EECE 321: Computer Organization

Mohammad M. Mansour
Dept. of Electrical and Compute Engineering
American University of Beirut

Lecture 34: Virtual Memory

Virtual Memory

- VM is an imaginary memory area supported by some operating systems (for example, Windows but not DOS) in conjunction with the hardware.
  - Use main memory as a "cache" for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)

- You can think of virtual memory as an alternate set of memory addresses. Programs use these *virtual addresses* rather than real addresses to store instructions and data.

- When the program is actually executed, the virtual addresses are converted into real memory addresses. One purpose of virtual memory is to *enlarge the address space*, the set of addresses a program can utilize.

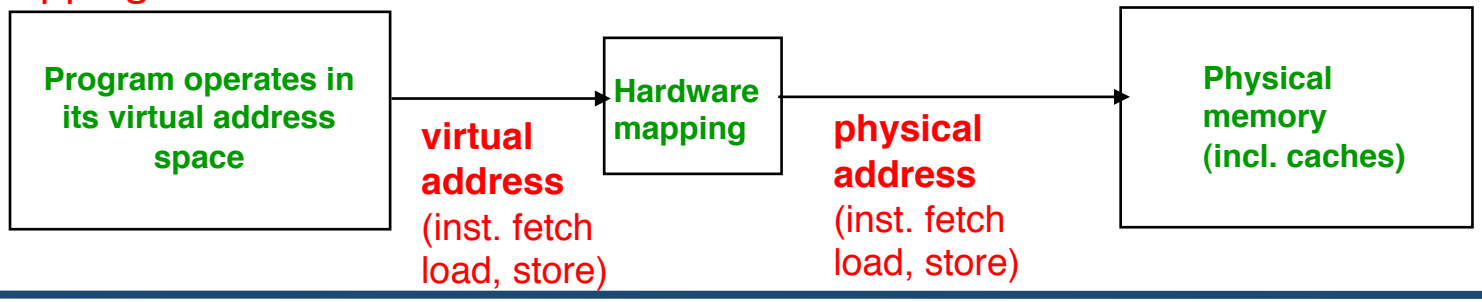
- For example, virtual memory might contain twice as many addresses as main memory. A program using all of virtual memory, therefore, would not be able to fit in main memory all at once. Nevertheless, the computer could execute such a program by copying into main memory those portions of the program needed at any given point during execution.

- To facilitate copying virtual memory into real memory, the operating system divides virtual memory into pages, each of which contains a fixed number of addresses.
  - Each page is stored on a disk until it is needed. When the page is needed, the operating system copies it from disk to main memory, translating the virtual addresses into real addresses.

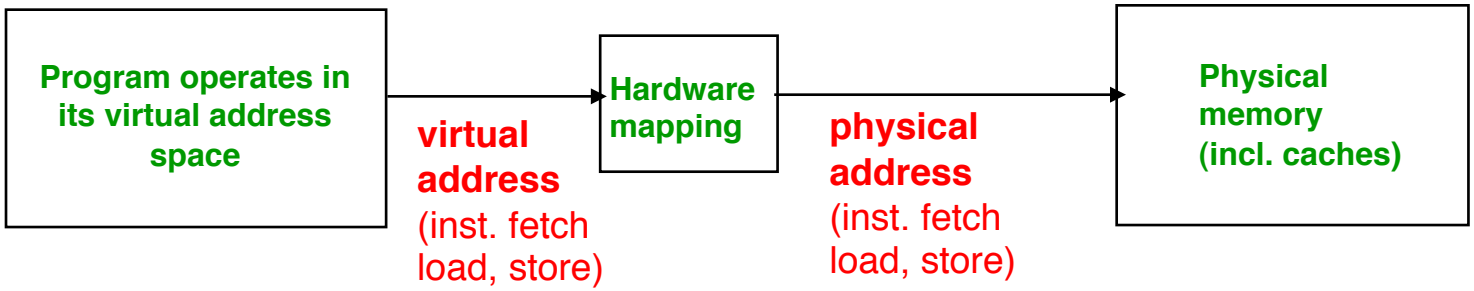
- Today with cheap memory, VM is more important for protection vs. just another level of memory hierarchy.

Virtual Memory

- Virtual memory manages the two levels of the memory hierarchy represented by main memory (sometimes called physical memory) and secondary storage.
- Historically, VM predates caches
  - In older days, main memory was very scarce.
- Employs same concept as caches, but has different terminology due to historical reasons:
  - A VM block is called a page.
  - A VM miss is called a page fault.
- With VM, CPU produces a virtual address (VA) which is translated in hardware into a physical address (PA)
  - That physical address is used to access main memory
- The process of translating virtual addresses into real addresses is called mapping. The copying of virtual pages from disk to main memory is known as paging or swapping.

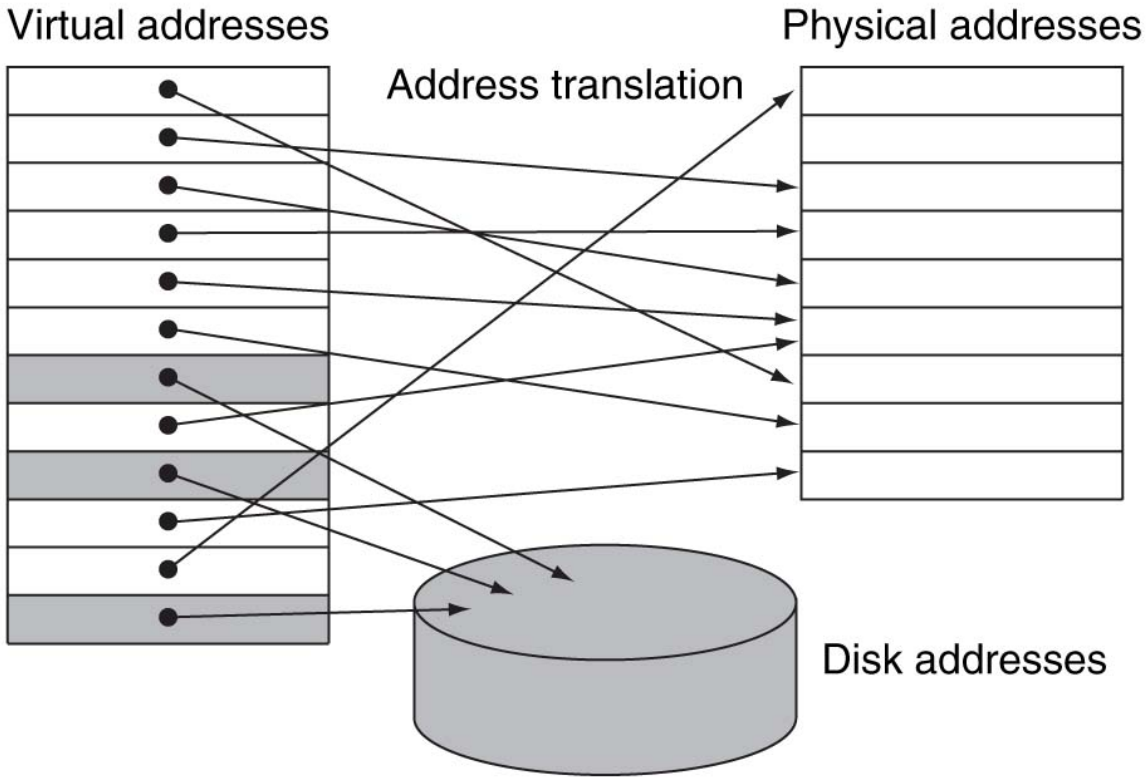


Virtual Memory



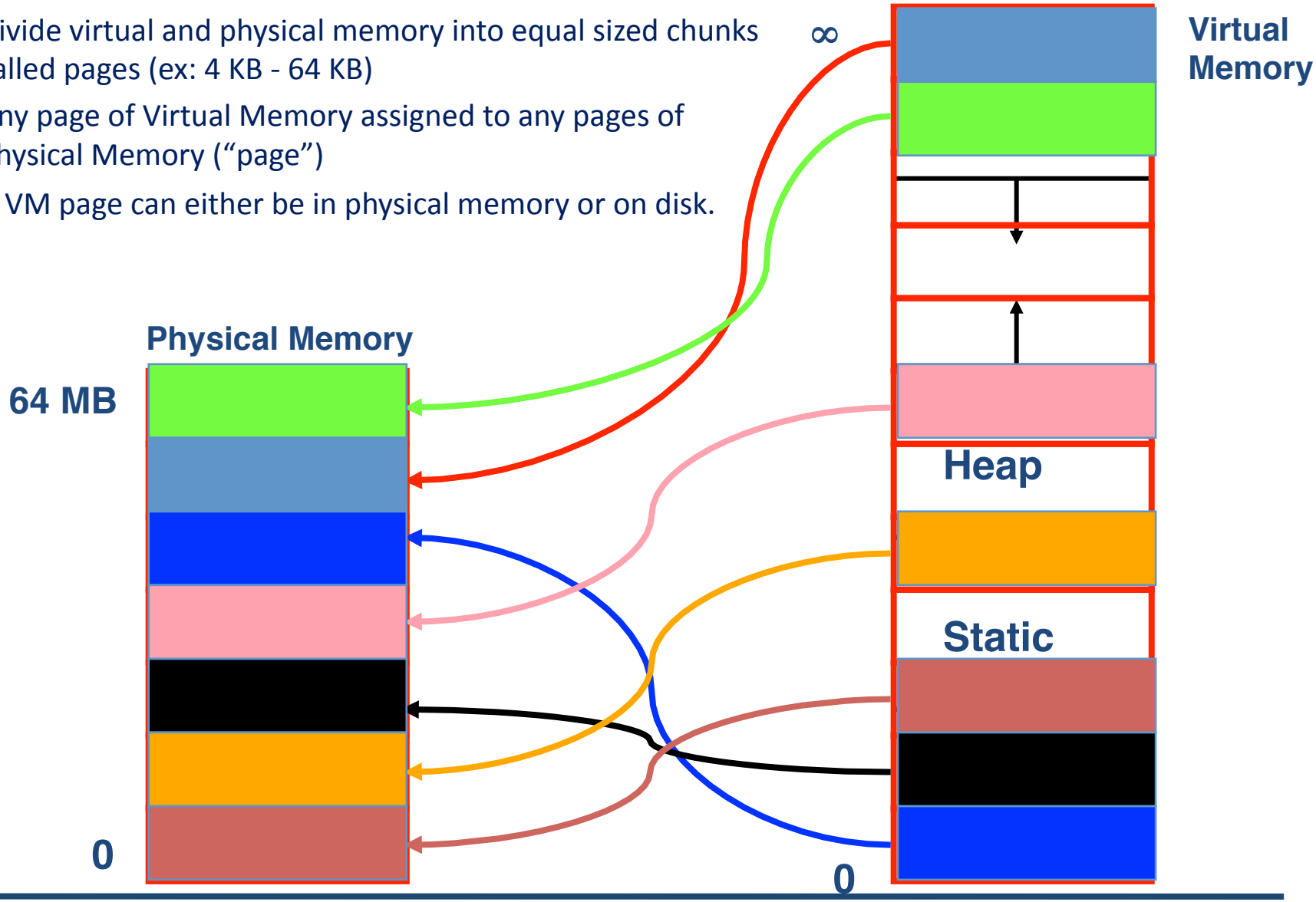
- Each program operates in its own virtual address space
  - Only portions of those programs that are running
- Each program is protected from other programs
- OS can decide where each program goes in memory
- Hardware (HW) provides virtual ⇒ physical mapping
- Historical note:
  - Early computers employed absolute addresses: virtual address = physical address
  - Only one program ran at a time, with unrestricted access to entire machine
  - Addresses in a program depended upon where the program was to be loaded in memory
  - Programmers wanted to write location-independent subroutines, and
  - Multiple programs should not affect each other inadvertently.

Virtual Memory

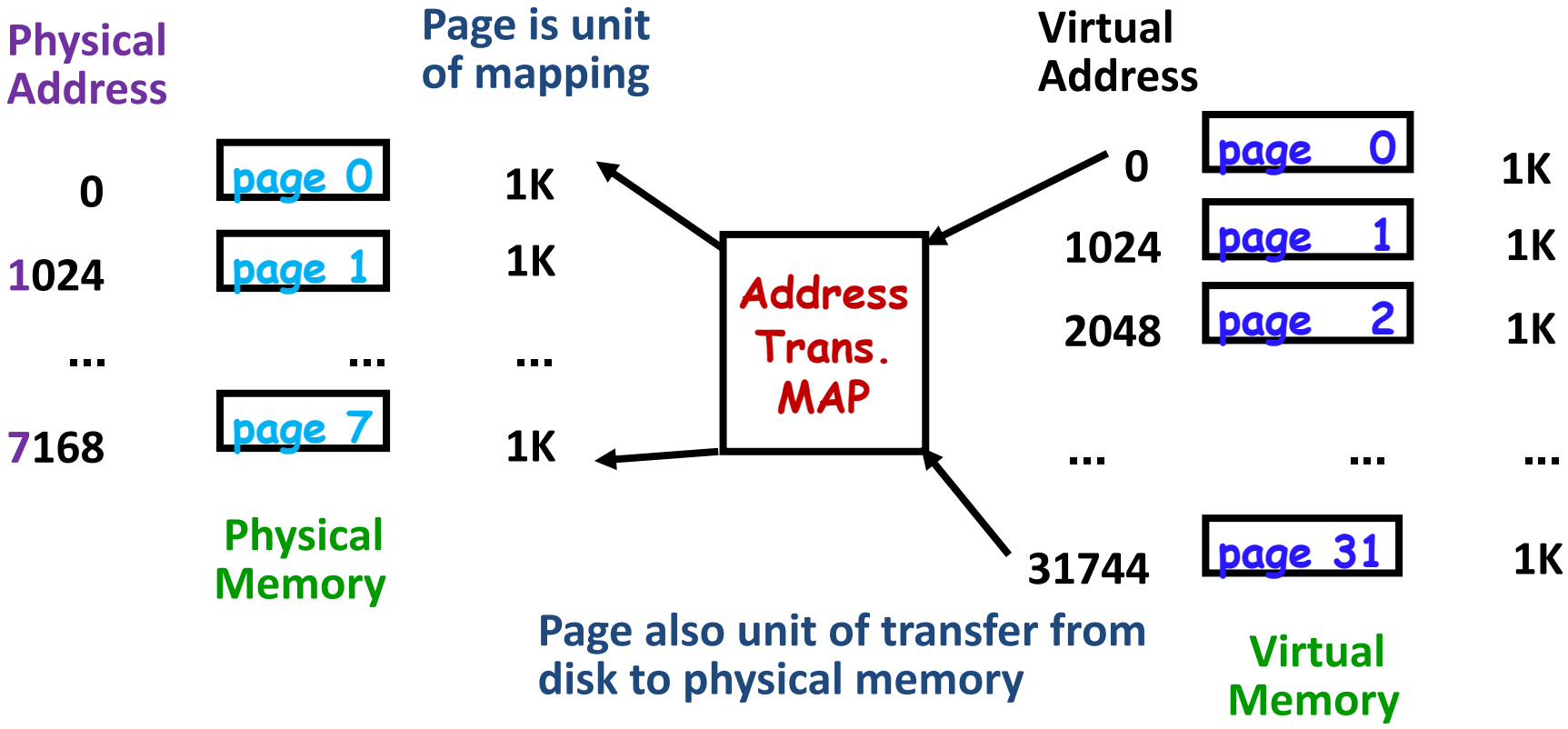


Example: Mapping Virtual Memory to Physical Memory

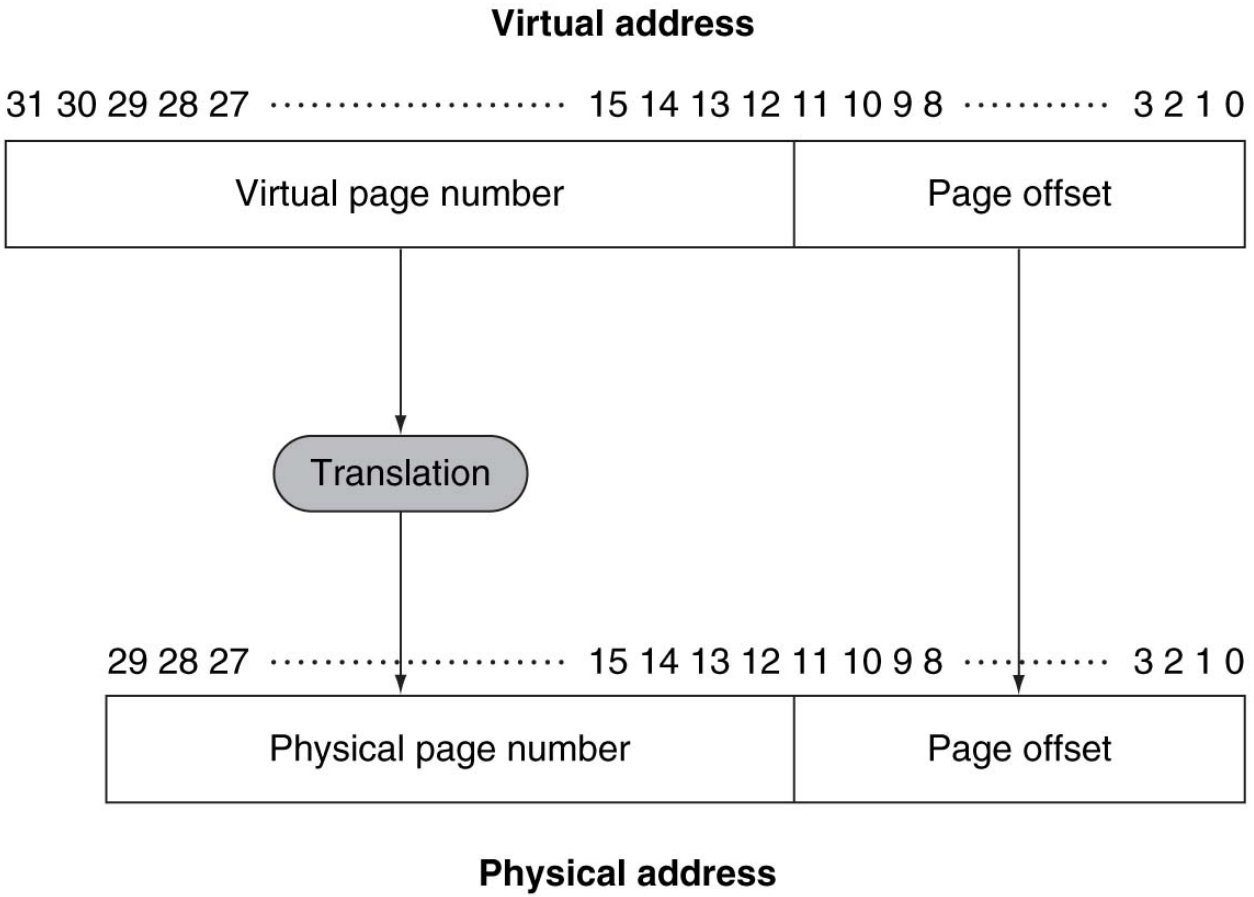
- Divide virtual and physical memory into equal sized chunks called pages (ex: 4 KB - 64 KB)
- Any page of Virtual Memory assigned to any pages of Physical Memory (“page”)
- A VM page can either be in physical memory or on disk.



Paging Organization (assume 1 KB pages)



Virtual to Physical Address Mapping



Virtual Memory Mapping Function

- Cannot have simple formula to predict arbitrary mapping: page can be anywhere
- Use a table to lookup the mappings
- Decompose virtual address into a page number and a page offset field.
- Virtual address:

- **Use table lookup ("Page Table") to store the mappings:**
  - Page number is used to index the page table

- **Virtual Memory Mapping Function**

  Physical Page Offset = Virtual Page Offset

  Physical Page Number = PageTable [Virtual Page Number]

  (P.P.N. also called "Page Frame")

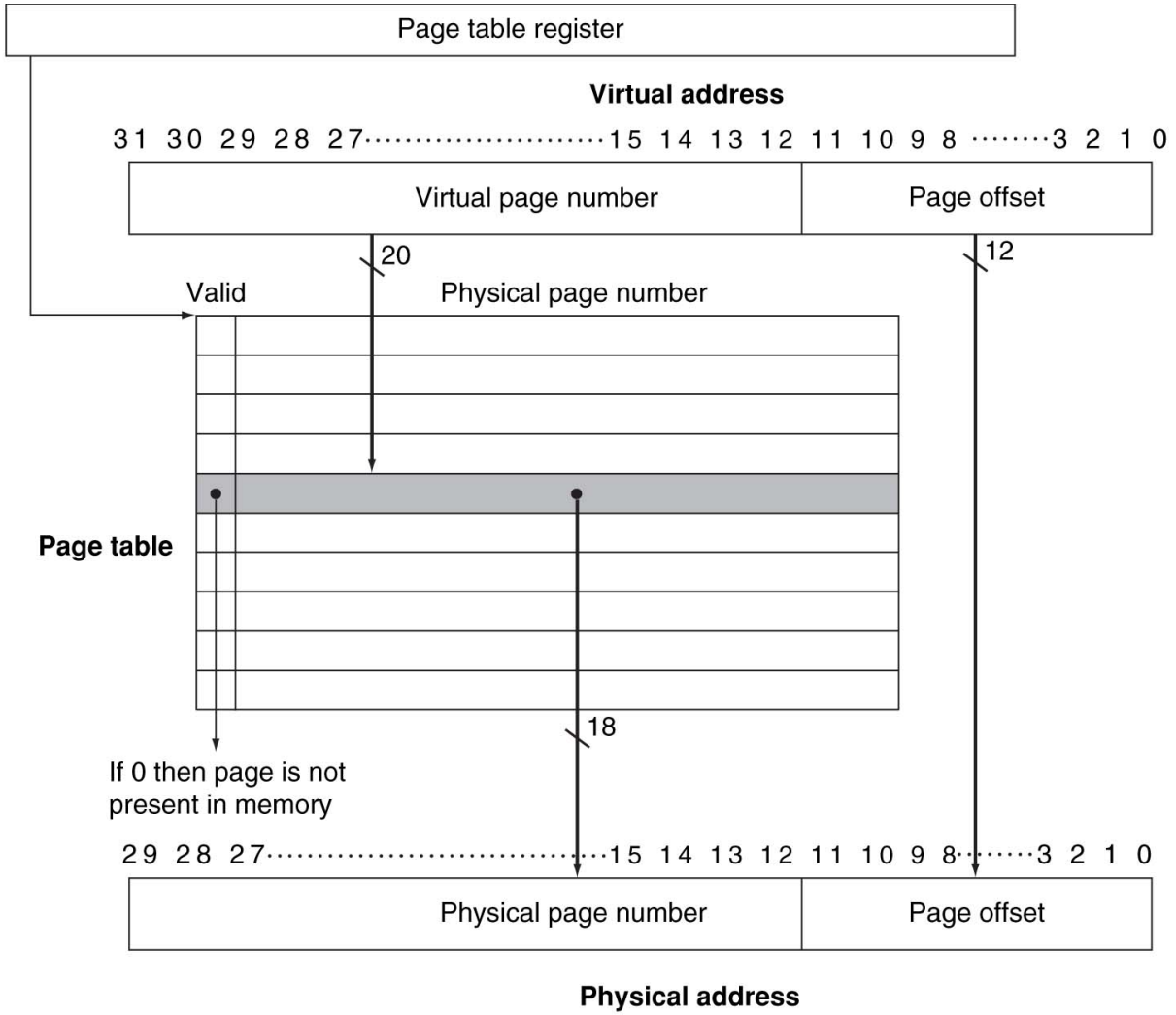
- **A page table is an operating system structure which contains the mapping of virtual addresses to physical locations**
  - There are several different ways, all up to the OS, to keep this data around

- **Each process running in the operating system has its own page table**

  "State" of a process is PC, all registers, plus page table

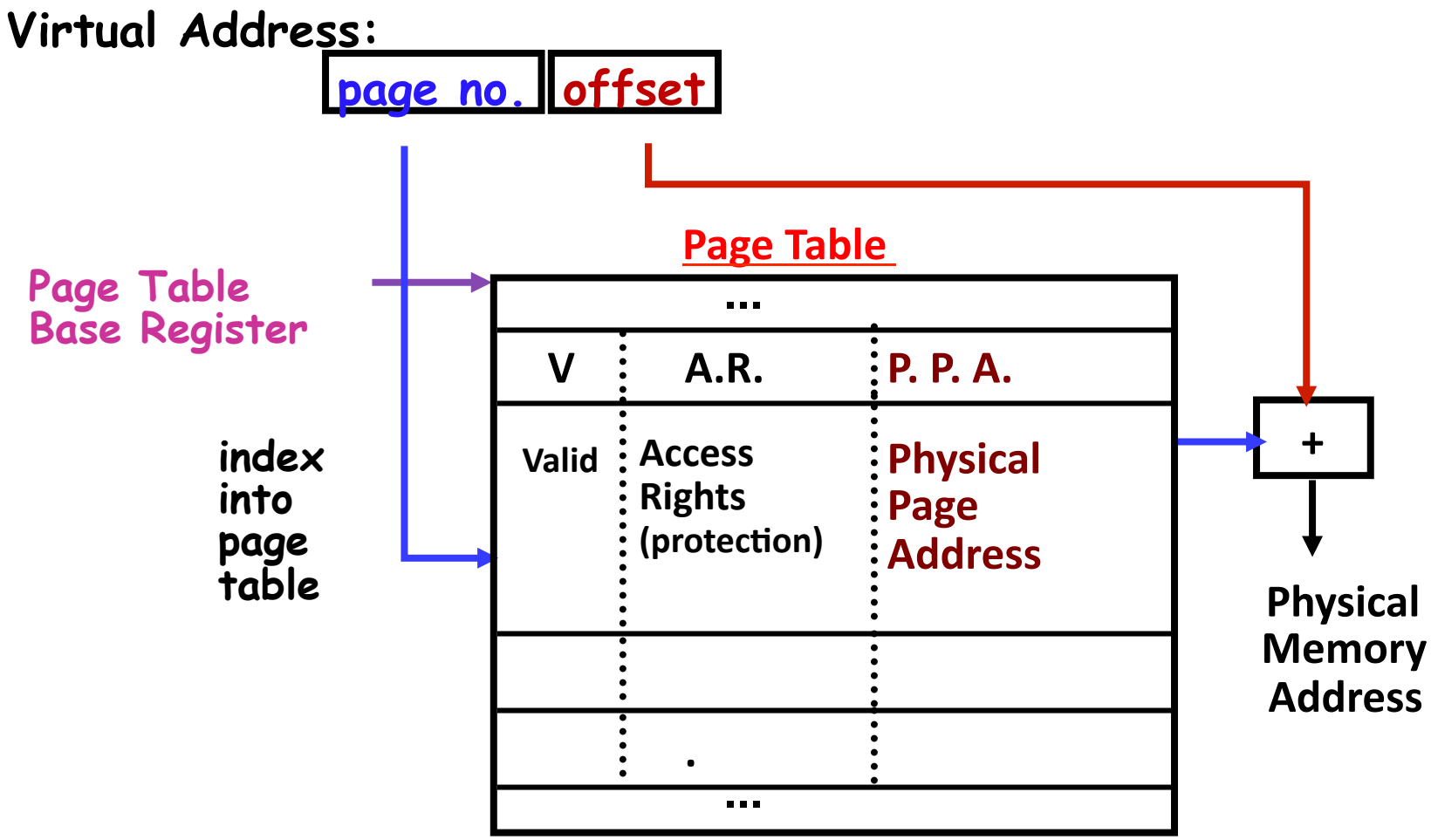
  OS changes page tables by changing contents of Page Table Base Register

Page Table Example



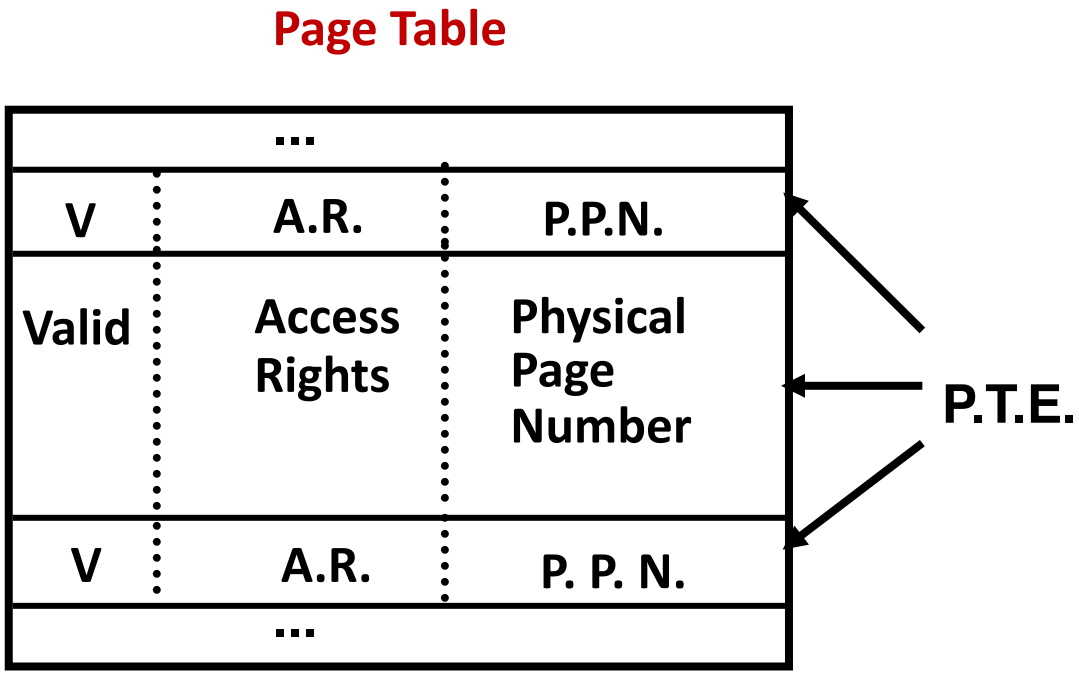
More Details about Page Table

- Page Table located in physical memory at address indicated by the Page Table Base Register



Page Table Entry (PTE) Format

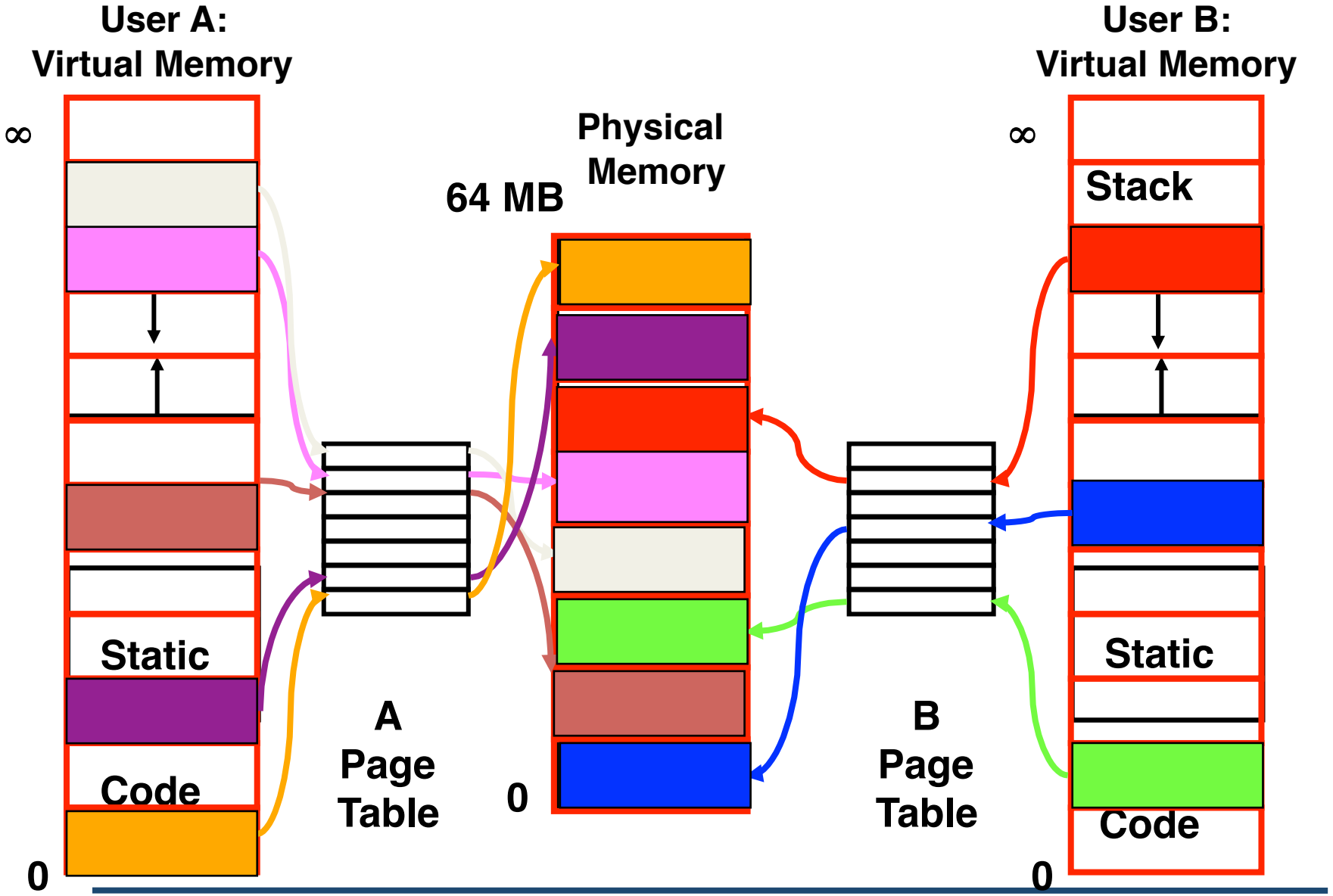
- Contains either Physical Page Number (PPN) or indication not in Main Memory
- OS maps to disk if Not Valid (V = 0)



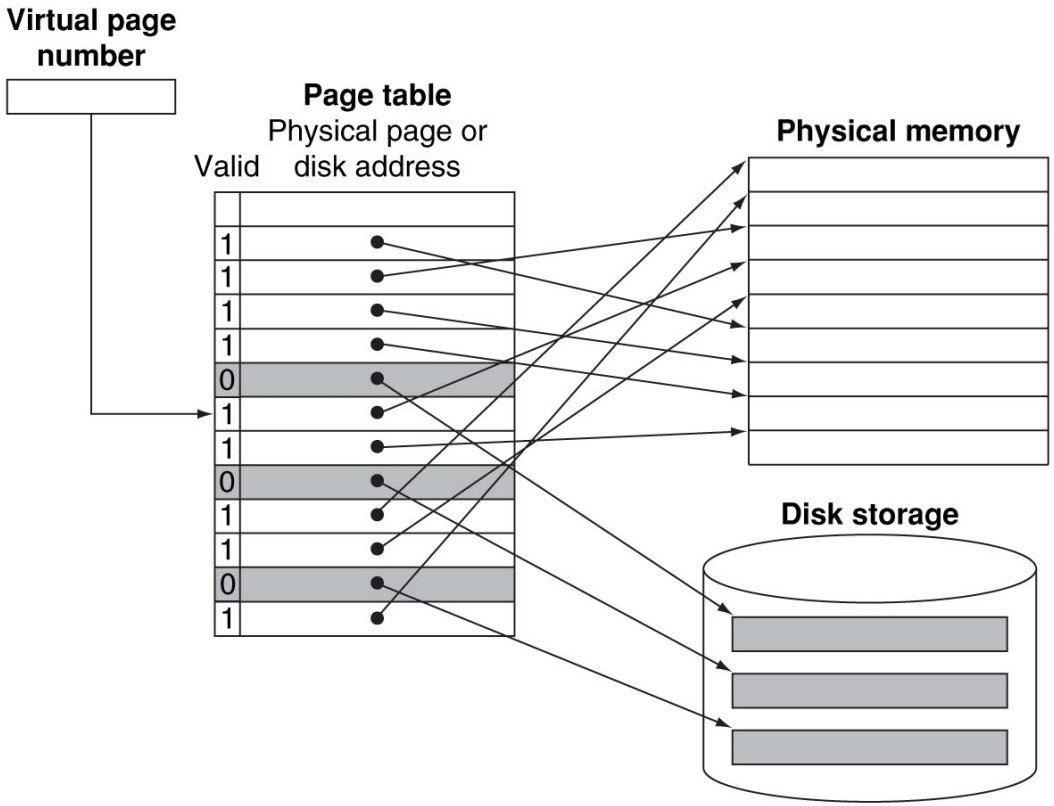
- **If valid, also check if have permission to use page:**

  Access Rights (A.R.) may be Read Only, Read/Write, Executable

Paging/Virtual Memory Multiple Processes



Page Table: Another View



Comparing the 2 Levels of Hierarchy: VM vs. Cache

Cache Version	Virtual Memory version
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-64KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU)
Write Thru or Back	Write Back

Page Size

- On Unix, use the system function sysconf() to get page size: (or simply type pagesize command)

```
#include <stdio.h>
#include <unistd.h> // sysconf(3)
int main() {
   printf("The page size for this system is %ld bytes.\n",
          sysconf(_SC_PAGESIZE)); // _SC_PAGE_SIZE is OK too.

   return 0;
}
```

- **On Windows, use the system function** the system function GetSystemInfo () from kernel32.dll

```
#include <stdio.h>
#include <windows.h>
int main() {
   SYSTEM_INFO si;
   GetSystemInfo(&si);
   printf("The page size for this system is %u bytes.\n", si.dwPageSize);
   return 0;
}
```