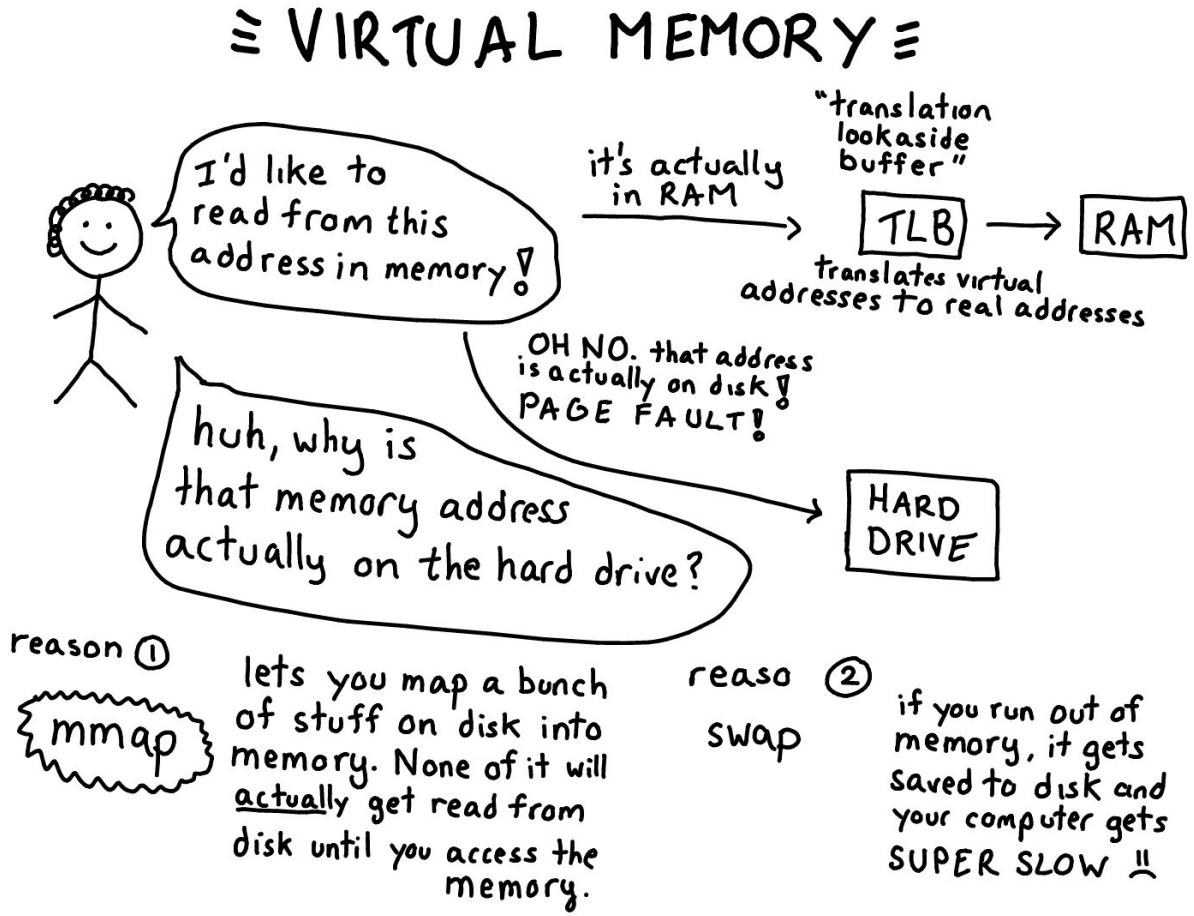
The Hardware/Software Interface

Virtual Memory III

Instructors:
Amber Hu, Justin Hsia

Teaching Assistants:

Anthony Mangus	Divya Ramu
Grace Zhou	Jessie Sun
Jiuyang Lyu	Kanishka Singh
Kurt Gu	Liander Rainbolt
Mendel Carroll	Ming Yan
Naama Amiel	Pollux Chen
Rose Maresh	Soham Bhosale
Violet Monserate	



https://twitter.com/b0rk/status/799472984939446272

Relevant Course Information

- HW 24 due tonight; HW 25 due Wednesday (12/3)
- HW 26 due December 3
  - Last HW! 🎉🎉
- Lab 5 due Thursday (12/5)
  - Recommended that you watch all the Lab 5 helper videos

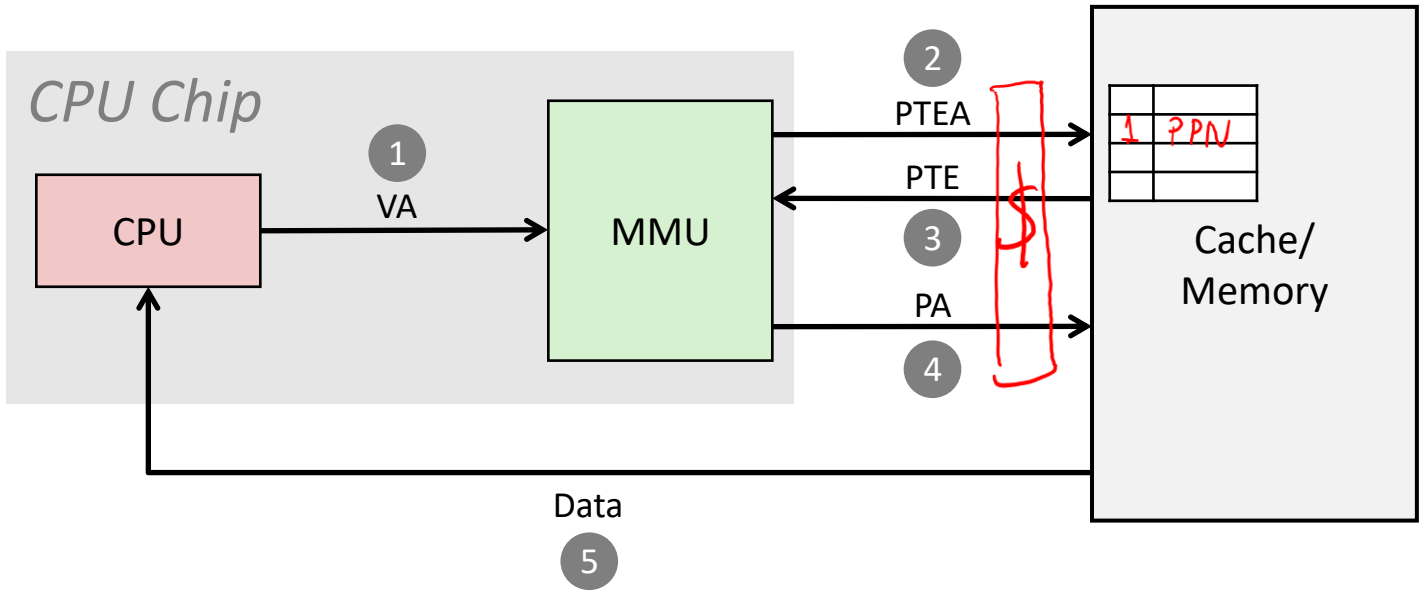
- **Final Exam:** Wednesday, December 10
  - Final review section on Thu, 12/4
  - Review Session: Fri, 12/5, evening
  - Ed post #547

Lecture Outline (1/3)

- **Translation Lookaside Buffer**
- Memory Translation Examples
- For Fun: DRAMMER Security Attack

(page does live in physical mem)

Address Translation: Page Hit (Review)



1) Processor sends *virtual* address to MMU (*memory management unit*)

2-3) MMU fetches PTE from page table in cache/memory
(Uses PTBR to find beginning of page table for current process)

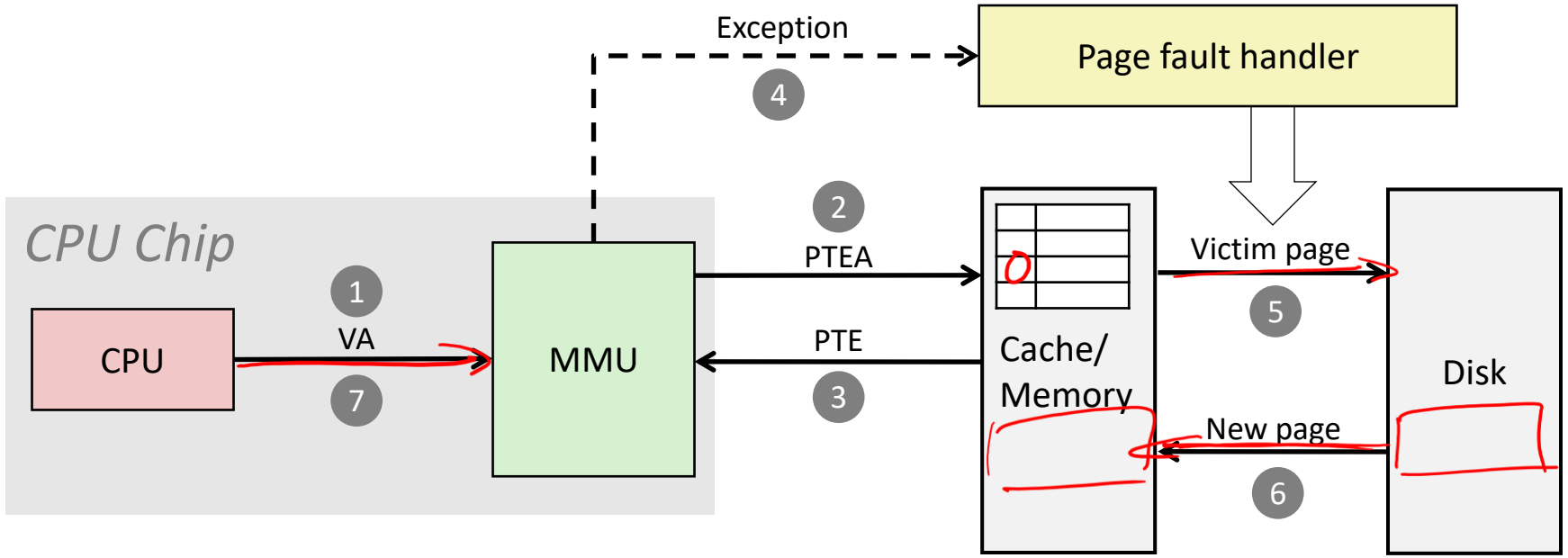
4) MMU sends *physical* address to cache/memory requesting data

5) Cache/memory sends data to processor

VA = Virtual Address PTEA = Page Table Entry Address PTE= Page Table Entry
PA = Physical Address Data = Contents of memory stored at VA originally requested by CPU

(page is NOT in physical mem)

Address Translation: Page Fault (Review)



1) Processor sends virtual address to MMU
2-3) MMU fetches PTE from page table in cache/memory
4) Valid bit is zero, so MMU triggers page fault exception
5) Handler identifies victim (and, if dirty, pages it out to disk)
6) Handler pages in new page and updates PTE in memory
7) Handler returns to original process, restarting faulting instruction

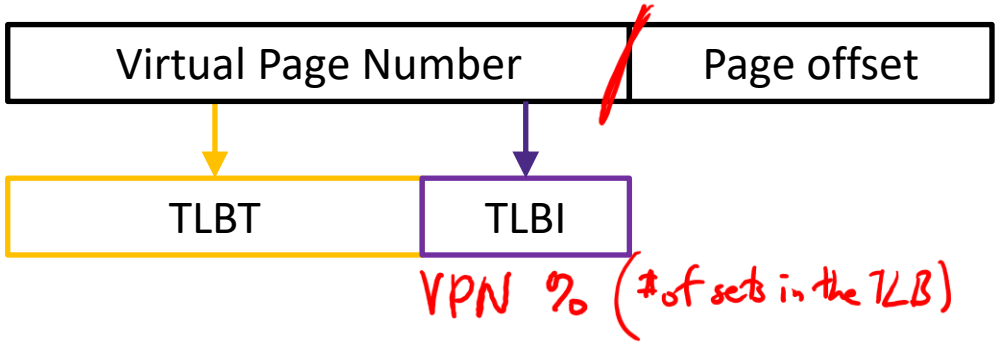
Hmm… Translation Sounds Slow

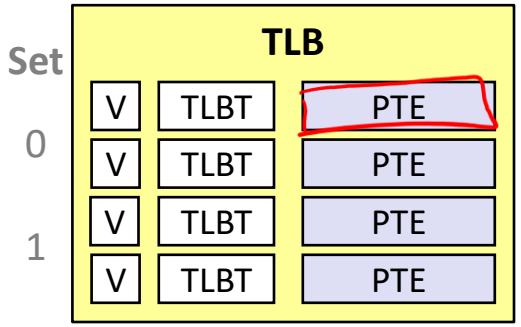
- The MMU accesses memory *twice*: once to get the PTE for translation, and then again for the actual memory request
  - The PTEs *may* be cached in L1 like any other memory word
    - But they may be evicted by other data references
    - And a hit in the L1 cache still requires 1-3 cycles

- *What can we do to make this faster?*
  - "Any problem in computer science can be solved by adding another level of **indirection**." – *David Wheeler, inventor of the subroutine*
  - Add another cache! 

Speeding Up Translation with a TLB

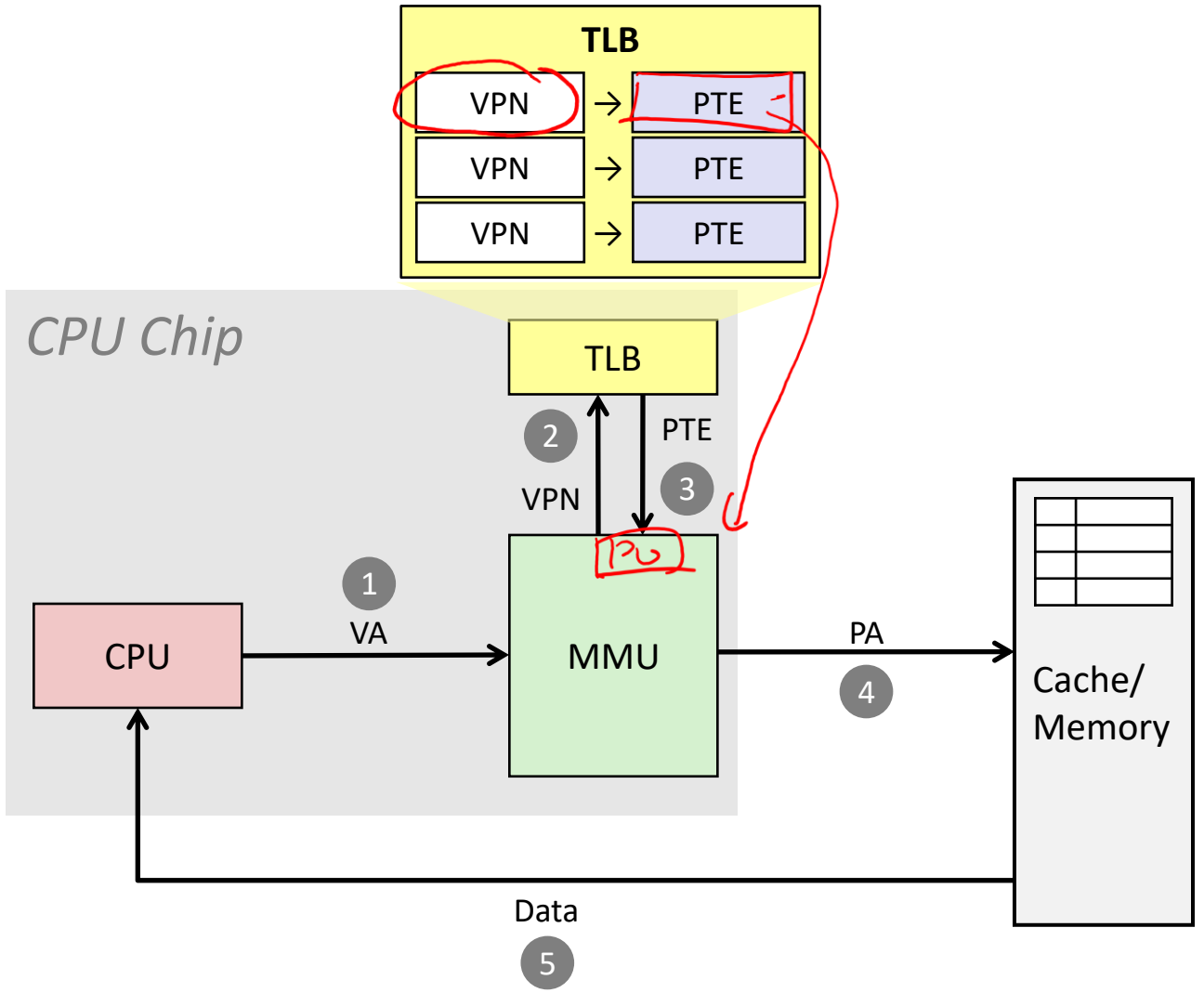
- *Translation Lookaside Buffer* (TLB):
  - Small hardware cache in MMU that maps virtual page numbers (VPNs) to physical page numbers (PPNs)
    - Much faster than a page table lookup in cache/memory
    - Access by splitting VPN into *TLB Tag* and *TLB Index* based on number of sets in TLB
  - Stores *page table entries* for a small number of pages
    - Modern Intel processors have 128 or 256 entries in TLB





TLB Hit

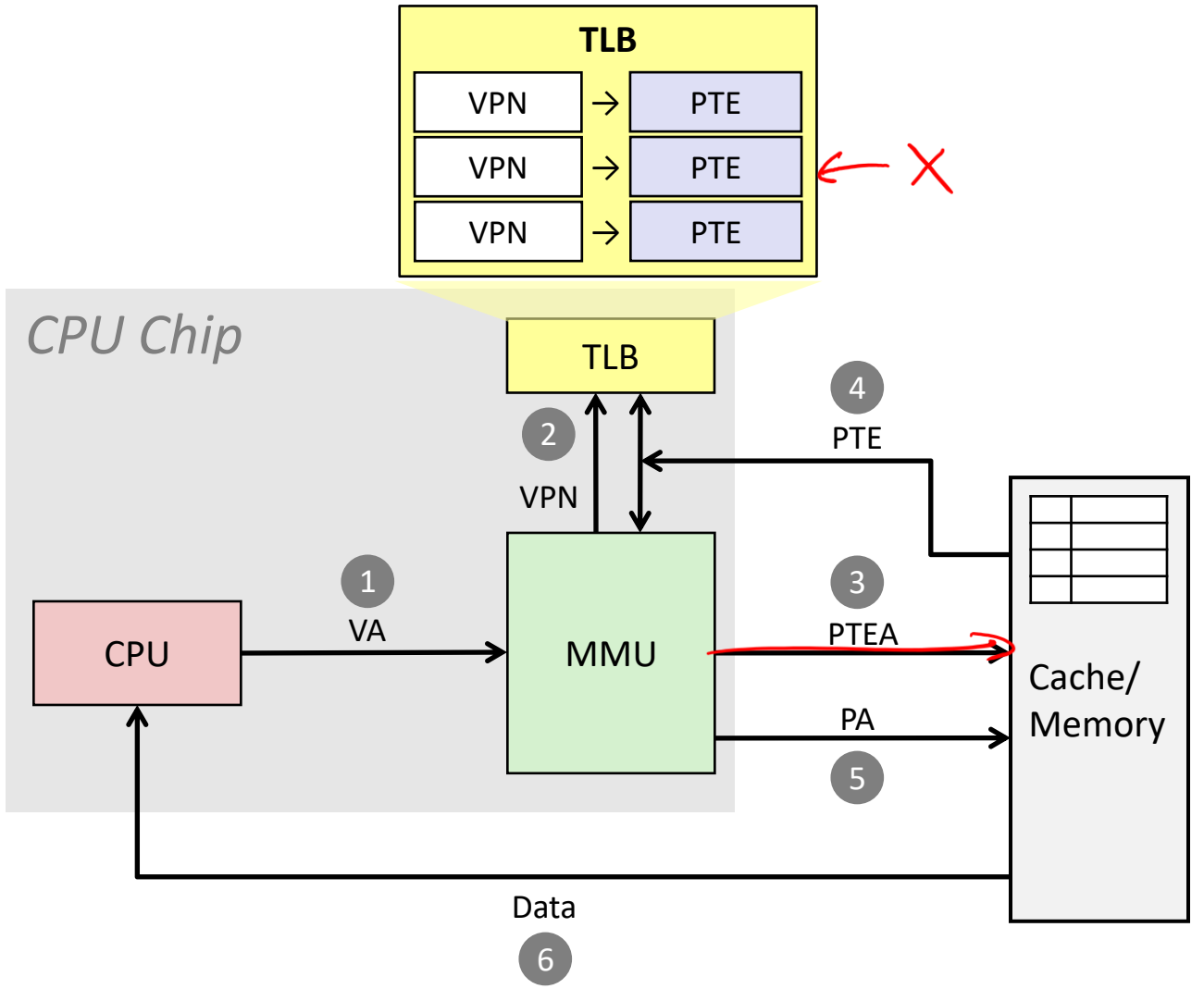
(page has been used recently)



- A TLB hit eliminates a memory access!

TLB Miss

(page has not been used recently)



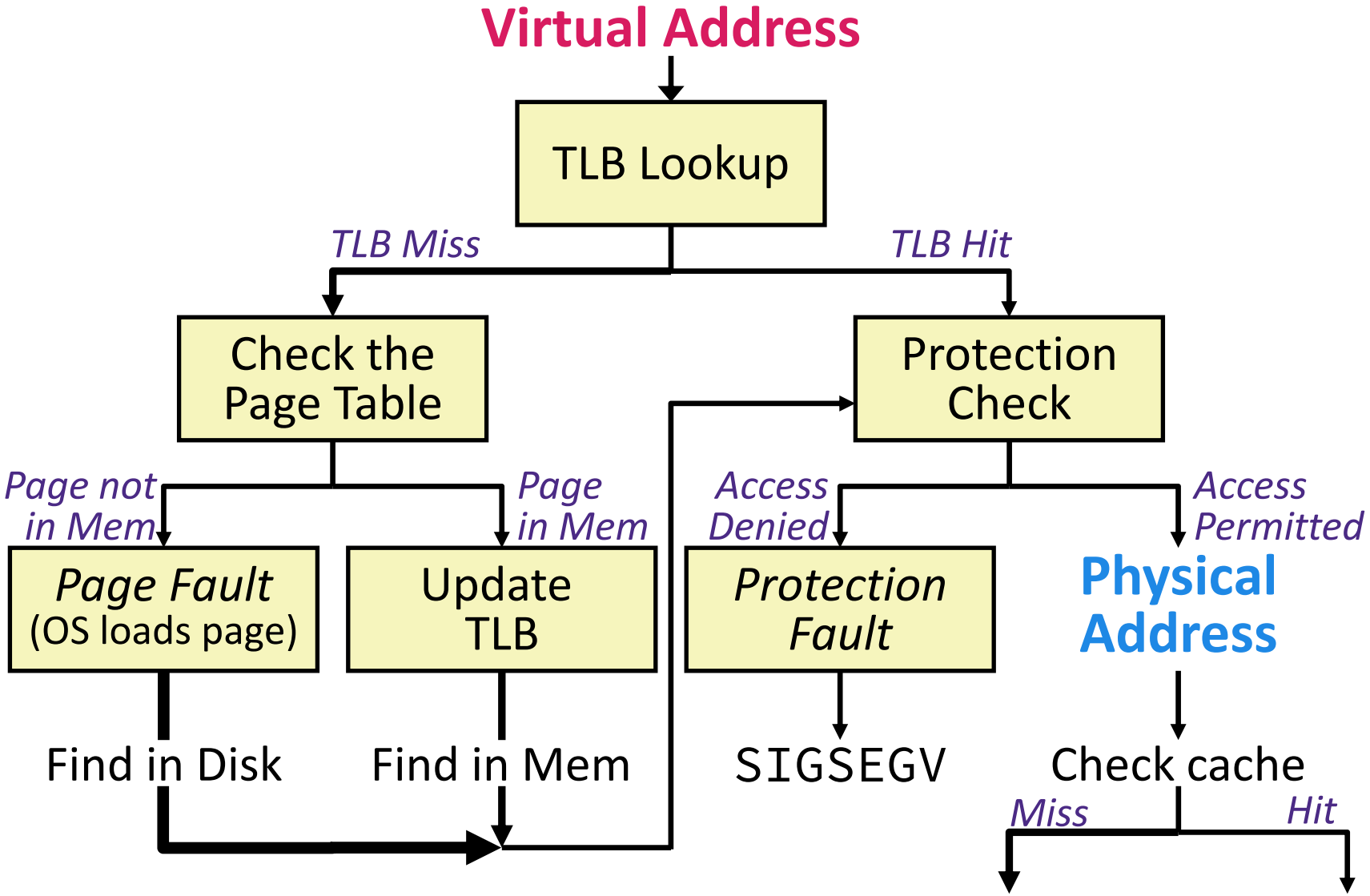
- A TLB miss incurs an additional memory access (the PTE)
  - Fortunately, TLB misses are rare

Fetching Data on a Memory Read

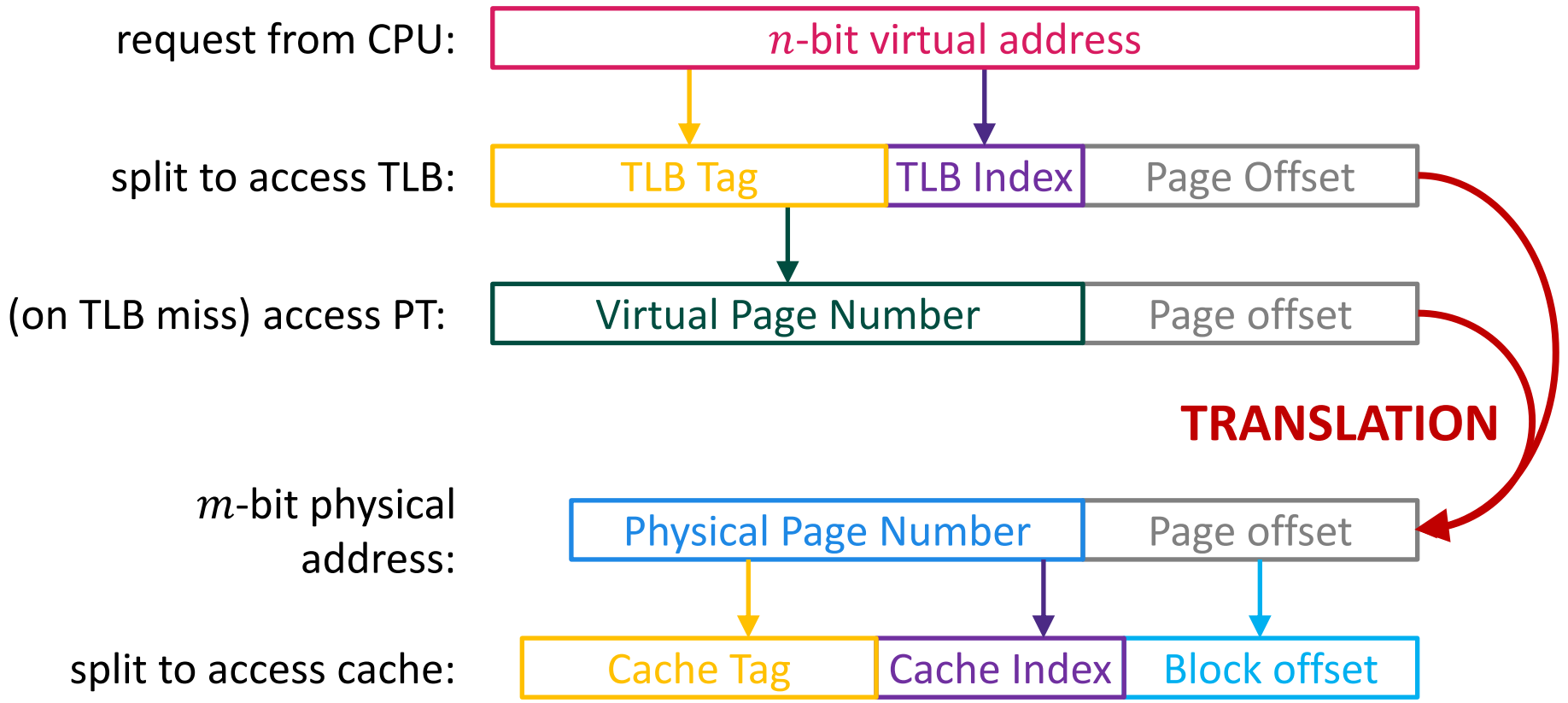
1) Address Translation (check TLB)
   - Input: VPN, Output: PPN
   - *TLB Hit:* Fetch translation, return PPN
   - *TLB Miss:* Check page table (in memory)
     - *Page Table Hit:* Load page table entry into TLB
     - *Page Fault:* Fetch page from disk to memory, update corresponding page table entry, then load entry into TLB
2) Fetch Data (check cache)
   - Input: physical address, Output: data
   - *Cache Hit:* Return data value to processor
   - *Cache Miss:* Fetch data value from memory, store it in cache, return it to processor

these serve different purposes!

Address Translation



Address Manipulation



Context Switching Revisited

- What needs to happen when the CPU switches processes?
  - Registers:
    - Save state of old process, load state of new process
    - Including the Page Table Base Register (PTBR)
  - Memory:
    - Nothing to do! Pages for processes already exist in memory/disk and protected from each other
  - TLB:
    - *Invalidate* all entries in TLB – mapping is for old process' VAs
  - Cache:
    - Can leave alone because storing based on PAs – good for shared data

Lecture Outline (2/3)

- Translation Lookaside Buffer
- **Memory Translation Examples**
- For Fun: DRAMMER Security Attack

Summary of Address Translation Symbols

- Basic Parameters
  - $\mathrm{N} = 2^n$ Number of addresses in virtual address space
  - $\mathrm{M} = 2^m$ Number of addresses in physical address space
  - $\mathrm{P} = 2^p$ Page size (bytes)
- Components of the virtual address (VA)
  - **VPO** Virtual page offset
  - **VPN** Virtual page number
  - **TLBI** TLB index
  - **TLBT** TLB tag
- Components of the physical address (PA)
  - **PPO** Physical page offset (same as VPO)
  - **PPN** Physical page number

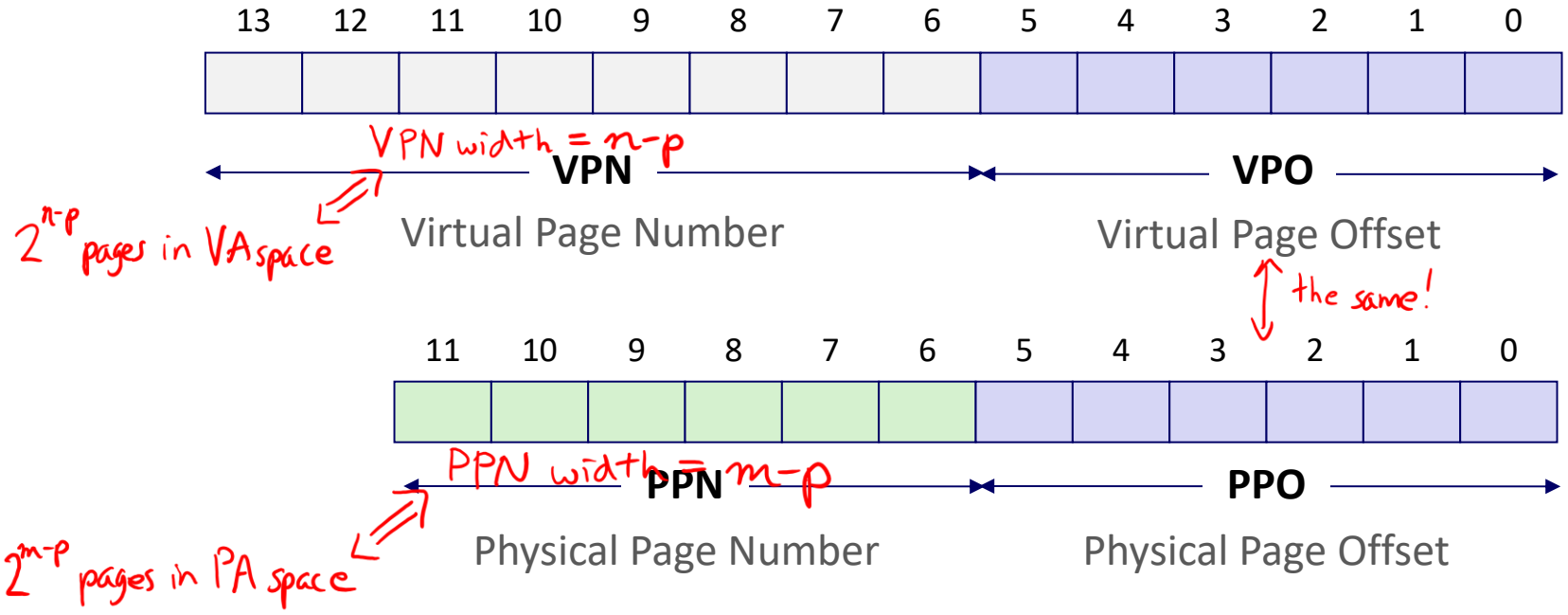
Small Memory System Example

- Addressing
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes

$n = 14$ bits $\iff$ $N = 16$ KiB VA space

$m = 12$ bits $\iff$ $M = 4$ KiB PA space

$P = 64$ B $\iff$ $p = 6$ bits



Small Memory System: Page Table

- Only showing first 16 entries (out of $2^8 = 256$) 2^{n-p} one for every virtual page
  - **Note:** showing 2 hex digits for PPN even though only 6 bits
  - **Note:** other management bits not shown, but part of PTE

(D, R, W, X)

VPN	Valid	PPN
0	1	28
1	0	–
2	1	33
3	1	02
4	0	–
5	1	16
6	0	–
7	0	–

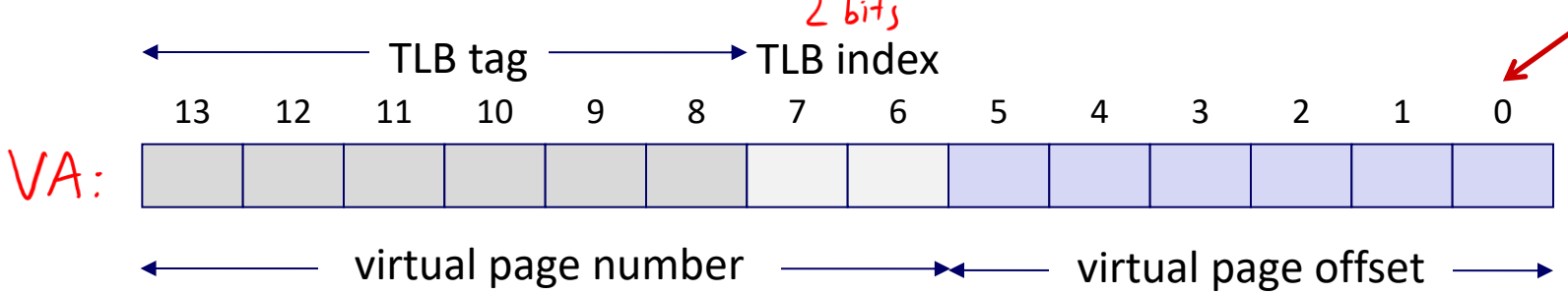
VPN	Valid	PPN
8	1	0x13
9	1	17
A	1	09
B	0	–
C	0	–
D	1	2D
E	0	–
F	1	0D
⋮	⋮	⋮

Small Memory System: TLB

- 16 entries total
- 4-way set associative

16/4 = 4 sets

2 bits



Why does the TLB ignore the page offset?

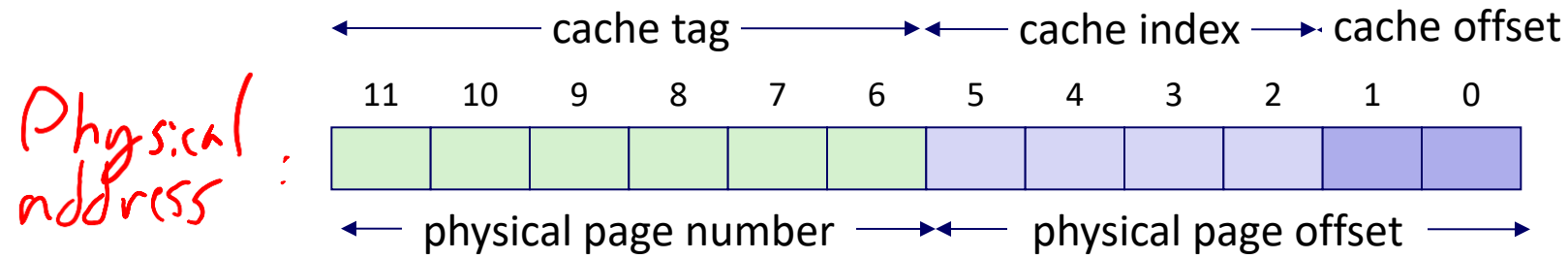
not part of its job! (address translation)

	Way 0			Way 1			Way 2			Way 3		
Set	***Tag***	***PPN***	***V***	***Tag***	***PPN***	***V***	***Tag***	***PPN***	***V***	***Tag***	***PPN***	***V***
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Small Memory System: Cache

Note: It is just coincidence that the PPN is the same width as the cache Tag

- Direct-mapped with K = 4 B, C/K = 16
- Physically addressed



Physical address:

Index	Valid	Tag	B0	B1	B2	B3
0	1	19	99	11	23	11
1	0	15	–	–	–	–
2	1	1B	00	02	04	08
3	0	36	–	–	–	–
4	1	32	43	6D	8F	09
5	1	0D	36	72	F0	1D
6	0	31	–	–	–	–
7	1	16	11	C2	DF	03

Index	Valid	Tag	B0	B1	B2	B3
8	1	24	3A	00	51	89
9	0	2D	–	–	–	–
A	1	2D	93	15	DA	3B
B	0	0B	–	–	–	–
C	0	12	–	–	–	–
D	1	16	04	96	34	15
E	1	13	83	77	1B	D3
F	0	14	–	–	–	–

Current State of Small Memory System

Circled #s refer to Memory Request Example #

TLB:

Set	V	Tag	PPN	V	Tag	PPN	V	Tag	PPN	V	Tag	PPN
③ 0	0	03	–	1	09	0D	0	00	–	1	07	02
④ 1	1	03	2D	0	02	–	0	04	–	0	0A	–
② 2	0	02	–	0	08	–	0	06	–	0	03	–
① 3	0	07	–	1	03	0D	1	0A	34	0	02	–

Page table (partial):

VPN	V	PPN	VPN	V	PPN
③ 0	1	28	8	1	13
1	0	–	9	1	17
2	1	33	A	1	09
3	1	02	B	0	–
4	0	–	C	0	–
5	1	16	D	1	2D
6	0	–	② E	0	–
7	0	–	F	1	0D

Cache:

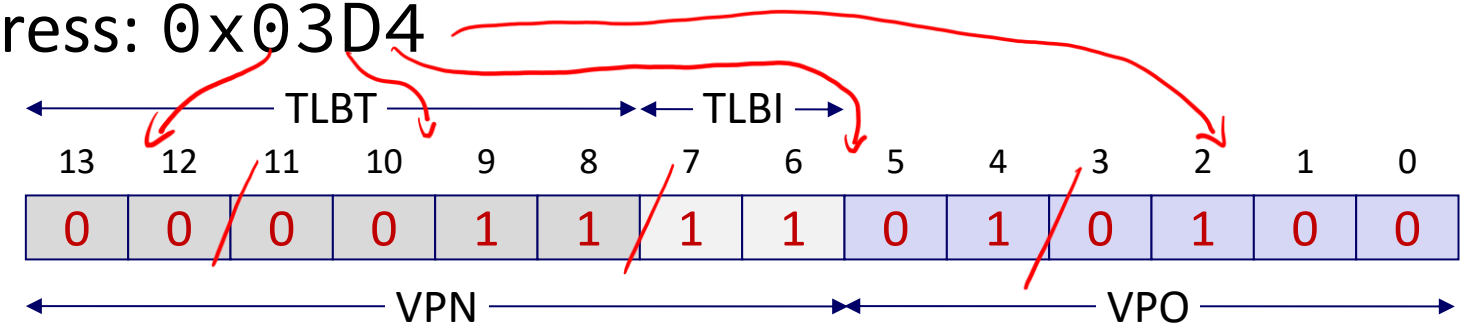
Index	V	Tag	B0	B1	B2	B3
0	1	19	99	11	23	11
1	0	15	–	–	–	–
2	1	1B	00	02	04	08
3	0	36	–	–	–	–
4	1	32	43	6D	8F	09
① 5	1	0D	36	72	F0	1D
6	0	31	–	–	–	–
7	1	16	11	C2	DF	03

Index	V	Tag	B0	B1	B2	B3
③ 8	1	24	3A	00	51	89
9	0	2D	–	–	–	–
④ A	1	2D	93	15	DA	3B
B	0	0B	–	–	–	–
C	0	12	–	–	–	–
D	1	16	04	96	34	15
E	1	13	83	77	1B	D3
F	0	14	–	–	–	–

Memory Request Example #1

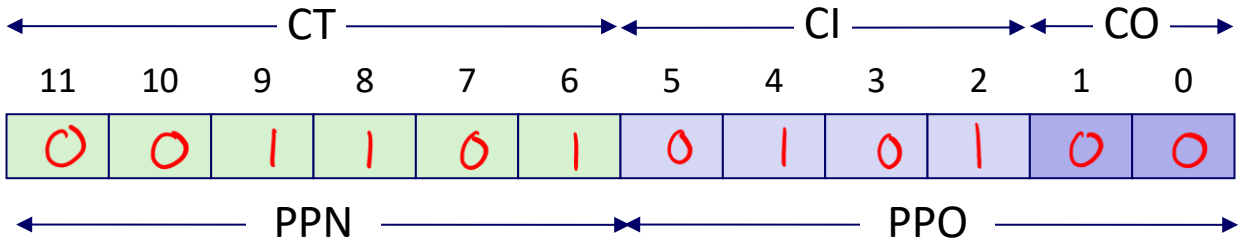
Note: It is just coincidence that the PPN is the same width as the cache Tag

- Virtual Address: 0x03D4



VPN 0xF (check this entry of the page table) TLBT 0x03 (look for this tag within TLB set) TLBI 3 (check this set of the TLB) TLB Hit? Y Page Fault? N PPN 0x0D

- Physical Address:

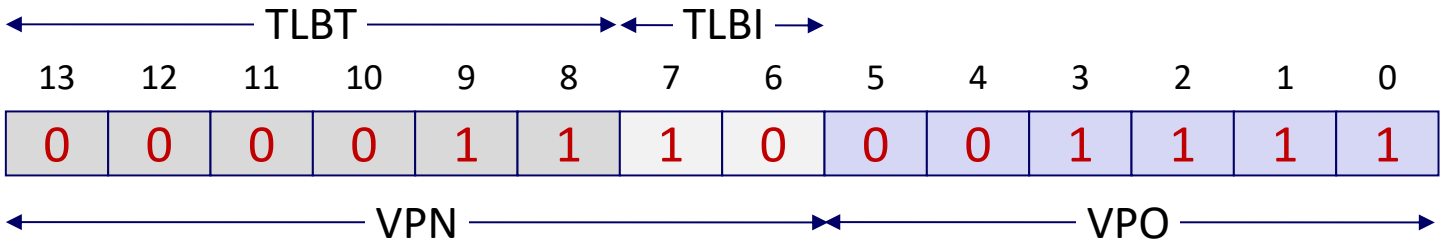


CT 0x0D (look for this tag within cache set) CI 5 (check this set of the cache) CO 0 (which byte of cache block) Cache Hit? Y Data (byte) 0x36

Memory Request Example #2

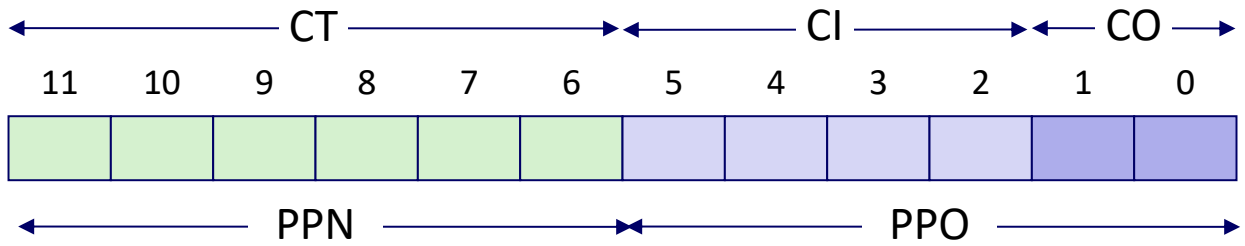
Note: It is just coincidence that the PPN is the same width as the cache Tag

- Virtual Address: 0x038F



VPN 0x0E TLBT 0x03 TLBI 2 TLB Hit? N Page Fault? Y PPN n/a

- Physical Address:

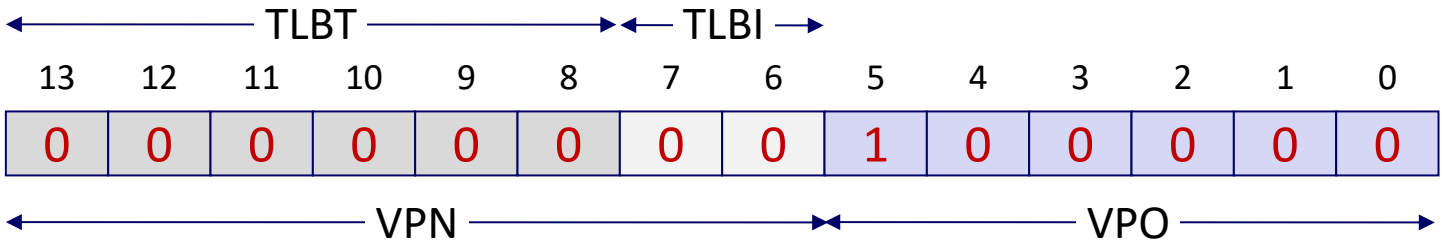


CT _____ CI _____ CO _____ Cache Hit? ___ Data (byte) _____

Memory Request Example #3

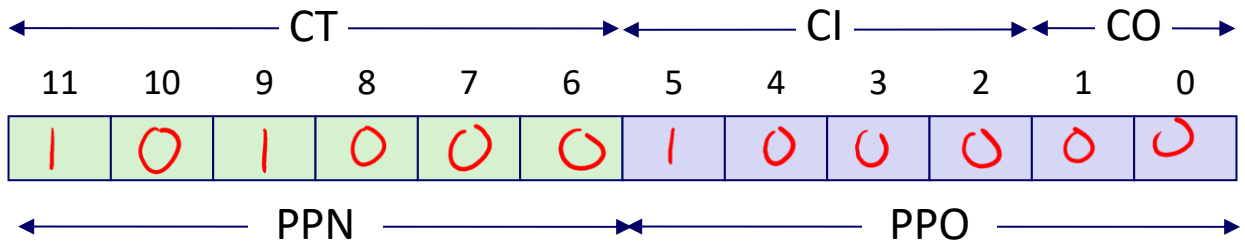
Note: It is just coincidence that the PPN is the same width as the cache Tag

- Virtual Address: 0x0020



VPN 0x00 TLBT 0x00 TLBI 0 TLB Hit? N Page Fault? N PPN 0x28

- Physical Address:

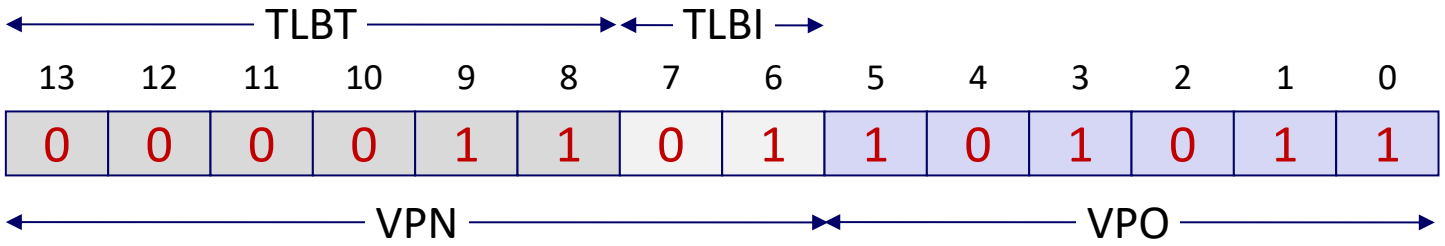


CT 0x28 CI 8 CO 0 Cache Hit? N Data (byte) n/a

Memory Request Example #4

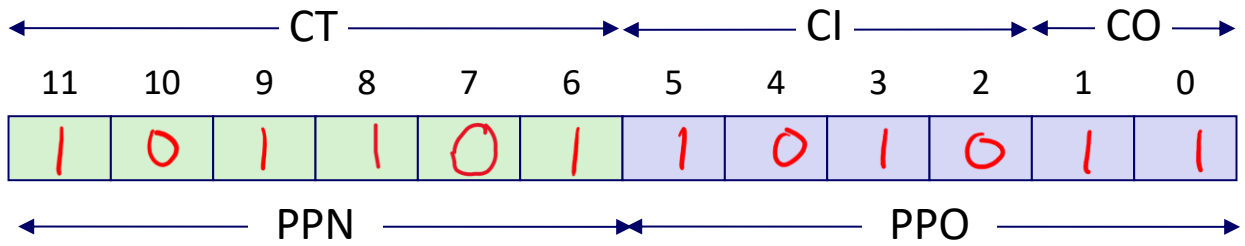
Note: It is just coincidence that the PPN is the same width as the cache Tag

- Virtual Address: 0x036B



VPN 0x0D TLBT 0x03 TLBI 1 TLB Hit? Y Page Fault? N PPN 0x2D

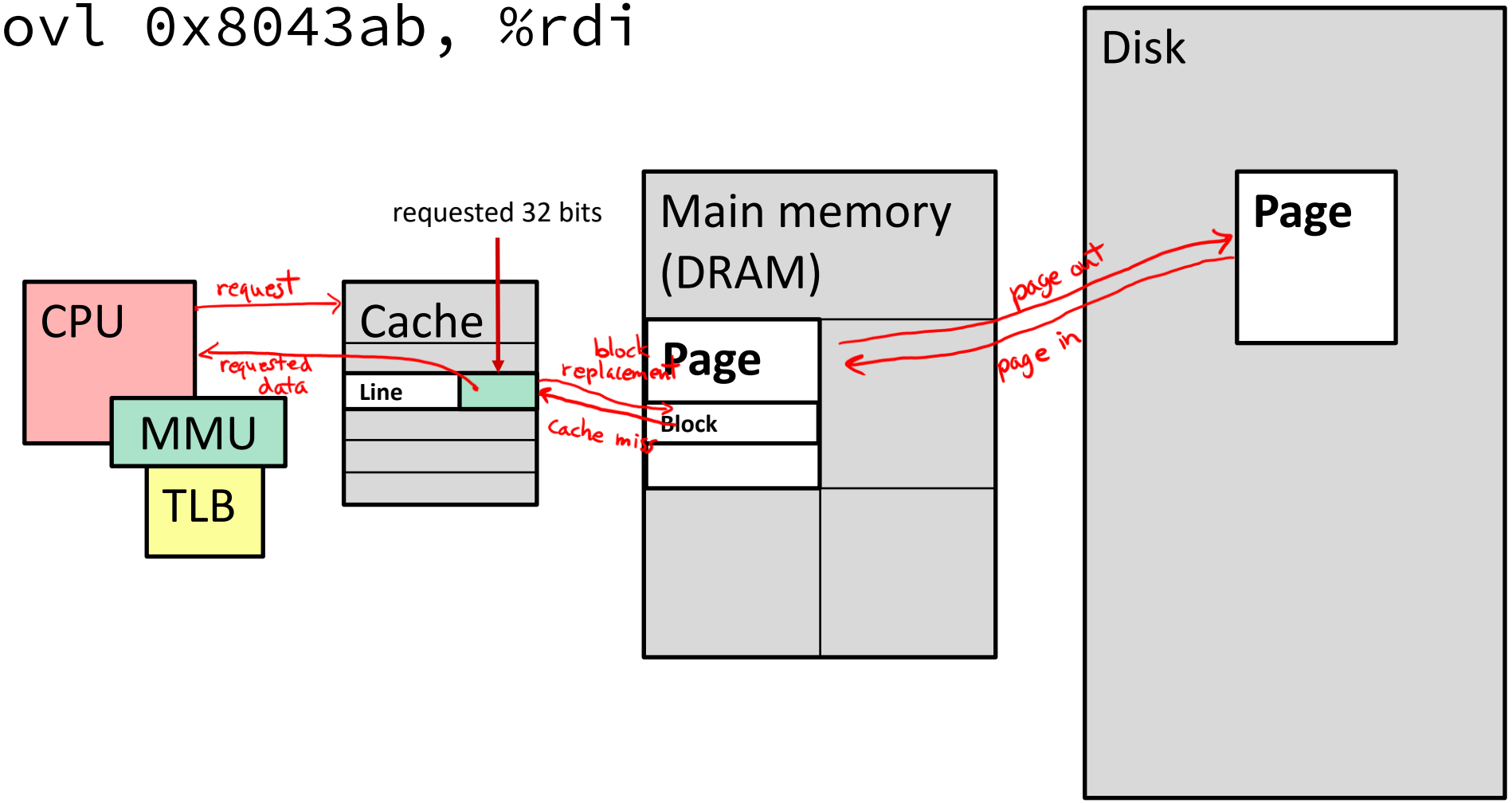
- Physical Address:



CT 0x2D CI A CO 3 Cache Hit? Y Data (byte) 0x3B

Memory Overview (Data Flow)

- `movl 0x8043ab, %rdi`



Lecture Outline (3/3)

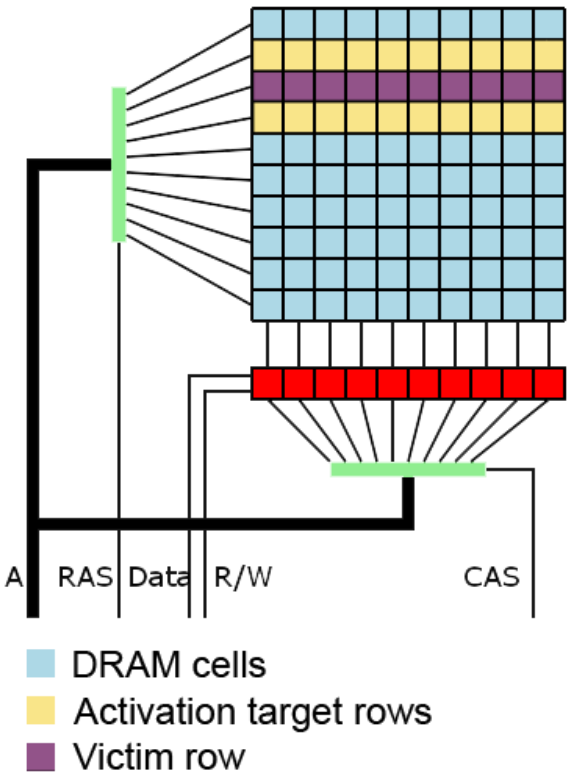
- Translation Lookaside Buffer
- Memory Translation Examples
- **For Fun: DRAMMER Security Attack**

DRAMMER Security Attack

- **Relevance:** Uses your system's memory setup to gain elevated privileges
  - Ties together some of what we've learned about virtual memory and processes
- **Interest:** It's a software attack that uses *only hardware vulnerabilities* and requires *no user permissions*
- **Recent:** DRAMMER announced in October 2016; latest attack variant (Half Double) announced May 2021
  - Other recent variants of underlying vulnerability (row hammer) as recent as 2018

Underlying Vulnerability: Row Hammer

- Dynamic RAM (DRAM) has gotten denser over time
  - DRAM cells physically closer and use smaller charges
  - More susceptible to "*disturbance errors*" (interference)
- DRAM capacitors need to be "refreshed" periodically (~64 ms)
  - Lose data when loss of power
  - Capacitors accessed in rows
- Rapid accesses to one row can flip bits in an adjacent row!
  - ~ 100K to 1M times



By Dsimic (modified), CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=38868341

Row Hammer Exploit

- Force constant memory access
  - Read then flush the cache
  - `clflush` – flush cache line
    - Invalidates cache line containing the specified address
    - Not available in all machines or environments
  - Want addresses X and Y to fall in activation target row(s)
    - Good to understand how *banks* of DRAM cells are laid out
- The row hammer effect was discovered in 2014
  - Only works on certain types of DRAM (2010 onwards)
  - These techniques target x86 machines

```
hammertime:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  jmp hammertime
```

Consequences of Row Hammer

- Row hammering process can affect another process via memory
  - Circumvents virtual memory protection scheme
  - Memory needs to be in an adjacent row of DRAM

- Worse: privilege escalation
  - Page tables live in memory!
  - Hope to change PPN to access other parts of memory, or change permission bits
  - **Goal:** gain read/write access to a page containing a page table, hence granting process read/write access to *all of physical memory*

Effectiveness?

- Doesn't seem so bad – random bit flip in a row of physical memory
  - Vulnerability affected by system setup and physical condition of memory cells
- **Improvements:**
  - Double-sided row hammering increases speed & chance
  - Do system identification first (*e.g.*, Lab 4)
    - Use timing to infer memory row layout & find "bad" rows
    - Allocate a huge chunk of memory and try many addresses, looking for a reliable/repeatable bit flip
  - Fill up memory with page tables first
    - `fork` extra processes; hope to elevate privileges in any page table

What's DRAMMER?

- No one previously made a huge fuss
  - **Prevention:** error-correcting codes, target row refresh, higher DRAM refresh rates
  - Often relied on special memory management features
  - Often crashed system instead of gaining control
- Research group found a *deterministic* way to induce row hammer exploit in a non-x86 system (ARM)
  - Relies on predictable reuse patterns of standard physical memory allocators
  - Universiteit Amsterdam, Graz University of Technology, and University of California, Santa Barbara

DRAMMER Demo Video

- https://youtu.be/x6hL-obNhAw
  - It's a shell, so not that sexy-looking, but still interesting
  - Apologies that the text is so small on the video

How did we get here?

- Computing industry demands more and faster storage with lower power consumption
- Ability of user to circumvent the caching system
  - `clflush` is an unprivileged instruction in x86
  - Other commands exist that skip the cache
- Availability of virtual to physical address mapping
  - **Example:** `/proc/self/pagemap` on Linux (not human-readable)

- Google patch for Android (Nov. 8, 2016)
  - Patched the ION memory allocator

More reading for those interested

- DRAMMER paper: https://vvdveen.com/publications/drammer.pdf
- Google Project Zero: https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html
- First rowhammer paper: https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf
- Latest non-uniform, frequency-based exploit: https://comsec.ethz.ch/research/dram/blacksmith/
- Wikipedia: https://en.wikipedia.org/wiki/Row_hammer

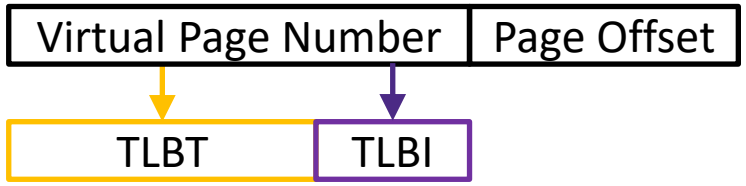
Discussion Question

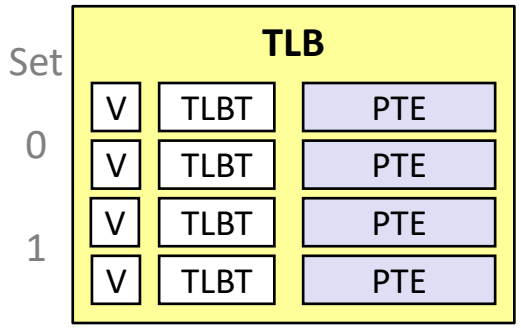
- Discuss the following question(s) in groups of 3-4 students
  - I will call on a few groups afterwards so please be prepared to share out
  - Be respectful of others' opinions and experiences

- We tend to think of software as existing in a realm abstracted away from hardware; however, this class has been about the interactions between hardware and software
  - Based on what we've learned in this class, what are some ways that your choice of hardware setup may affect your "day-to-day" as a software engineer?

Lesson Summary (1/2)

- Introduced the ***translation lookaside buffer*** (TLB) as a cache for page table entries (PTEs)
  - Try to avoid accessing the page table in memory
  - Split VPN into **TLB Tag** and **TLB Index** based on # of sets in TLB
  - Management bits include valid (like $, not PT) and TLB tag





Lesson Summary (2/2)

- The virtual memory system is complicated, but also elegant and effective
  - Level of indirection to provide isolated memory & caching
  - TLB as a cache of page tables avoids two trips to memory for every memory access