CSE4201
Computer Architecture

Virtual memory

Slides are based on slides by Prof. Shaaban (RIT) and Prof. Paterson (UCB)

Virtual Memory

- **Virtual memory first used to relive programmers from the burden of managing *overlays*.**
- **Later, it was used to share memory between processes.**
- **Also used to allow the program to be relocated (to run in a different physical space than the one it was compiled for).**

Virtual Memory

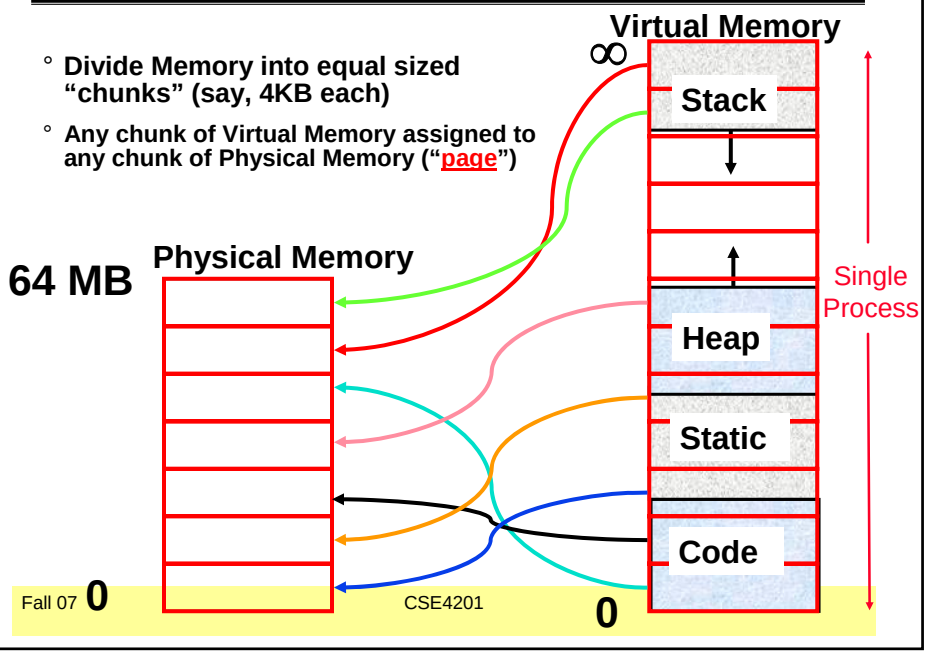
- Each process has its own private “virtual address space” For example, the CPU issues a 32-bit memory address
- Each computer has a “physical address space” (e.g., 512 MegaBytes DRAM); 29 bits and not all of the RAM will be available for a single process.
- Address translation: mapping virtual to physical
  - Allows multiple programs to use (different chunks of physical) memory at same time
  - Also allows some chunks of virtual memory to reside on disk, (to exploit memory hierarchy)
  - Compiler generate addresses independent of the physical memory size.

Virtual Memory

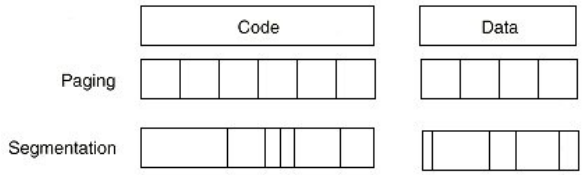
- A program can be run in any location in main memory or disk by using a relocation/mapping mechanism controlled by the operating system which maps the address from virtual address space (logical program address) to physical address space (main memory, disk).
- A program code/data block needed for process execution and not present in main memory result in a page fault (address fault) and the page has to be loaded into main memory by the OS from disk (demand paging).

Mapping Physical Memory to Physical Memory

- Divide Memory into equal sized "chunks" (say, 4KB each)
- Any chunk of Virtual Memory assigned to any chunk of Physical Memory ("page")



Paging vs. Segmentation

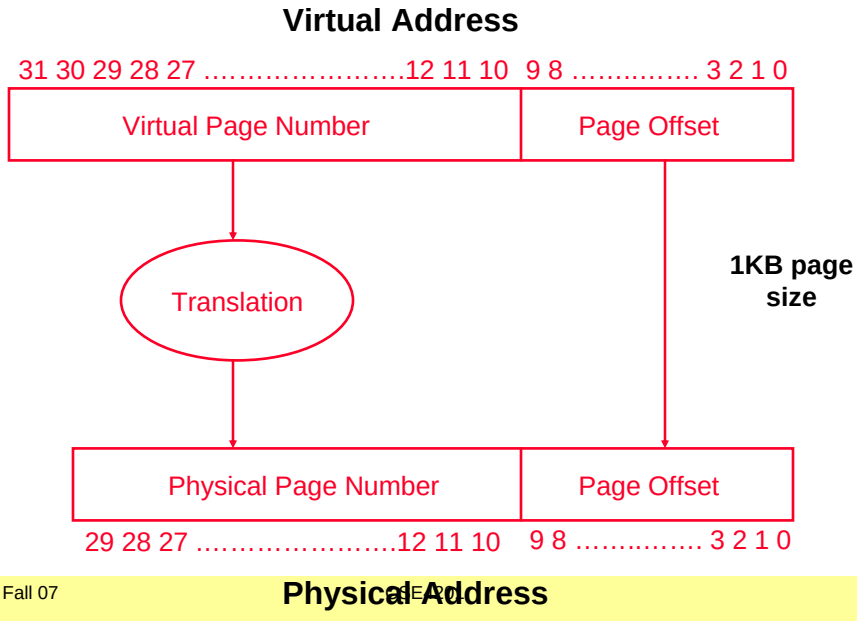


	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial (all blocks are the same size)	Hard (must find contiguous, variable-size, unused portion of main memory)
Memory use inefficiency	Internal fragmentation (unused portion of page)	External fragmentation (unused pieces of main memory)
Efficient disk traffic	Yes (adjust page size to balance access time and transfer time)	Not always (small segments may transfer just a few bytes)

VM vs. Cache

Parameter	First-level cache	Virtual memory
Block (page) size	16–128 bytes	4096–65,536 bytes
Hit time	1–2 clock cycles	40–100 clock cycles
Miss penalty	8–100 clock cycles	700,000–6,000,000 clock cycles
(Access time)	(6–60 clock cycles)	(500,000–4,000,000 clock cycles)
(Transfer time)	(2–40 clock cycles)	(200,000–2,000,000 clock cycles)
Miss rate	0.5–10%	0.00001– 0.001%
Data memory size	0.016–1MB	16–8192 MB

Mapping Physical Memory to Physical Memory



Handling Page Faults

- A page fault is like a cache miss
  - Must find page in lower level of hierarchy
- If valid bit is zero, the Physical Page Number points to a page on disk
- When OS starts new process, it creates space on disk for all the pages of the process, sets all valid bits in page table to zero, and all Physical Page Numbers to point to disk
  - called Demand Paging - pages of the process are loaded from disk only as needed

Cache vs. Virtual Memory

Cache	*Virtual Memory*
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-16KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU) approximation
Write Thru or Back	Write Back
How Managed: Hardware	Hardware + Software (Operating System)

Virtual Memory

° **Where can a block be placed in MM?**

- **Since the miss penalty (page miss) is sever, lower miss rate is the main concern.**
- **A block can be placed anywhere in the main memory.**
- **Similar to *fully associative* cache**

Virtual Memory

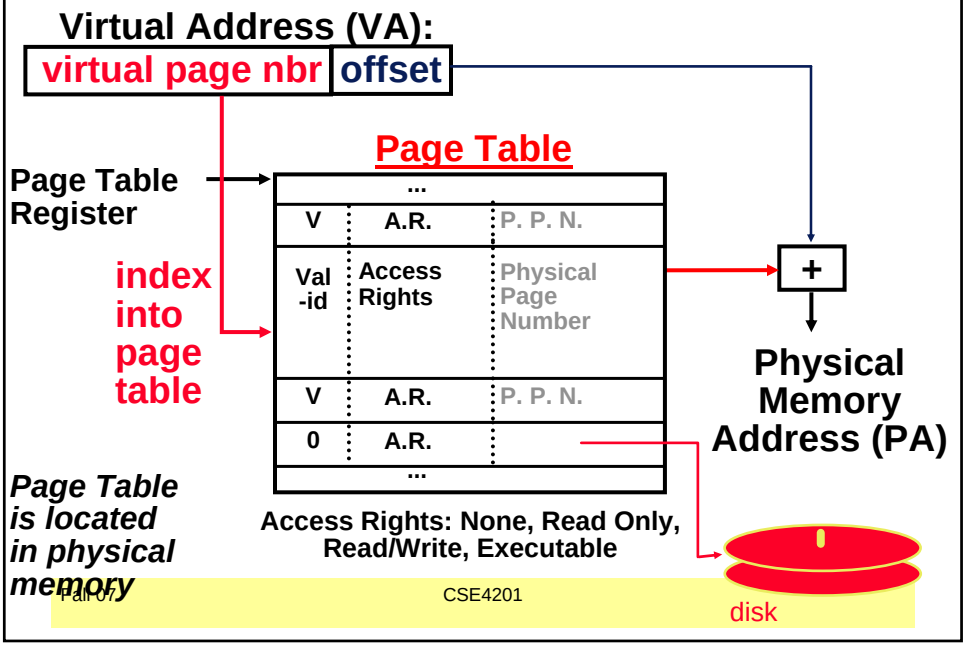
° **How is a block found if it is in the MM?**

- **A data structure is used to translate between virtual and physical address**
- **For paging, the virtual address is composed of a page number and an offset in the page.**
- **The page number is used to index a page table, that gives the beginning physical address of the page.**
- **That address concatenated with the offset produces the physical address.**

Address Translation

- Want fully associative page placement
- How to locate the physical page?
- Search impractical (too many pages)
- A page table is a data structure which contains the mapping of virtual pages to physical pages
  - There are several different ways, all up to the operating system, to keep this data around
- Each process running in the system has its own page table

Page Table



Handling Page Faults

- A page fault is like a cache miss
  - Must find page in lower level of hierarchy
- If valid bit is zero, the Physical Page Number points to a page on disk
- When OS starts new process, it creates space on disk for all the pages of the process, sets all valid bits in page table to zero, and all Physical Page Numbers to point to disk
  - called Demand Paging - pages of the process are loaded from disk only as needed

Optimizing for Space

- Page Table too big!
  - 4GB Virtual Address Space ÷ 4 KB page
    ⇒ 2^{20} (~ 1 million) Page Table Entries
    ⇒ 4 MB just for Page Table of single process!
- Variety of solutions to tradeoff Page Table size for slower performance when miss occurs in TLB

  Use a limit register to restrict page table size and let it grow with more pages,Multilevel page table, Paging page tables, etc.

How to Translate Fast?

- Problem: Virtual Memory requires two memory accesses!
  - one to translate Virtual Address into Physical Address (page table lookup)
  - one to transfer the actual data (cache hit)
  - But Page Table is in physical memory!
- Observation: since there is locality in pages of data, must be locality in virtual addresses of those pages!
- Why not create a cache of virtual to physical address translations to make translation fast? (smaller is faster)
- For historical reasons, such a "page table cache" is called a Translation Lookaside Buffer, or TLB

TLB

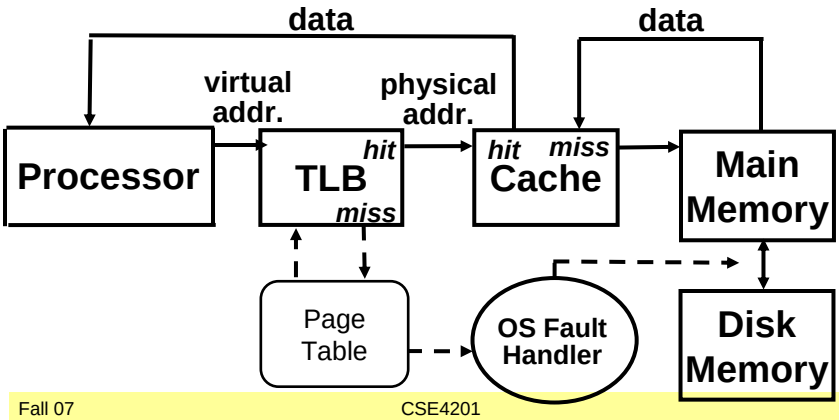
Virtual Page Nbr	Physical Page Nbr	Valid	Ref	Dirty	Access Rights

"tag" "data"

- TLB just a cache of the page table mappings
- Dirty: since use write back, need to know whether or not to write page to disk when replaced
- Ref: Used to calculate LRU on replacement
- TLB access time comparable to cache (much less than main memory access time)

Translation

- TLB is usually small, typically 32-4,096 entries
- Like any other cache, the TLB can be fully associative, set associative, or direct mapped



Replacement

- Which block should be replaced?
- To minimize page fault, LRU is used
- A *use* bit or *reference* bit is used (in the page table), logically set when the page is referenced.
- Periodically, the OS clears the reference bit and later record them to know least-recently used page.

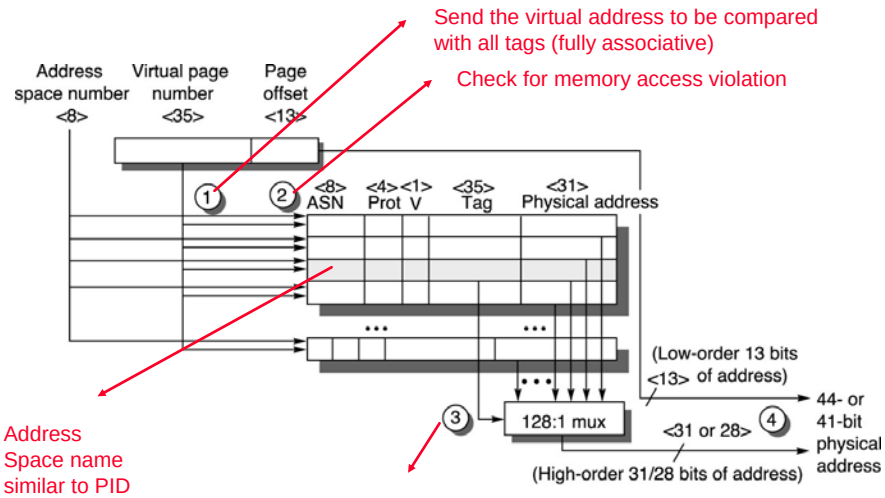
Writes?

- What happens on writes?
- It would be crazy to do *write-through*
- A dirty bit is used to minimize writing on replacement.

Segmented Memory System

- Programs that are block structured may include procedures or functions which call each other.
- The set of logically related contiguous data elements are referred to as a *segment*.
- Segments are allowed to grow and shrink almost arbitrarily, unlike pages which have fixed size.
- Addresses are defined as *(<s>,<i>)*, s is translated into a segment address by the OS.
- Segmentation is a possibility since the segment size varies.

Alpha 21264 Data TLB



Page Size

- **Bigger pages**
  - **Small size of page table (small number of pages)**
  - **Transferring from and to desk is more efficient for large page sizes.**
  - **Large pages means reducing TLD misses**
- **Smaller pages**
  - **Less wasted space per page (internal fragmentation), as opposed to external fragmentation for segmented virtual memory**