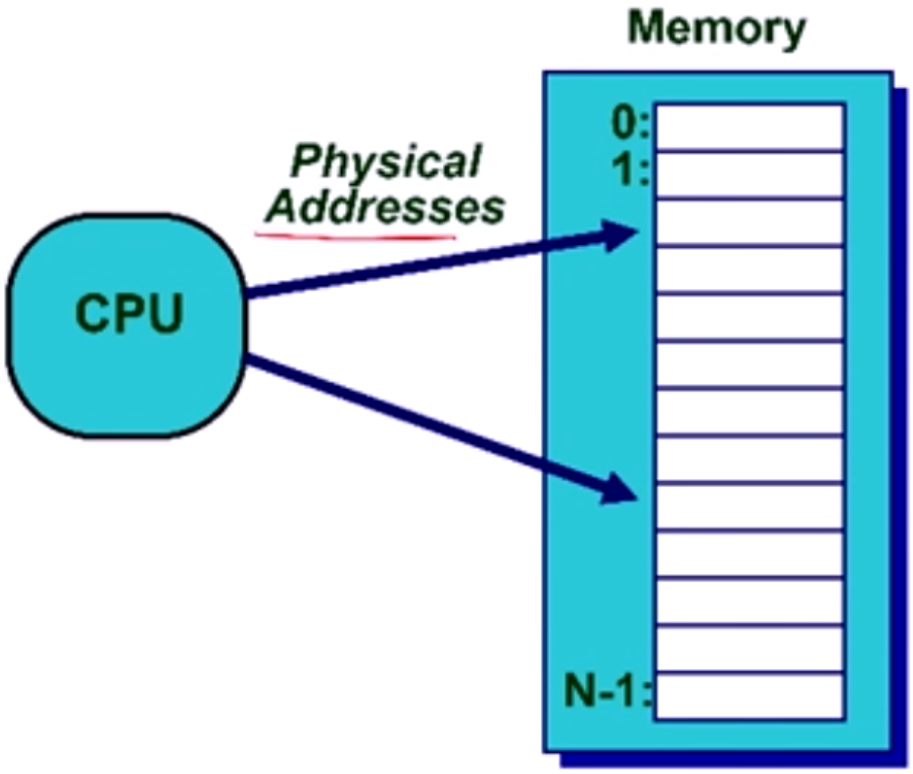
Physical Addresses Only



- Programmer needs to manage physical memory space
  - Inconvenient & hard
  - Harder when you have multiple processes
- Difficult to support code and data relocation
  - Addresses are directly specified in the program
- Difficult to support multiple processes
  - Protection and isolation between multiple processes
  - Sharing of physical memory space
- Difficult to support data/code sharing across processes

Abstraction Virtual [[Memory]]

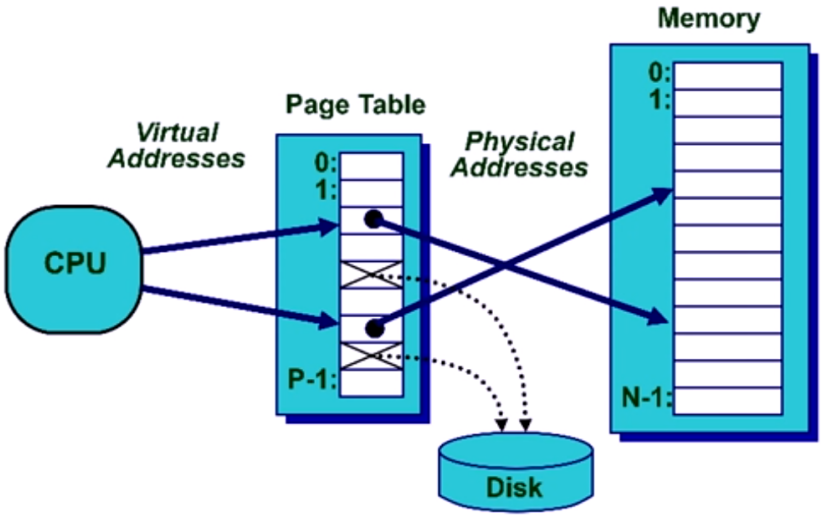
- Programmer sees virtual memory
  - Can assume the memory is "infinite"
- Reality: Physical memory size is much smaller than what the programmer assumes
- The system (system software + hardware, cooperatively) maps virtual memory addresses to physical memory
  - The system automatically manages the physical memory space transparently to the programmer

+ Programmer does not need to know the physical size of memory nor manage it → A small physical memory can appear as a huge one to the programmer → Life is easier for the programmer

-- More complex system software and architecture

- Programmer does not deal with physical addresses
- Each process has its own mapping from virtual→physical addresses
- Enables
  - Code and data to be located anywhere in physical memory (relocation)
  - Isolation/separation of code and data of different processes in physical memory (protection and isolation)
  - Code and data sharing between multiple processes (sharing)

Virtual to Physical Address Translation



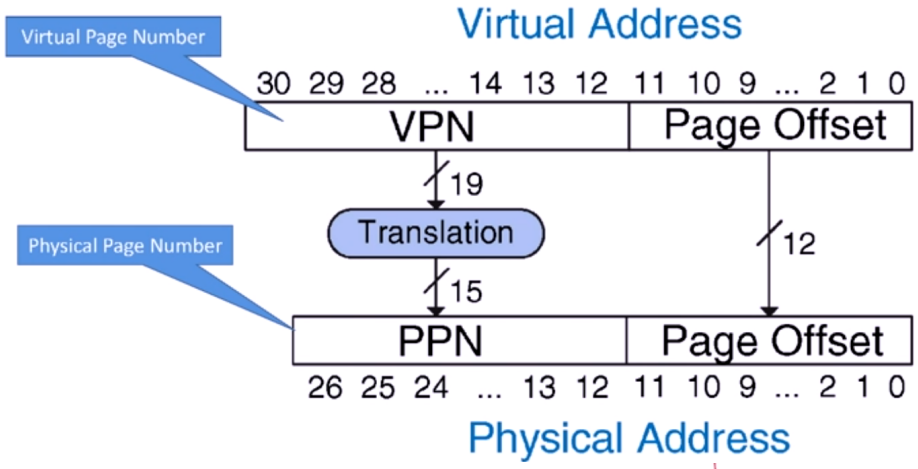
- Address Translation: The hardware converts virtual addresses into physical addresses via an OS-managed lookup table (page table)
- similar to [[Caching]]

- Virtual address space divided into pages
- Physical address space divided into frames

- A virtual page is mapped to
  - A physical frame, if the page is in physical memory
  - A location in disk, otherwise

- If an accessed virtual page is not in memory, but on disk
  - Virtual memory system brings the page into a physical frame and adjusts the mapping → this is called demand paging

Cache	Virtual Memory
Block	Page
Block Size	Page Size
Block Offset	Page Offset
Miss	Page Fault
Tag	Virtual Page Number



- Each page table entry has:
  - Valid bit: whether the virtual page is located in physical memory (if not, it must be fetched from the hard disk)
  - Physical page number: where the virtual page is located in physical memory
  - (Replacement policy, dirty bits)

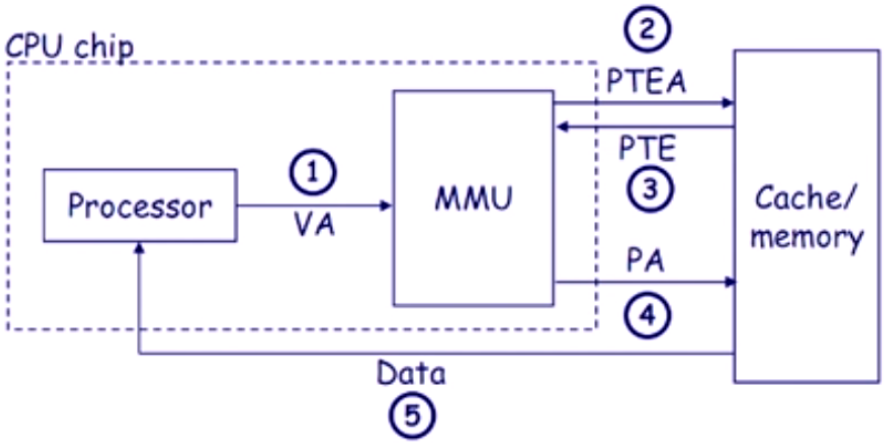
Challenges

- Challenge 1: Page table is large
  - at least part of it needs to be located in physical memory
  - solution: multi-level (hierarchical) page tables
- Challenge 2: Each instruction fetch or load/store requires at least two memory accesses:
  1. one for address translation (page table read)
  2. one to access data with the physical address (after translation)

- Translation Lookaside Buffer TLB
  - Idea: Cache the page table entries (PTEs) in a hardware structure in the processor to speed up address translation
  - Small cache of most recently used translations (PTEs)
  - Reduces number of memory accesses required for *most* instruction fetches and loads/stores to only one

Virtual Memory Support

- Virtual memory requires both HW+SW support
  - Page Table is in memory
  - Can be cached in special hardware structures called Translation Lookaside Buffers (TLBs)



1) Processor sends virtual address to MMU
2-3) MMU fetches PTE from page table in memory
4) MMU sends physical address to L1 cache
5) L1 cache sends data word to processor