
EECE 321: Computer Organization

Mohammad M. Mansour

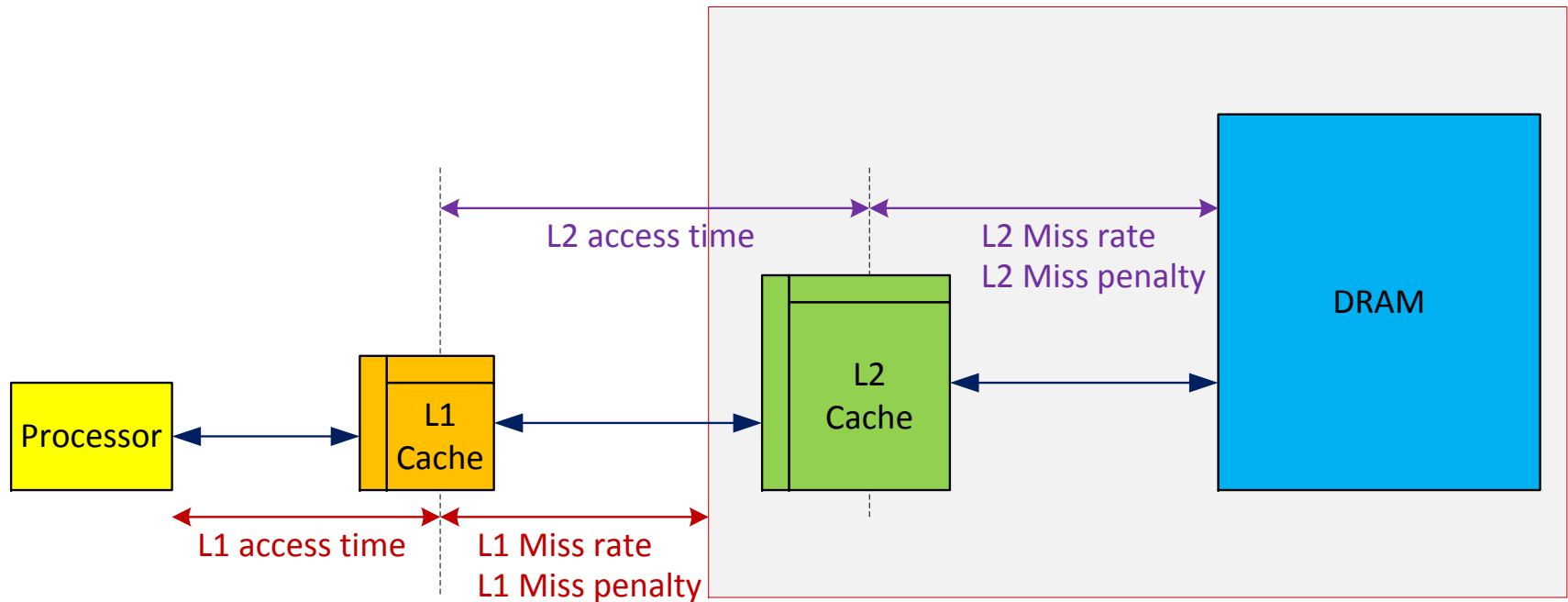
Dept. of Electrical and Compute Engineering

American University of Beirut

Lecture 33: Multi-Level Caches

Multilevel Caches

- One effective method of reducing miss penalty is to use a multilevel cache.
 - For a two-level cache, if the second-level cache contains the desired data, the miss penalty will be the access time of the second-level cache.
- This access time is typically much less than the access time of main memory.



Avg. Mem Access Time = L1 Access Time + L1 Miss Rate * L1 Miss Penalty

L1 Miss Penalty = L2 Access Time + L2 Miss Rate * L2 Miss Penalty

Miss Rates

- L1 miss rate: Fraction of memory accesses from processor that miss in L1
 - These may hit in L2, or may miss in L2 as well
- L2 miss rate: (looking from the L1 side) Fraction of L1 misses that also miss in L2.
 - Ratio of all misses in L2 cache over the number of accesses to L2.
- Global miss rate: (looking from the processor side) Fraction of all memory accesses from processor that miss in all cache levels.

- Example: Assume L1 miss rate of 5%, L2 miss rate of 20%
 - Out of 1000 memory accesses issued by processor to L1, 50 instructions miss in L1
 - These 50 instructions will be the number of memory accesses that L2 sees
 - Out of these 50, $50 \times 20\% = 10$ instructions miss in L2 and go to Memory
 - Therefore, out of the total 1000 memory accesses, only 10 go to memory (i.e. miss in L1 and L2), resulting in a global miss rate of $10/1000 = 1\%$.
 - Or directly, global miss rate = L1 miss rate \times L2 miss rate

Multilevel Caches: Typical Scale

- A 2-level cache structure allows **L1** cache to focus on minimizing **hit time** to yield a shorter clock cycle, while allowing **L2** cache to focus on **(global) miss rate** to reduce the penalty of long memory access times.
- Typical Scales:
- L1 Cache:
 - size: tens of KB
 - hit time: must complete in one clock cycle
 - miss rates: 1-5%
- L2 Cache:
 - size: hundreds of KB
 - hit time: few clock cycles
 - miss rates: 10-20%
- Why is L2 miss rate so high?
 - L1 already filters accesses with good temporal and spatial locality
- Benefit: L2 lowers the global miss rate:
 - Ex: L1 miss rate = 4%, L2 miss rate = 30%, Global miss rate = 1.2%
 - Hence L2 lowered the miss rate from 4% to 1.2%, a reduction of 70% in miss rate

Multilevel Caches Example

- Example:
 - Processor with ideal CPI=1, a clock rate of 500 MHz
 - Main memory access time of 200ns (including all the miss handling).
 - L1-only cache system: 5% miss rate per instruction
 - L1-L2 cache system: L2 has a 20-ns access time for either a hit or a miss
 - Global miss rate to main memory is reduced to 2%.
 - How much faster will the L1-L2 system compared to the L1 system?
- Miss penalty to main memory is $200\text{ns}/2\text{ns} = 100$ clock cycles
 - $\text{CPI}_{1\text{-level}} = 1 + 5\% \times 100 = 6$
- Miss penalty to secondary cache is $20\text{ns}/2\text{ns} = 10$ clock cycles
 - $\text{CPI}_{2\text{-level}} = 1 + \text{Primary cache stalls} + \text{Secondary cache stalls}$
 - Primary cache stalls = % instructions that miss in L1 & hit in L2 $\times L2_{\text{access-time}}$
 - Sec. cache stalls = % instructions that miss in L1 & miss in L2 $\times L2_{\text{access-time}} +$
% instructions that miss in L1 & miss in L2 $\times \text{MM}_{\text{access-time}}$
 - $\text{CPI}_{2\text{-level}} = 1 + (5\%-2\%) \times 10 + 2\% \times (10+100) = 3.5$
- Speedup: $6 / 3.5 = 1.7$

Multilevel Caches Example

- **Alternatively,**

$$\begin{aligned} \text{CPI}_{2\text{-level}} &= 1 + \% \text{ instructions that miss in L1 \& hit in L2} \times L2_{\text{access-time}} \\ &\quad + \% \text{ instructions that miss in L1 \& miss in L2} \times L2_{\text{access-time}} \\ &\quad + \% \text{ instructions that miss in L1 \& miss in L2} \times MM_{\text{access-time}} \\ &= 1 + \% \text{ instructions that miss in L1} \times L2_{\text{access-time}} \\ &\quad + \% \text{ instructions that miss in L1 \& miss in L2} \times MM_{\text{access-time}} \end{aligned}$$

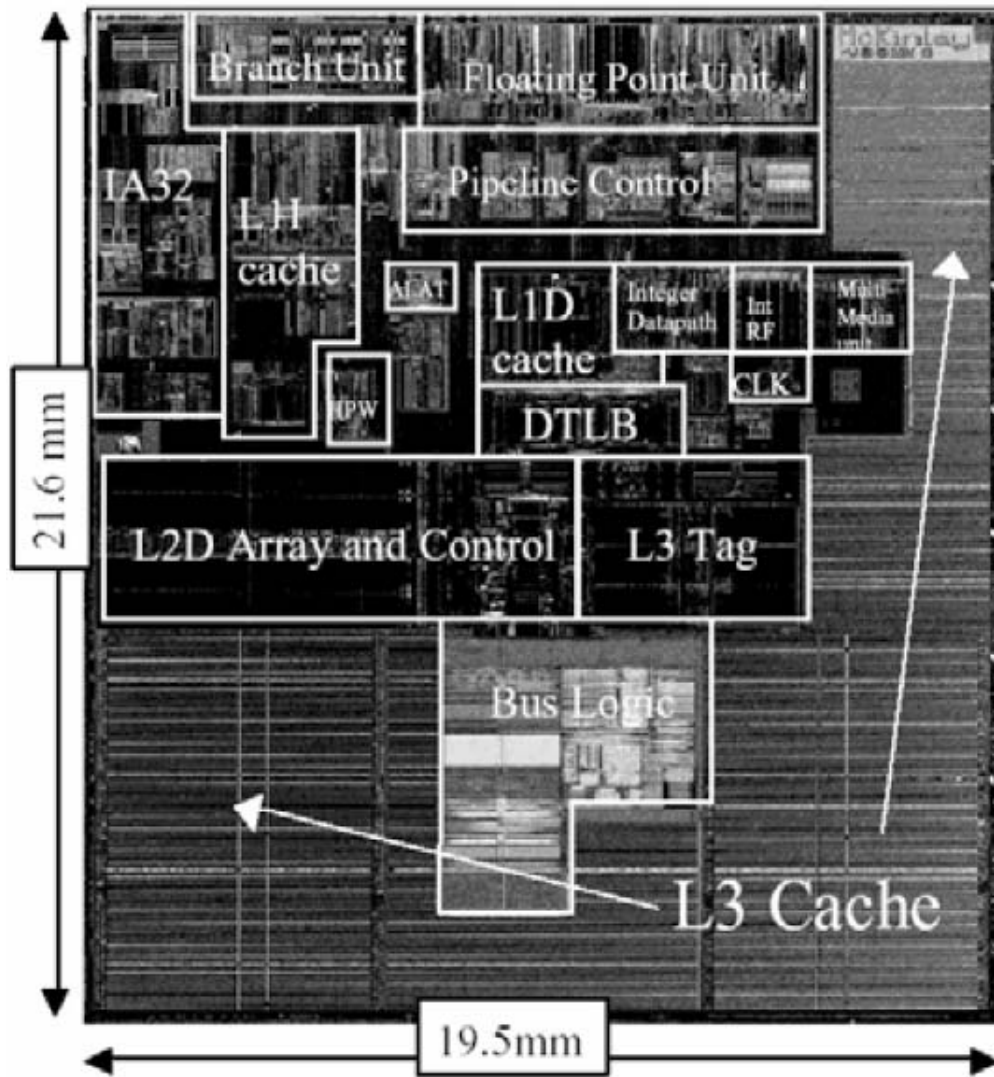
- **Typically, primary cache is smaller than secondary cache (~10 times) to keep hit time small.**

Block size is small to go with the smaller cache size and reduced miss penalty.

- **Secondary cache is often larger since its access time is less critical**

It uses larger block sizes.

Itanium-2 On-Chip Caches



Level 1, 16KB, 4-way s.a., 64B line, quad-port (2 load+2 store), single cycle latency

Level 2, 256KB, 4-way s.a, 128B line, quad-port (4 load or 4 store), five cycle latency

Level 3, 3MB, 12-way s.a., 128B line, single 32B port, twelve cycle latency

Cache Things to Remember

- Caches are NOT mandatory:
 - Processor performs arithmetic
 - Memory stores data
 - Caches simply make data transfers go *faster*
- Each level of Memory Hierarchy is subset of next higher level
- Caches speed up due to **temporal locality**: store data used recently
- Block size > 1 word **spatial locality** speedup:
Store words next to the ones used recently
- Cache design choices:
 - size of cache: speed v. capacity
 - N-way set assoc: choice of N (direct-mapped, fully-associative just special cases for N)
- Cache performance:
 - Miss Rate
 - Miss Penalty

Generalized Caching

- We've discussed memory caching in detail. Caching in general shows up over and over in computer systems:
 - File system cache
 - Web page cache
 - Game Theory databases / table-bases
 - Software *memoization* (automatically modifying functions to include caching behavior)
 - Search engines (~ Google)
 - BIND DNS daemon caches a mapping of domain names to IP addresses
- Main idea: if something is expensive to obtain but is needed repeatedly, do it once and cache the result.
- Memory hierarchy requirements:
 - If Principle of Locality allows caches to offer (close to) speed of cache memory with size of DRAM memory, then recursively why not use same idea at next level to give speed of DRAM memory, size of Disk memory?

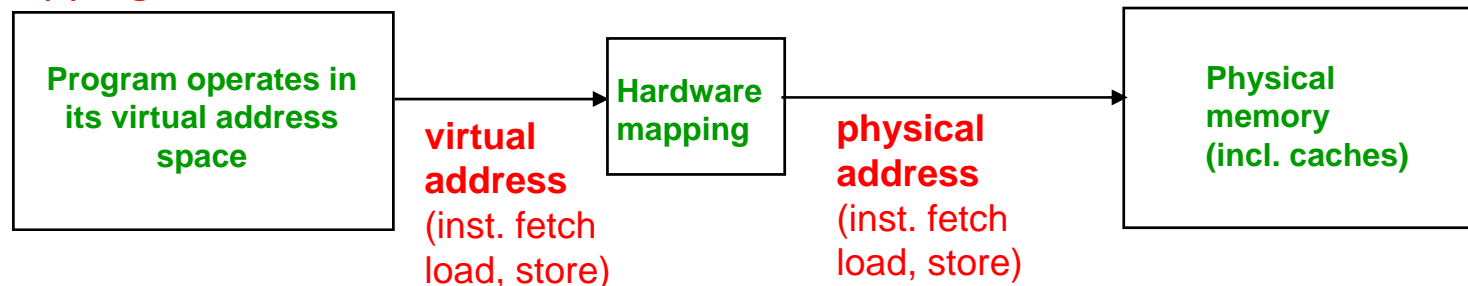
Virtual Memory

Virtual Memory

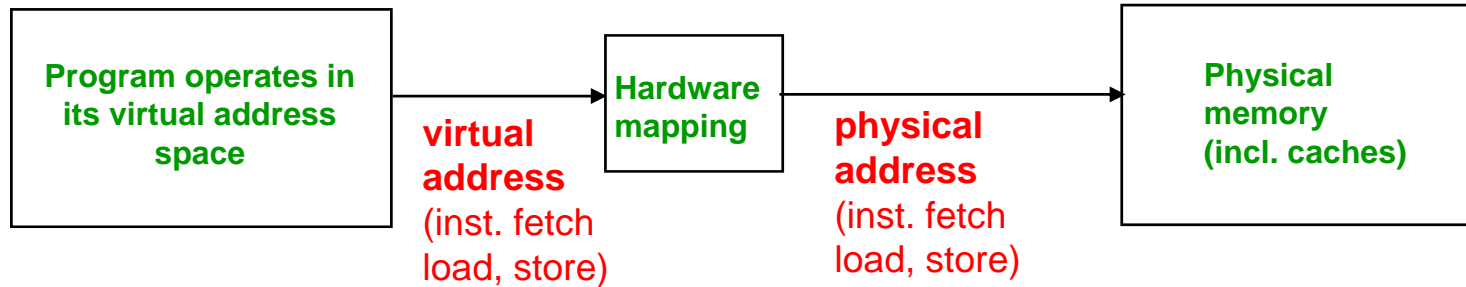
- VM is an imaginary memory area supported by some operating systems (for example, Windows but not DOS) in conjunction with the hardware.
- You can think of virtual memory as an alternate set of memory addresses. Programs use these *virtual addresses* rather than real addresses to store instructions 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**.



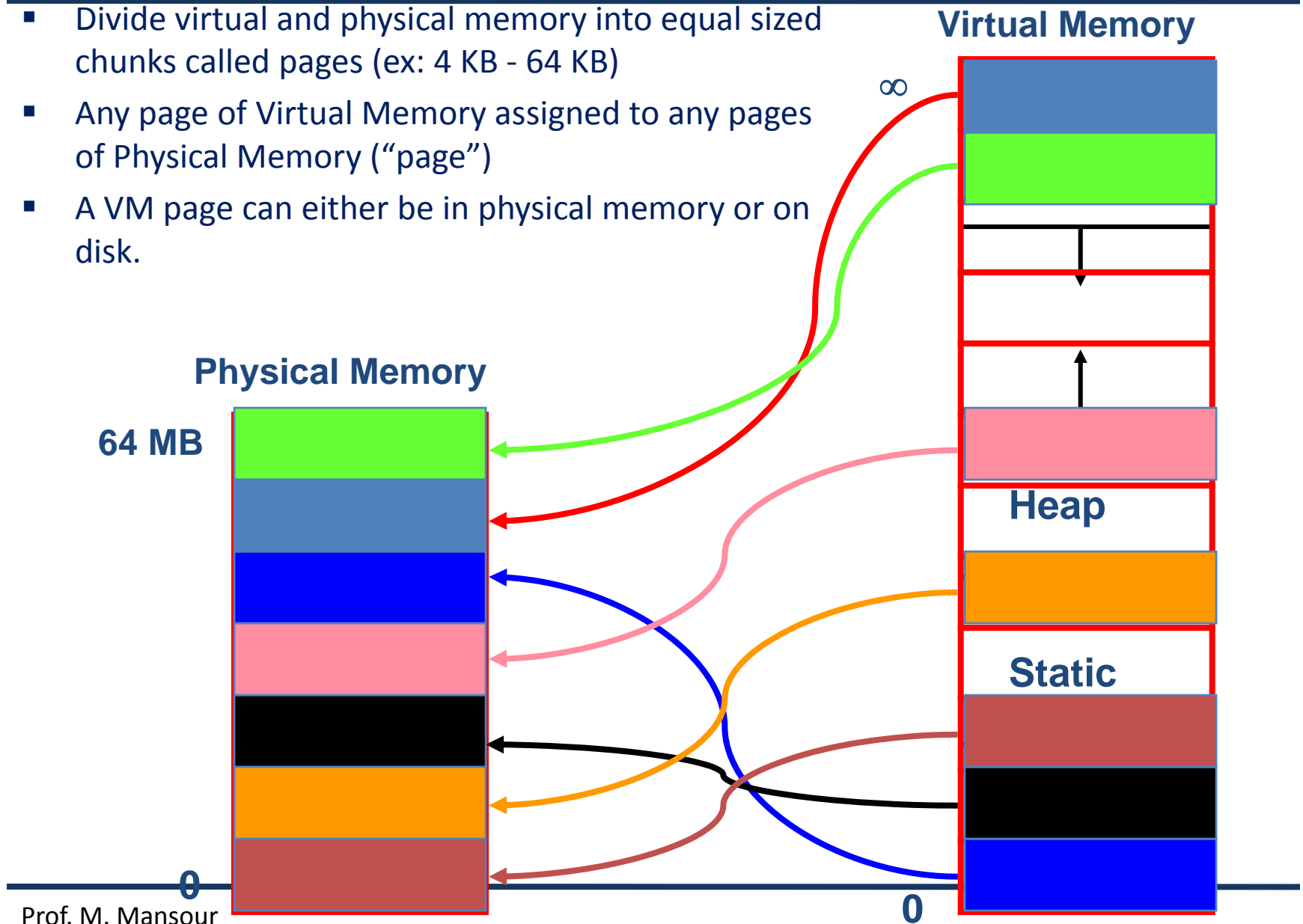
Virtual Memory



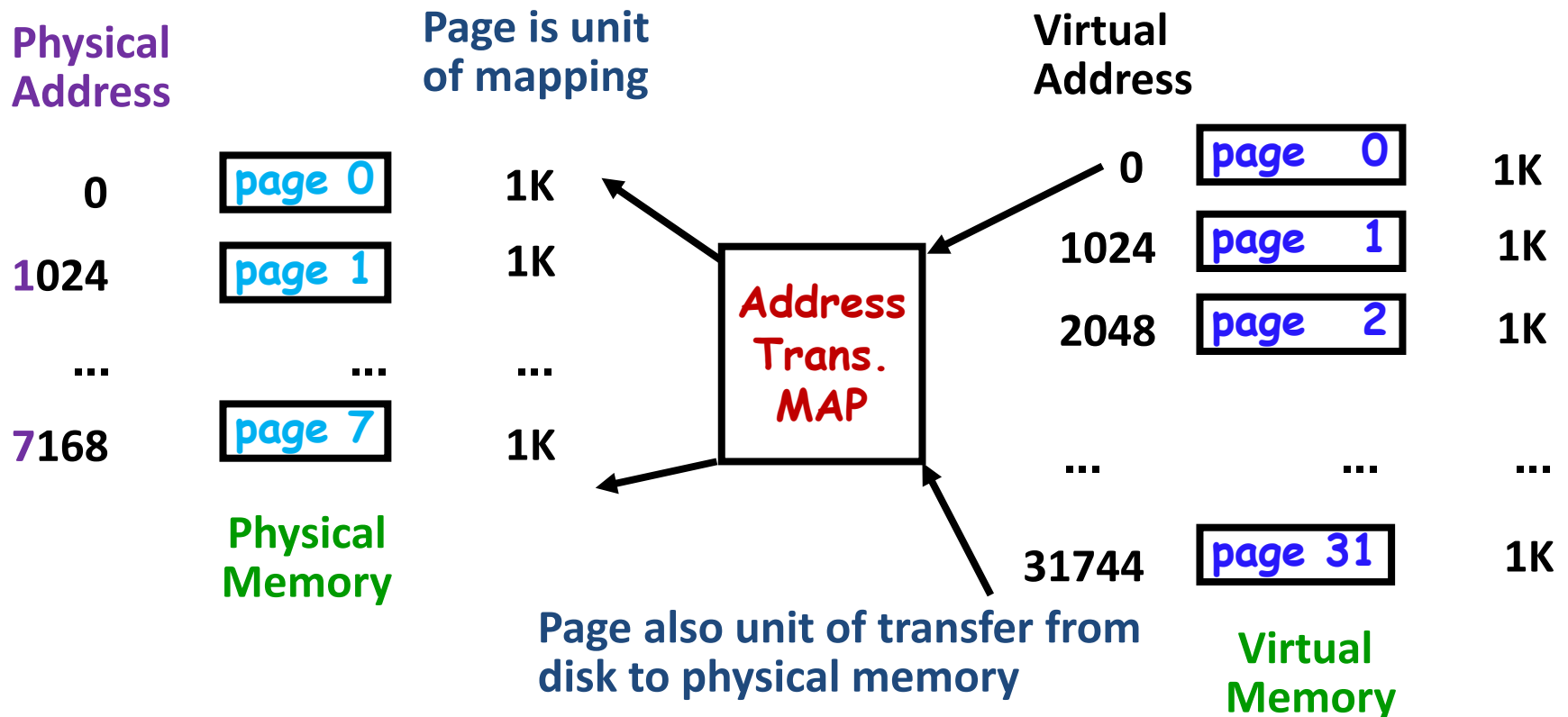
- Each program operates in its own virtual address space;
 - Only portions of those programs that are running
- Each program is protected from other programs
- OS can decide where each program goes in memory
- Hardware (HW) provides virtual \Rightarrow physical mapping
- Historical note:
 - Early computers 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.

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 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:

Page Number	Offset
--------------------	---------------

- ❑ **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

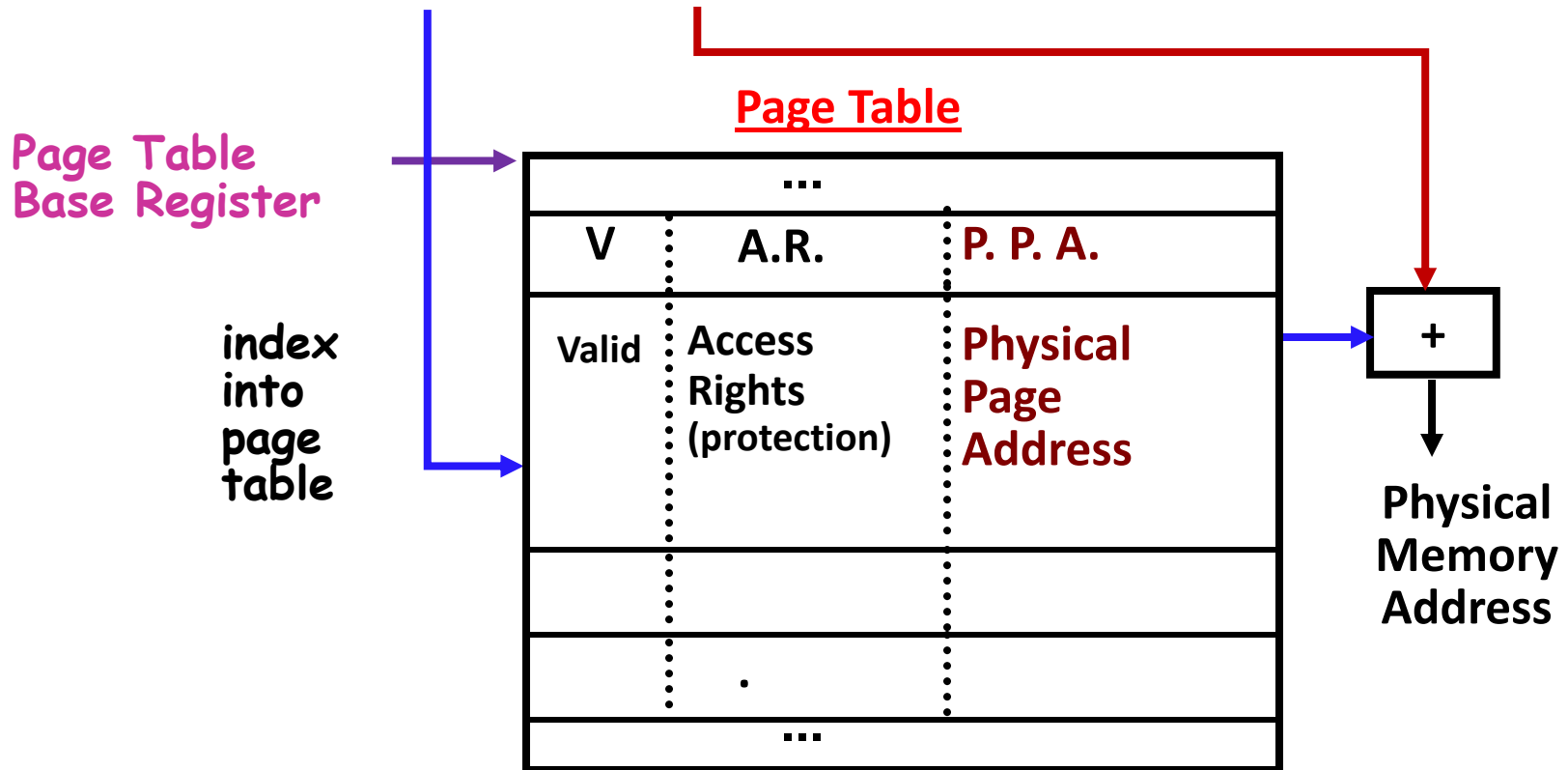
OS changes page tables by changing contents of **Page Table Base Register**

Address Mapping: **Page Table**

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

Virtual Address:

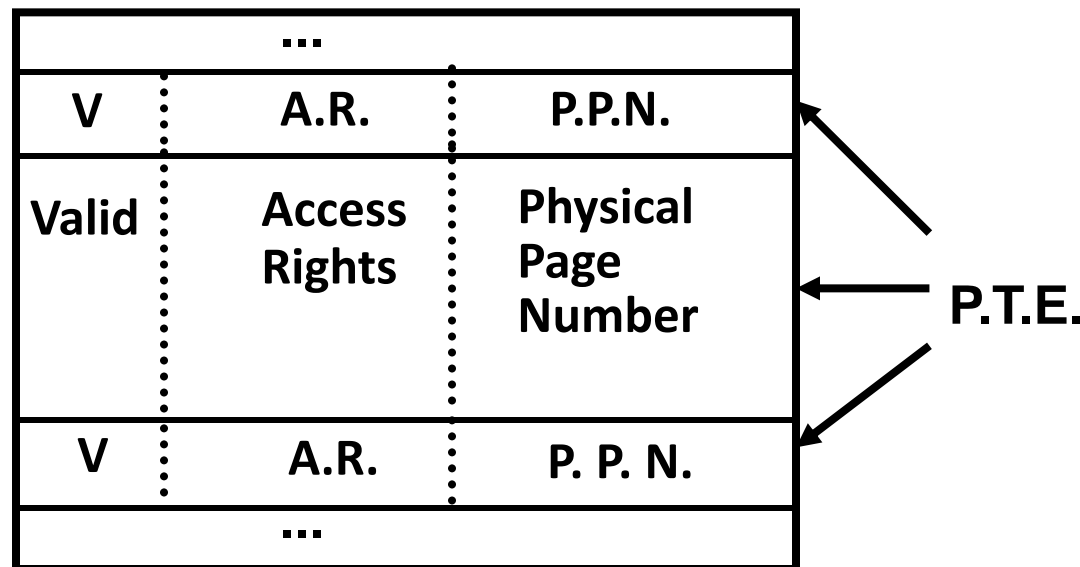
page no. **offset**



Page Table Entry (PTE) Format

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

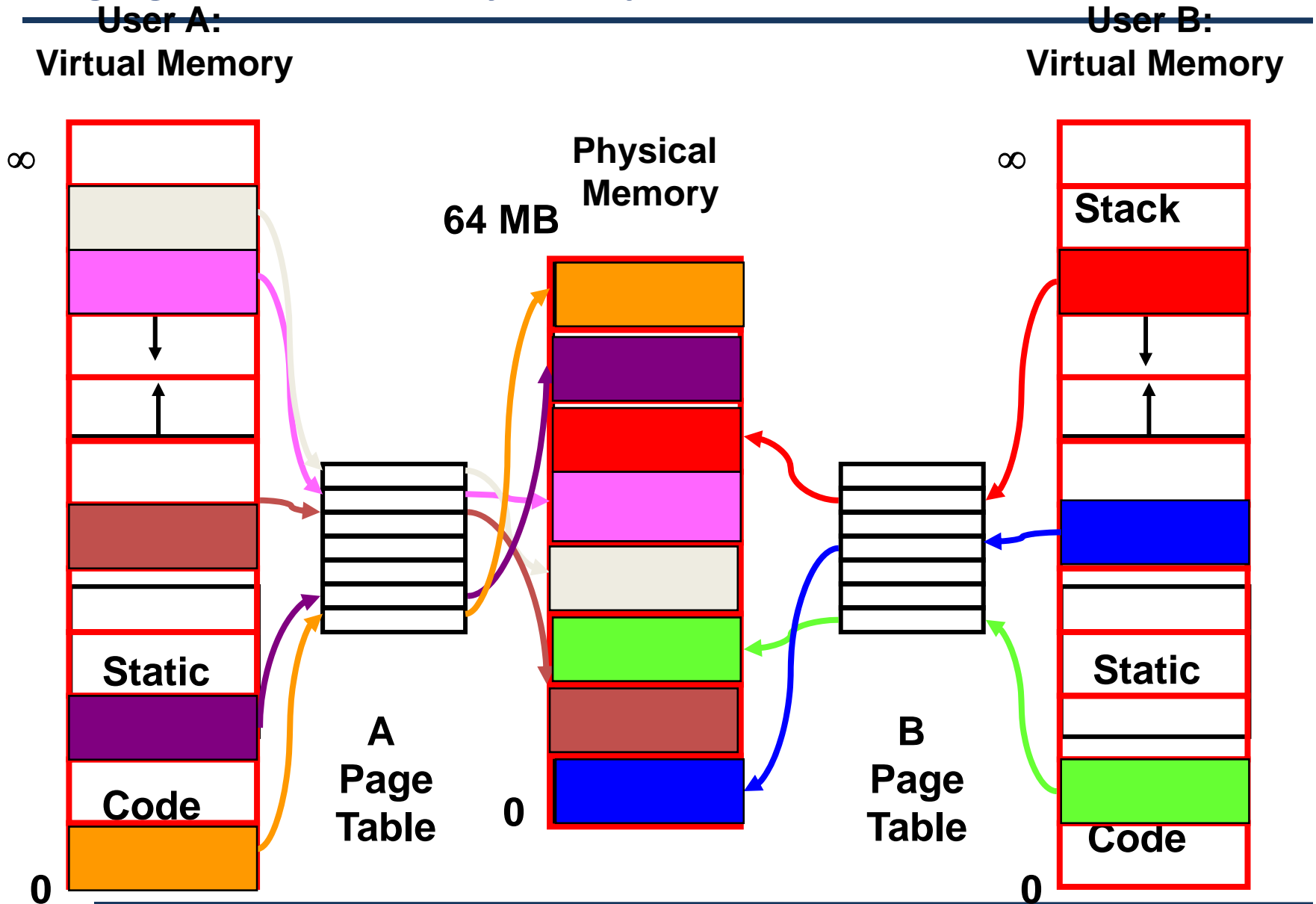
Page Table



- 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



Comparing the 2 Levels of Hierarchy: VM vs. Cache

Cache Version

Block or Line

Miss

Block Size: 32-64B

Placement:

Direct Mapped,
N-way Set Associative

Replacement:

LRU or Random

Write Thru or Back

Virtual Memory version

Page

Page Fault

Page Size: 4K-64KB

Fully Associative

Least Recently Used
(LRU)

Write Back