Virtual Memory

Outline

- Why virtual memory?
- Basic issues
- Demand paging
- Virtual to Physical Address Mapping
- Page faults
- Page replacement
- Page table entries
- Translation Lookaside Buffer (TLB)
- Page table organization
- Replacement algorithms
- Memory management

Why Virtual Memory?

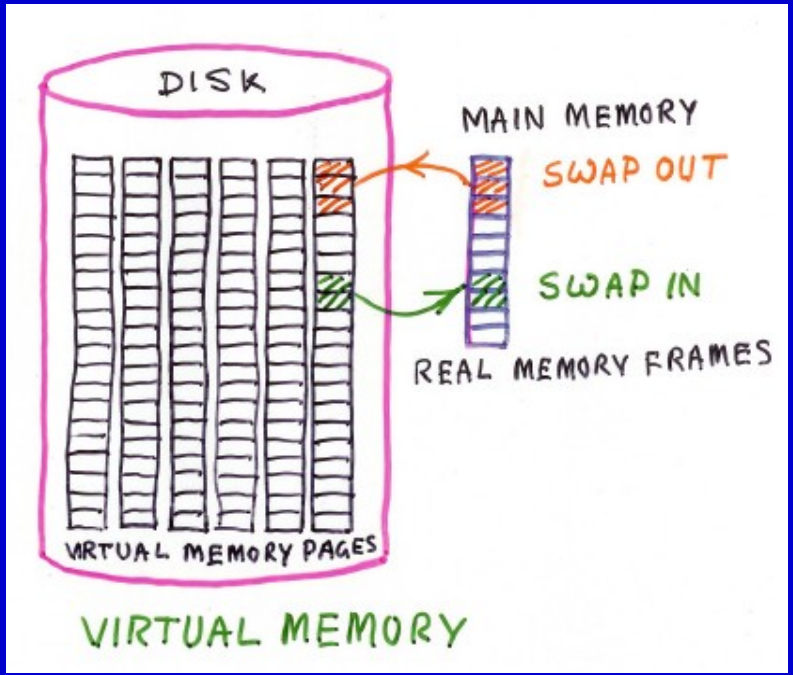
- Observations:
  - A large number of programs or inactive programs are stored on the disk.
  - A small portion of them are used at any instant of time.
  - A program must reside in the main memory while they are running.
- Consider a multiprogramming environment
  - Which programs will share memory with others is not known at compile time.
  - It changes dynamically as programs are running.
  - How do we ensure sharing as well as protection?

Why Virtual Memory?

- Observations:
  - Formerly, if the size of a program is larger than the main memory size, it was up to the user to take care of it.
    - How?
    - Programs were divided into pieces and identified the pieces that were mutually exclusive.
    - These overlays were loaded and unloaded under user program control during execution
    - Calls between procedures in different modules would lead to overlaying of one module with the other
- Imposes a substantial burden on the programmer

Why Virtual Memory?

- To fulfill dream of unlimited memory size
- To execute program of any size (bigger than the main memory)
- To allow efficient and safe sharing of memory by many programs (to support multiprogramming)



Motivations for Virtual Memory

- Use Physical DRAM as a Cache for the Disk
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory
- Simplify Memory Management
  - Multiple processes resident in main memory.
    - Each process with its own address space
  - Only “active” code and data is actually in memory
    - Allocate more memory to process as needed.
- Provide Protection
  - One process can’t interfere with another.
    - Because they operate in different address spaces.
  - User process cannot access privileged information
    - Different sections of address spaces have different permissions.

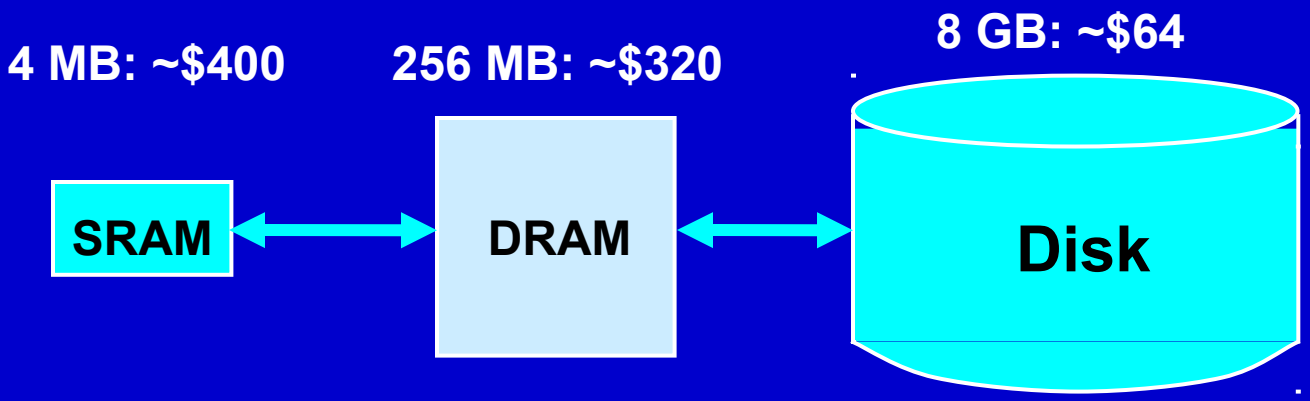
Basic Issues

- Developed to alleviate the burden of programmers.
- A technique that uses main memory as a 'cache' of secondary storage.
- Shares the basic concepts of cache memory, but differ in terminology.

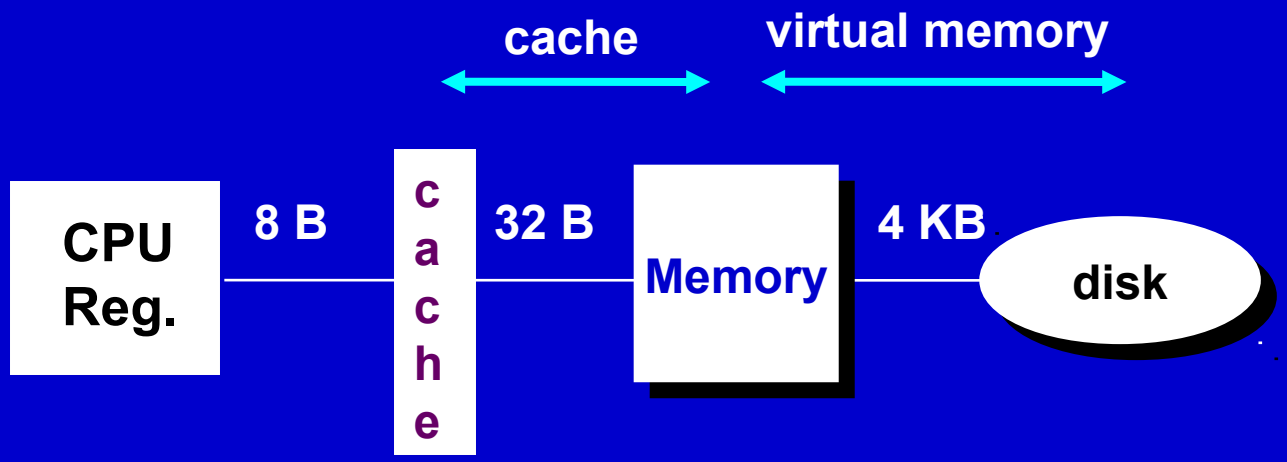
- Basic issues:
  - Mapping: Translation of virtual address to Physical address
  - Management: Controlled sharing and protection and protection in multiprogramming environment
- Mapping Techniques:
  - Paging (Demand Paging)
  - Segmentation

DRAM a Cache for Disk

- Full address space is quite large:
  - 32-bit addresses: ~4,000,000,000 (4 billion) bytes
  - 64-bit addresses: ~16,000,000,000,000,000,000 (16 quintillion) bytes
- Disk storage is ~156X cheaper than DRAM storage
  - 8 GB of DRAM: ~ $10,000
  - 8 GB of disk: ~ $64
- To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk



Levels in Memory Hierarchy

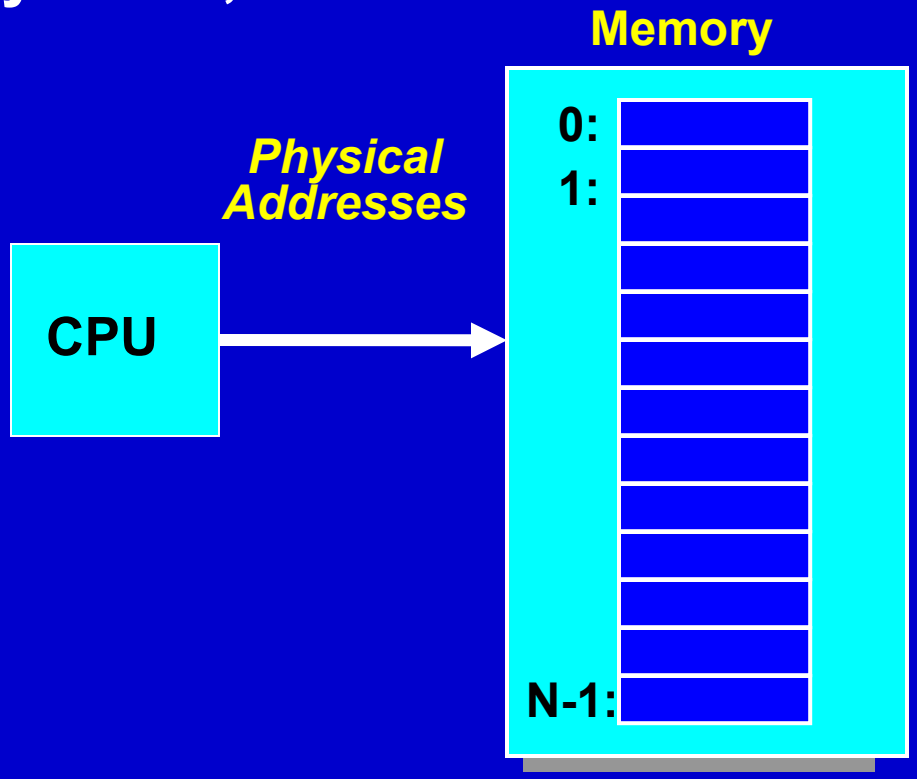


	Register	Cache	Memory	Disk Memory
Size:	32 B	32 KB-4MB	128 MB	30 GB
Speed:	3 ns	6 ns	60 ns	8 ms
$/Mbyte:		$100/MB	$1.25/MB	$0.008/MB
Line size:	8 B	32 B	4 KB	

larger, slower, cheaper

A System with Physical Memory Only

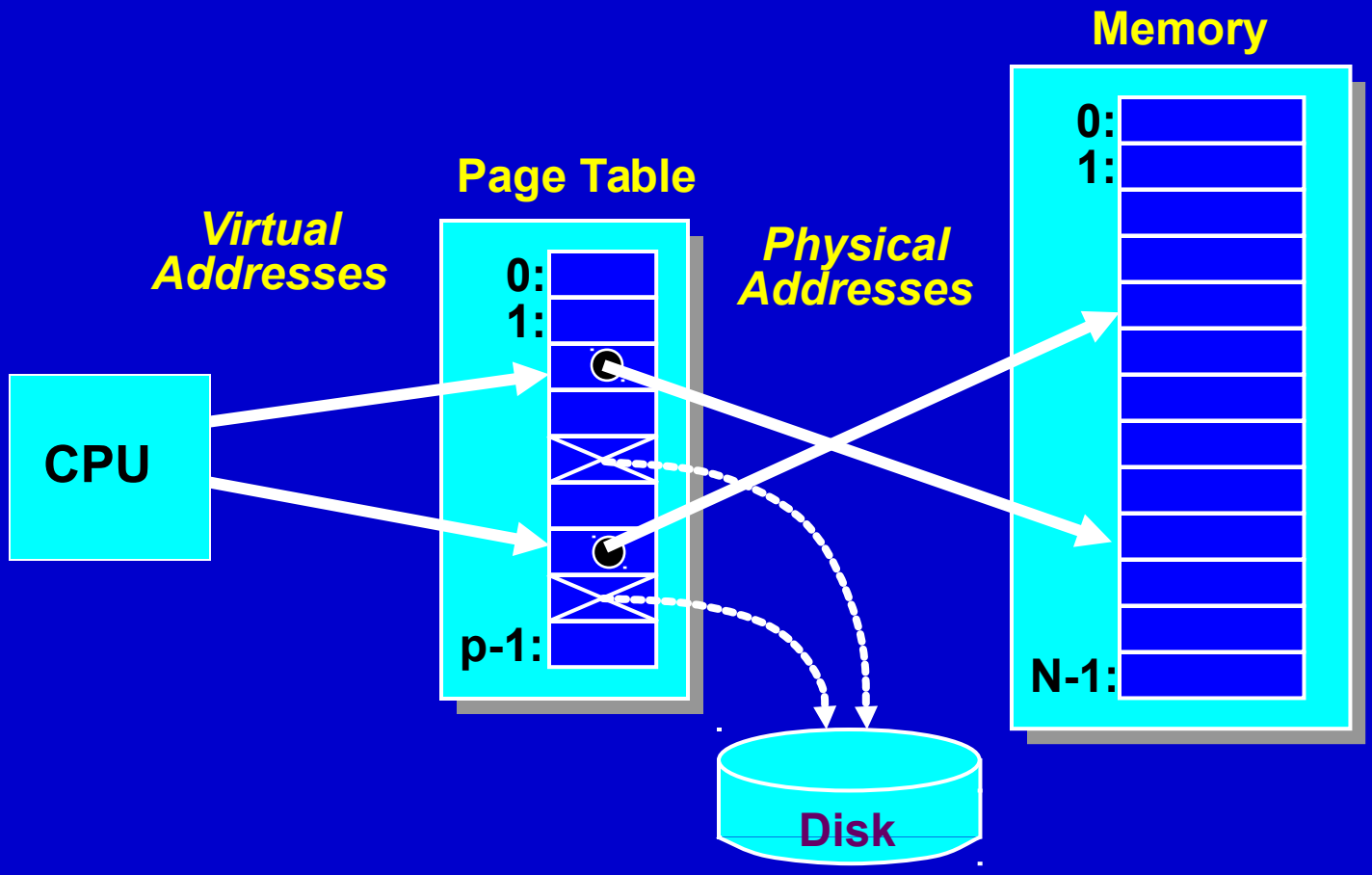
- Examples: most Cray machines, early PCs, nearly all embedded systems, etc.



- Addresses generated by the CPU point directly to bytes in physical memory

A System with Virtual Memory

- Examples: workstations, servers, modern PCs, etc.



Address Translation: Hardware converts virtual addresses to physical addresses via an OS-managed lookup table (page table)

Cache Memory Vs Virtual Memory

Parameter	First-level cache	Virtual Memory
Block (page) size	**16-28 bytes**	**4096-65536 bytes**
Hit time	**1-2 clock cycles**	**40-100 clock cycles**
Miss penalty	**8-100 clock cycles**	**700,000-6,00,000 clock cycles**
Access time	50-60 clock cycles	**5,000,000 - 20,000,000 clock cycles**
Miss rate	0.5-10%	0.00001-0.001 %
Data memory size	0.016-1 MB	16-8192 MB

Cache Memory Vs Virtual Memory

- Terminology:
  - Block -> Page
  - Cache miss -> page fault
- Replacement on cache memory misses by hardware, whereas virtual memory replacement is by the operating system
- The size of processor address determines the size of virtual memory, whereas cache size is independent of address size
- Virtual memory can have fixed or variable size block

DRAM vs. SRAM as a "Cache"

- DRAM vs. disk is more extreme than SRAM vs. DRAM
  - Access latencies:
    - DRAM ~10X slower than SRAM
    - Disk ~100,000X slower than DRAM
  - Importance of exploiting spatial locality:
    - First byte is ~100,000X slower than successive bytes on disk
      - versus ~4X improvement for page-mode vs. regular accesses to DRAM
  - Bottom line:
    - Design decisions made for DRAM caches driven by enormous cost of misses

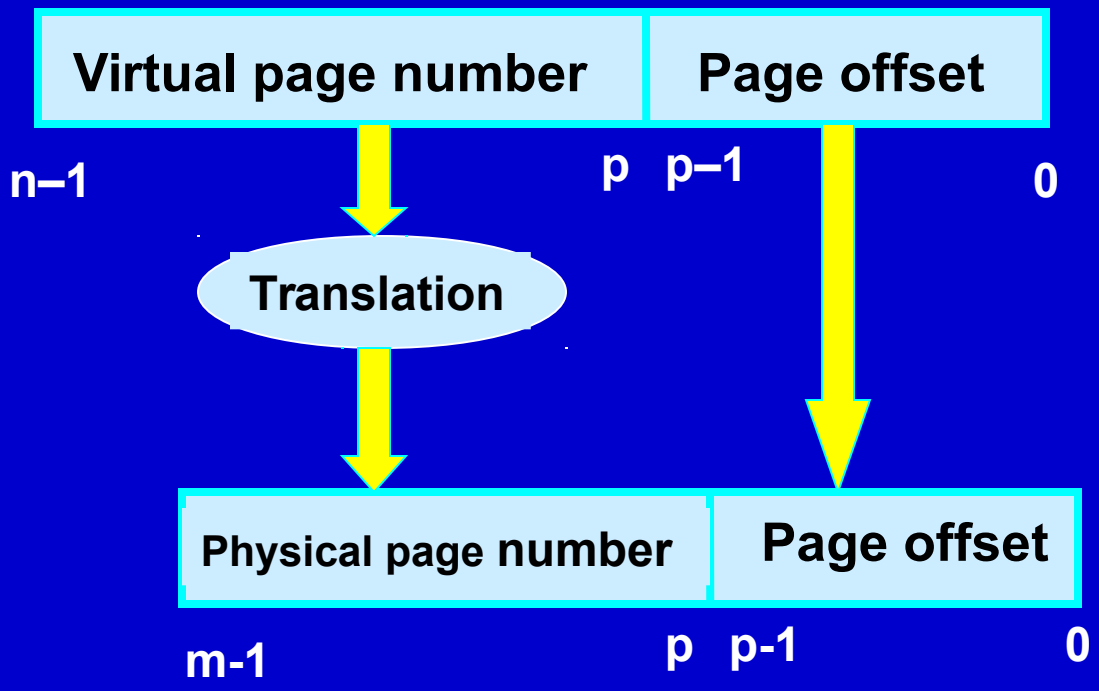
Virtual Memory Design Issues

- Page size should be large enough to try to amortize the high access time.
  - Typical size: 4KB to 16KB
- Organization that reduce page fault rate is important.
  - Fully associative placement of pages in memory
- Page faults need not be handled by hardware.
  - Software (OS) can afford to use cleaver algorithm for page replacement to reduce page fault rate
- Write through approach cannot be used.
  - Write back approach is used

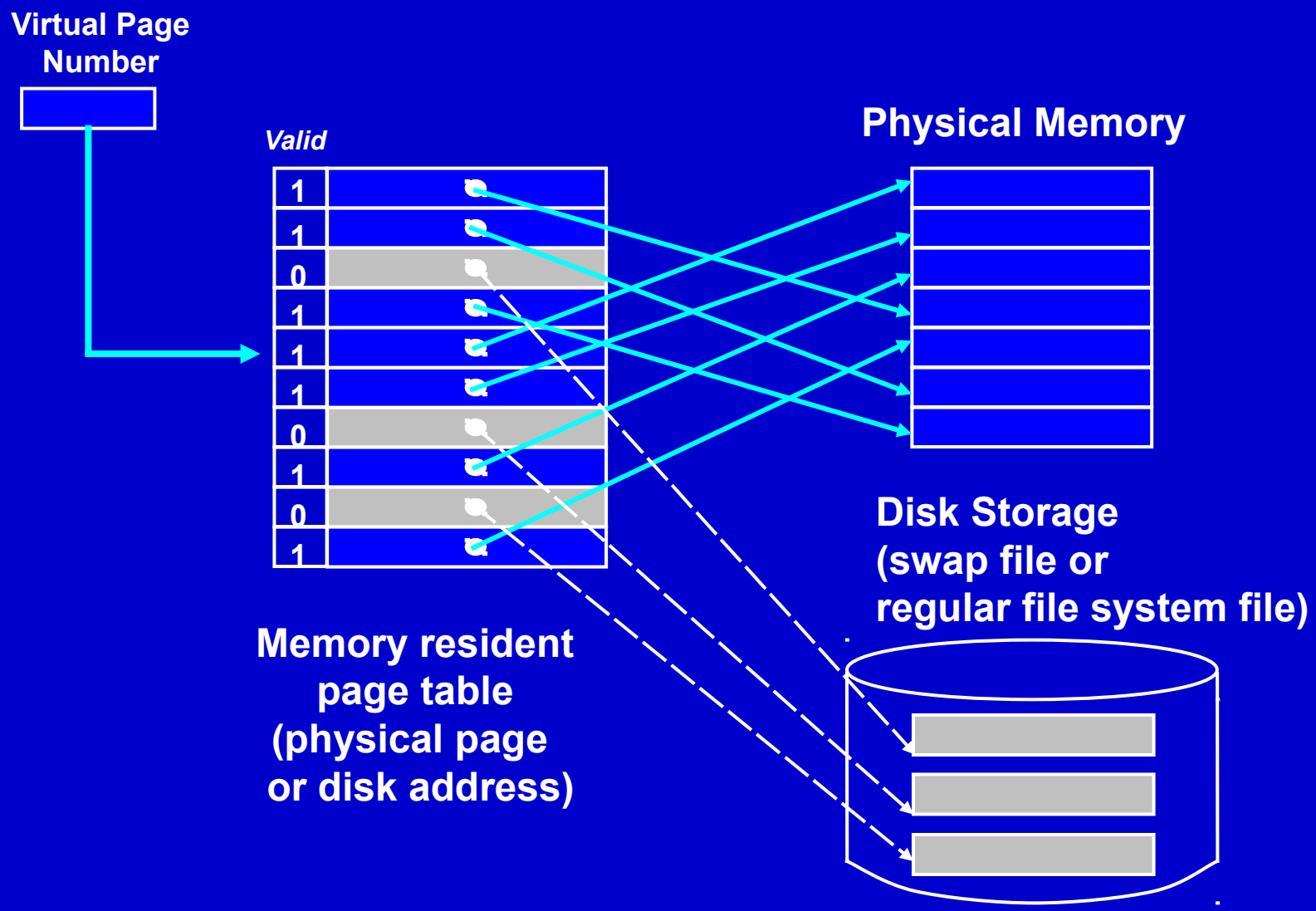
Virtual to Physical Address Mapping

- Virtual address:

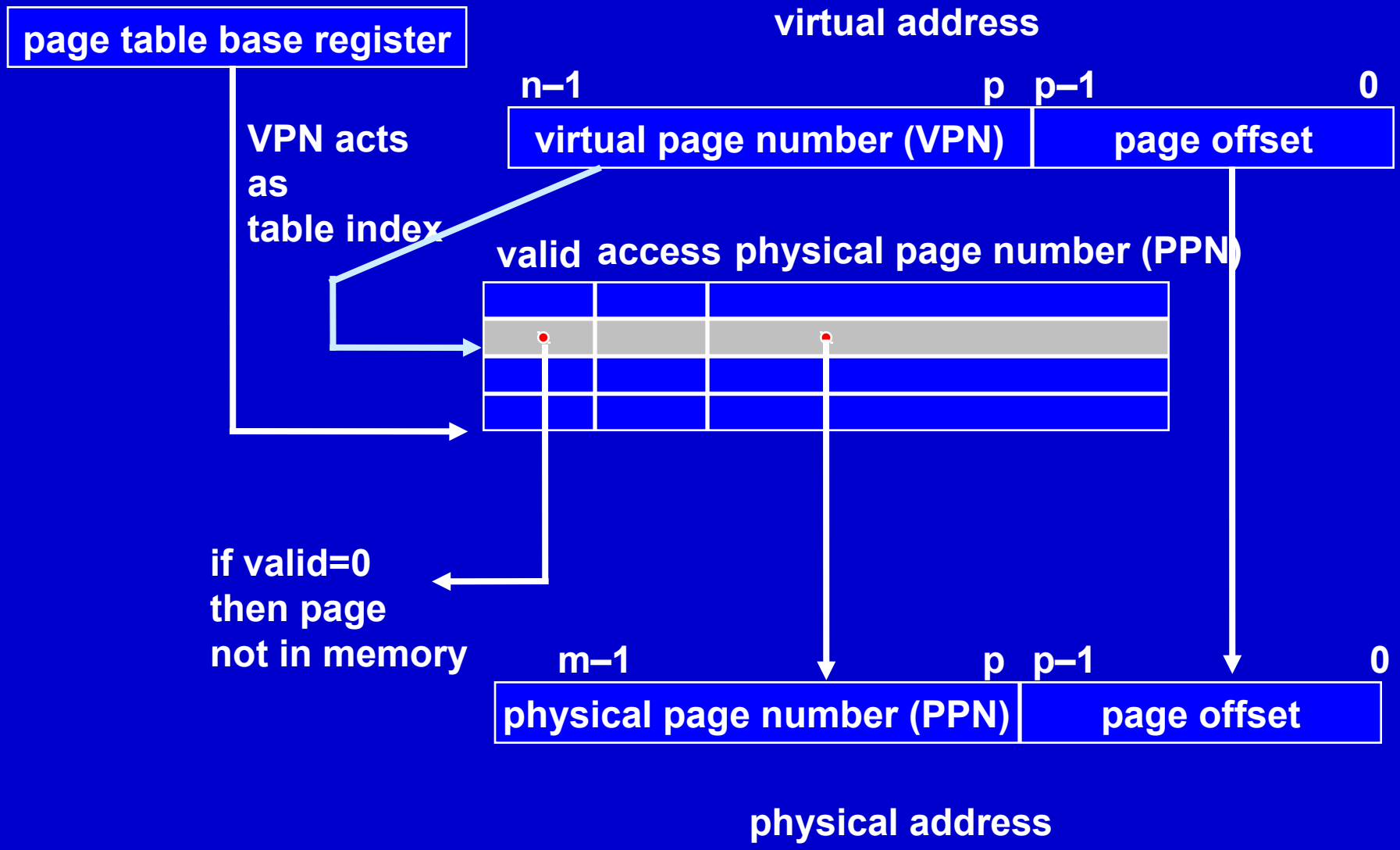
- The number pages addressable with the virtual address is virtually unlimited, where as the number of pages addressable by the physical address is limited.
- Mapping from a virtual to a physical address



Address Mapping



Address Translation via Page Table



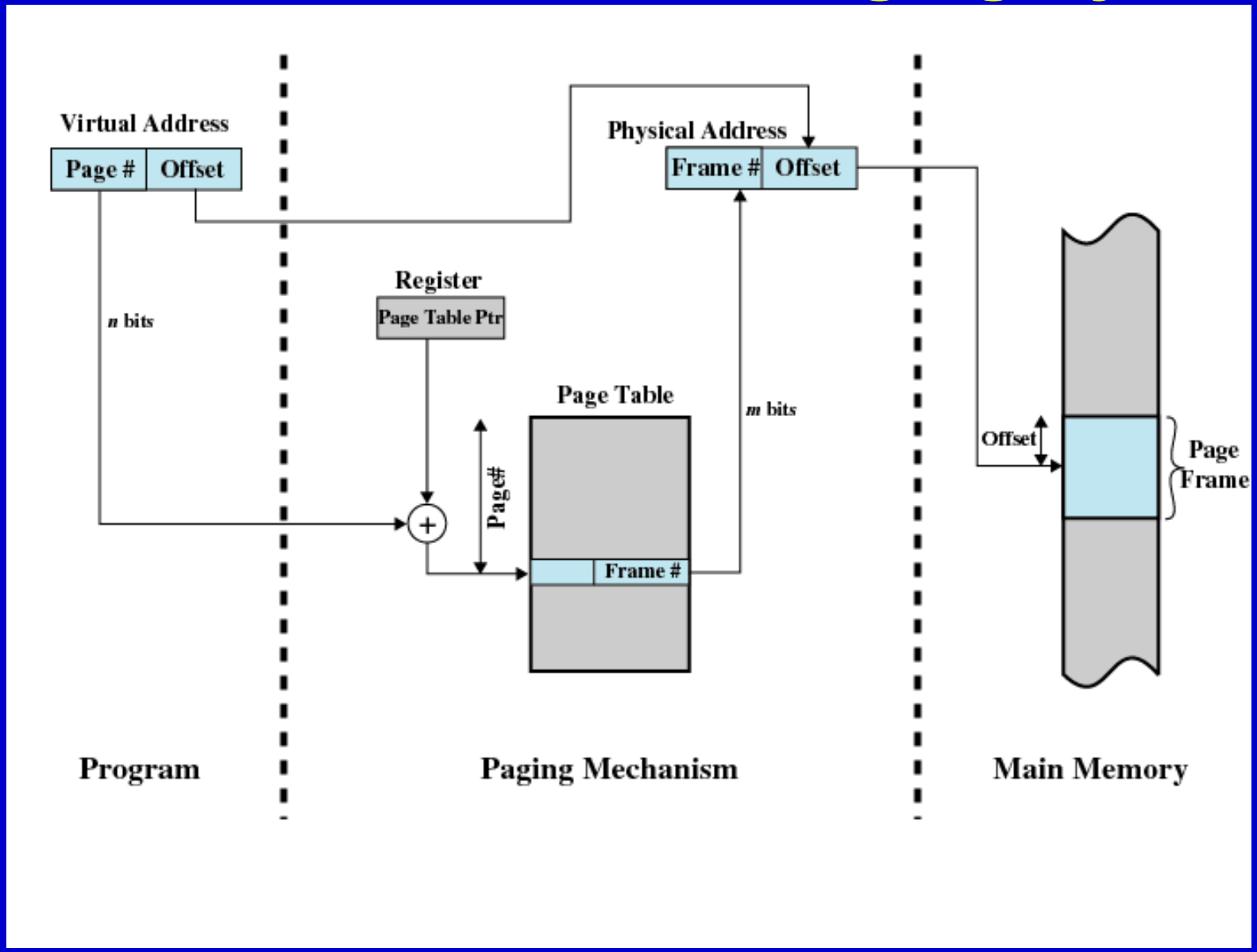
Page Table

- Reading a word from memory involves translation of the virtual address to physical address comprising frame number and offset using a table called page table

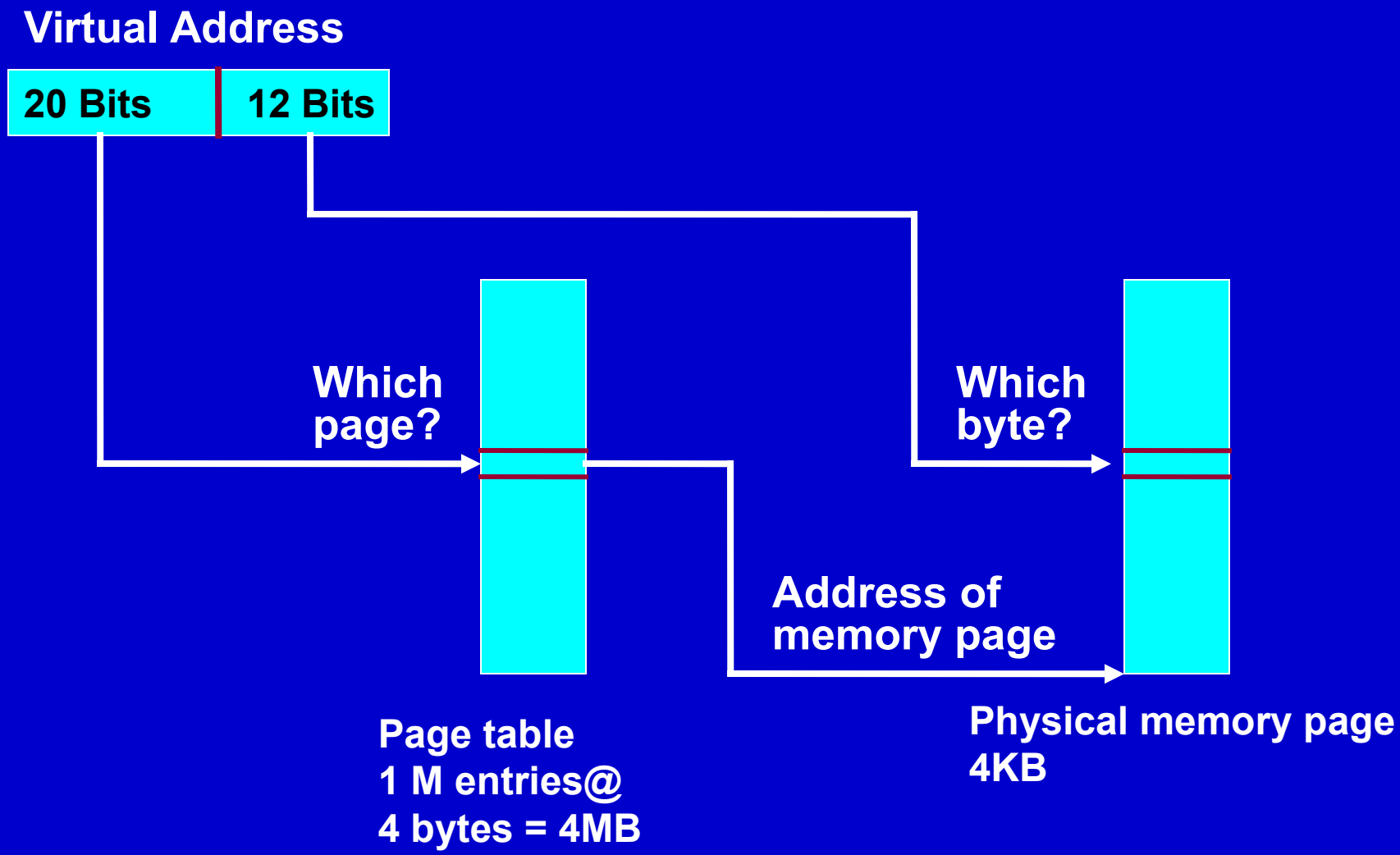
Page Table Operation

- Translation
  - Separate (set of) page table (s) per process
  - VPN forms index into page table (points to a page table entry)
- Computing Physical Address
  - Page Table Entry (PTE) provides information about page
    - if (valid bit = 1) then the page is in memory.
      - Use physical page number (PPN) to construct address
    - if (valid bit = 0) then the page is on disk
      - Page fault: Must load page from disk into main memory before continuing

Address Translation in Paging System

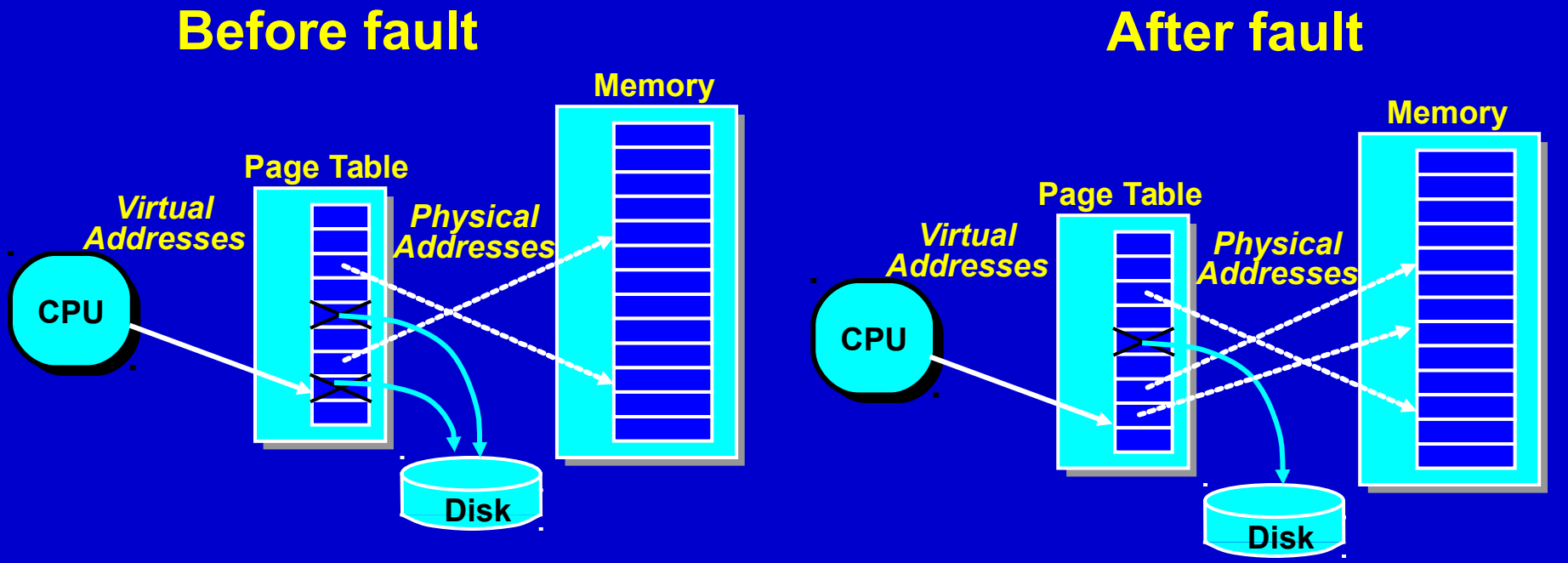


Address Translation



Page Faults

- What if an object is on disk rather than in memory?
  - Page table entry indicates virtual address not in memory
  - OS exception handler invoked to move data from disk into memory
    - current process suspends, others can resume
    - OS has full control over placement, etc.

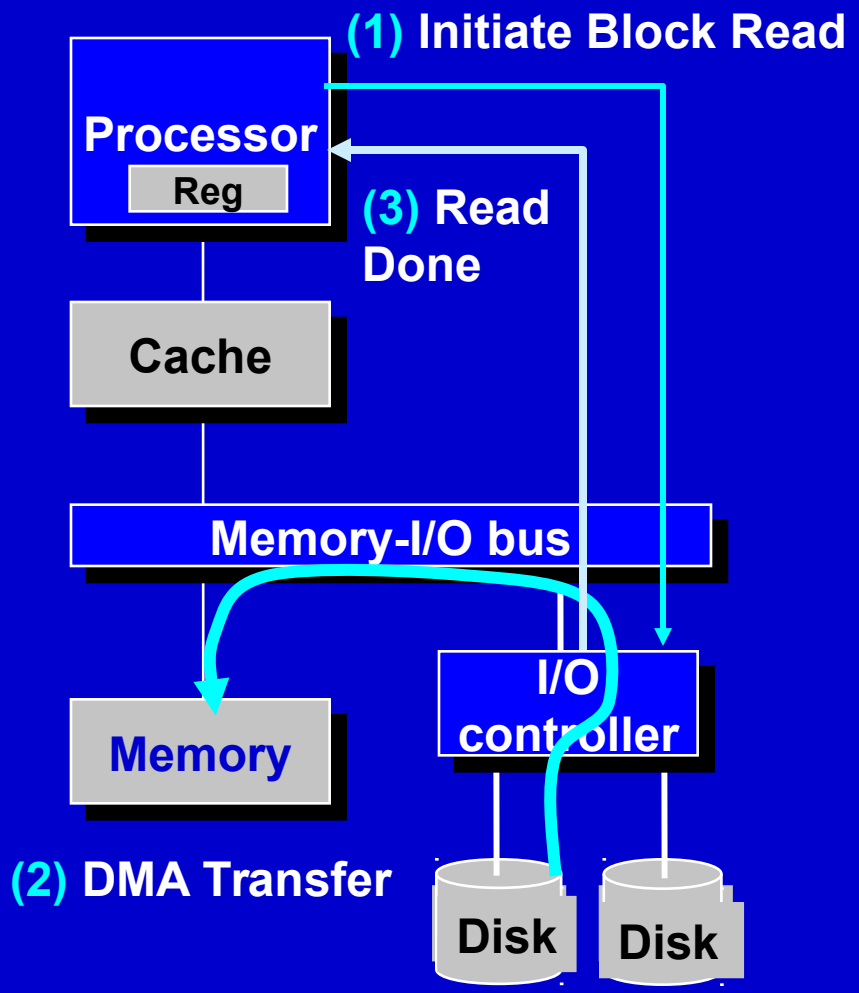


Page Faults

- The OS takes control when the valid bit of a physical page is 0.
- How does it finds the requested page in disk?
  - When a process is created, the OS creates space for all the pages of a process in the disk.
  - This disk space is called the swap space.
  - It also creates a data structure to record where each virtual page is stored in the disk.
  - This data structure may be a part of the page table or separate.
  - The OS also creates a data structure that tracks which processes and which virtual addresses use each physical page.
  - If all the pages in main memory are in use, the OS must choose a page to replace based on the past history.
  - The replaced pages are written to the swap space of the disk.

Servicing a Page Fault

- Processor Signals Controller
  - Read block of length P starting at disk address X and store starting at memory address Y
- Read Occurs
  - Direct Memory Access (DMA)
  - Under control of I/O controller
- I / O Controller Signals Completion
  - Interrupt the processor
  - OS resumes suspended process



Page Replacement

- Find a free frame to store a desired page
  - If there is a free frame, use it
  - Otherwise, use a page replacement algorithm to select a victim frame
  - Write the victim page to the secondary storage; change the page and page table accordingly
- Read the desired page into the free (newly) frame; change the page and frame tables
- Restart the user process

Write Miss

- Write through approach cannot be used.
  - Write back approach is used
  - Page table requires an additional bit known as dirty bit
  - Many writes can take place before a page is replaced

Protecting a Page

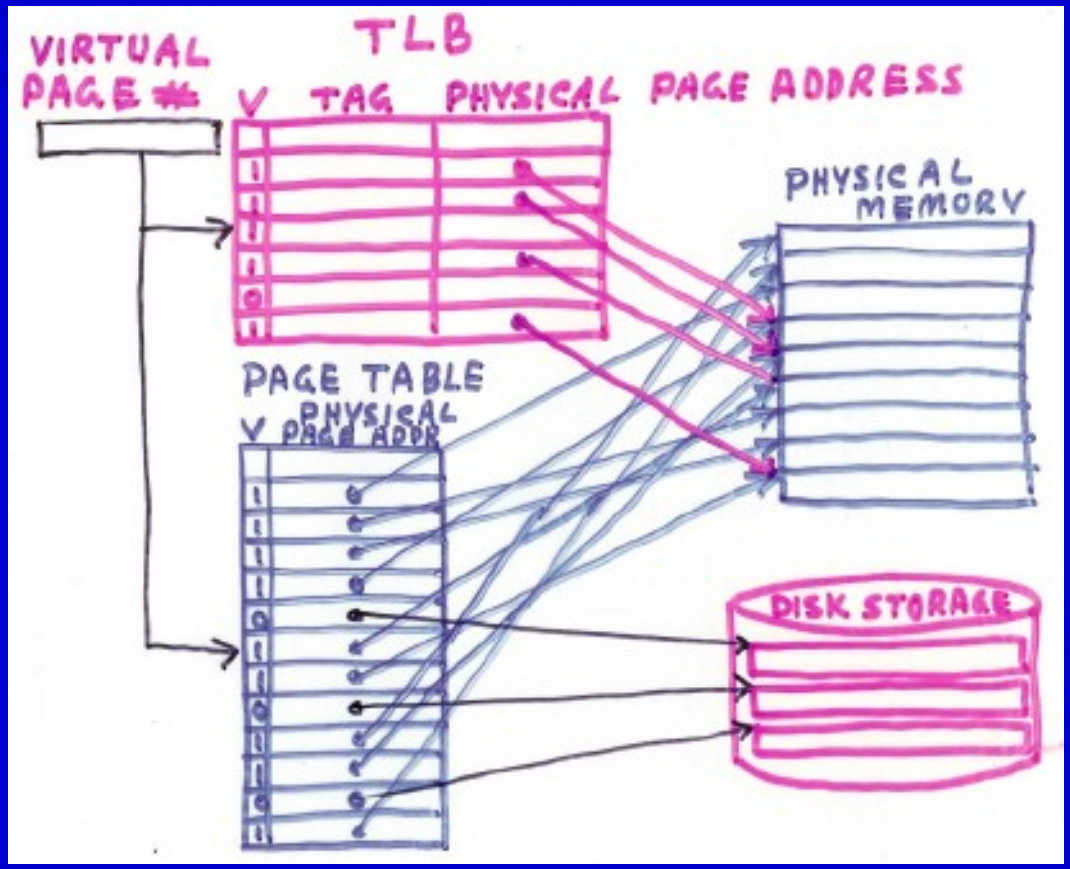
- Checking Protection
  - Access rights field indicate allowable access
    - e.g., read-only, read-write, execute-only
    - typically support multiple protection modes (e.g., kernel vs. user)
  - Protection violation fault if user doesn't have necessary permission

Page Table Entries

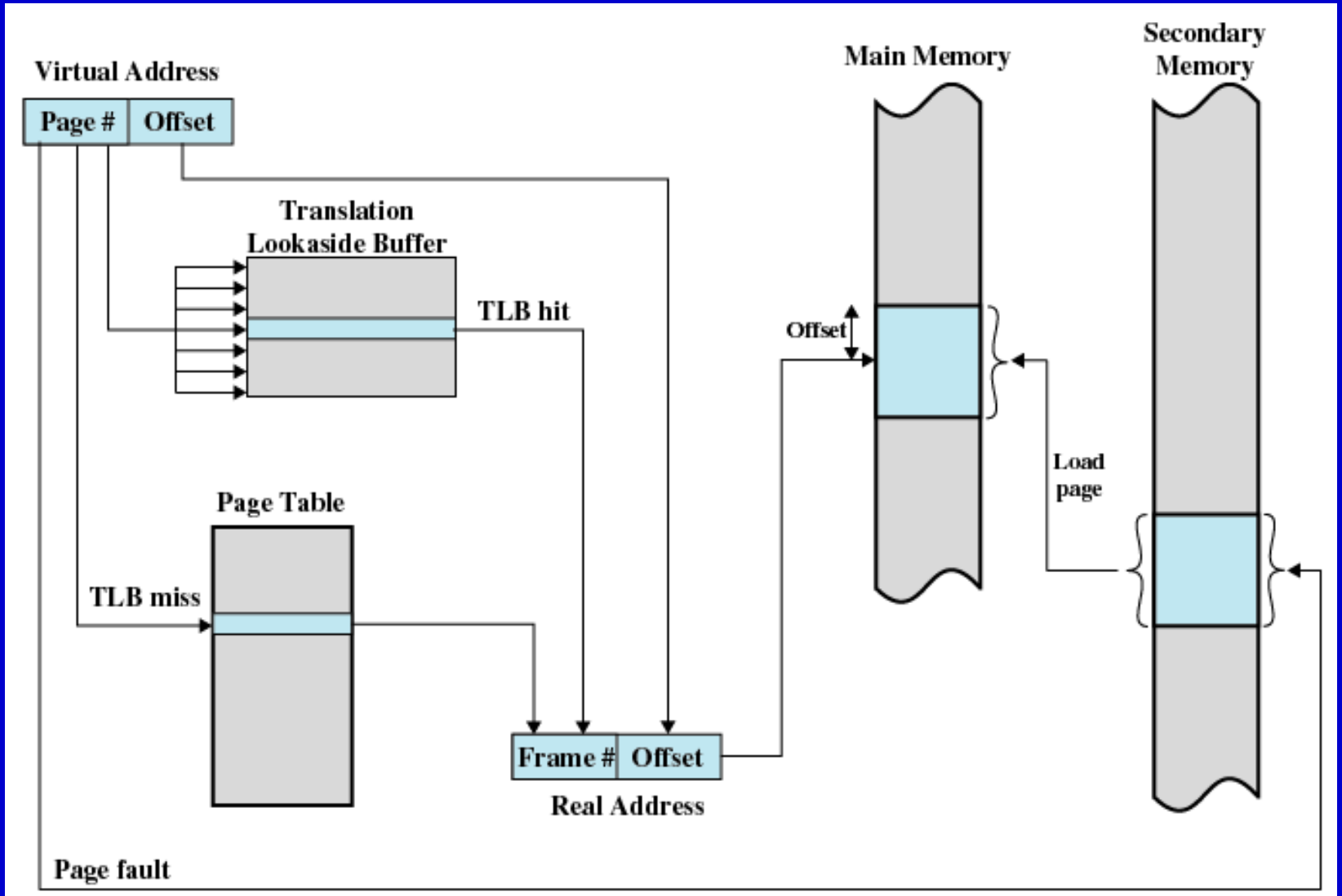
- Accomplishes mapping via use of look-up table
- Address
  - Pointer to location of page in memory
  - Or, if page is swapped, can be disk address instead of Control bits
- Valid/Present bit
  - If set, page being pointed to is resident in memory
- Modified/dirty bit
  - Set if at least one word in page has been modified
- Referenced bit
  - Set if page has been referenced (with either read or write)
  - Used to support software replacement policies
- Protection bits
  - Used to restrict access
  - For example, read-only access, or system-only access

Making Address Translation Faster

- If page tables are stored in the main memory, each memory access requires two memory reads
- To speed up the process, a special address translation cache called Translation Lookaside Buffer (TLB)

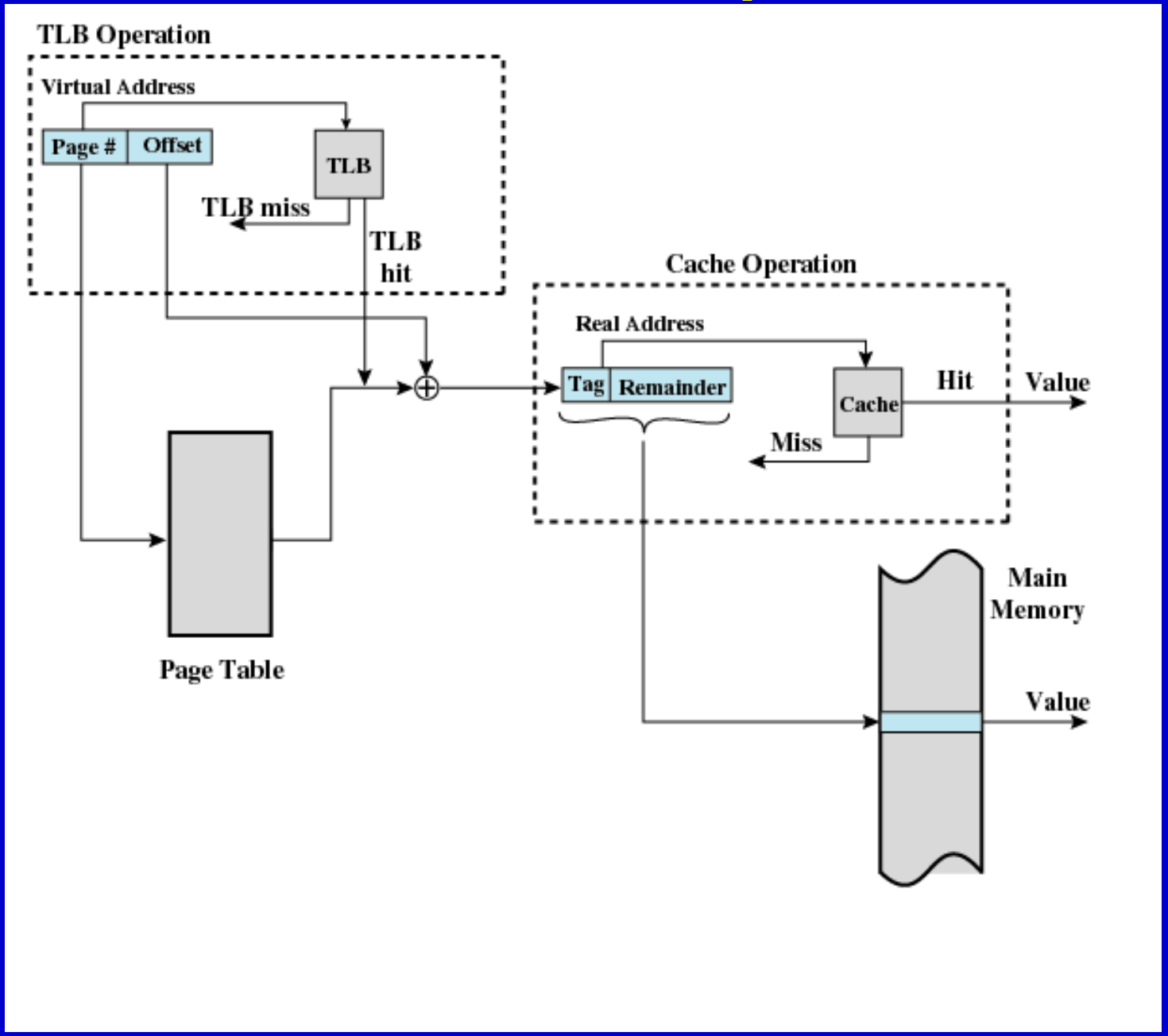


Use of a Translation Lookaside Buffer

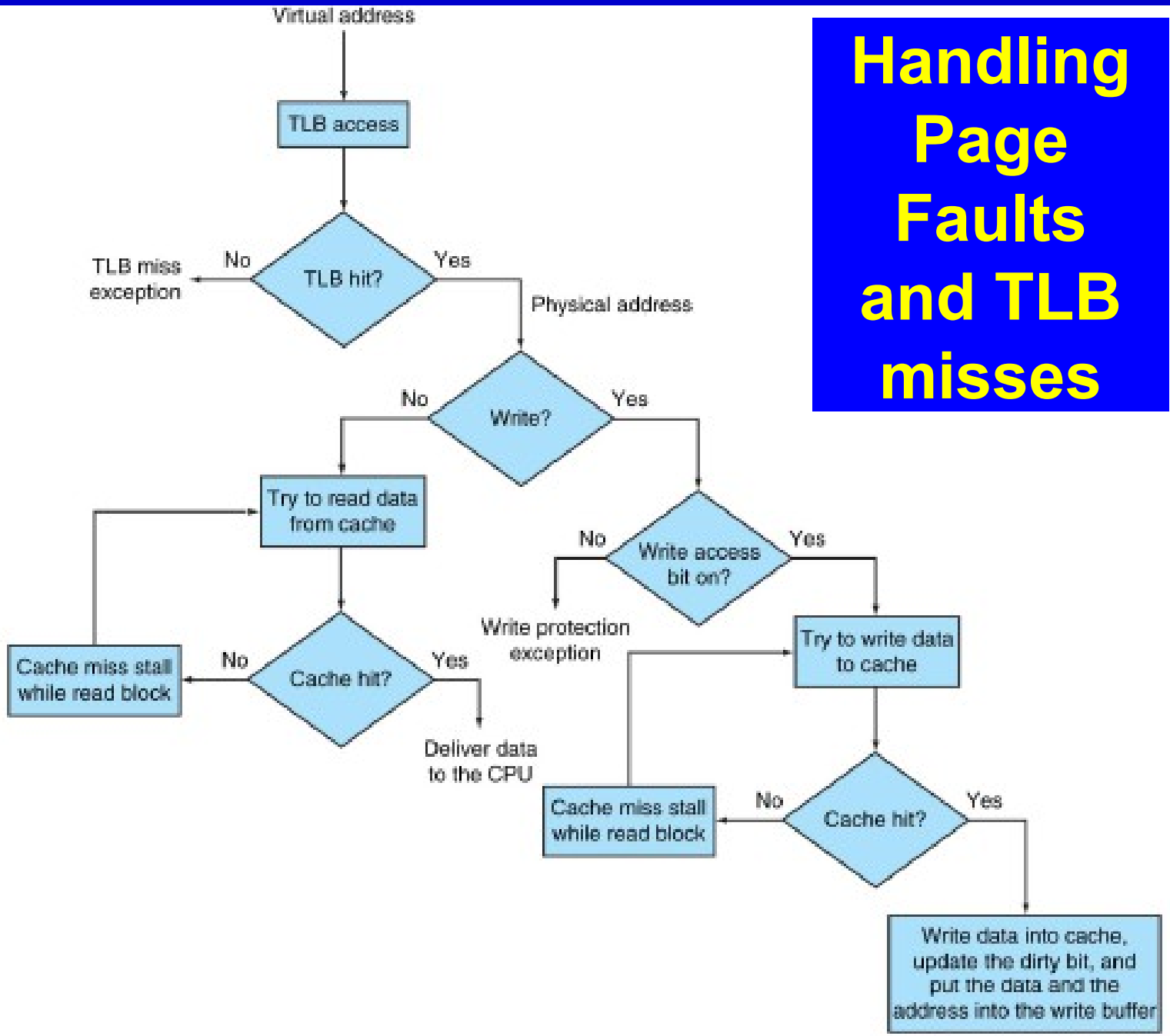


The page table is walked for every TLB miss,
but not every TLB miss indicates a page fault

TLB and Cache Operation



Handling Page Faults and TLB misses



Handling Page Faults and TLB misses

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

Memory Management

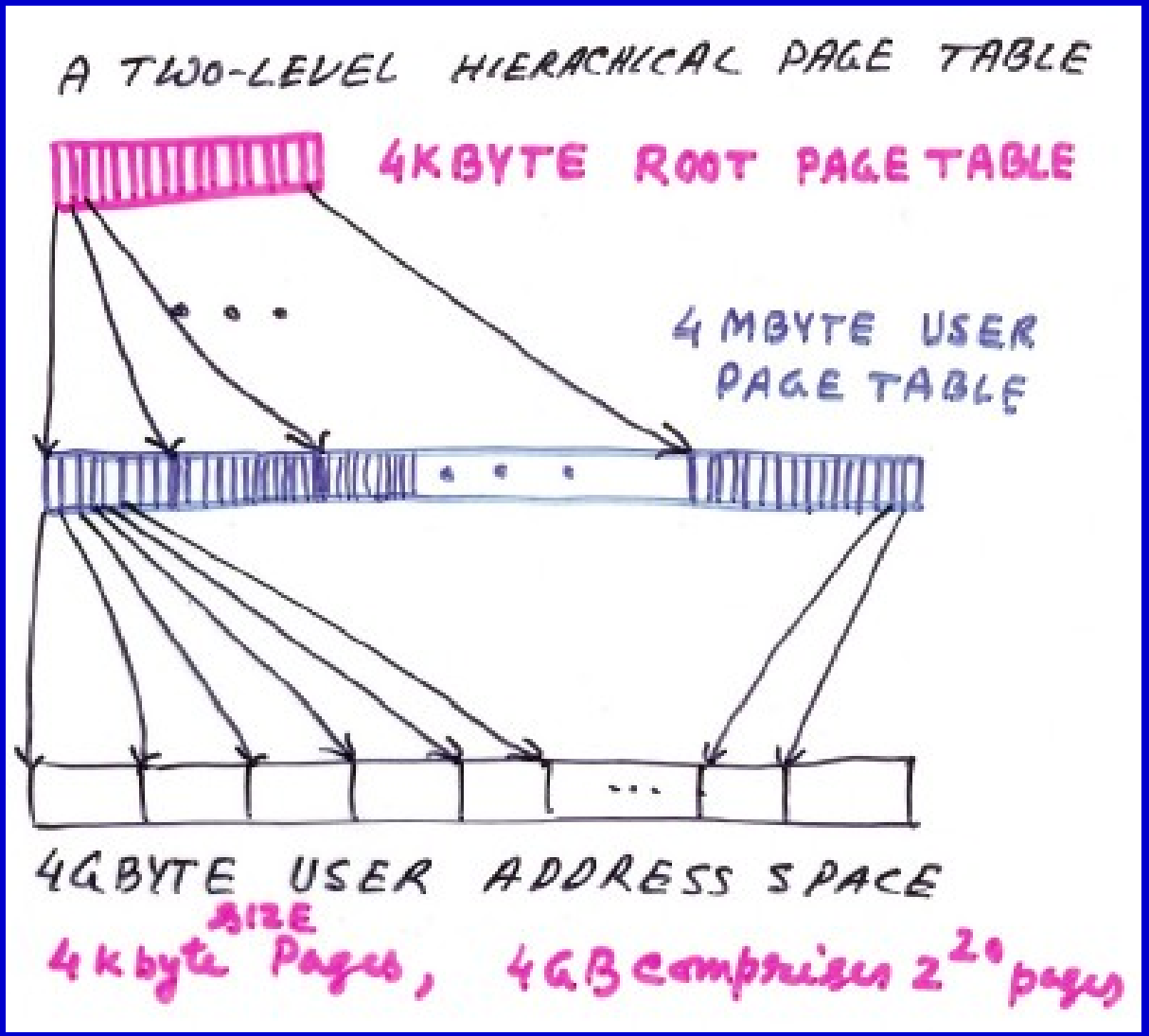
- To allow sharing of a single memory by multiple processes
- To enable the OS to implement protection, the hardware must provide three basic support
  - Two modes of operation; User, Supervisory
  - Provide a portion of the CPU state, which user can read but not write
  - Mechanism to switch from one mode to the other and context switching

Page Table Organization

- A single-level table called direct table
  - maintained entirely in hardware
  - as the table size grew it is moved to memory
  - search: page table walking (part of TLB miss)
  - suitable organization to minimize overhead
- Primary Approaches:
  - forward mapped or hierarchical page table – Indexed by virtual page number
  - inverse-mapped or inverted page table – Indexed by page frame number

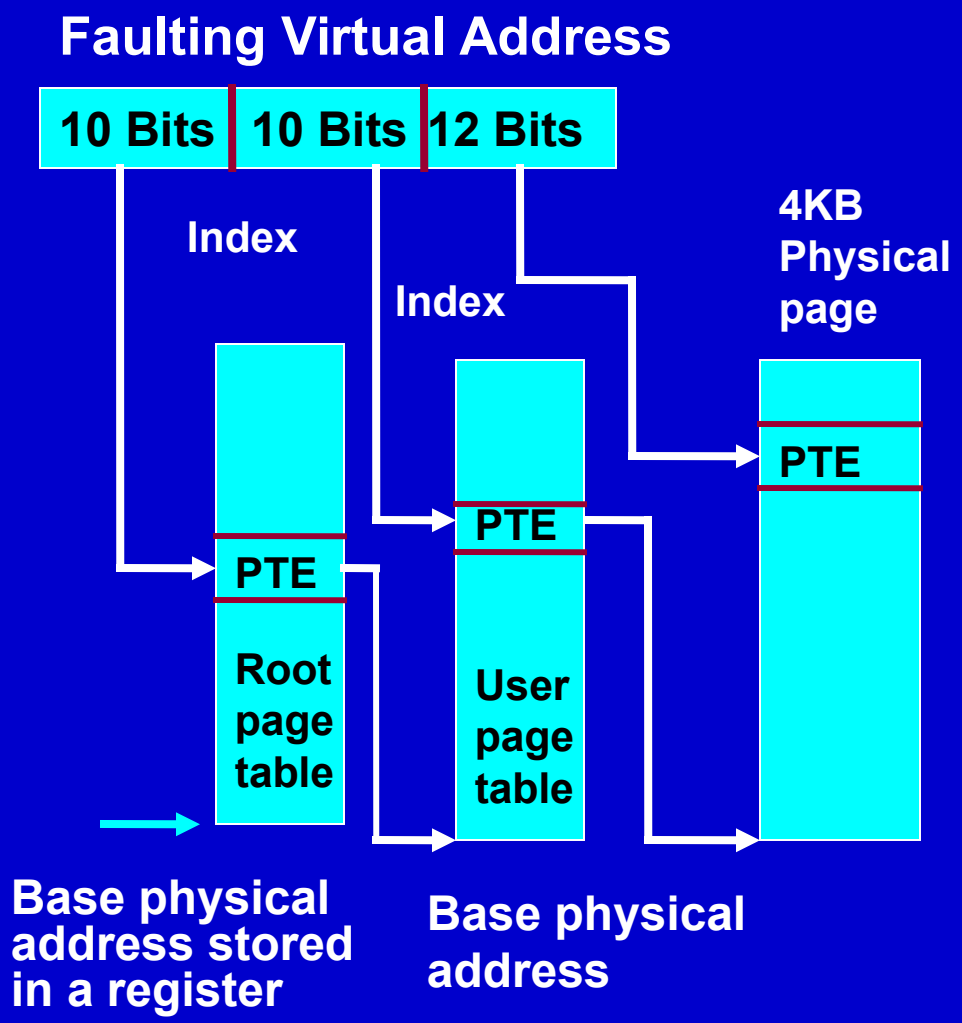
Forward Mapped or Hierarchical Page Table

- Basic idea: a large data array can be mapped by a smaller array
- A hierarchical structure with several levels
- DEC Alpha supports a four-tiered page table



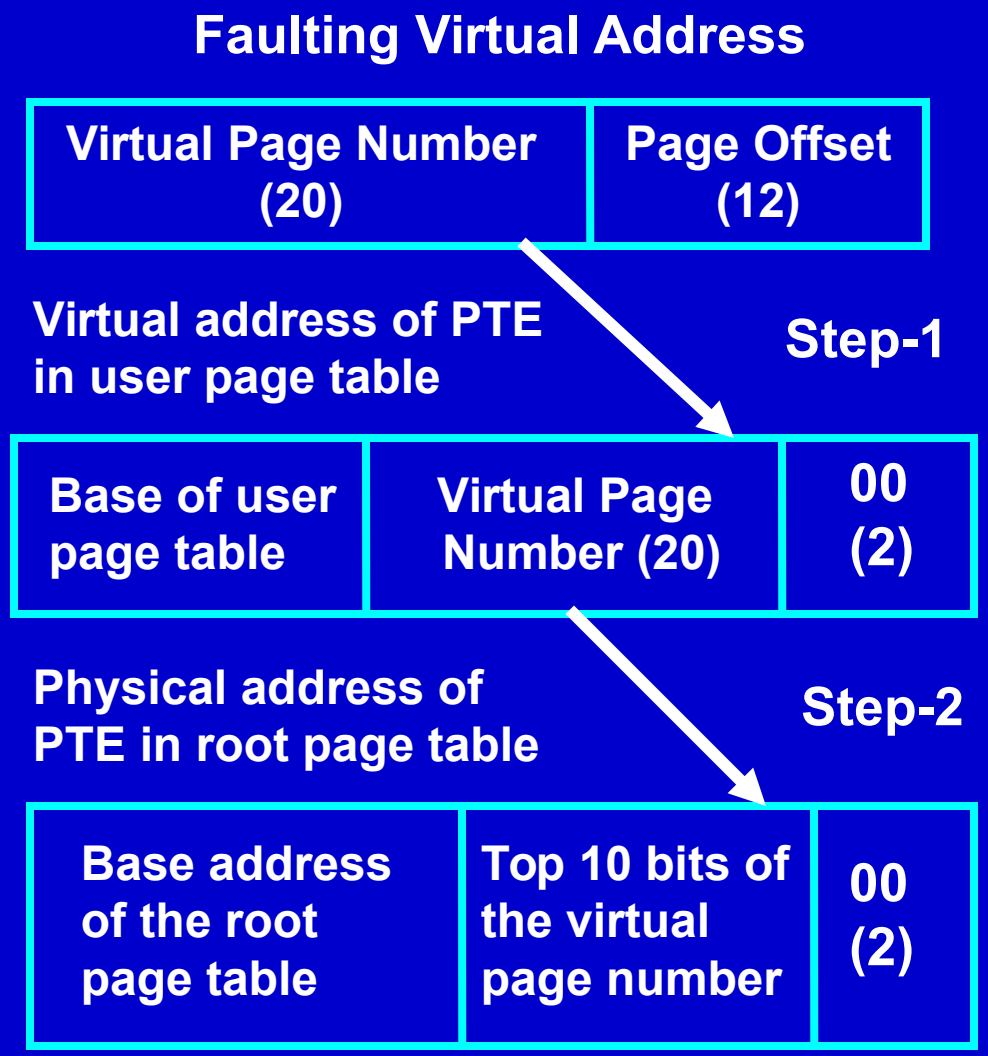
Forward Mapped or Hierarchical Page Table

- Access methods:
  - TOP DOWN
  - BOTTOM UP
- Top-down traversal
  - Top 10 bits index the 1024 entry root page table with base address stored in a hardware register
  - The referenced PTE gives the physical address of a 4-kbyte PTE
  - The top-down access method requires three memory references to satisfy one memory request



Forward Mapped or Hierarchical Page Table

- Top-down traversal requires as many memory references as there are table levels
- Bottom-up traversal reduces this overhead and typically accesses memory only once
- The top 20 bits of the virtual address are offset into the 4-Mbyte user page table
- In step 1, the top twenty bits of a faulting virtual address are concatenated with the virtual offset of the user page table.
- In step 2, the operating system generates a second address when the virtual PTE access fails.



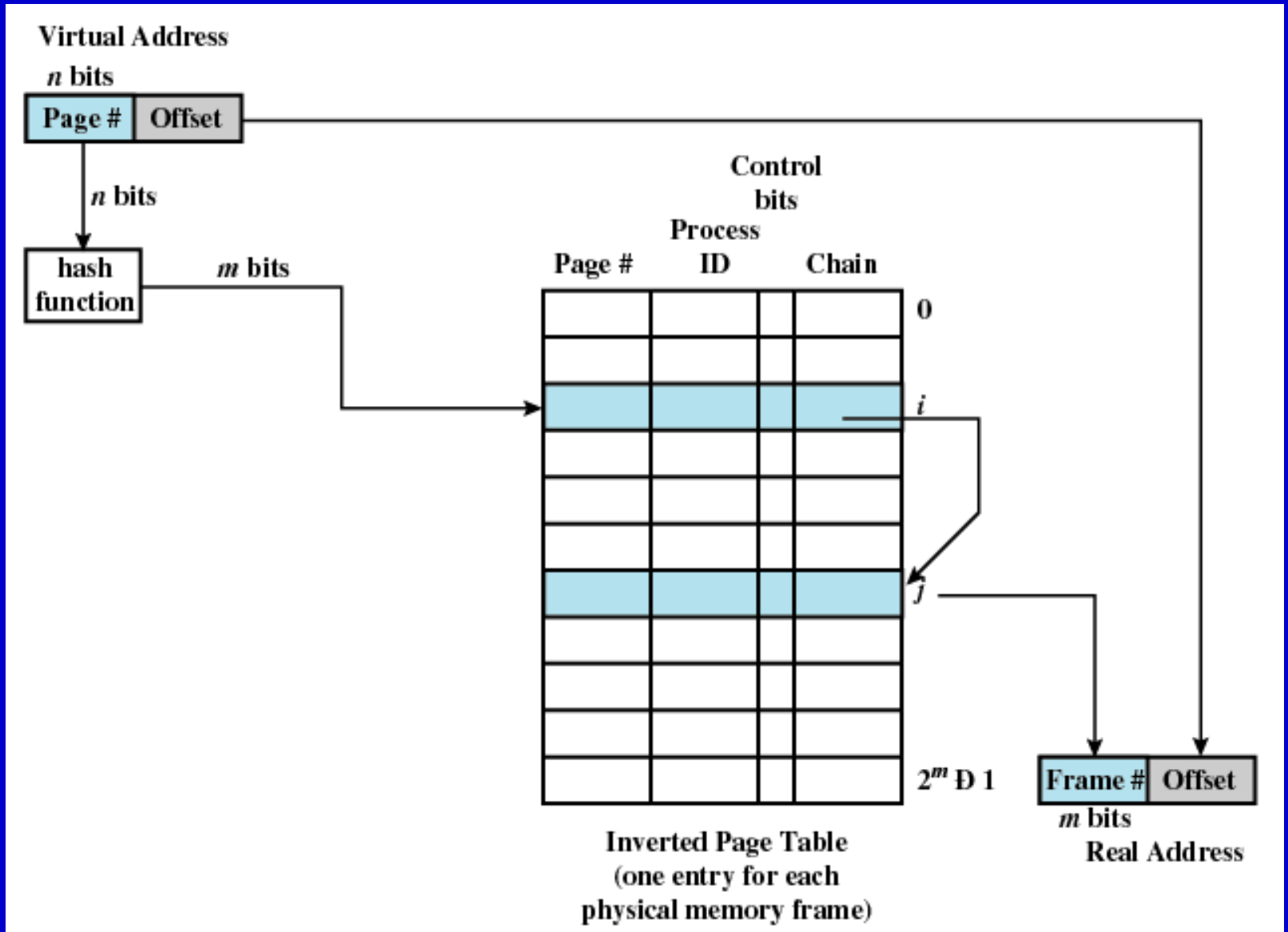
Two-level Page Table Advantages

- Only page directory and active page tables need to be in main memory
  - Page tables can be created on demand
  - Page tables can be one page in size: uniform paging mechanism for both virtual memory management and actual memory contents!
- More sophisticated protection
  - Can have a different page directory per process
  - Don't have to check owner of page table against process ID
  - Facilitates sharing of pages
- Can scale up with 3-level scheme and keep page directory size small
  - Example -- Alpha: say an 8 KB page holds 1K directory entries
    - 1 level: 1K page table = 8 MB
    - 2 levels: 1K page directory * 1K page tables = 8 GB
    - 3 levels: 1K page directory * 1K intermediate directories * 1K page tables = 8 TB
- Alpha 21164 is specified to have a 43-bit virtual address space with 8KB pages

Inverted Page Table

- The inverted page table has one entry for every page frame in the main memory
- It is called inverted because it indexes PTEs by page frame number rather than virtual page numbers
- It is compact in size and good candidate for hardware managed mechanisms
- Fewer memory references
- Collision-chain mechanism for multiple mapping

Inverted Page Table Structure

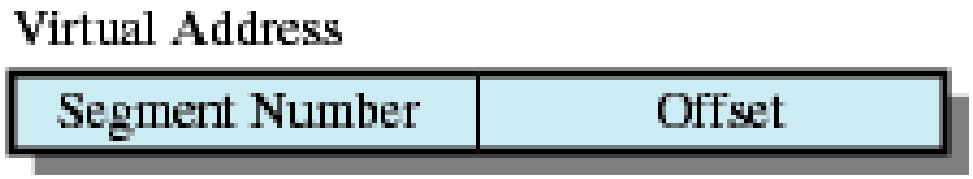


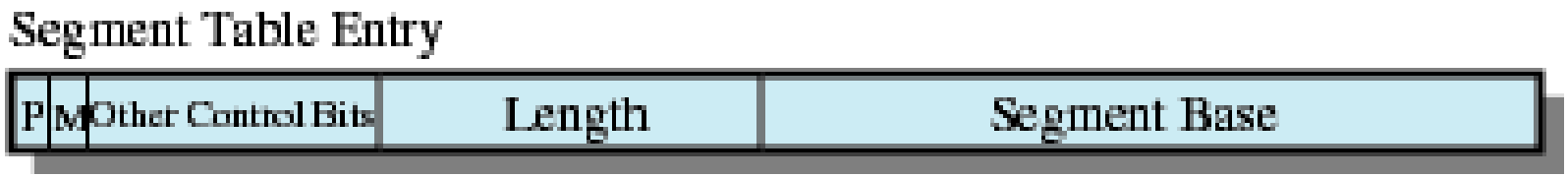
Segmentation

- A segment is a set of logically related instructions or data elements of variable size associated with a given name
- Advantages:
  - It simplifies the handling of growing data structures
  - It allows programs to be altered and recompiled independently
  - It lends itself to sharing among processes
- As paging gives efficient memory management, both can be combined to have advantages of both

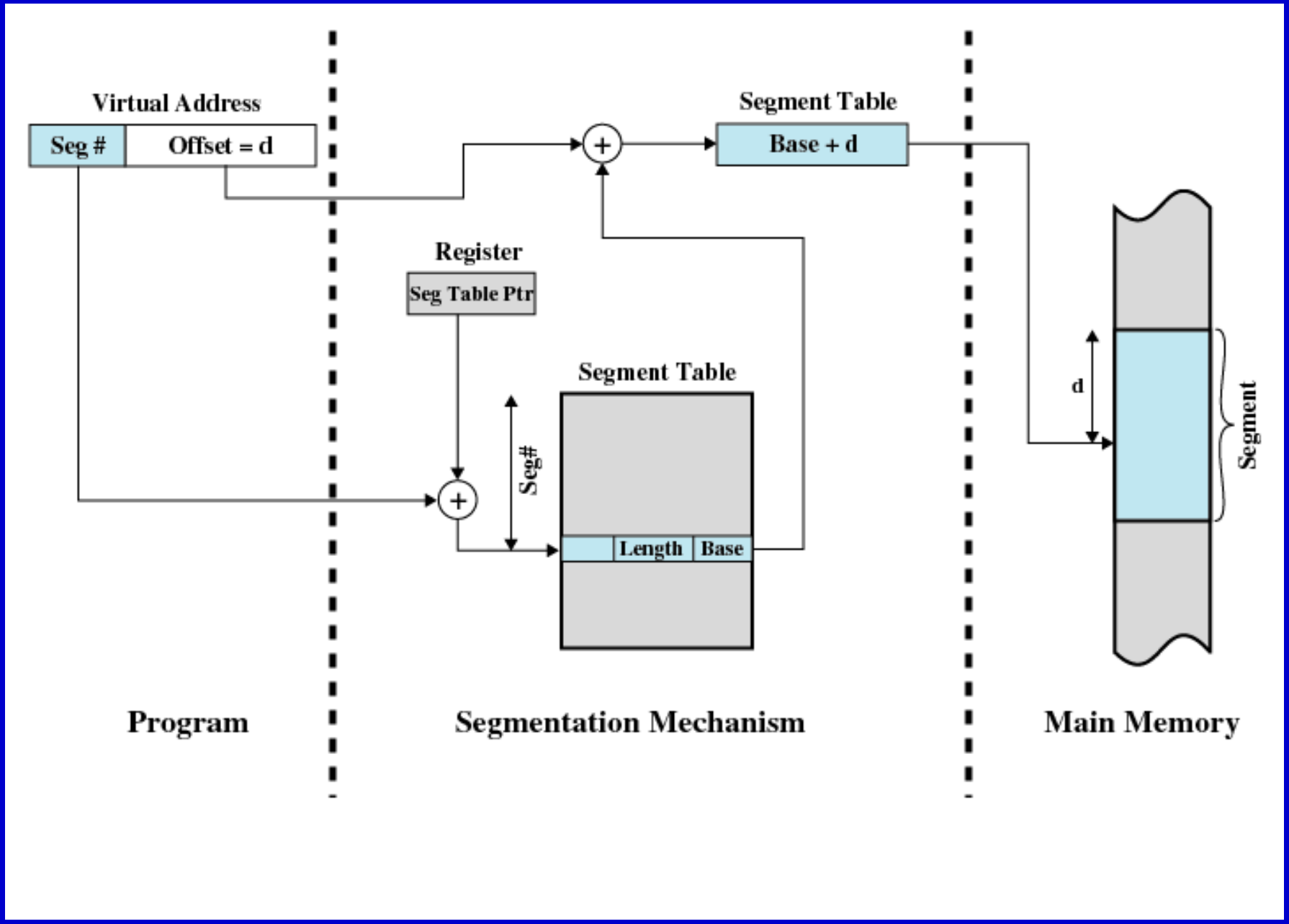
Segment Table

- One entry corresponding to each segment is present in the main memory
- Each entry contains the length of the segment
- A bit is needed to determine if segment is already in main memory (present)
- Another bit is needed to determine if the segment has been modified since it was loaded in main memory



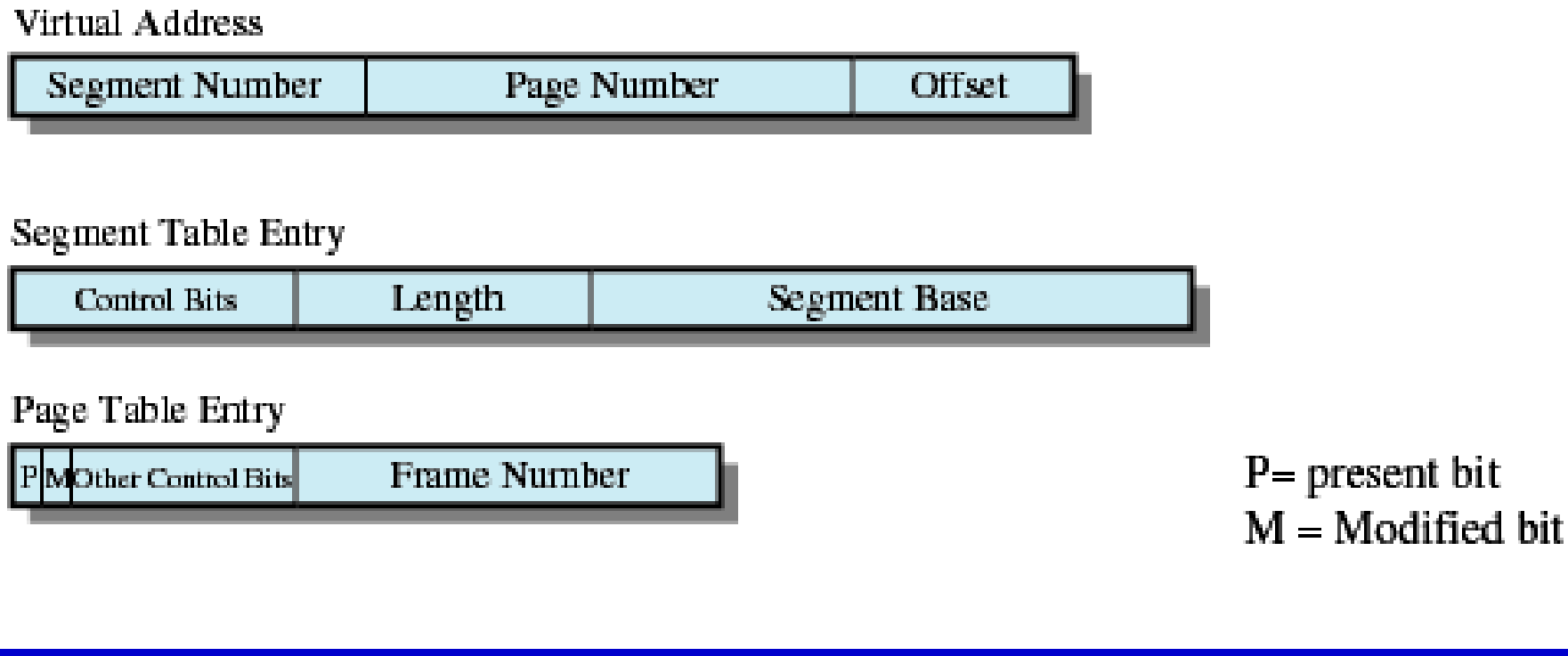


Address Translation in a Segmented System



Combined Paging and Segmentation

- Paging is transparent to the programmer
- Segmentation is visible to the programmer
- Each segment is broken into fixed-size pages



Address Translation in a Segmented/Paging System