澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

School of Applied Sciences (B.Sc. in Computing)

Notes #7: Virtual Memory

COMP 213
(211/212)
Operating Systems

2019-2020 1st Semester

Notice Two Characteristics (1)

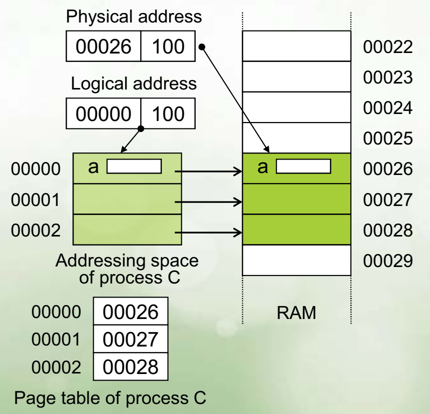
- Memory references are translated by the CPU dynamically
  - logical/physical address translation is done at each memory access (for data and program)
  - OS can put the pages of a process anywhere in RAM, and it can move the pages without interfering the process

Notice Two Characteristics (2)

- Each page is mapped to a frame independently
  - A process may be broken up into pieces that do not need to be located contiguously in RAM
  - the OS may partially swap a process. E.g. page 0 and page 2 are in RAM, while page 1 is swapped out to hard disk

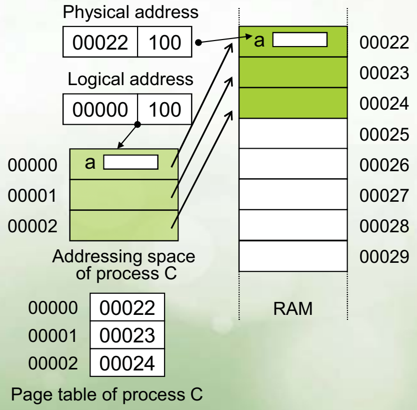
Address Translated Dynamically (1)

Suppose “a=0” is compiled to the assembly “mov [00000100], 0”. At the moment that the CPU executes this instruction, the logical address 0x00000100 will be translated to the physical address 0x00026100 using the page table.



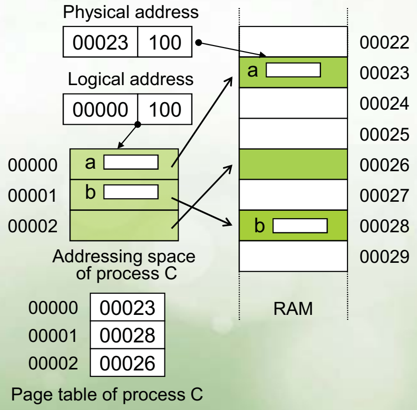
Address Translated Dynamically (2)

After swapping, the process image moves to a different location in RAM. When the CPU executes the instruction "mov [00000100], 0" (a=0) again, logical address 00000100 will be translated to new location automatically (physical address 00022100).



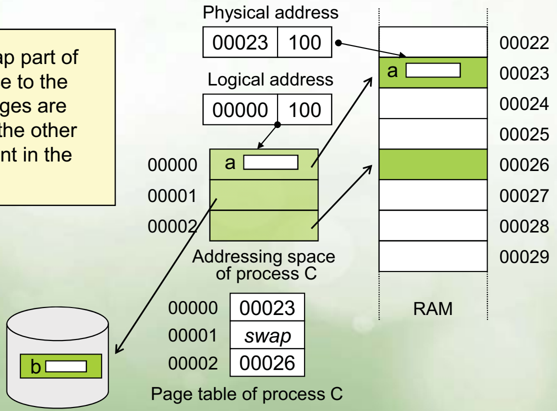
Each Page is Mapped Independently (1)

The three pages 00000, 00001 and 00002 (the first 12k in the addressing space) appear to be contiguous to process C. But, in fact, they are mapped to separate frames in RAM.



Each Page is Mapped Independently (2)

And the OS can swap part of the addressing space to the hard disk. Some pages are swapped out, while the other pages are still present in the RAM.



What We'll Learn

- Virtual memory
  - what, advantages, how, page fault handling, efficiency
- Hardware support: page table, page table point, TLB
- OS support
  - esp. Page replacement algorithm
- Requirements of memory management revisited

Virtual Memory

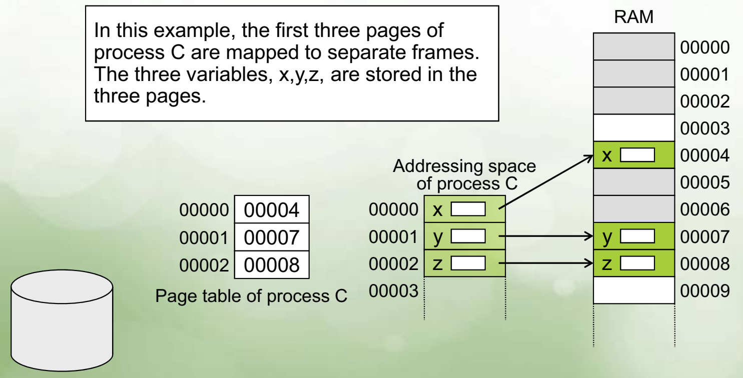
- "Simulate more RAM using hard disk space"
- The program/data used by a process do not have to be present in the RAM all the time. Pages not currently used are swapped out to disk.
- When needed, these pages are swapped in, back to some frames in RAM. This operation is invisible to the process

Advantages of Virtual Memory

- More processes can be kept in main memory
  - only some pieces of each process are loaded
  - better utilization of the CPU
- It is possible for a process to be larger than all the main memory
  - easier life for programmer

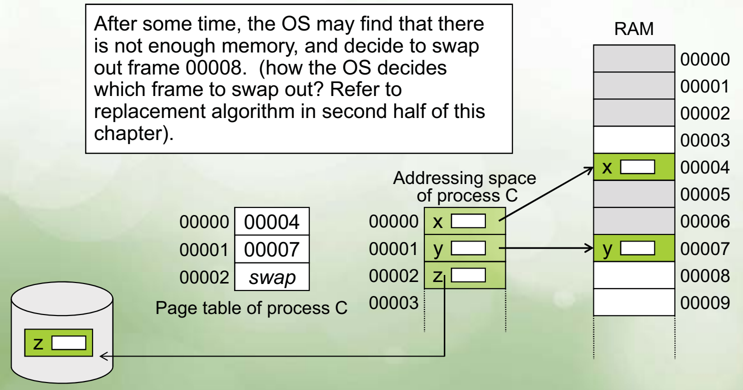
How Virtual Memory Works (1)

In this example, the first three pages of process C are mapped to separate frames. The three variables, x,y,z, are stored in the three pages.



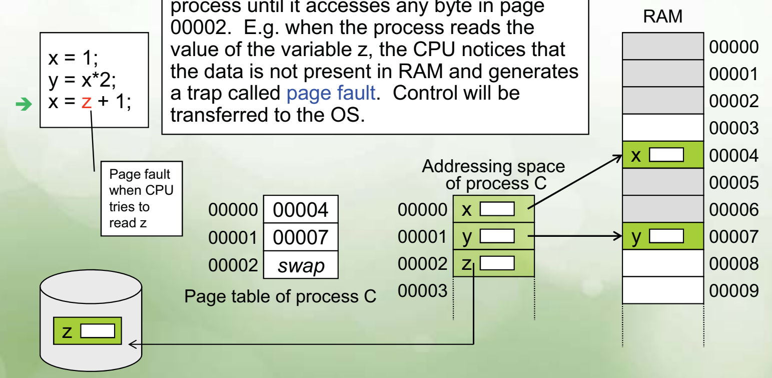
How Virtual Memory Works (2)

After some time, the OS may find that there is not enough memory, and decide to swap out frame 00008. (how the OS decides which frame to swap out? Refer to replacement algorithm in second half of this chapter).



How Virtual Memory Works (3)

This does not interfere the execution of the process until it accesses any byte in page 00002. E.g. when the process reads the value of the variable z, the CPU notices that the data is not present in RAM and generates a trap called page fault. Control will be transferred to the OS.



How Virtual Memory Works (4)

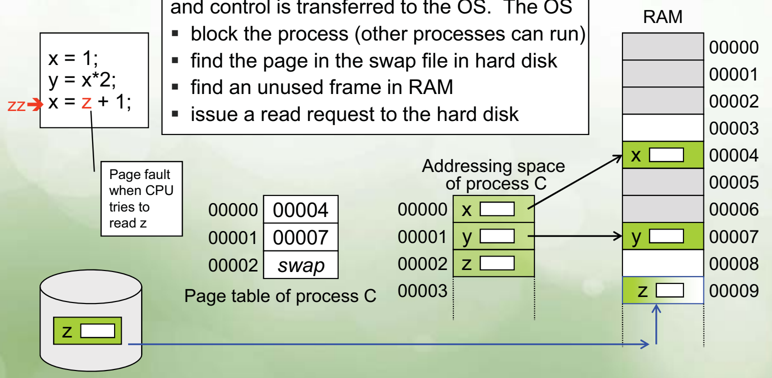
This does not interfere the execution of the process until it accesses any byte in page 00002. E.g. when the process reads the value of the variable z, the CPU notices that the data is not present in RAM and generates a trap called page fault. Control will be transferred to the OS.

How? We will learn later in this chapter that the CPU detects page fault by checking the P bit in the page table entry of the page in address translation.

How Virtual Memory Works (5)

… The CPU generates a page fault trap, and control is transferred to the OS. The OS

- block the process (other processes can run)
- find the page in the swap file in hard disk
- find an unused frame in RAM
- issue a read request to the hard disk

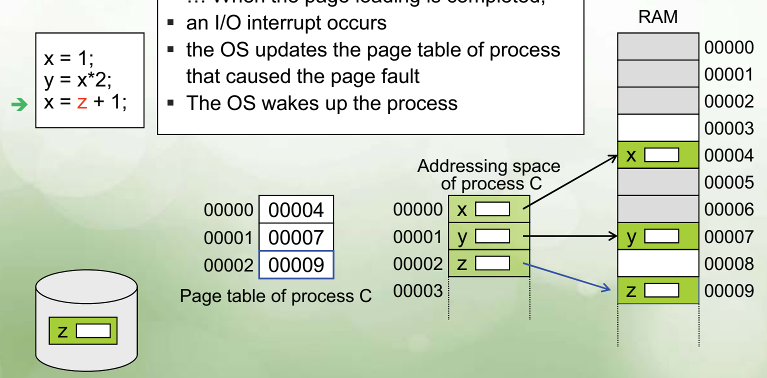


How Virtual Memory Works (6)

No suspended state?
- When swap space used up

… When the page loading is completed,

- an I/O interrupt occurs
- the OS updates the page table of process that caused the page fault
- The OS wakes up the process



Page Fault Handling, the Steps (1)

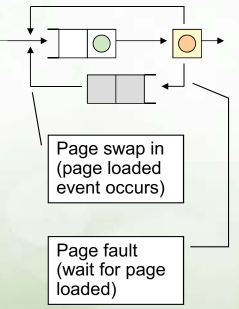
- A page fault occurs when a reference memory address is not present in RAM
- The CPU issues a trap, and the OS handles it:
  - block the process (while other processes can run)
  - find the page in the swap file in hard disk
  - find an unused frame in RAM
  - issue a read request to the hard disk

Page Fault Handling, the Steps (2)

- When page loading is completed (disk I/O finishes)
  - an interrupt occurs
  - OS updates the page table of the blocked process that causes the page fault
  - OS wakes up the process

Process State Changes in Page Fault Handling

- Running ⇨ blocked ⇨ ready
- The process does not know that a page fault occurs (similar to an ordinary process switching)
- While the page is being read into memory, other processes can use the CPU


Page swap in (page loaded event occurs)
Page fault (wait for page loaded)

Efficiency of Virtual Memory

- Hard disk I/O is much slower than RAM
- If the OS swaps out a page just before using it, it has to get that page almost immediately. This is known as *thrashing*
- Thrashing is a state in which the system spends most of its time swapping pieces rather than executing instructions

Efficiency of Virtual Memory (cont’d)

- To avoid thrashing, the OS tries to guess (based on recent history) which pieces are least likely to be used in the near future.
- In other words, can we just keep a few pages in RAM for a process without thrashing?

Principle of Locality

- Program and data references within a process tend to cluster
  - Over a long period of time, the clusters in use change
  - Over a short period of time, the processor is primarily working with fixed clusters of memory references

Locality in a Memory-Reference Pattern

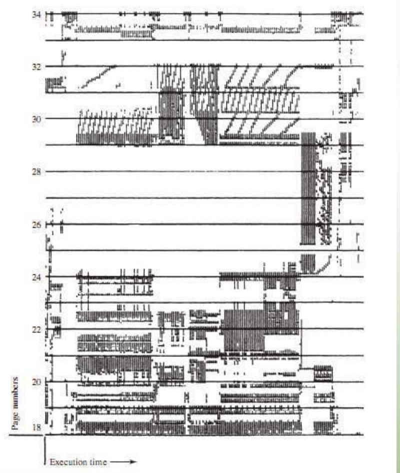


Figure 8.1 Paging Behavior

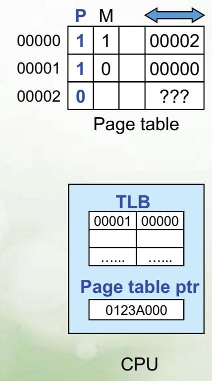
- Note that during the lifetime of the process, references are confined to a subset of pages

Principle of Locality (cont’d)

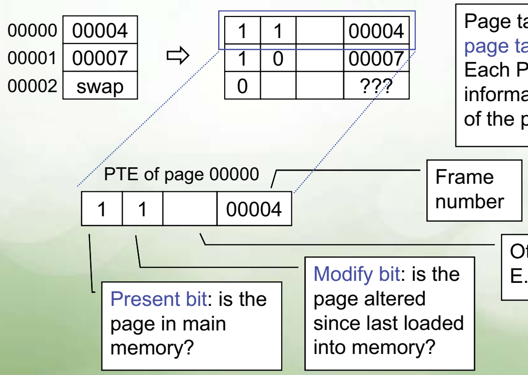
- **Temporal locality** – tendency to access data that are accessed recently
  - Pieces used frequently will likely be used in the near future
  - The OS can make intelligent guess of which page to swap out (page replacement algorithm)
- **Spatial locality** – tendency to access data near data that are recently accessed
  - Only a few pages of a process are needed within a short period of time

Hardware Support to Virtual Memory

- Detail structure of the page table
  - How the CPU detects page fault
- How the CPU accesses the page table of the running process, the ***page table pointer register***
- How the TLB caches most recently used entries of the running process



Page Table Entry, PTE



Page table is an array of page table entries (PTE). Each PTE stores information about a page of the process.

Page Table

- Each process has its own page table
  - What about threads within this process?
- Each page table entry contains
  - Frame number – where this page maps to
  - **P**resent bit – the page is in RAM?
  - **M**odify bit – any byte modified in the page since last loaded?
  - Some other control bits...

Present Bit

- "Is the page in RAM?"
- In address translation, the CPU first checks the Present bit of the page
  - If P=1, the page is present, and address translation continues. The process continues without blocking.
  - If P=0, a page fault occurs, and the OS is invoked to handle this

For detail, refer to "page fault handling" in the beginning of this chapter.

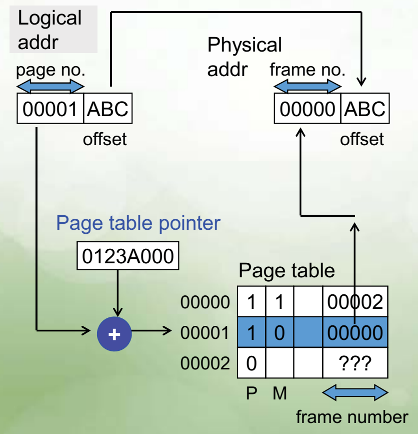
Modify Bit

- "Any byte modified in the page since last loaded?"
  - Example usage:
  - If no change has been made since last loaded into memory, the page content is the same as the copy in the swap file (hard disk)
  - Therefore, when it is swapped out, the OS does not need to write the page to the swap file

Other usage of the M bit is discussed in "page replacement algorithm" later in this chapter.

Page Table Pointer

For virtual memory management
Virtual address maps to logical address
Logical address = Page number + Offset

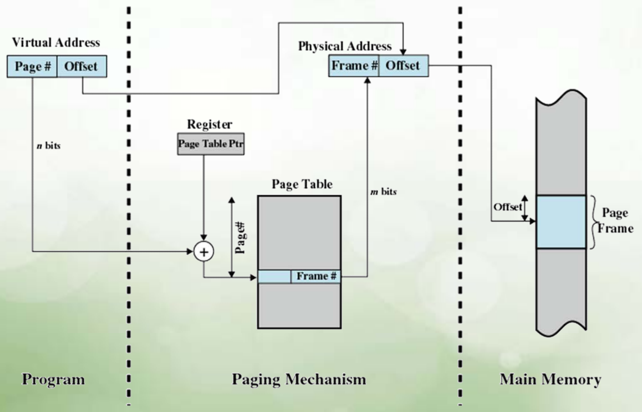


The page table pointer is a register that holds the starting address of the page table of the running process.

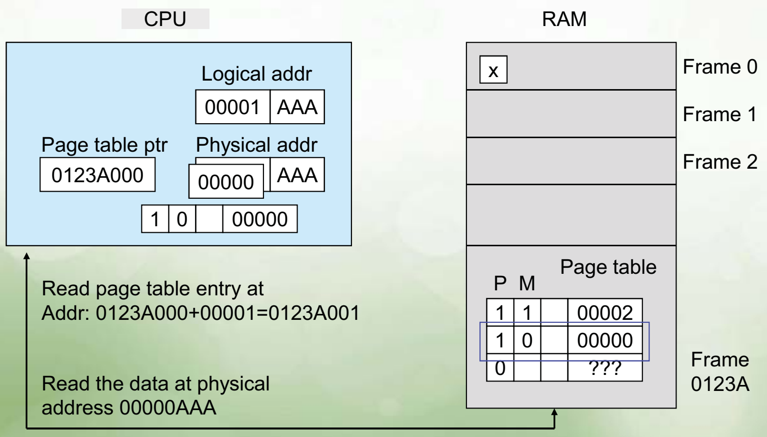
In address translation, CPU computes the address of the relevant PTE by adding the page number to the page table pointer.

What happens to page table pointer in process switching? In thread switching?

Address Translation in a Paging System



How CPU reads PTE



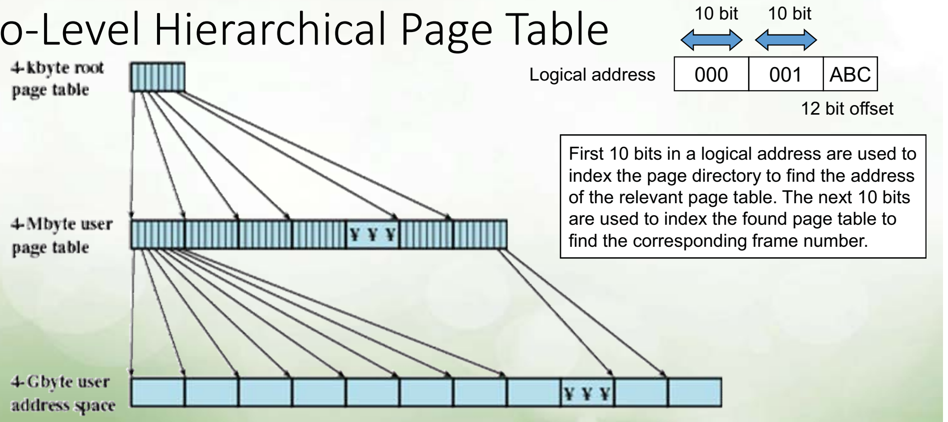
Read page table entry at
Addr: 0123A000+00001=0123A001

Read the data at physical address 00000AAA

Page Table can be Big...

- The entire page table may take up too much memory, e.g.,
  - A process in Pentium can have at most 2^{20} pages
  - Each page table entry takes 32-bit (4-byte)
  - So a page table can occupy up to 4MB!
- Solutions:
  - Page table stored in virtual memory
  - Two-level page table

Two-Level Hierarchical Page Table



First 10 bits in a logical address are used to index the page directory to find the address of the relevant page table. The next 10 bits are used to index the found page table to find the corresponding frame number.

Virtual Memory Access can be Very Slow...

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table entry
  - One to fetch the data

Translation Lookaside Buffer

- Translation Lookaside Buffer, TLB is an associative cache that contains page table entries most recently used
- TLB is fast (cache inside the CPU)
- TLB is small (16*4 entries in Pentium)
- Why such a small cache is enough?

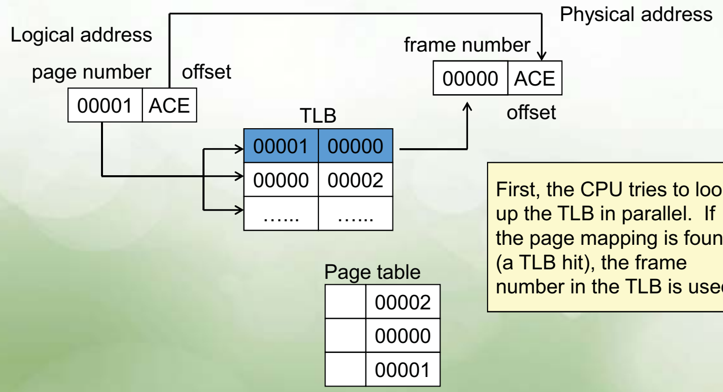
TLB

00001	00000	…
00000	00002	…
……	……	…

E.g. page 00001 maps to frame 00000 (other info in PTE omitted…)

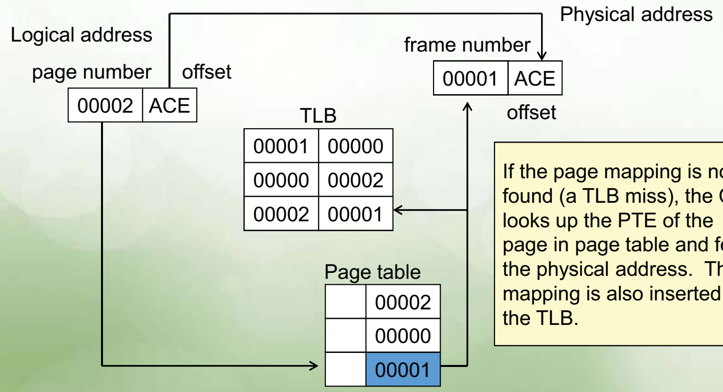
What happens to TLB in process switching? In thread switching?

How TLB Works, TLB Hit



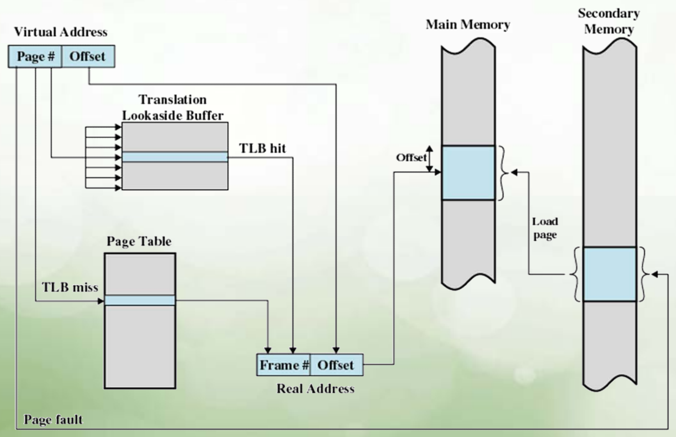
First, the CPU tries to look up the TLB in parallel. If the page mapping is found (a TLB hit), the frame number in the TLB is used.

How TLB Works, TLB Miss



If the page mapping is not found (a TLB miss), the CPU looks up the PTE of the page in page table and form the physical address. This mapping is also inserted into the TLB.

Use of a Translation Lookaside Buffer



How TLB works (1)

- Given a logical address, processor first checks the TLB
- If the page table entry is found there (a hit)
  - the frame number is retrieved and the physical address is formed
  - no memory reference to the page table

How TLB works (2)

- If the page table entry is not found (a miss)
  - CPU uses the page number to index the page table of the process to obtain the corresponding frame number
    - Page fault might occur and is handled by the OS
  - The TLB is updated to include the new page entry
  - the frame number is used to complete the physical address

At Each Memory Access (1)

At each virtual memory access, the MMU (memory management unit) performs the following:
- split the logical address into ***page number*** and ***offset***
- try to look up the TLB for the page number

1 - TLB hit: use the frame number cached to complete the physical address
- TLB miss: ...

- add the page number to the page table pointer register to obtain the address of the relevant page table entry
- try to read the PTE
- add the mapping to TLB
- check P bit

2 - present: use the frame number in PTE to complete physical address

3 - absent: page fault ...

At Each Memory Access (2)

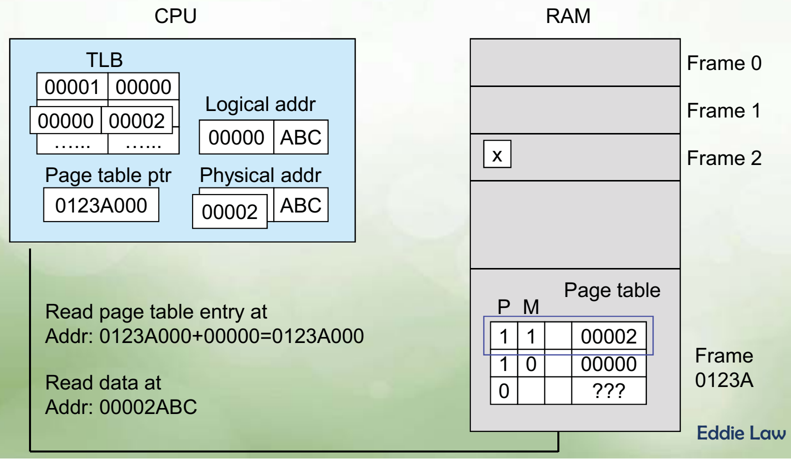
...

3 - absent: page fault ...

Page fault handling:
The CPU issues an interrupt/trap, and the OS handles it:
- block the process (other processes can run)
- find the page in the swap file in hard disk
- find an unused frame in RAM (w/ page replacement algorithm)
- issue a read request to the hard disk

When the disk I/O finishes:
- an interrupt occurs
- OS updates the page table of the blocked, page fault process
- OS wakes up the process

Example: Memory Access at TLB Miss (Page Present)



Page Size

- Smaller page size means
  - Less amount of internal fragmentation
- But … smaller page size also means
  - More pages required per process
  - More pages per process means larger page tables
  - Larger page tables means large portion of page tables in virtual memory
  - Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better

Page Size (cont'd)

- Small page size, large number of pages will be found in main memory
- As time goes on during execution, the pages in memory will all contain portions of the process near recent references
  - Page faults low
- Increased page size causes pages to contain locations further from any recent reference
  - Page faults rise

Segmentation

- May be unequal, dynamic size
- Simplifies handling of growing data structures
- Easier to sharing data among processes
- Easier to protect data

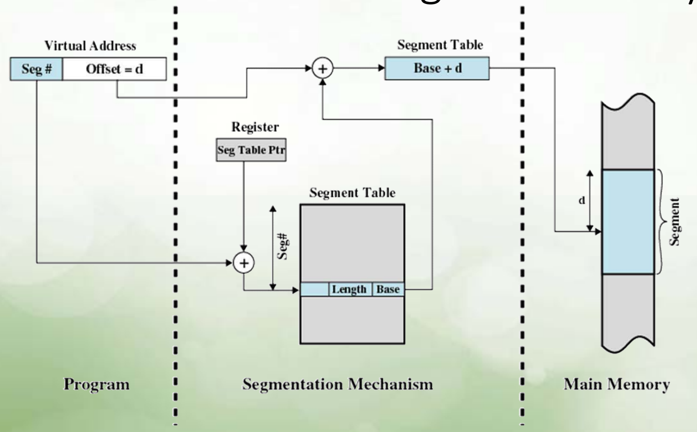
Segment Table

- Each entry contains the length of the segment, P bit, M bit, and the base address of the segment

Segment table

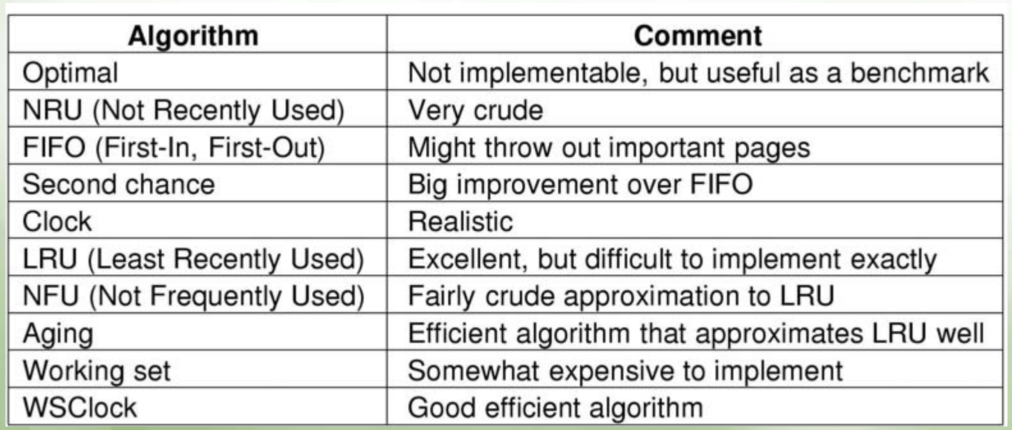
	P	M	length	base
0	1	0	40k	10101000
1	1	1	50k	20202000
2	1	1	20k	30303000

Address Translation in a Segmentation System



Address Translation in a Segmentation System

Page Replacement Algorithms: Reference



Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Next Topic

- I/O Management and Disk Scheduling