

Virtual Memory (VM)

CIT 595
Spring 2007

Motivation for Virtual Memory

- A **process** is an instance of a computer program that is being executed
- Monoprogramming
 - One process in memory at a time runs till completion
 - With OS as supervisor
- What if process address space is larger than Physical Main Memory (DRAM)?
 - Process Address space is 0 to 2^{n-1} where n = machine size
 - Main memory is temporary storage not made as big as process address space
 - Instead your program is usually stored on some form of permanent storage (disk or tape)

Why not just make main memory large enough?

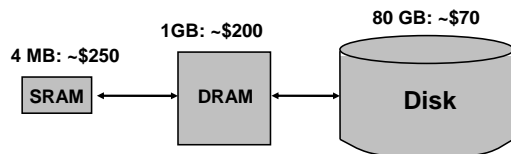
- Due to Memory Technology – cost, speed and capacity factors

CIT 595

12 - 2

Memory Comparison

- Full address space is quite large:
 - e.g. 32-bit address (with 1 byte storage): ~4 GB
- Disk storage is ~300X cheaper than DRAM
 - 80 GB of DRAM: ~ \$21,000 vs. 80 GB of disk: ~ \$70
- To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk



CIT 595

12 - 3

Original Motivation for VM

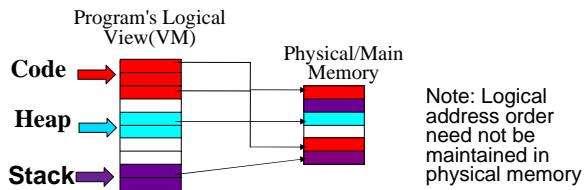
- IBM wanted one software suite for a family of System 370 computers
 - Family line of computers is a suite of compatible designs for same command set or ISA
 - Allowed customers to purchase a smaller system with the knowledge they could upgrade to larger system
- Allowing same program run on machines with different memory sizes
 - Earlier programmers had to do explicit memory management
- Idea was to create an illusion for a process that it has memory as big as its address space
 - Hence the **concept of Virtual Memory** i.e. memory appears to be but isn't

CIT 595

12 - 4

Big Picture: How VM works?

- Virtual Address (VA)
 - Is the address generated by your program
 - Address range 0 to $2^n - 1$ where n is machine width
- Physical Address (PA)
 - Where the virtual address is physically stored in Main Memory (DRAM)
 - Address ranges from 0 to $2^m - 1$ where $m < n$

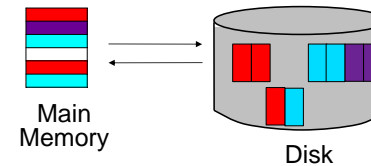


CIT 595

12 - 5

Big Picture: How VM works?

- Use the disk as an extension to main memory
- Need some address mapping scheme which will tell us where the Virtual Address is actually located
 - Mapping need not preserve continuity of data in both physical memory and disk
 - Logical address is only from program's point of view
- Address Translation from is done by OS + hardware



CIT 595

12 - 6

Uses of VM: Multiprogramming

Multiple processes resident in main memory

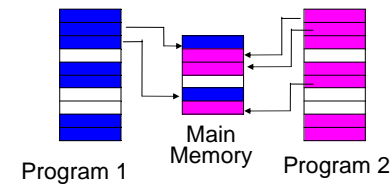
- Can have many different active processes at the same time
 - E.g. Word processor and paint application are being simultaneously used by the user
- Same program is run in different processes i.e. multiple instances of the same program
 - the program is the same i.e. the instructions *but* the data is different
 - the program counters are different
- Only *active* part of the code and data of process is in main memory
 - Allocate more memory to process as needed

CIT 595

12 - 7

Uses of VM: Multiprogramming (contd..)

- This gives an appearance to the user that different processes are being executed at the same time
- But in reality one process can be executing at a time per CPU
 - This known as time-sharing the processor
 - O.S decides which processes gets the CPU based on a scheduling algorithm (more on this in chp 8)



CIT 595

12 - 8

Uses for VM: Program Isolation/Protection

- VM creates an illusion i.e. each process thinks
 - It has 2^n address space
 - It has its own stack starting at address (say 0x3FFF)
- With transparent memory management we can map virtual addresses of a process to a physical address that is separate from another process

Hence, one process cannot interfere with another because they operate in different address spaces

Data Size Transfer between Disk and Main Memory

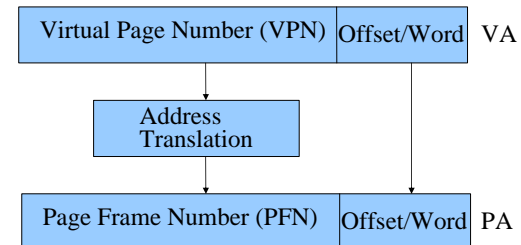
- According to Principle of Locality
 - Large chunks or pieces of data is transferred between Disk and Main Memory
- This large chunk or unit of transfer is known as a *page*
 - *Analogous* to block transfer between cache and main memory
 - However the page is much larger than block
 - E.g. Transfer 8-32 bytes vs. 2048-8192 bytes for most modern systems
- If you ask for a word of main memory and find that it is not present, then whole page's worth of data from the disk is loaded into physical main memory

Virtual Memory Technique: Paging

- Allocate physical memory to processes in fixed size chunks called *page frame*
- The virtual address space is divided into pages of equal size
 - The page size is also same as the frame size
 - Each page has same number of words (like cache)
- The entire address space required by a process need not be in memory at once
 - Some parts can be on disk, while others are in main memory
- Pages allocated to a process do not need to be stored contiguously- either on disk or in main memory
 - In both cases O.S. keeps a data structures to track actual locations

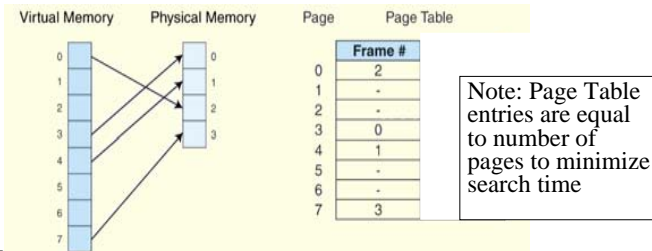
Paging: Address Translation Overview

- The Virtual (Logical) Address (VA) is translated into Physical Address (PA) as follows:



Address Translation using Page Table

- Information concerning the location of each page, is maintained in a data structure called a *page table*
 - Operating System (OS) maintains this data structure
 - The page table is placed in memory at a known location
 - The Virtual Page number is used as an offset into the table to find which frame in Physical Memory is the data located

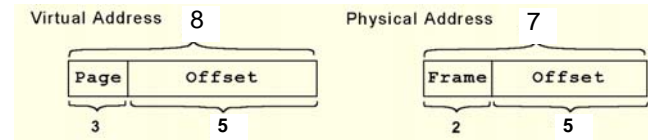


CIT 5

12 - 13

Example

- A system has a virtual address space of 2^8 and a physical address space of 2^7 . Further assume that each page has 32 words.
 - A virtual address has 8 bits
 - Of the 8 bits, 5 bits are used for offset ($2^5 = 32$ words)
 - Remaining 3 bits are used for Virtual Page Number (VPN)
 - Since physical memory address is 7 bits, 5 bits go to offset (offset part is never modified) and hence 2 bits are used for Page Frame Number (PFN)

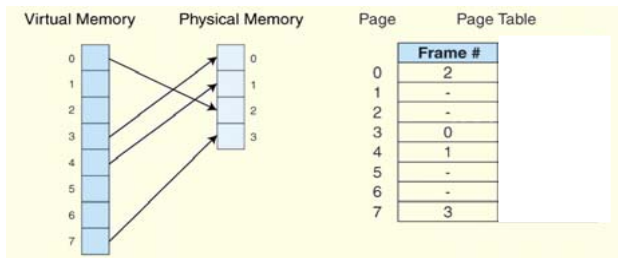
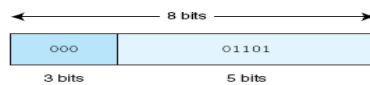


CIT 595

12 - 14

Example (contd..)

Virtual Address 13 is produced by the processor for a process, what is the Physical Address?



CIT 595

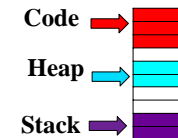
Physical Address = 1001101 (77_{10})

12 - 15

Keeping track of VA's in Multiprogramming

- Each program's logical view is the same

Program's Logical View



- Problem: Different processes use the same virtual addresses
- Solution: Each process has its *own* page table
- OS responsible for *updating* page tables so that virtual address spaces of different processes do not collide

CIT 595

12 - 16

Multiprogramming Environment

- The processes existent in the physical memory time share the CPU
- When the OS switches between process, the *state* of the existing process is saved before the new process is allowed to use the CPU
 - In addition to the program counter and registers, the page table is also saved
- Instead of storing entire page table, only the address where the first page table entry is located gets stored
 - When a new process gets the CPU, the OS loads the state of the process
 - Like PC register is used to load the PC, the page table address is loaded into an internal register called *Page Table Pointer*

CIT 595

12 - 17

Overview of Paging Technique

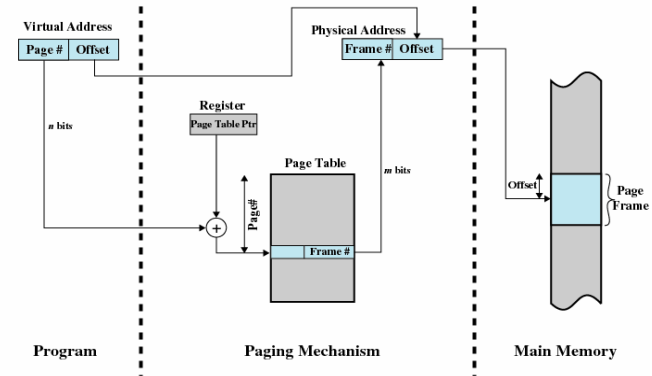


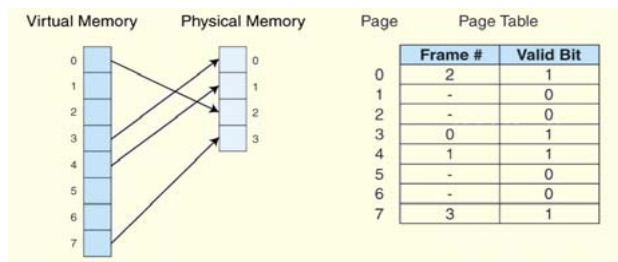
Figure 8.3 Address Translation in a Paging System

CIT 595

12 - 18

Valid Bit

- A bit is needed to indicate whether the page is in main memory or not
 - Use a valid bit like we used in cache
 - Valid bit = 1, means page is in physical memory



CIT 595

12 - 19

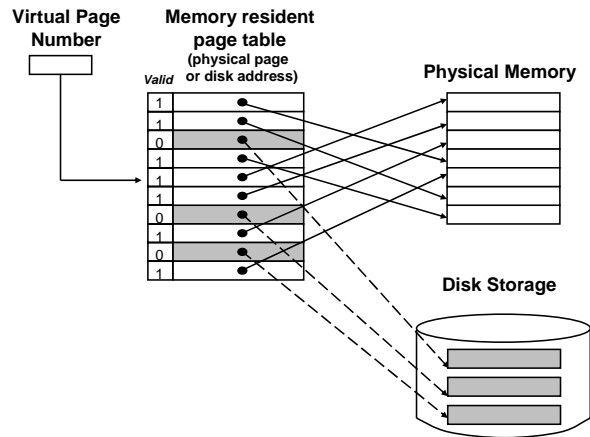
Valid Bit (contd..)

- If valid bit is 0, then page is not in physical memory
 - This is known as an occurrence of *page fault*
- When page fault occurs an exception is thrown and the OS intervenes
 - Job of the exception routine is to find the page required from disk and put it in main memory
 - OS also maintains a data structure to record where each virtual page is stored on disk
 - This data structure can be part of the page table or be a separate structure all together

CIT 595

12 - 20

Page Table with Physical Page/Disk Address



CIT 595

12 - 21

Bringing in a page when Physical Memory is full

- If all pages in main/physical memory are occupied then O.S must chose a page to replace
- O.S keeps track of which processes and which virtual addresses map to a physical page
- According principle of locality, O.S should chose the page that will not likely be used in the future
 - If page replaced is going to accessed again then we will incur a page fault penalty
- Replacement schemes: LRU, FIFO etc.

CIT 595

12 - 22

Write Policy

- Data modified data in physical memory must also be updated in the disk
- Disk access time is very slow
 - 5ms – 20ms compared to DRAM access time of 30-90ns
- **Write-Back** policy is employed i.e. update disk only when the page is going to be replaced
- **Dirty** (Modify) bit is needed to indicate if the page has been altered since it was last loaded into main memory
 - If dirty = 1, then write-back page to disk upon replacement

D	V	Page Frame # or Disk Addr
1	1	
0	1	
0	0	
0	1	
1	1	
1	1	

CIT 595

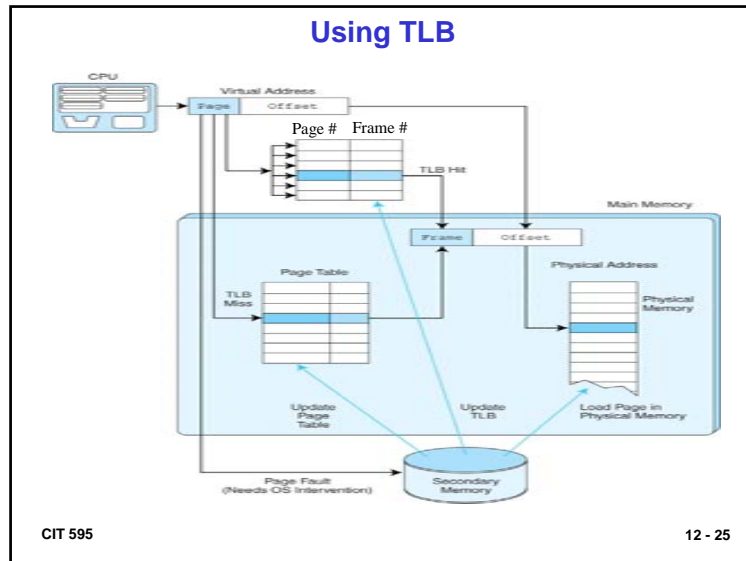
12 - 23

Speeding up data access further

- Each virtual memory reference can cause two physical memory accesses
 - One to fetch the page table
 - One to fetch the data
- To overcome this problem a high-speed memory called **Translation Lookaside Buffer** (TLB) is used
 - TLB is made of SRAM technology since speed is what we need
- The TLB stores the **most frequently** used mappings i.e. stores page number and corresponding frame number
 - SRAM is costly in terms of storage, hence use it store the data that you most need

CIT 595

12 - 24

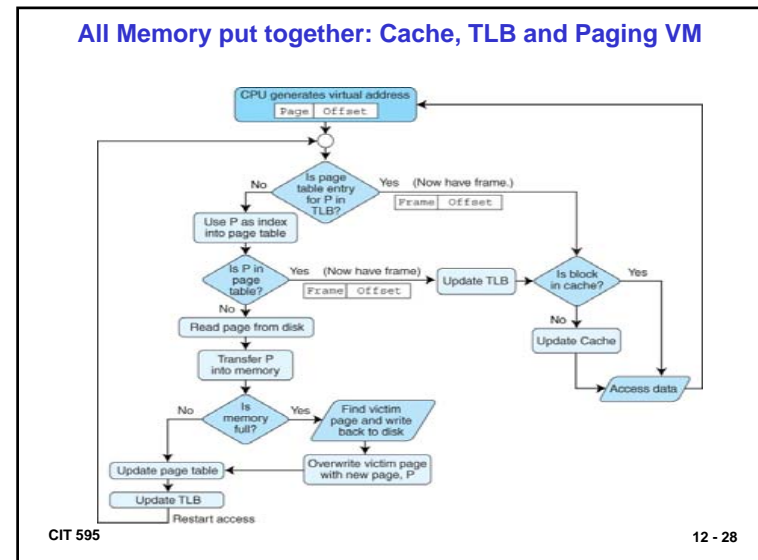
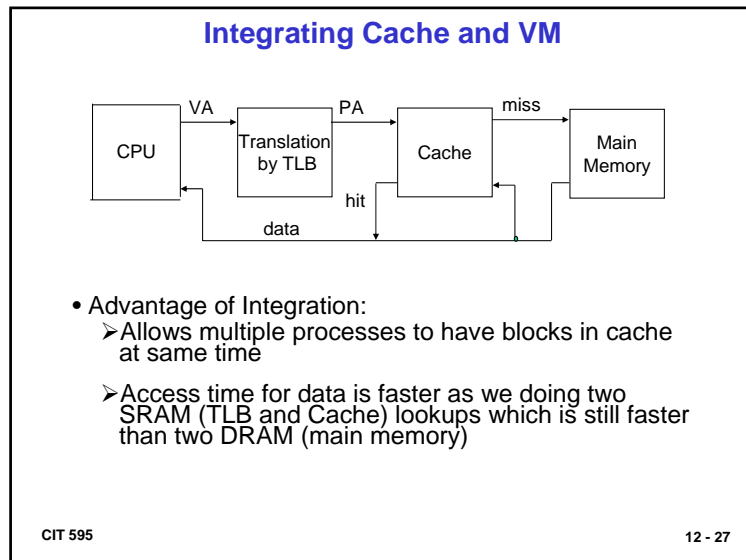


Modern Processor with Cache and Virtual Memory

- Studied Cache and Physical memory interaction
- Found Caches indexed using physical/main memory address
- But processor (CPU) generates a logical/virtual address
- So then how does the whole memory system work?
- Require Address Translation before Cache Lookup
 - Problem: Involve a memory access itself (page table lookup)
 - Solution: We know that page table entries can also be cached. Hence use TLB to translate VA to PA and then use cache.

CIT 595

12 - 26



Effective Access Time (EAT)

- The performance of hierarchical memory is measured by its *effective access time* (EAT)
- EAT is a weighted average that takes into account the hit ratio and relative access times of successive levels of memory
- The EAT for a two-level memory is given by:
$$\text{EAT} = H \times \text{Access Time for Level } i + (1-H) \times \text{Access for Level } i+1$$
 - H is the hit rate i.e. % time data is found in level *i*
- This equation can be extended to any number of memory levels

CIT 595

12 - 29

EAT Example I

Consider a system with a main memory access time of 200ns supported by a cache having a 10ns access time and a hit rate of 99%.

$$\text{EAT} = 0.99(10\text{ns}) + 0.01(200\text{ns}) = 9.9\text{ns} + 2\text{ns} = 11\text{ns}$$

CIT 595

12 - 30

EAT Example II

- Suppose a main memory access takes 200ns, the page fault rate is 1%, and it takes 10ms to load a page from disk. We have:

$$\text{EAT} = 0.99(200\text{ns} + 200\text{ns}) + 0.01(10\text{ms}) = 100,396\text{ns}$$

- Note: There are two access to the main memory, one for page table lookup and the other to get the actual data
- Having a TLB will alleviate EAT between main memory and disk to some extent

CIT 595

12 - 31

Disadvantage of Paging Technique

- A process may not need the entire range of addresses contained within the page
- Thus, there may be many pages containing unused fragments of memory
- This known as *internal fragmentation*
- The unused fragments could have allow more processes to be existent in physical memory
- But due to fixed size page constriction, the unused fragments of the page cannot be freed up unless the page is replaced

CIT 595

12 - 32

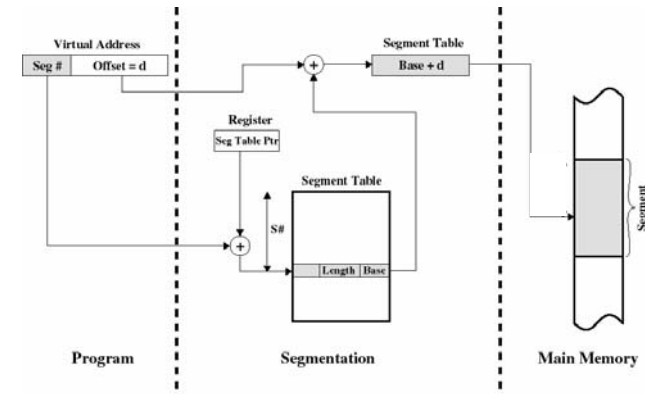
VM Technique: Segmentation

- The Virtual/Logical Address space into variable-length units called *segments*
- The Physical Memory is in not partitioned
- When segment needs to be copied into physical memory, O.S looks for free chunk of memory large enough to fit the segment
- Each segment contains <base, bound> pair
 - *Base* address indicating where in memory it is located
 - *Bound* (or length) indicates the size of the segment
 - This information is stored in *segment table*
- Logical to physical address translation is similar to paging except that the offset is added to the starting address (instead of being appended)

CIT 595

12 - 33

VM with Segmentation



CIT 595

12 - 34

Disadvantage of Segmentation

- Suffers from *external fragmentation*
- Segments are allocated and de-allocated, the free chunks that are in memory become broken into small chunks
- Eventually there are can many small chunks but none larger to store an entire segment
- Even though space may exists to allocate a new page but it not contiguous enough to fit the entire segment
- If segmentation is used, system has to use some sort of garbage collection
 - Where it tries to collect small free chunks into large one by moving around existing occupied chunks
 - Similar to disk fragmentation

CIT 595

12 - 35

Sharing Pages

- Why share pages?
 - If we share the same code or data among different processes, it is sufficient to keep only one copy in main memory
 - E.g. Common subroutines are shared among processes
- Sharing is implemented by making shared pages read-only (protection setting)
- Sharing is easier to implement in segmentation technique
 - As segments semantically can represent defined portions to be shared
 - E.g. Code, Heap, Stack segments
 - The protection on the segmented regions then can be set according
- In paging, semantic portions can be across different pages
 - Harder to set protection as you might want part of the data sharable and part to be exclusive

CIT 595

12 - 36

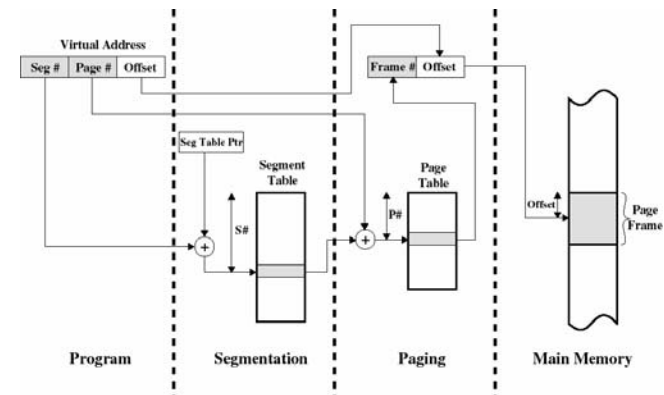
Paging Combined with Segmentation

- Modern systems employ combined paging and segmentation
- Take advantage of the best features of both by assigning fixed-size pages within variable-sized segments
- Each segment has a page table. This means that a memory address will have three fields, one for the segment, another for the page, and a third for the offset.

CIT 595

12 - 37

Paging Combined with Segmentation



CIT 595

12 - 38

Advantages of VM

- Enhances performance by providing greater memory capacity, without the expense of adding main memory
- The programmer does not have to worry about address space i.e. illusion of *very large memory*
 - The movement of data is transparent i.e. data between the two levels are exchanged by O.S. without the knowledge of the user
- To a process it appears as if it "owns" machine
 - Has private address space
 - Unaffected by behaviour of other processes

CIT 595

12 - 39