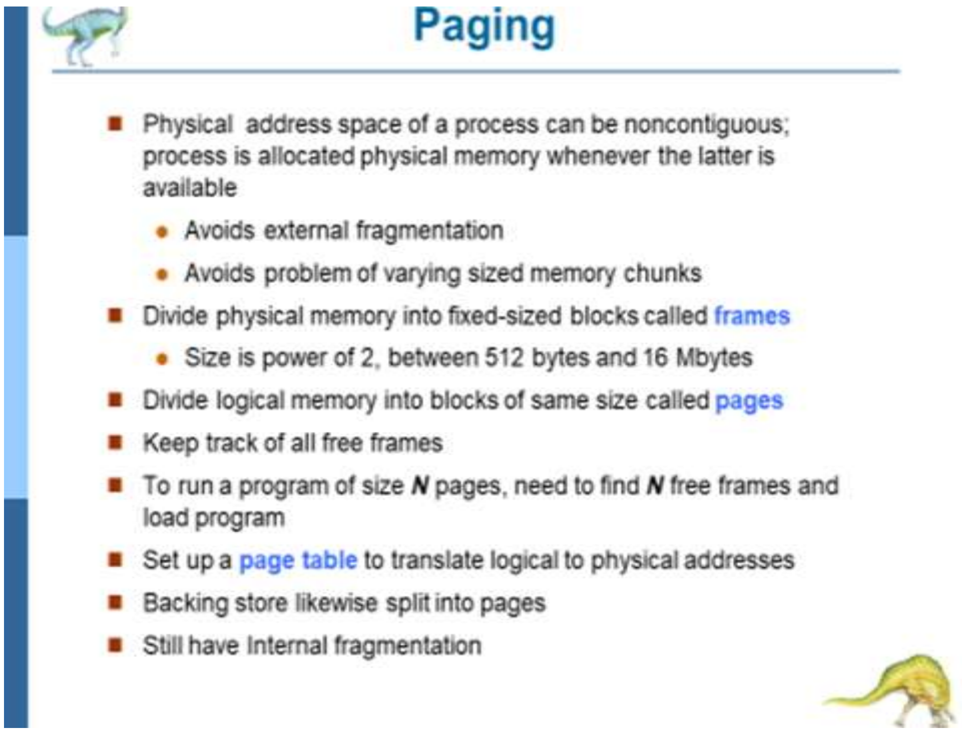
Topics: Memory Management

- **Paging**

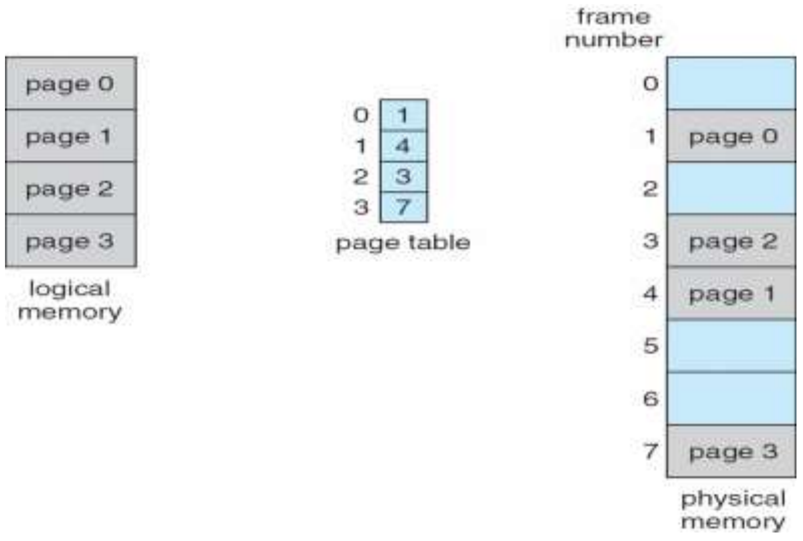
A solution to the external fragmentation problem is to permit the logical address to be non-contiguous. This can be implemented through the use of a paging scheme.



Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called frames
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called pages
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a page table to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation

Physical memory is broken into fixed-sized blocks called **frames**.
Logical memory is broken into blocks of the same size called **pages**



Translation between logical address and physical address

Page Table provides a mapping that assigns to each virtual page a page frame number or an auxiliary storage address number.

Frame Table contains information about the physical memory: how many total frames there are, which frames are allocated and to which page of which process, these frames are available.
When a process arrives to be executed, its size expressed in pages is examined. Each page of the process needs one frame.

If the process requires *n* pages, there must be at least *n* frames available in memory.
The first page of the process is loaded into one of the allocated frames, and the frame number is entered in the page table for this process.
The second page is loaded in the same manner and so on until the entire process is loaded.

The user's program address space will be scattered throughout the physical memory.

SOFTWARE SOLUTION

If	**S**	page size in bytes
	R	logical reference
	f	the frame number

then **Physical Address = f x S + offset = f x S + R mod S**

This is a mapping of the logical address into physical address using software computations.

Let's consider a hypothetical example:

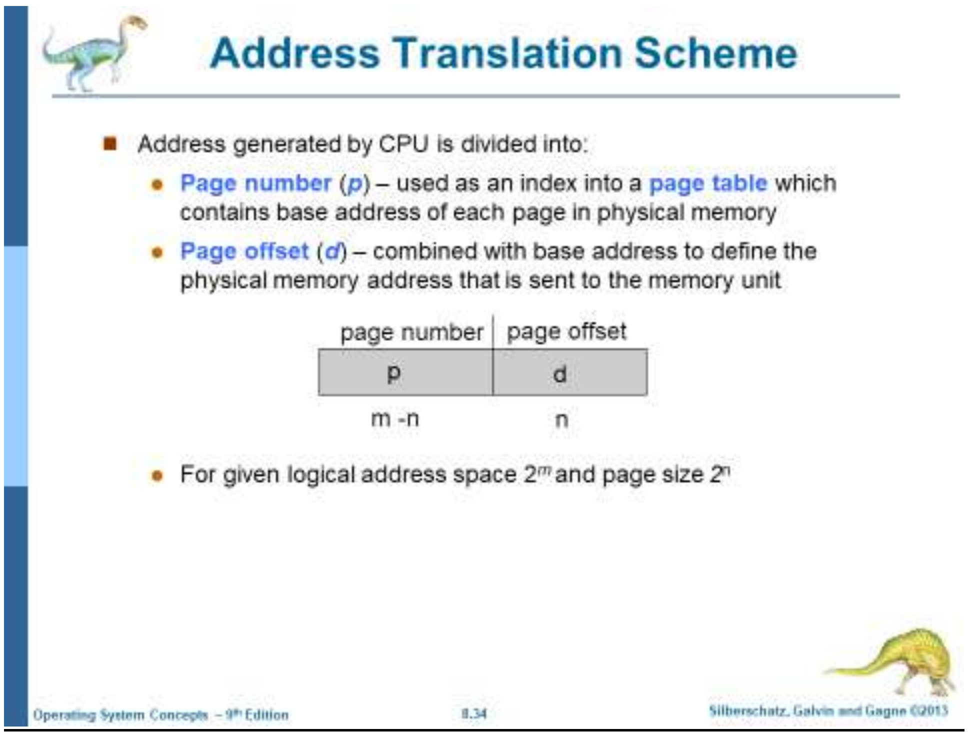
S	4
R	11

HARDWARE SOLUTION
Address translation that uses **Hardware Support** Is more convenient

physical address: frame number (f) | frame offset (displacement) d

The OS maintains a copy of the page table for each process. This copy is used by the OS for address mapping.

logical address: page number (p) | page offset (d)



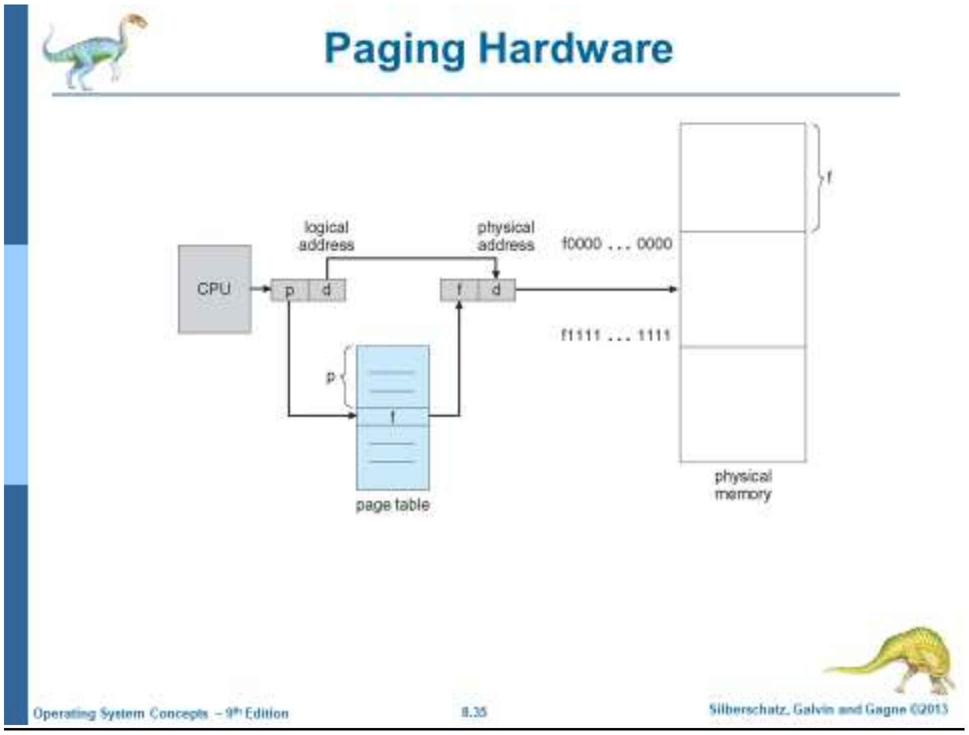
Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number** (*p*) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (*d*) – combined with base address to define the physical memory address that is sent to the memory unit

page number	page offset
p	d
m -n	n

  - For given logical address space 2^m and page size 2^n

Operating System Concepts – 9th Edition 8.34 Silberschatz, Galvin and Gagne ©2013



Paging Hardware

Operating System Concepts – 9th Edition 8.35 Silberschatz, Galvin and Gagne ©2013

THE COST OF PAGING FOR **Address Translation with Hardware Support**

Structure of the Page Table

Case 1: DIRECT MAPPING
The page table is in primary memory.

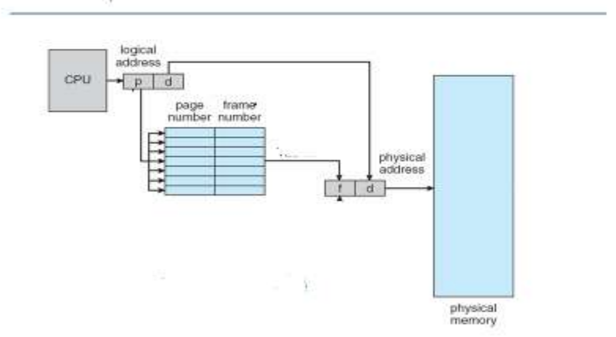
Two memory accesses are needed to access a byte.
Slowdown: the page table has to be accessed in memory.

the page table is kept in main memory and a **page-table-base-register (PTBR)** points to the page table. Changing page tables requires changing only this one register reducing the context switch time.

Case 2: ASSOCIATIVE MAPPING
Each register contains a key and a value (the frame number). Every entry in the associative storage is searched simultaneously.   Page table is implemented as a set of **dedicated registers**. The use of registers for the page table is satisfactory if the page table is reasonably small.

Extra cost: the time necessary to search the associative registers. One memory reference is made.

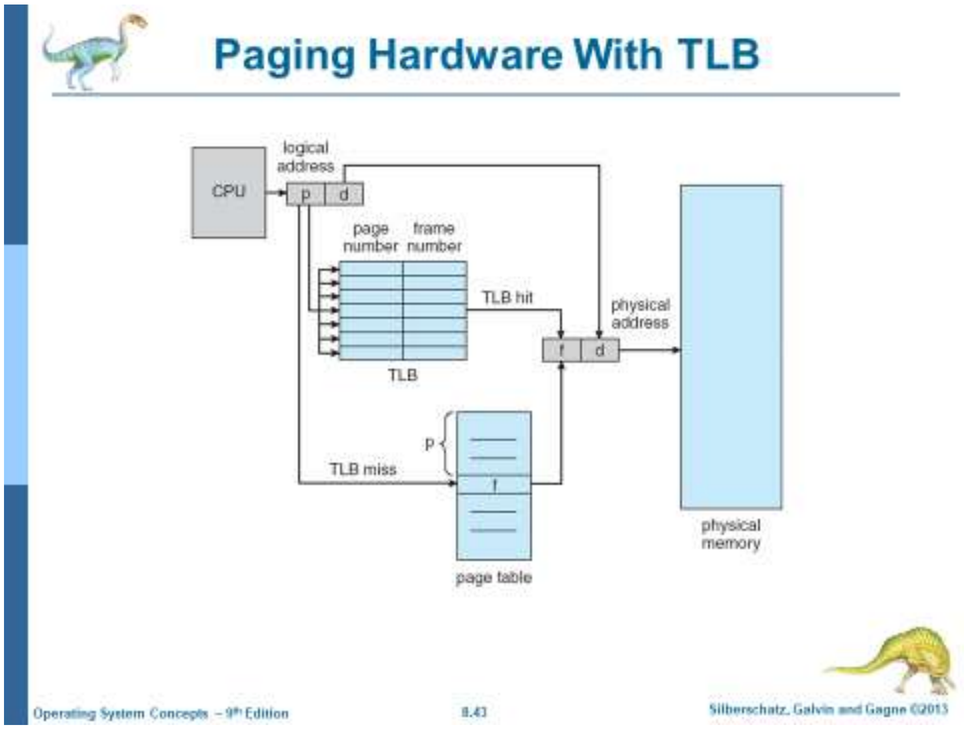
Effective Access Time:
Advantages:
Disadvantages:



<u>Case 3: Combined ASSOCIATIVE/DIRECT – USING TLB</u>
If the frame number is in the associative mapping the searching process stops, otherwise search further on the page table stored in the main memory.

It uses a special, small fast-lookup hardware cache, called **associative registers** or **translation look-aside buffers (TLBs)**

Usually the associative registers contain only a few of the page-table entries. If the page number is found in the associative registers, its number is immediately available. If the page number is not in the associative registers, a memory reference to the page table must be made. Search is fast, the hardware is expensive.



Paging Hardware With TLB

Operating System Concepts – 9th Edition 8.43 Silberschatz, Galvin and Gagne ©2013

h – TLB hit ratio the percentage of times that a page number is found in the associative registers.

effective access time = h * TLB hit access time + (1 - h) * TLB miss access time

TLB hit access time:
TLB miss access time

Advantage
Disadvantages

<u>Protection:</u>
Memory protection in a paged environment is accomplished by protection bits that are associated with each frame.
One bit can define a page to be read-write or read-only or execute-only.

valid-invalid bit: if the bit is set to valid then the reference is legal.
if the bit is set to 'invalid' the page is not in the process's logical address space.

<u>Shared Pages</u>
In multiprogrammed computer systems, especially in timesharing systems, it is common for users to execute the same programs. If individual copies of these programs were given to each user, then too much of primary storage would be wasted.

Nonmodifiable code is named reentrant code. Nonmodifiable code can be shared.
If the code is reentrant if it never changes during execution.
Heavily used programs can be shared: compilers, database systems etc.
Modifiable code cannot be shared.
All this points to the need to identify each page as either sharable or nonsharable.

COVER THE PROBLEMS OF THE 1ST LINK – Exercises Part 1

Segmentation

The user prefers to view memory as a collection of variable-size segments, with no necessary ordering among segments.
Segmentation is a memory-management scheme by which a program is allowed to occupy many separate segments (blocks) of physical memory. The segments (blocks) themselves need not be the same size, must consist of contiguous locations but the separate segments need not to be adjacent to one another.
The logical address will be a collection of segments. These segments can represent procedures, functions or various data structures.
Each segment has a length and a name. Elements within a segment are identified by their offset from the beginning of the segment.

logical address   < segment number, offset >

Hardware Implementation / Implementation of Segment Tables

The mapping from a user-defined address into a physical address is done using the segment table.
A segment table can be implemented either in fast registers or in memory.
The segment table has variable length, that's why it is more feasible to be in main memory.
Each entry of the segment table base contains a segment *base* and a segment *limit*.
When a particular process is running, a special register (segment table base register) will hold the Starting address of the segment table for that process.
Because the number of segments used by a program may vary, a segment table length register is used (STLR).