CENG 3420
Computer Organization & Design
Lecture 14: Virtual Memory

Textbook: Chapter 5.7

Zhengrong Wang

CSE Department, CUHK

zhengrongwang@cuhk.edu.hk

Virtual Memory

Motivation

- Physical memory may not be as large as "possible address space" spanned by a processor, e.g.
  - A processor can address 4G bytes with 32-bit address.
  - But installed main memory may only be 1GB.

- How if we want to simultaneously run many programs which require a total memory consumption greater than the installed main memory capacity?

- Terminology:
  - A running program is called a process or a thread
  - Operating System (OS) controls the processes

Virtual Memory

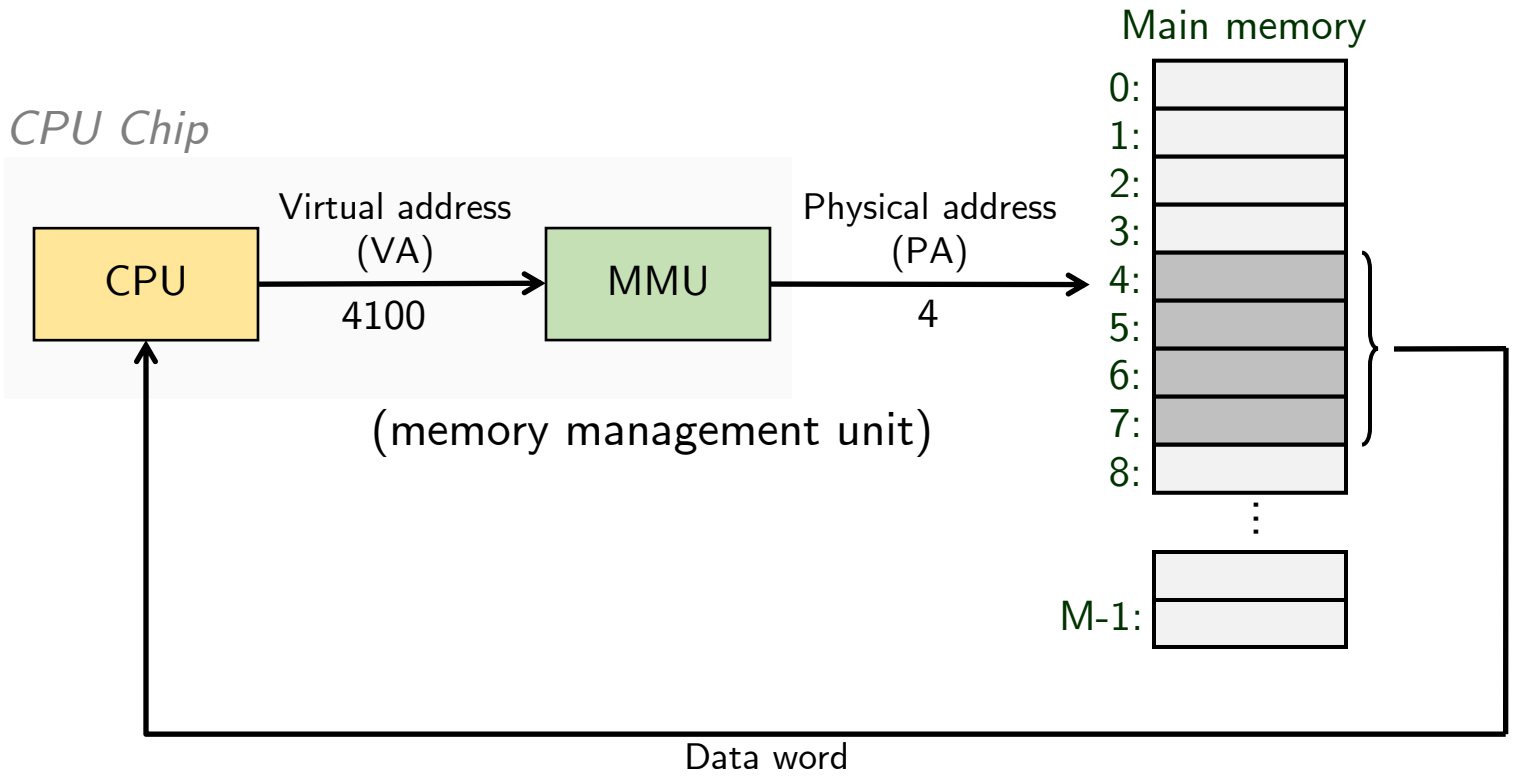
- Use main memory as a "cache" for secondary memory
- Each program is compiled into its own virtual address space
- What makes it work? Principle of Locality

- Why virtual memory?
  - During run-time, virtual address is translated to a physical address
  - Efficient & safe sharing memory among multiple programs
  - Ability to run programs larger than the size of physical memory
  - Code relocation: code can be loaded anywhere in main memory

Bottom of the Memory Hierarchy

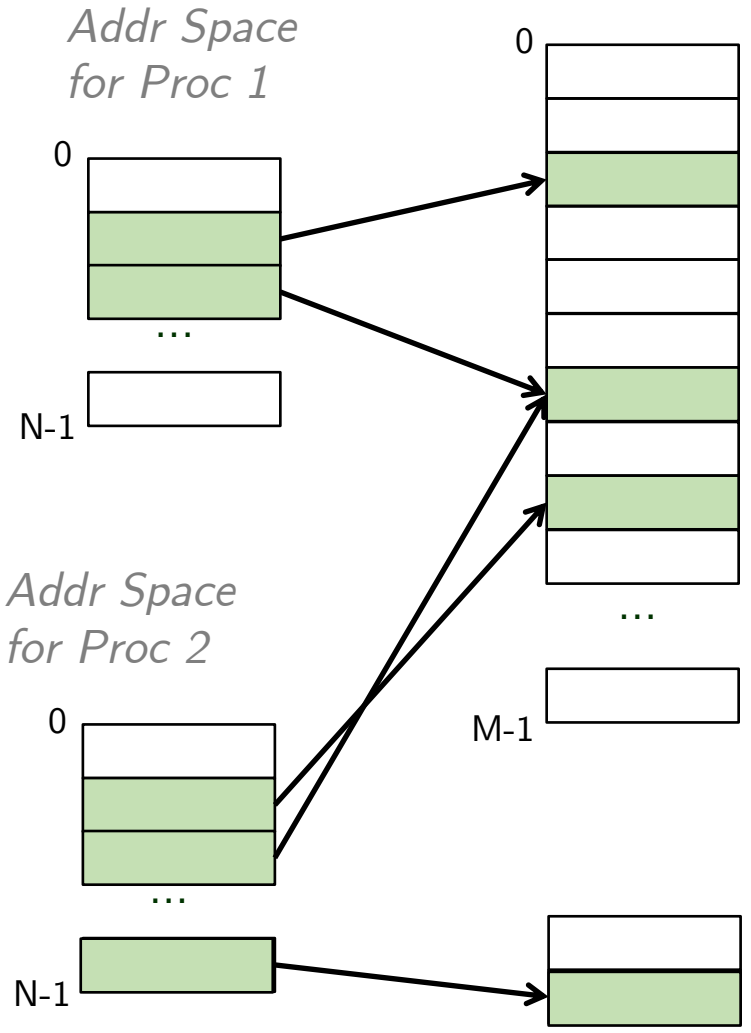
- Consider the following example:
  - Suppose we hit the limit of 1GB in the example, and we suddenly need some more memory on the fly.
  - We move some main memory chunks to the hard disk, say, 100MB.
  - So, we have 100MB of "free" main memory for use.
  - What if later on, those instructions / data in the saved 100MB chunk are needed again?
  - We have to "free" some other main memory chunks in order to move the instructions/ data back from the hard disk.

High Level Idea



Overview and A Few Details

- Paging
  - Logically divide memory into chunks called "Pages"
  - Make Decision of where to put memory on Page Granularity
  - **Size: typically 4 KB, sometimes larger (2 MB)**
- Flexible Translation
  - Any virtual page (VP) can be mapped to any physical page (PP)
    - Fully associative
    - Requires a "large" mapping function – different from cache memories
    - Too complicated and open-ended to be implemented in hardware
  - Some pages mapped to disk if you run out of DRAM



Terminology in Virtual Memory

- VA and PA: virtual and physical address.
- VPN and PPN: virtual and physical page number.
- VPO and PPO: virtual and physical page offset.

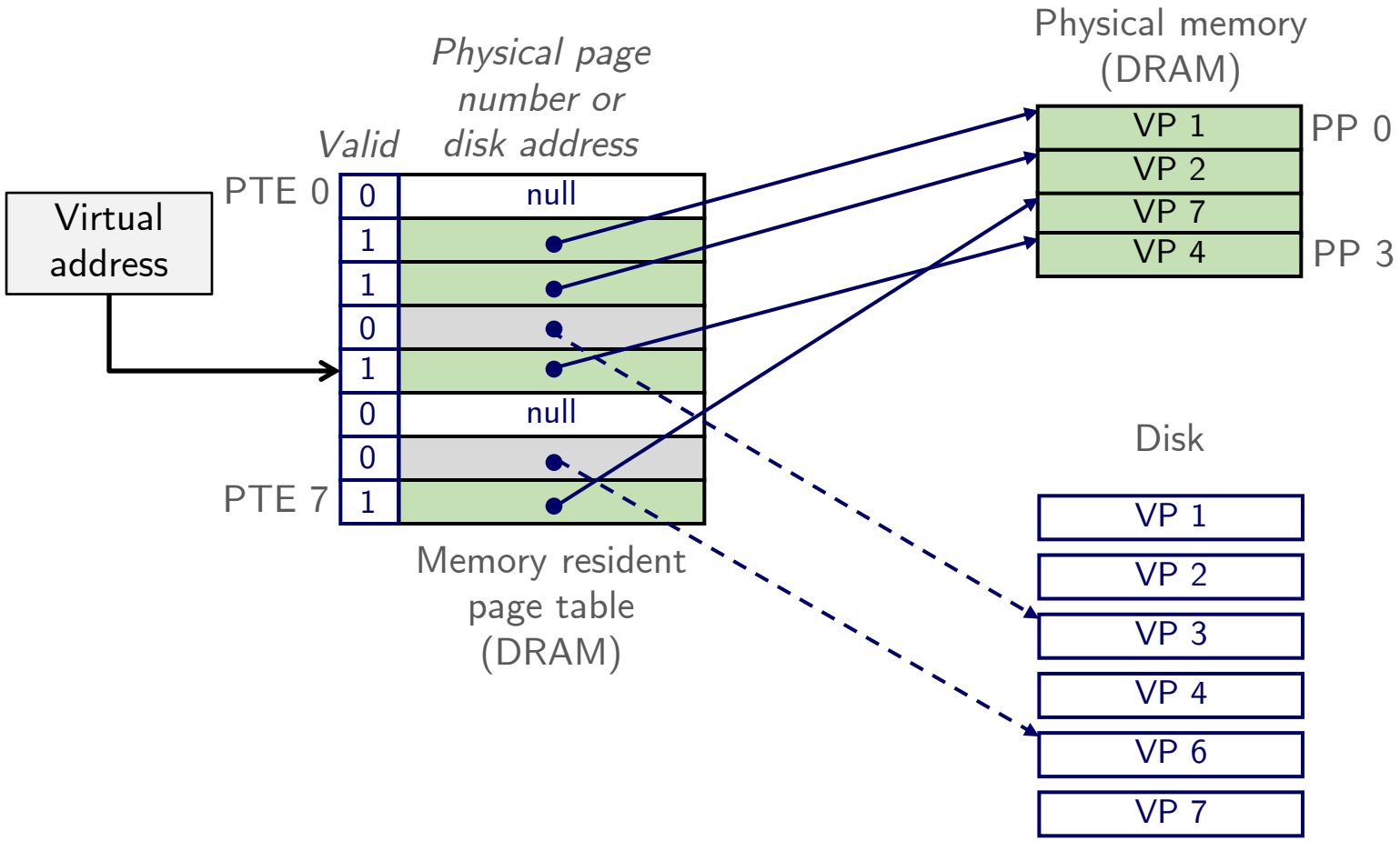
E.g. VA=0x1234BEEF → PA=0x457EEF, Page size 4KiB.

- Because page size is 4KiB, it takes 12 bits.
- Bit [0, 11] is page offset, so VPO=PPO=0xEEF. (VPO always equals PPO).
- From bit 12 to MSB is page number, so VPN=0x1234B, PPN=0x457E.

We are essentially translating the page number.

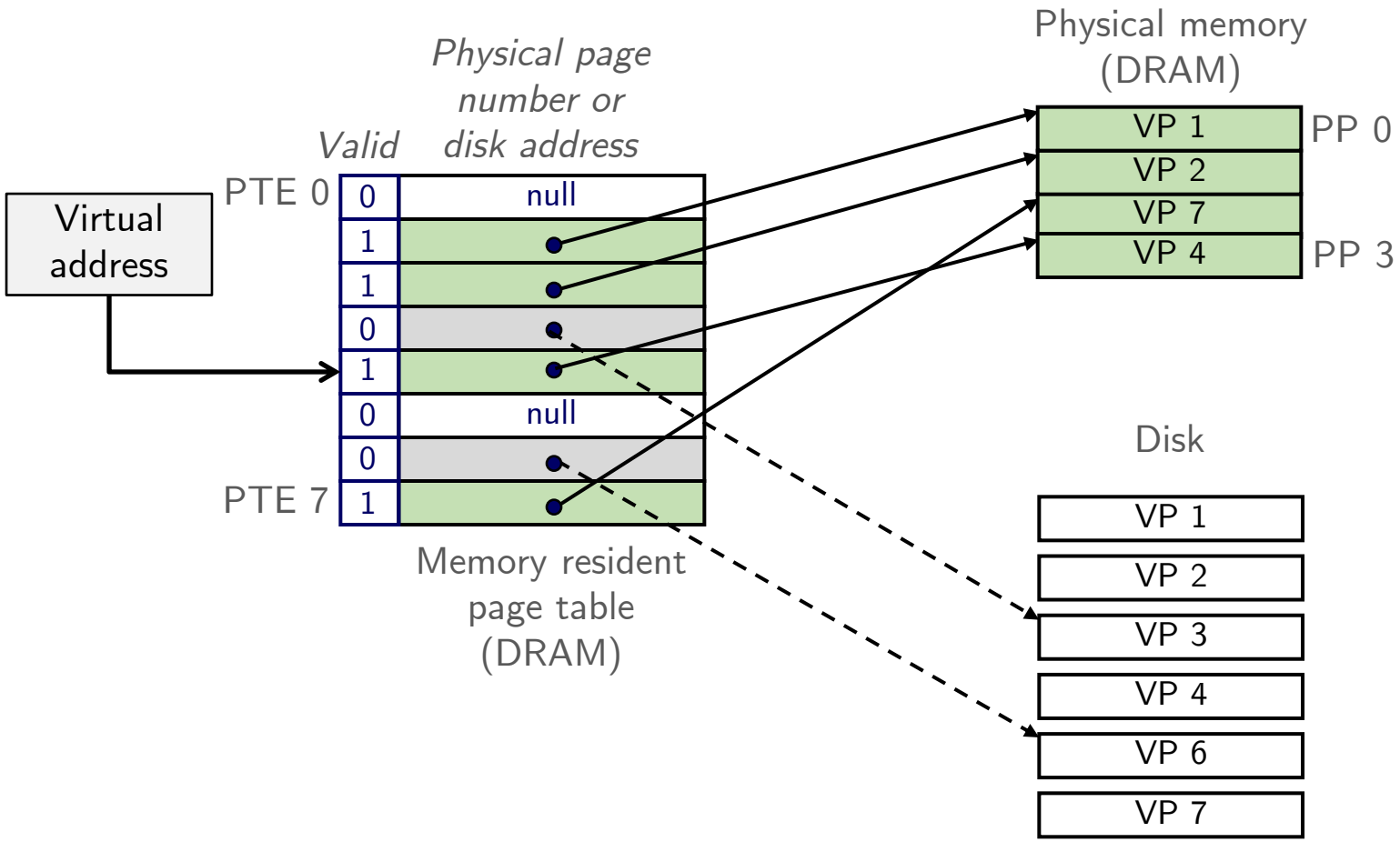
Enabling Data Structure: Page Table

- A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages.
  - Per-process kernel data structure in DRAM



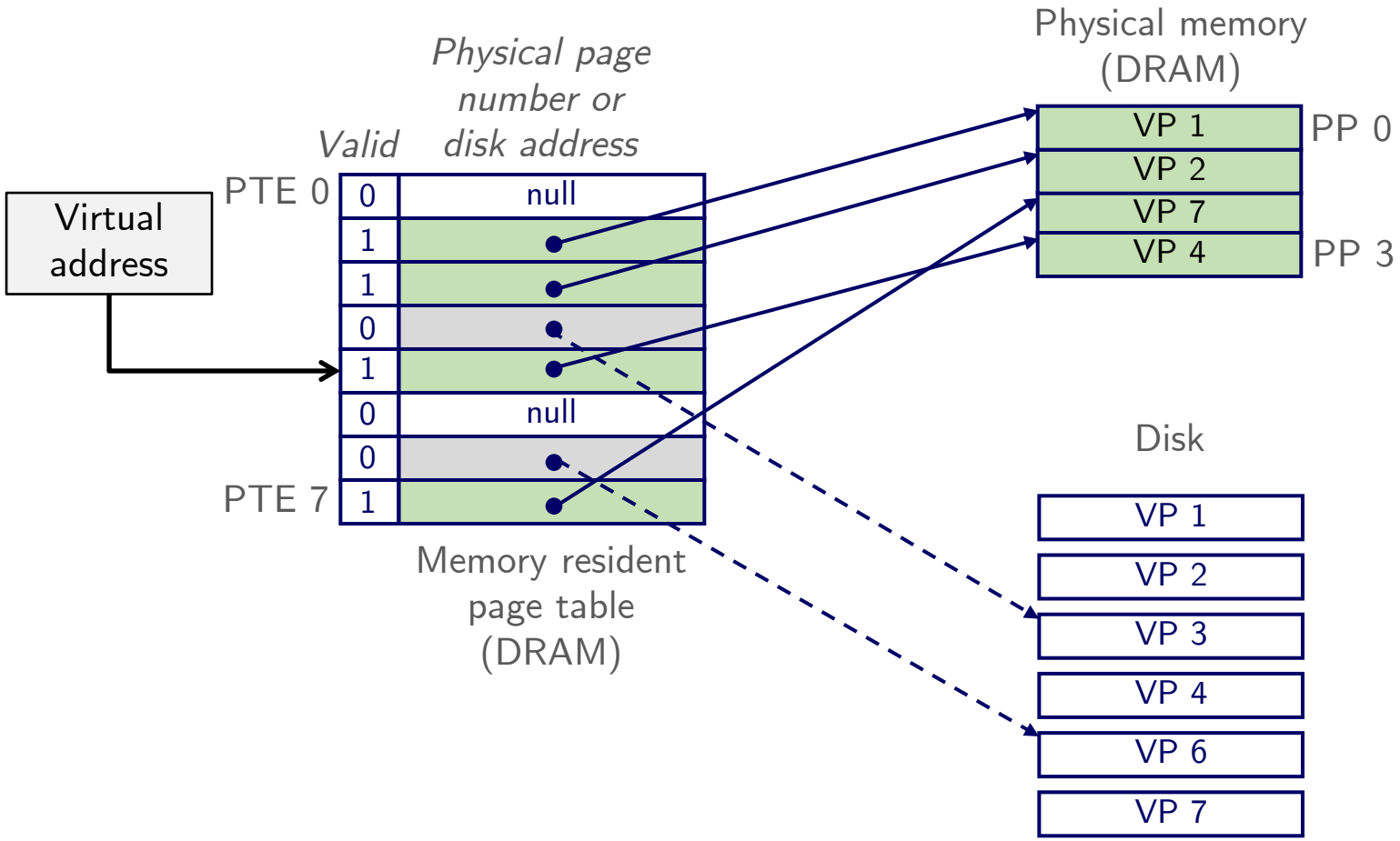
Page Hit

- *Page hit:* reference to VM word that is in physical memory (DRAM cache hit)



Page Fault

- *Page fault:* reference to VM word that is not in physical memory (DRAM cache miss)

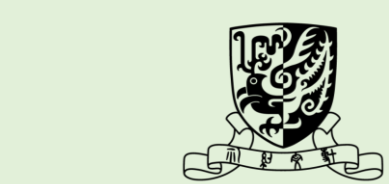


Address Translation Mechanism

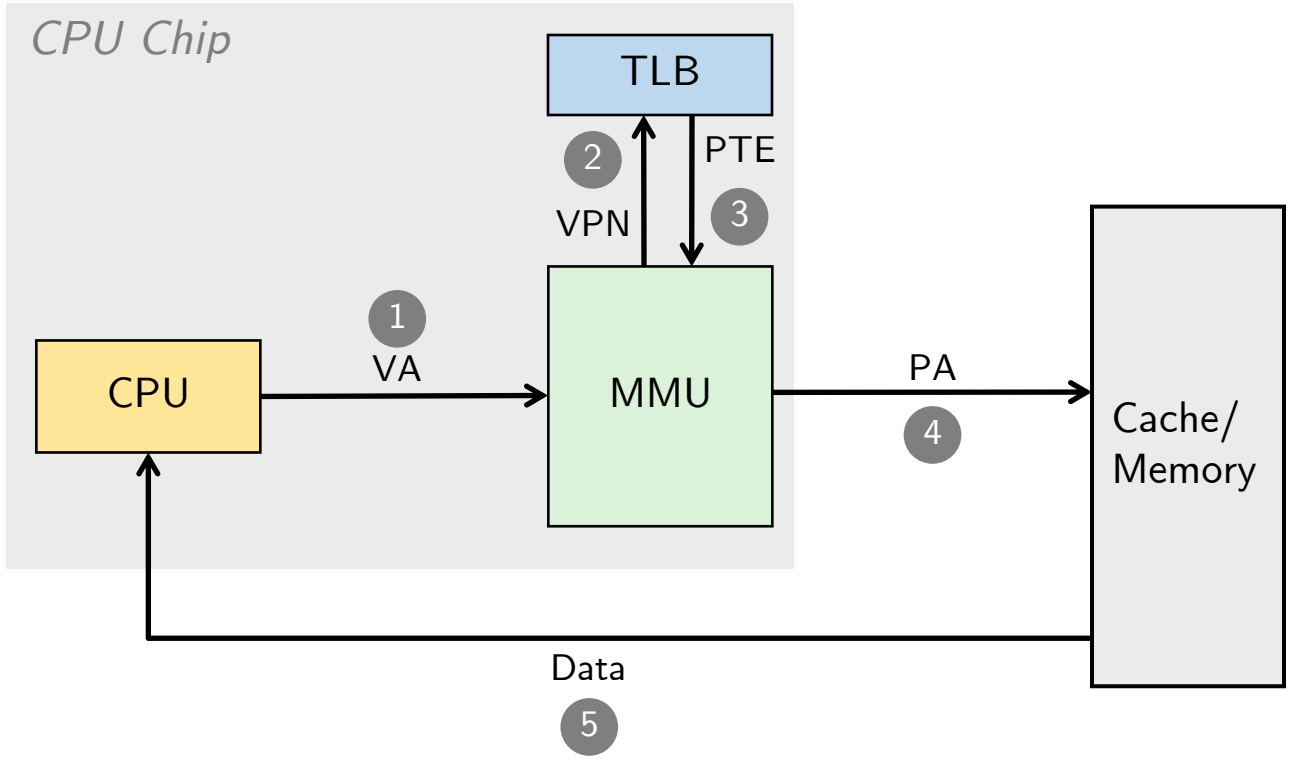
- Processor generates virtual addresses
  - MS (high order) bits are the virtual page number
  - LS (low order) bits are the offset
- Information about where each page is stored is maintained in a data structure in the main memory called the page table
  - Starting address of the page table is stored in a page table base register
  - Address in physical memory is obtained by indexing the virtual page number from the page table base register
- Every memory access requires extra memory accesses to the page table!
  - Because page table is in main memory.
  - Wouldn't this be super slow and bad for performance? Yes...

Translation Look-aside Buffer (TLB)

- Page table is too big to fit on-chip (must be in main memory).
- But we can add a small cache to accessed page table entries.
  - Small set-associative hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages
- Avoid frequent page table lookup.

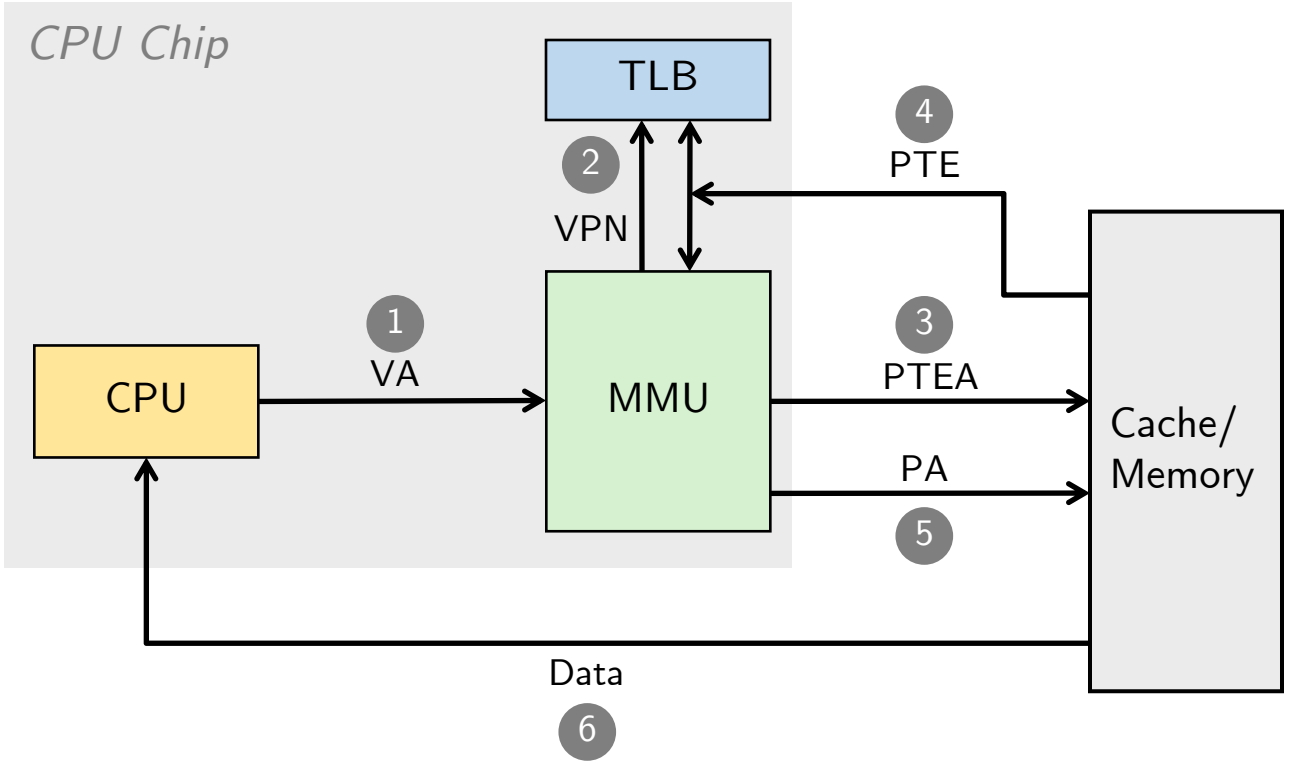


TLB Hit



A TLB hit eliminates a memory access

TLB Miss



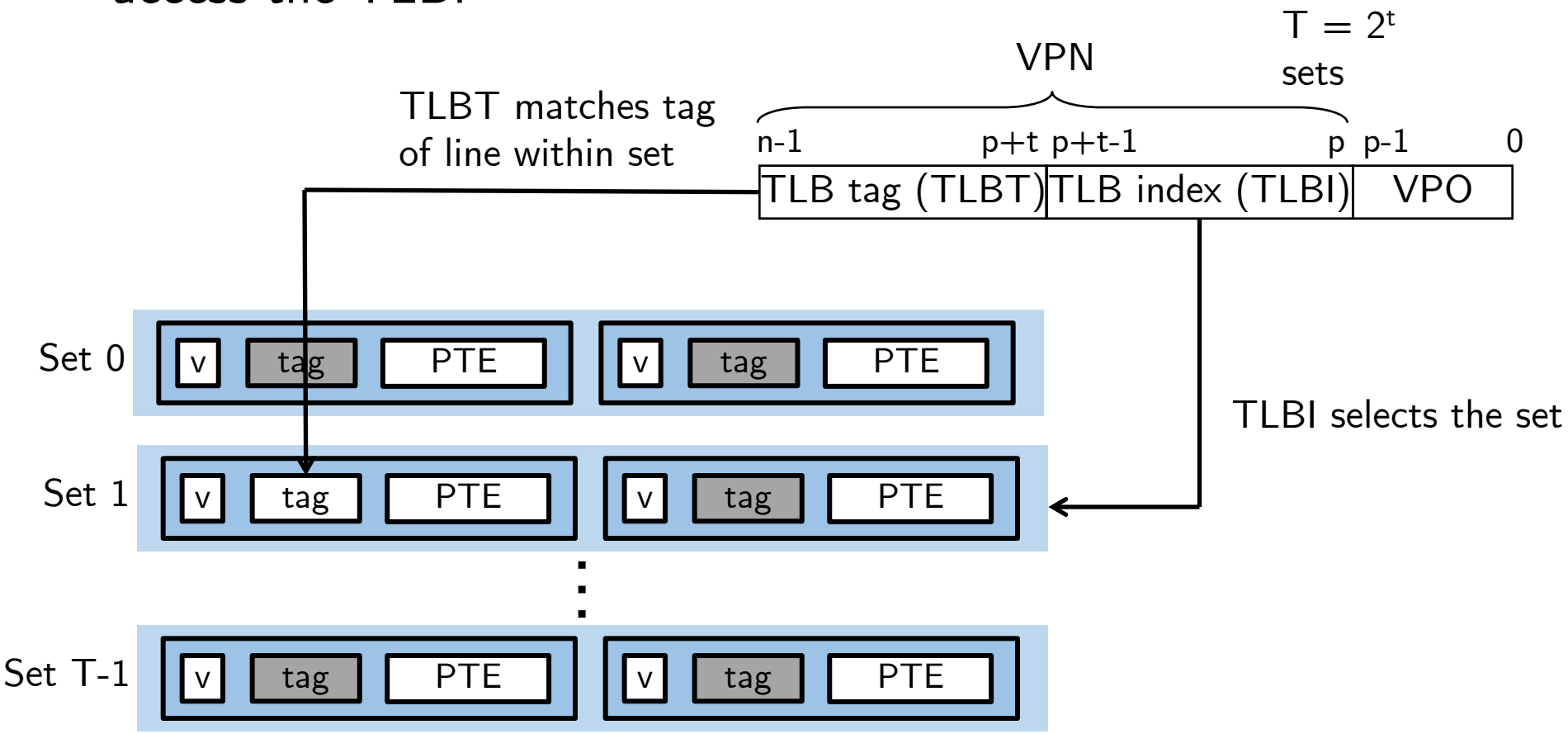
A TLB miss incurs an additional memory access (the PTE)

If the page is NOT in main memory: page fault.

Fortunately, TLB misses are rare. Why?

Accessing the TLB

- MMU uses the VPN portion of the virtual address to access the TLB:



TLB and Cache Event

- TLB/Cache miss: page table entry (PTE)/cache block not in “cache”
- Page Table miss: page NOT in memory (page fault).

TLB	Page Table	Cache	Possible? When?
Hit	Hit	Hit	
Hit	Hit	Miss	
Miss	Hit	Hit	
Miss	Hit	Miss	
Miss	Miss	Miss	
Hit	Miss	Miss/Hit	
Miss	Miss	Hit	

TLB and Cache Event

- TLB/Cache miss: page table entry (PTE)/cache block not in "cache"
- Page Table miss: page NOT in memory (page fault).

TLB	Page Table	Cache	Possible? When?
Hit	Hit	Hit	Yes – what we want!
Hit	Hit	Miss	Yes – although page table is not checked if TLB hits
Miss	Hit	Hit	Yes – TLB miss, PA in page table
Miss	Hit	Miss	Yes – TLB miss, PA in page table and data not in cache
Miss	Miss	Miss	Yes – Page fault, data is already swapped out to disk
Hit	Miss	Miss/Hit	No – TLB should be cleared if page swapped to disk (TLB shutdown)
Miss	Miss	Hit	No – Cache should also be flushed it page swapped to disk

QUESTION: Why not a VA Cache?

- Access Cache using virtual address (VA)
- Only address translation when cache misses