Virtual Memory

Muhammad Tahir

Lecture 22-23

Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

Virtual Memory and Its Importance

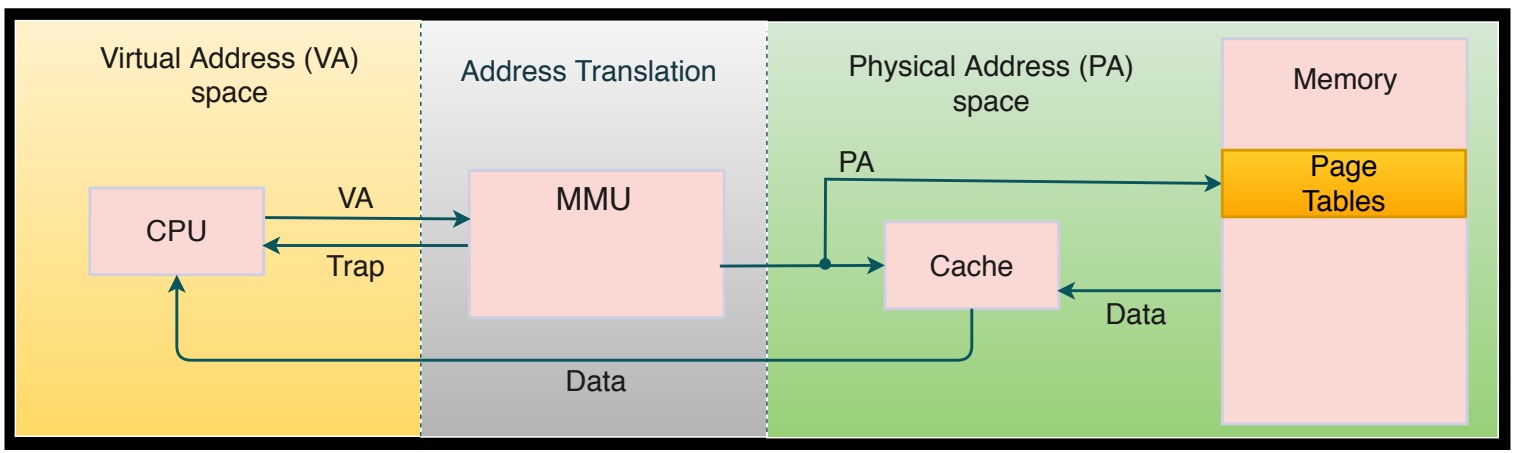
- `Virtual Memory` allows to execute a program that may not reside completely in the main memory (RAM)
- Allows efficient utilization of the available main memory resources
- Simplifies memory management
- Relieves the programmer from the burden of memory resource management

Working of Virtual Memory

- Types of memory space
  - Virtual memory space: What the program "sees"
  - Physical memory space: Where the program "resides" and executes from (size of RAM)
- On program startup
  - OS copies the program into RAM
  - If RAM is not enough, OS stops copying and starts execution (with partly loaded program in RAM)
  - When the program accesses a part not in the RAM, OS gets a page fault, and OS copies the missing part from disk to RAM

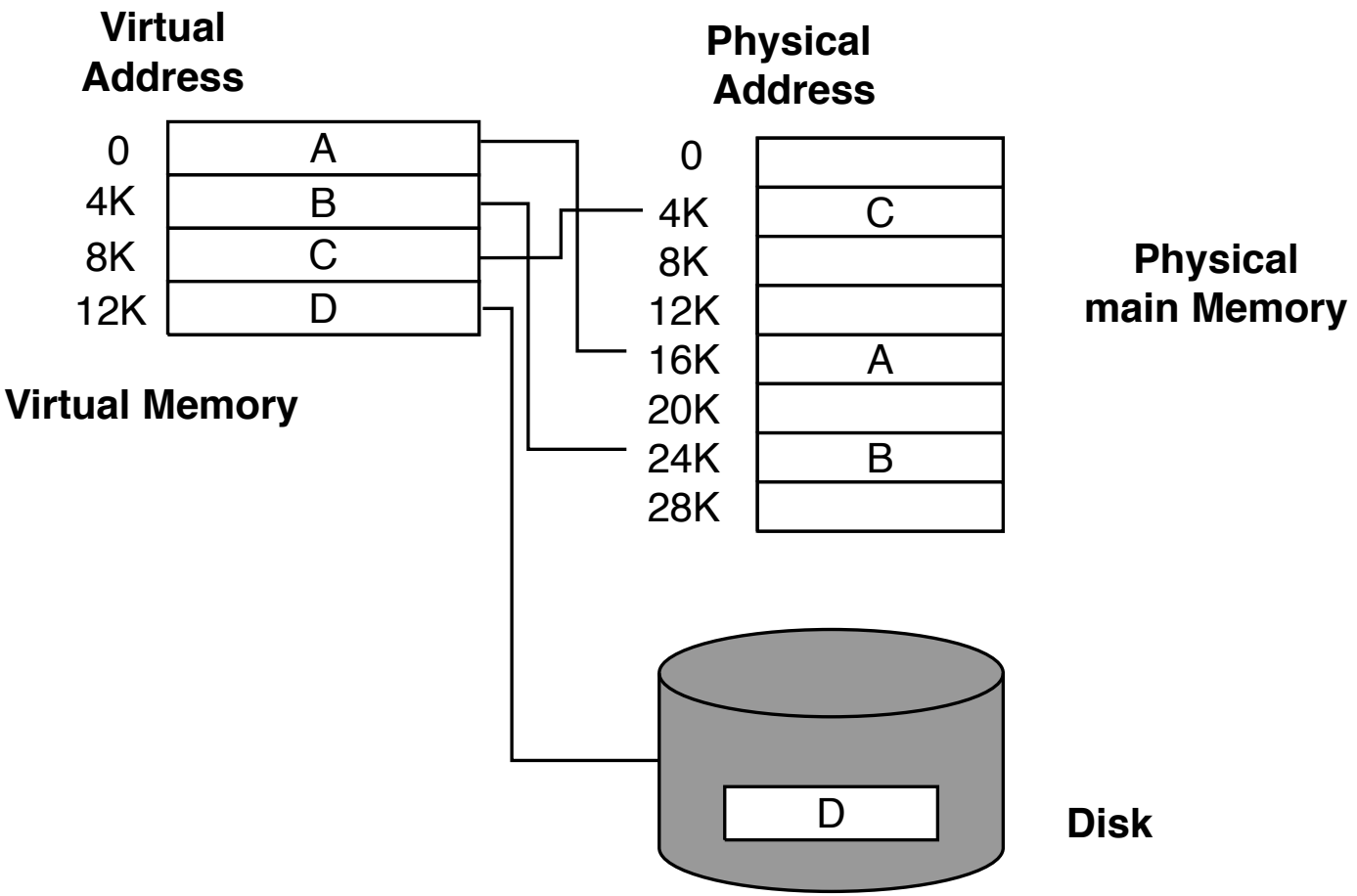
Working of Virtual Memory Cont'd

- On program startup cont'd
  - To copy the missing program page(s) from disk to RAM, OS may evict parts of the program already in the RAM
  - OS copies the evicted parts of the program back to disk



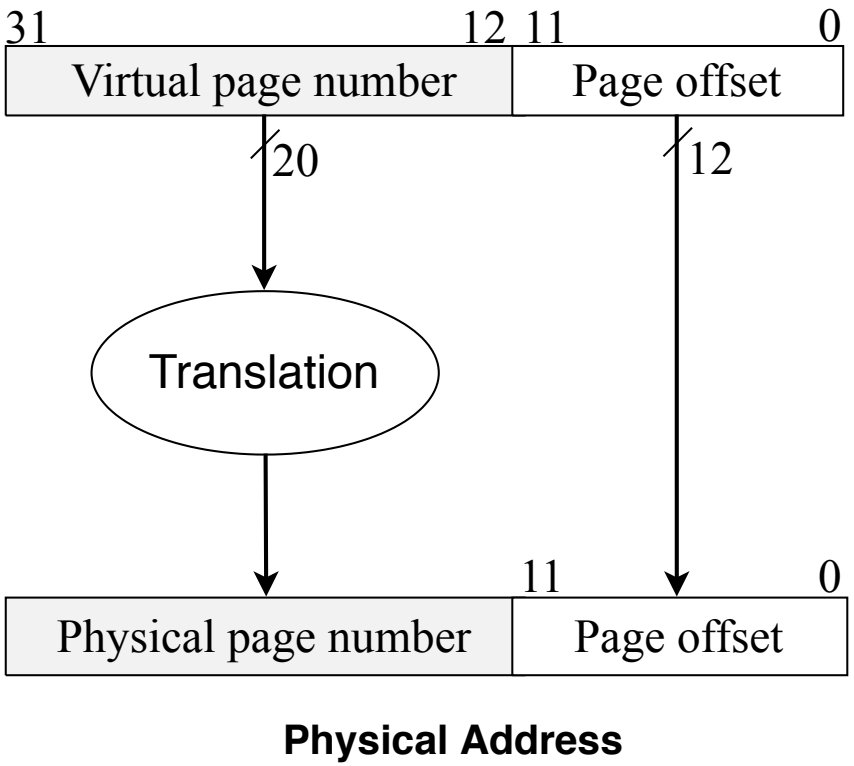
Virtual Memory: Logical View

- Virtual vs Physical memory view

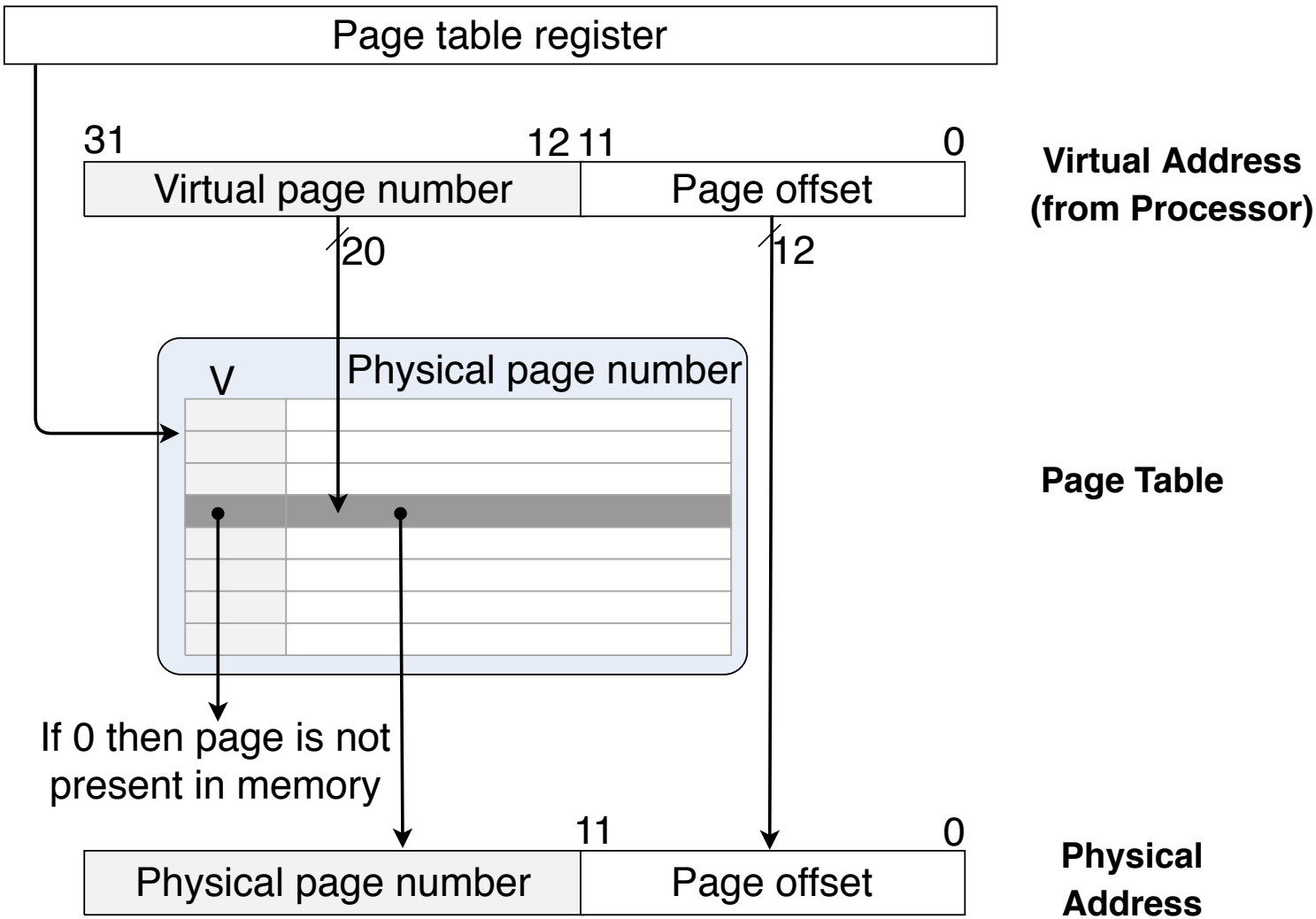


Address Translation

- The mapping between virtual addresses and physical addresses is stored in page table



Address Translation Cont'd



Address Translation Cont'd

- Page table is stored in **main memory**
- It is maintained by the **operating system**
- Page table entry example

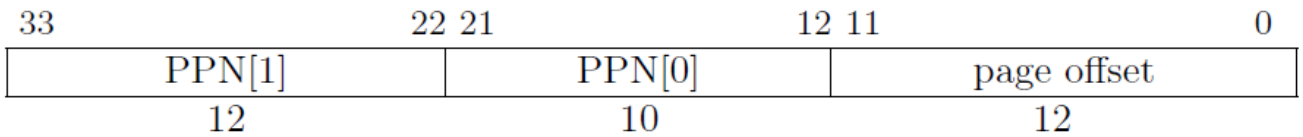


Figure 4.17: Sv32 physical address.

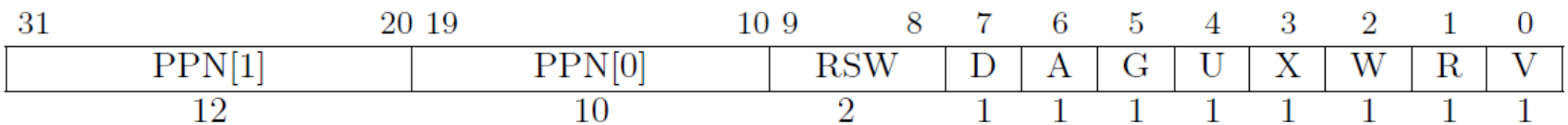
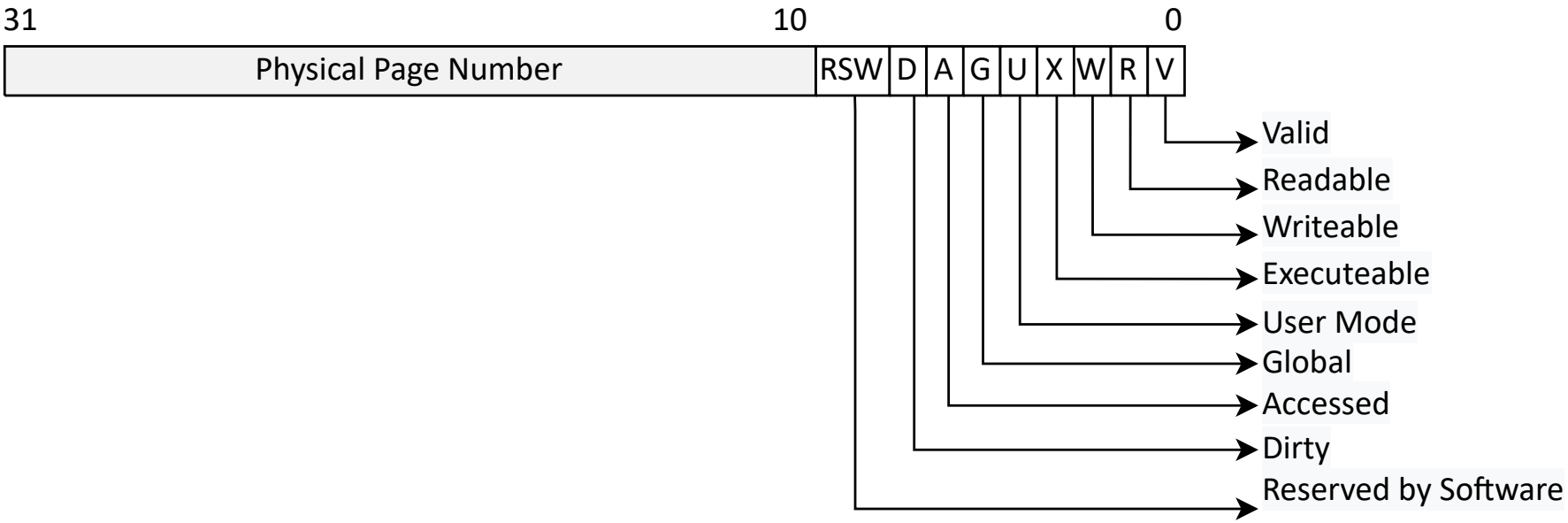


Figure 4.18: Sv32 page table entry.

Figure 1: Source: Privileged architecture manual.

Address Translation Cont'd



Page Table Size: The Problem

- For every *virtual page number* there is an entry in the page table
- One entry per page for the entire virtual address space, even for pages the program never uses!
- Entry contains: *physical page number* and page attribute bits, e.g. presence, protection bits etc.
- Overall size:

$$\frac{\textit{Virtual memory size}}{\textit{page size}} \times \textit{size of page table entry}$$

Page Table Size: The Problem cont'd

- Example
  - Virtual memory = 4GB, Page size = 4KB, Size of entry is 4B. 4MB Page table/process
  - Some processes may actually be smaller than 4MB but their page table is 4MB!
- How big would the page table be for a 64-bit machine?

Page Table Size: The Problem cont'd

- The problem with flat page tables is:
  - The page table size is determined by the size of the virtual memory and not by the actual amount of memory a program uses
  - For 32bit virtual address space, *several MBs* which in most cases is larger than the actual application
  - For 64bit virtual address space, *its too big*

Multi Level Page Table: The Solution

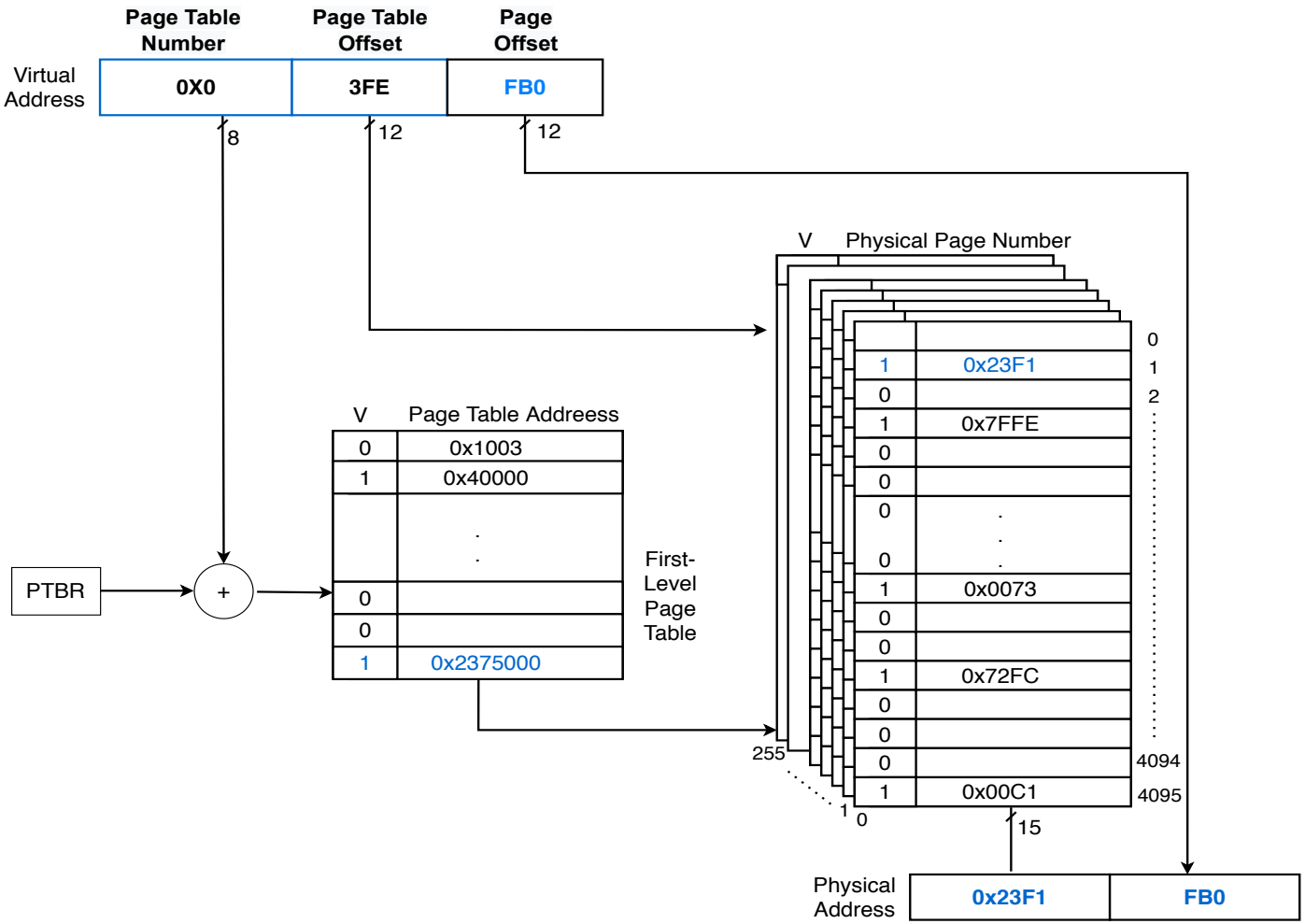
- One way to solve this is via multi-level page tables which solves these problems
  - The overall page table size is $\approx$ to size of program
  - Provides a solution for large virtual addresses
- The reason it works is because of the way large virtual address spaces actually get partitioned & used by processes

Multi Level Page Table: The Solution cont'd

- The VA is partitioned into two parts: Page offset & virtual page number
  - Virtual page number is further divided into two parts:
  - **Outer virtual page number (page table number) :** which part of the large page table will be used – first level page table
  - Entry indicates which of the second level page tables to use
  - **Inner virtual page number (page table offset) :** which specific entry in that portion of the page table will be used – second level page table
  - Entry gives us physical frame number

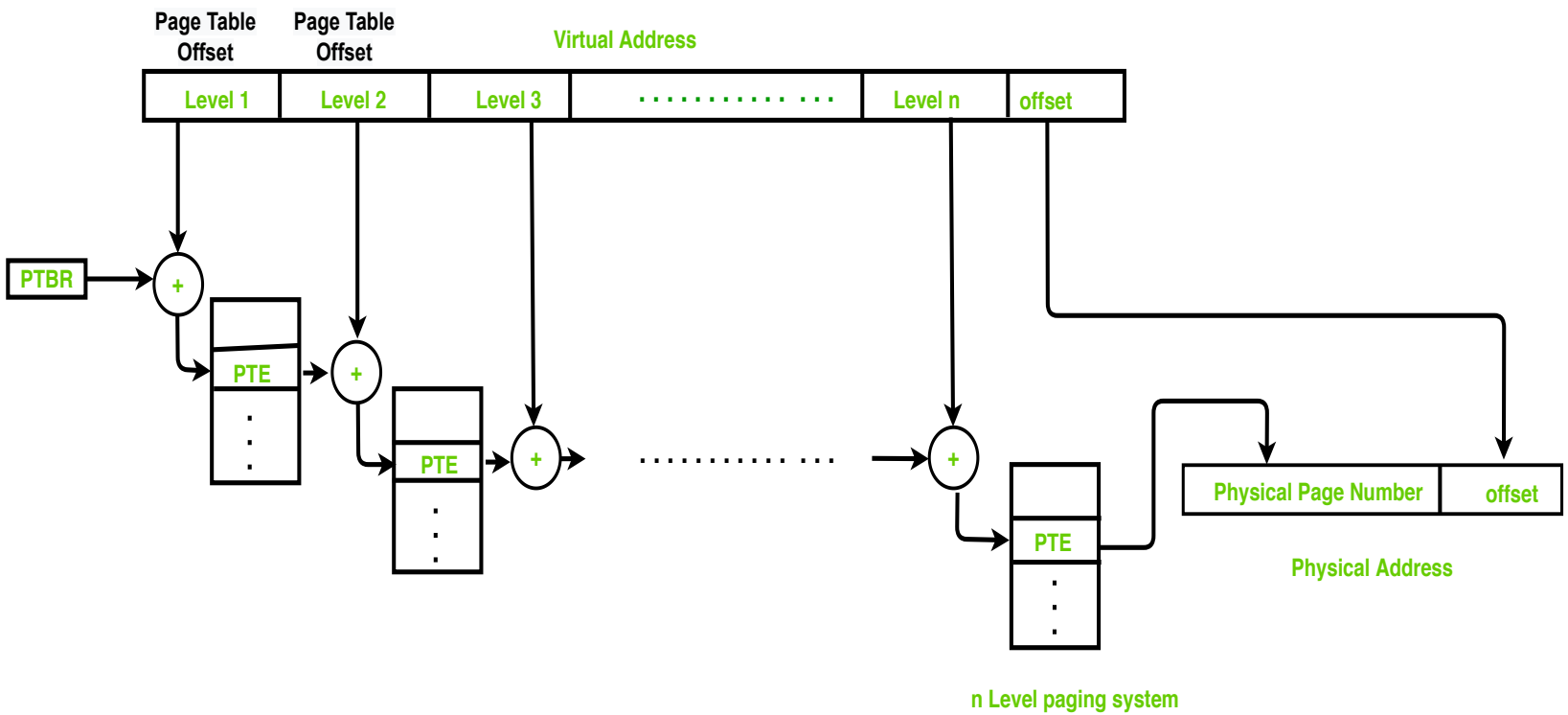
2 Level Page Table: The Solution cont'd

- The VA is partitioned into two parts: Page offset & virtual page number



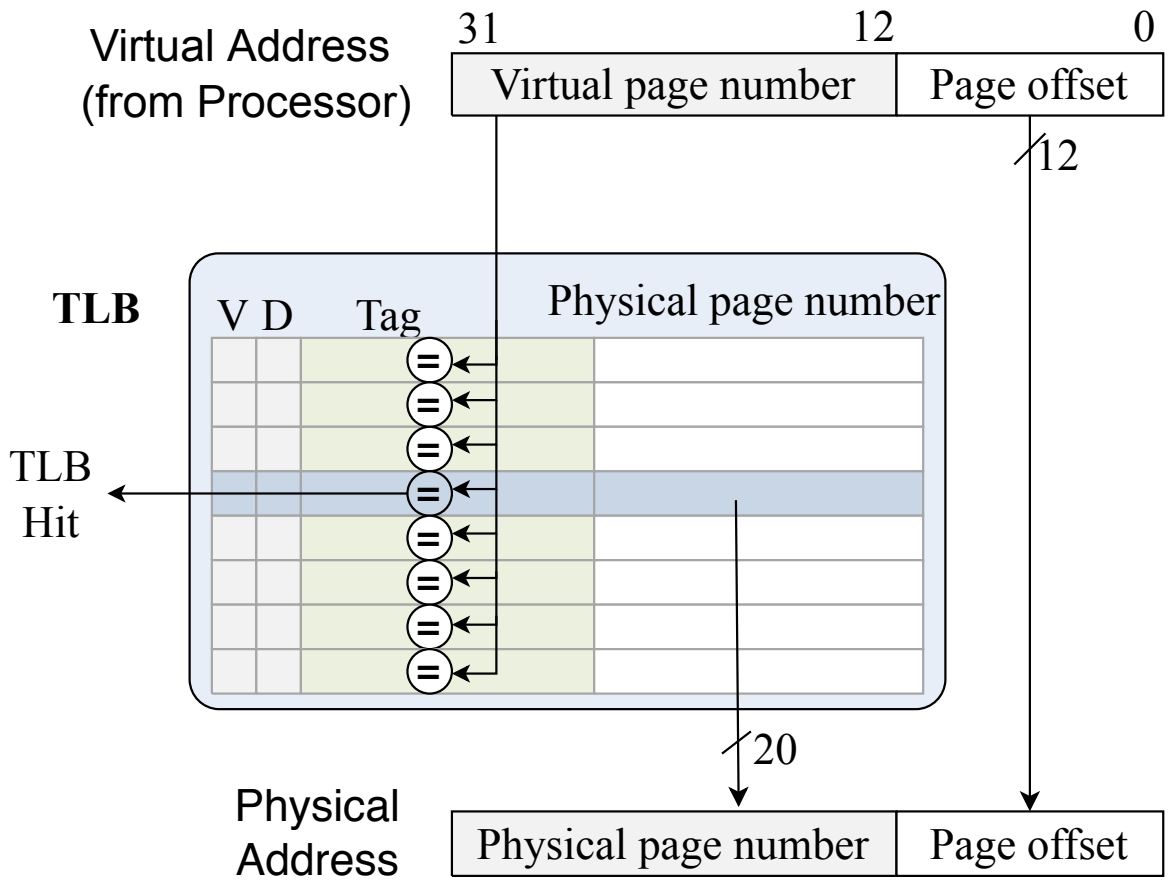
N Level Page Table: The Solution cont'd

- In multilevel paging all the page tables will be stored in main memory
- One access for each level needed
- Each page table entry except the last level page table entry contains base address of the next level page table

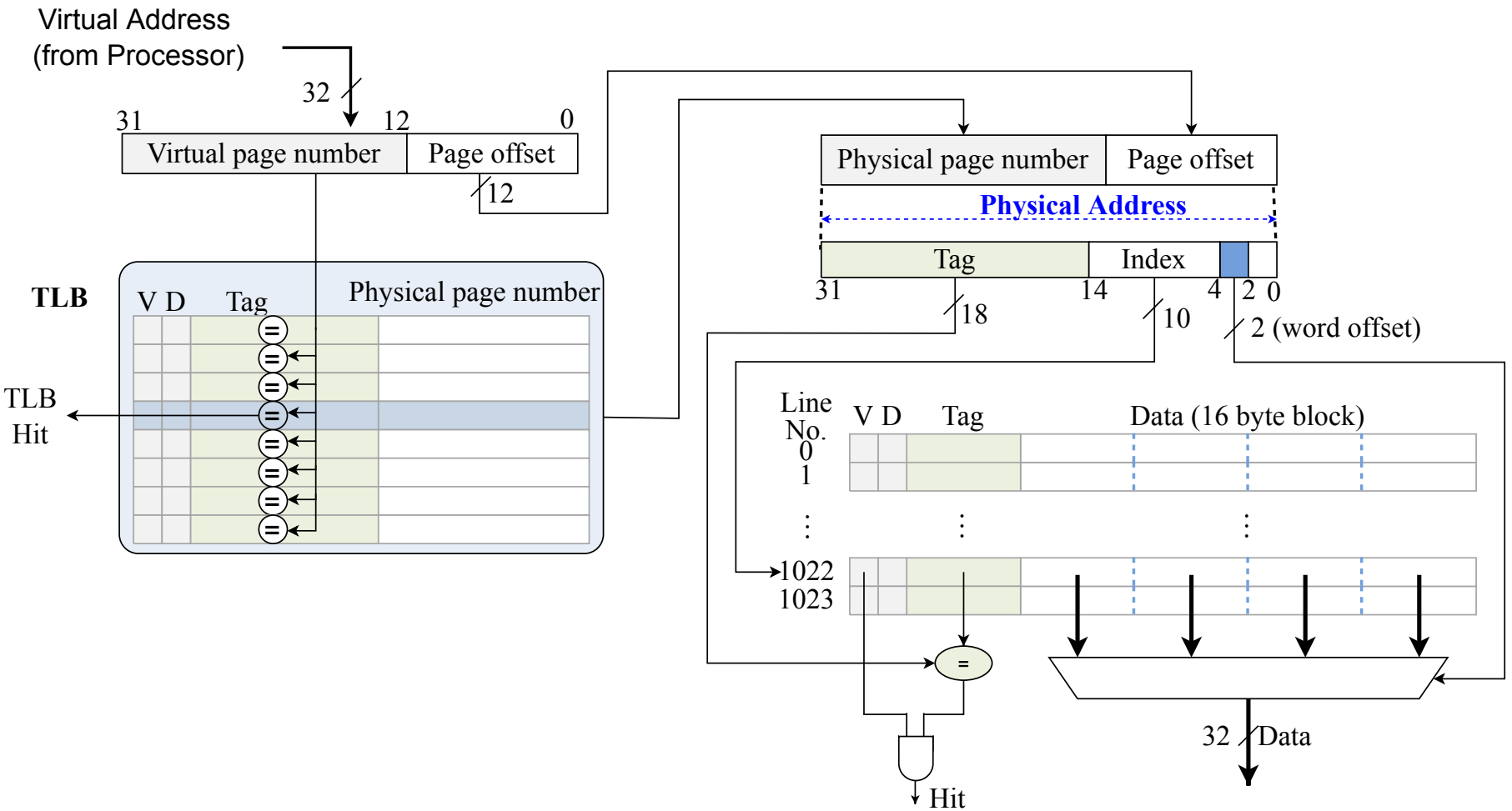


Making Address Translation Fast: The TLB

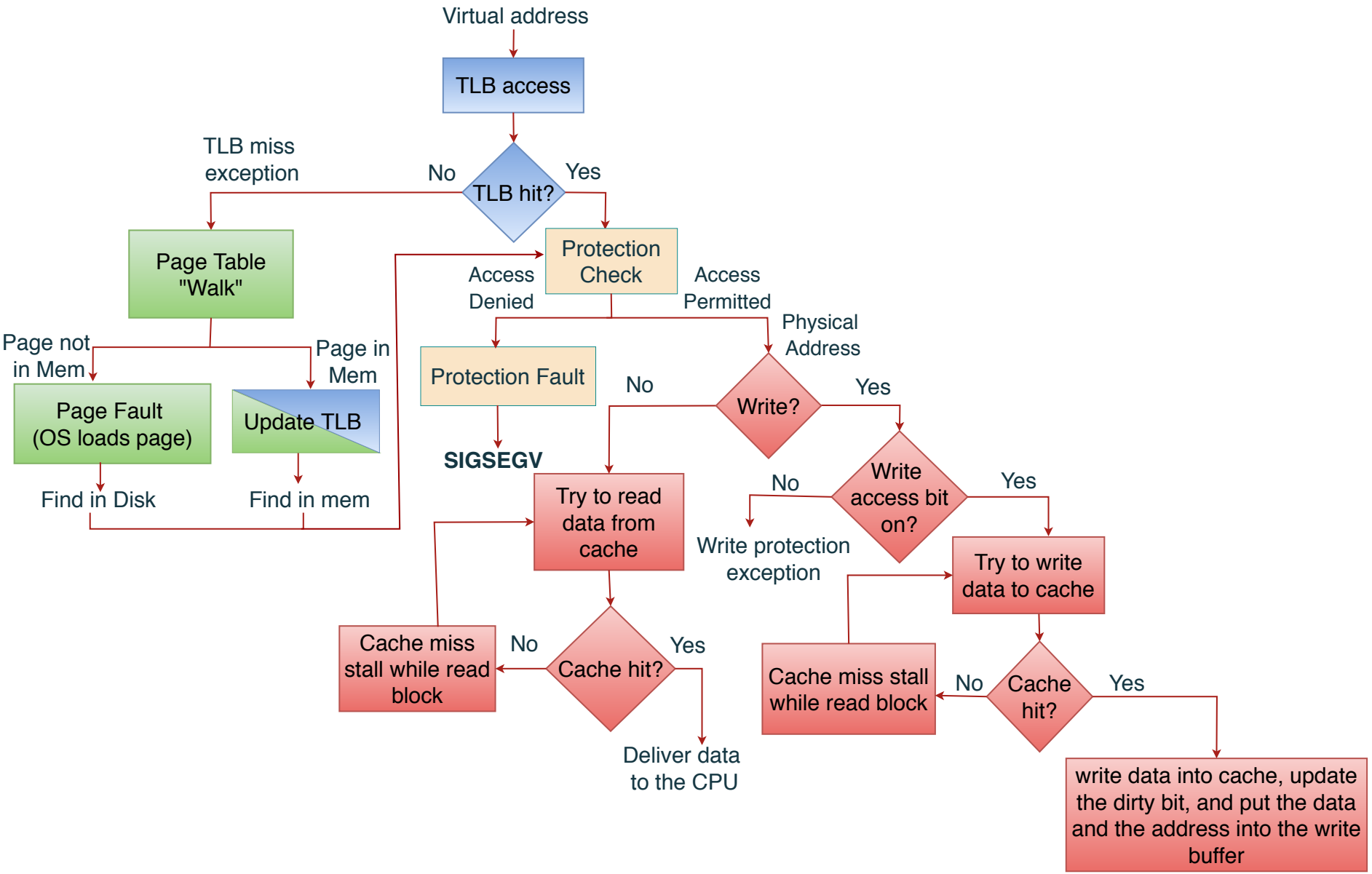
- The memory management unit maintains a buffer (cache memory) of recently used page table mappings
- This cache is called Translation look-aside buffer (TLB)



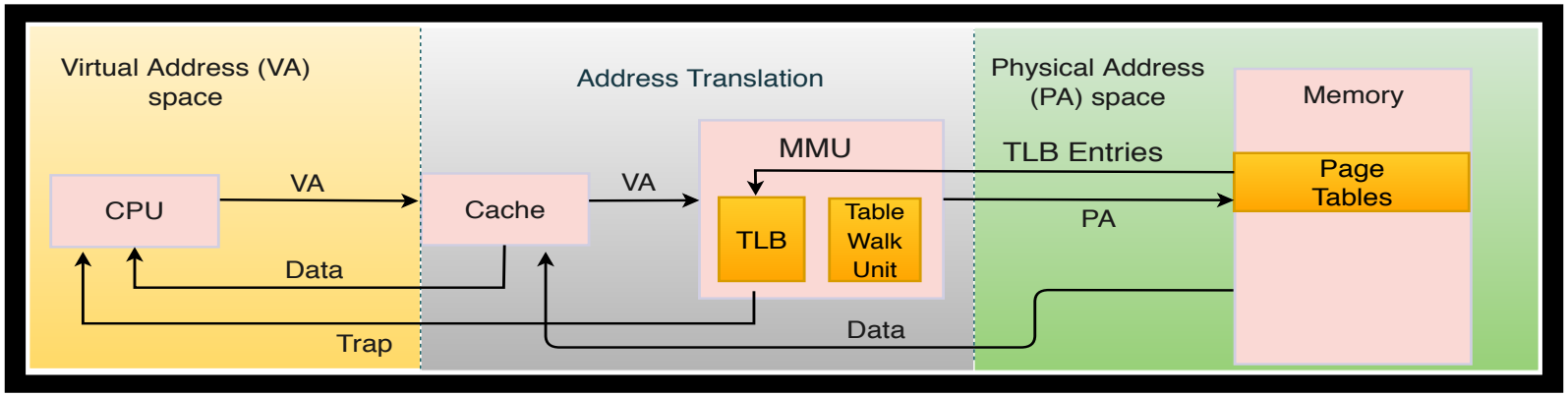
Real Life Example



Handling Page Faults



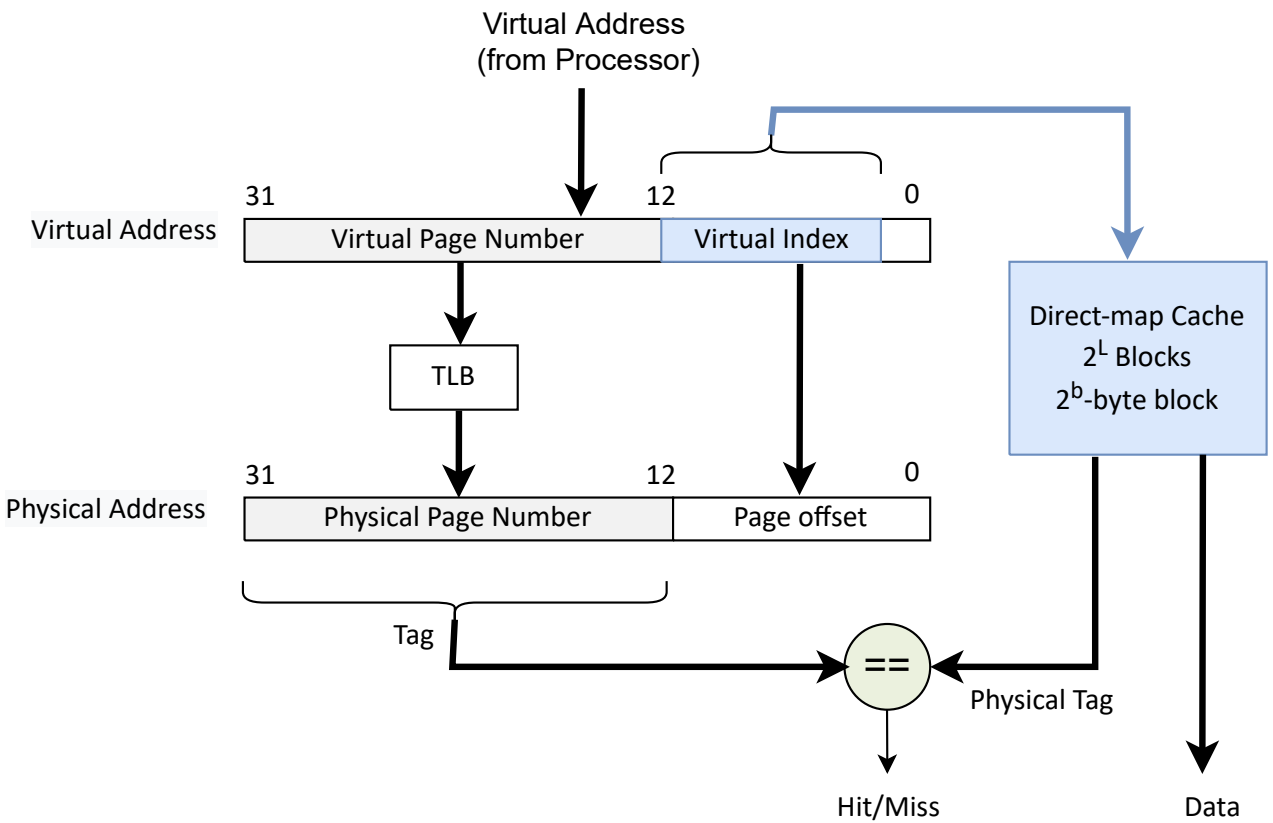
Caches with Virtual Addressing



- One-step process in case of a hit (+)
- Cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- Aliasing problems due to the sharing of pages (-)
- Maintaining cache coherence (-)

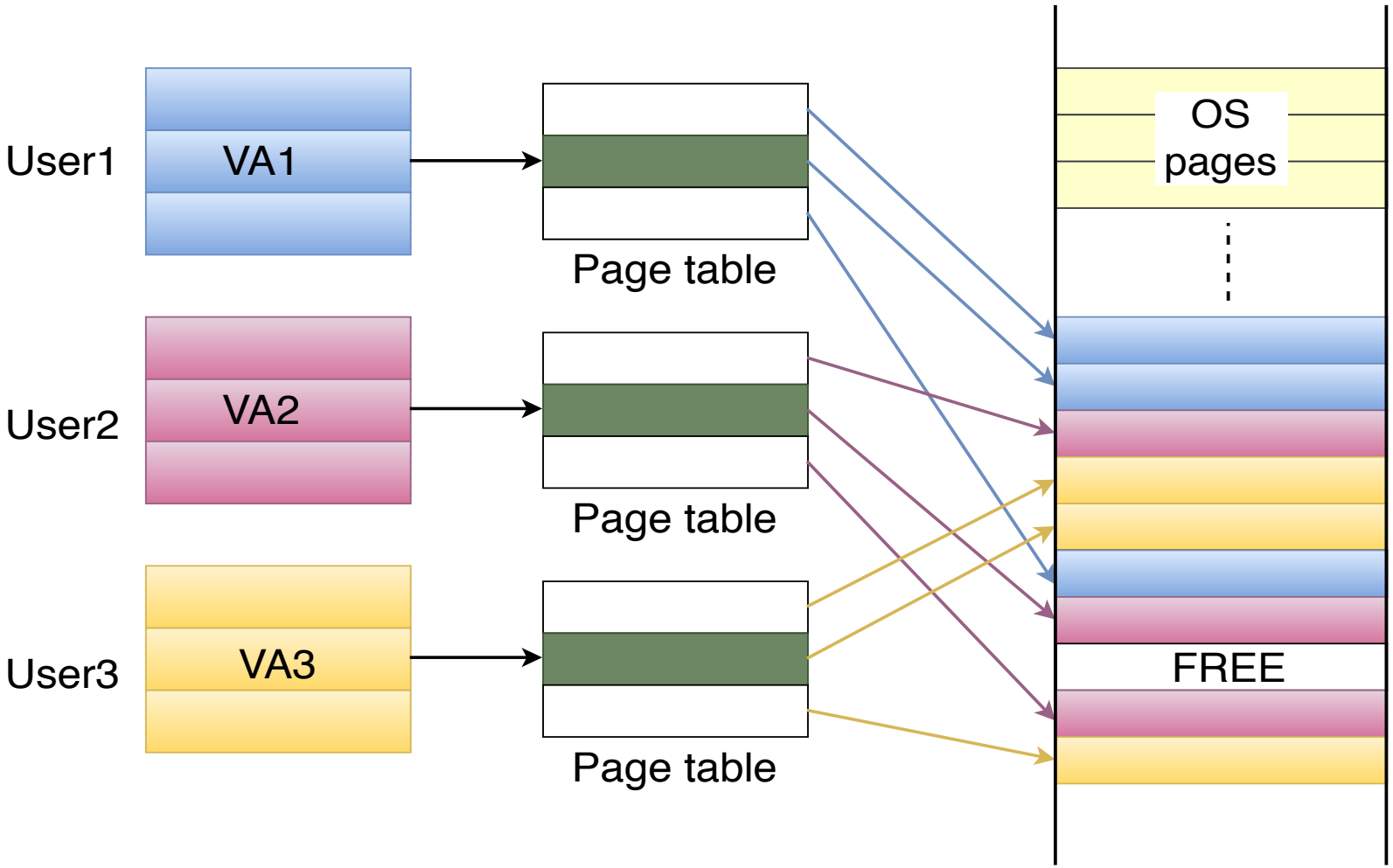
Caches with Virtual Addressing Cont'd

- To resolve aliasing problems:
  - Lookup in the cache with a virtual address
  - Verify the data is right with a physical tag
  - Look in the TLB at the same time as the cache



Multiple Page Tables

- Multiple page tables for different processes



Suggested Reading

- Read relevant sections of Chapter 5 of [Patterson and Hennessy, 2021].
- Read Section 2.4 of [Patterson and Hennessy, 2019].

Acknowledgment

- Preparation of this material was partly supported by Lampro Mellon Pakistan.

References

Patterson, D. and Hennessy, J. (2021).

Computer Organization and Design RISC-V Edition: The Hardware Software Interface, 2nd Edition.

Morgan Kaufmann.

Patterson, D. and Hennessy, J. (6th Edition, 2019).

Computer Architecture: A Quantitative Approach.

Morgan Kaufmann.