CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

UNIVERSITY INSTITUTEOF ENGINEERING

Bachelor of Engineering (Computer Science & Engineering)

Operating System (CST-328)

Subject Coordinator: Er. Puneet kaur (E6913)

Introduction to Operating System
Font size 24

DISCOVER . LEARN . EMPOWER

Lecture 16

Paging
&
Paging Techniques

PAGING

Permits a program's memory to be physically noncontiguous so it can be allocated from wherever available. This avoids fragmentation and compaction.

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever that memory is available and the program needs it.
- Divide **physical** memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8192 bytes).
- 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.
- Internal fragmentation.

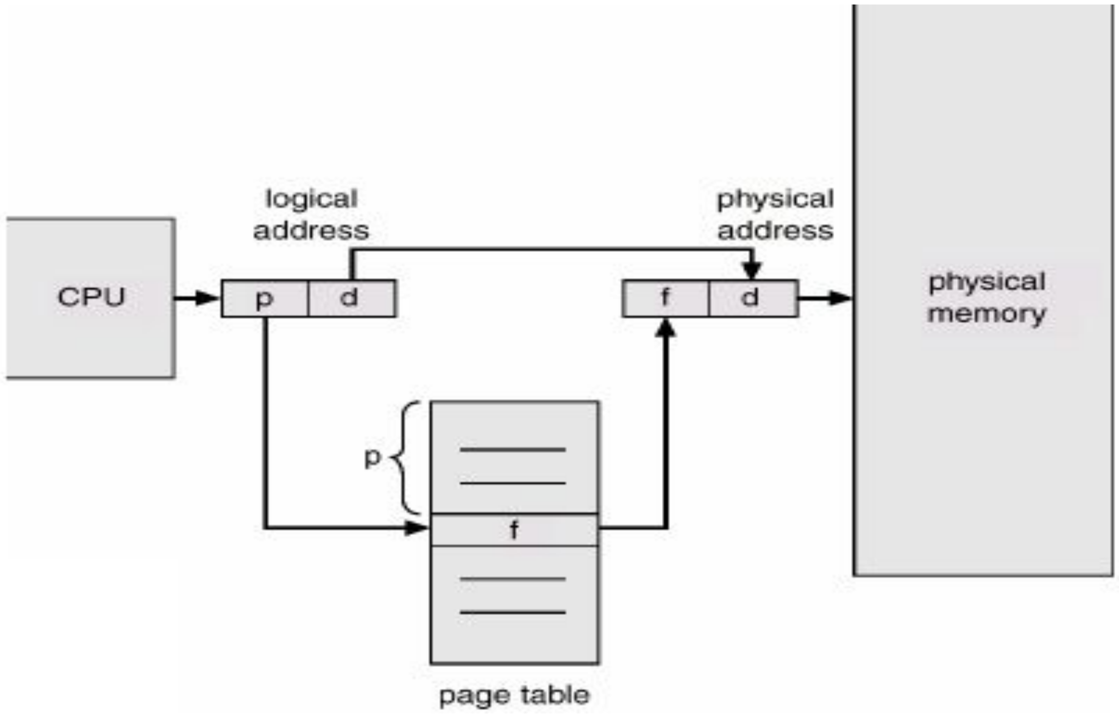
PAGING

- An address is determined by:

  page number (index into table) + offset

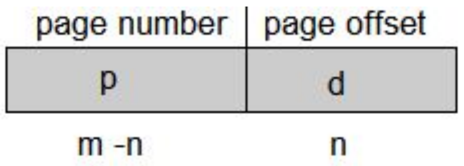
  ---> mapping into --->

  base address (from table) + offset.



PAGING

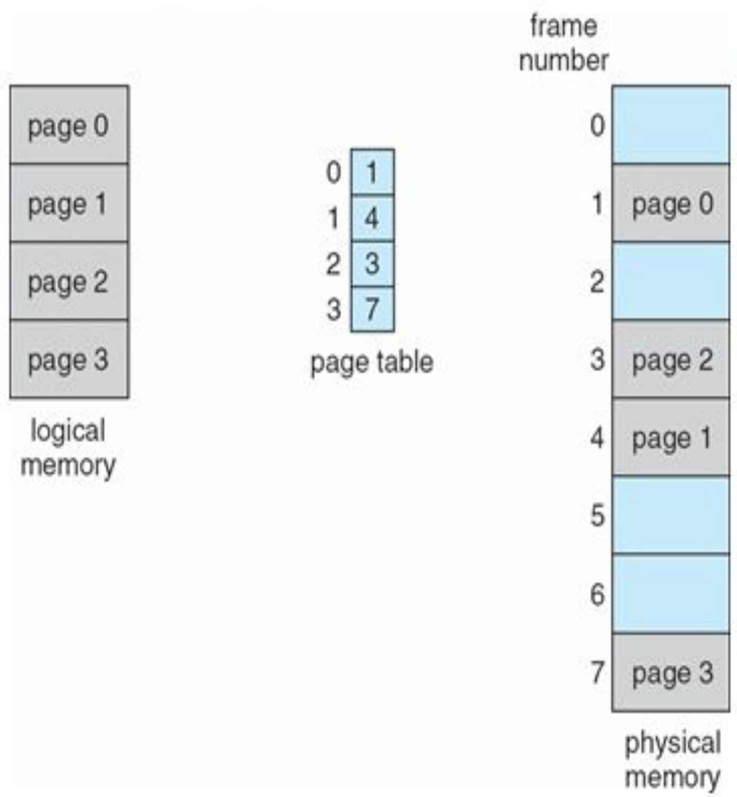
- **Address generated by the CPU is divided into:**
  - *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory.
  - *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit.

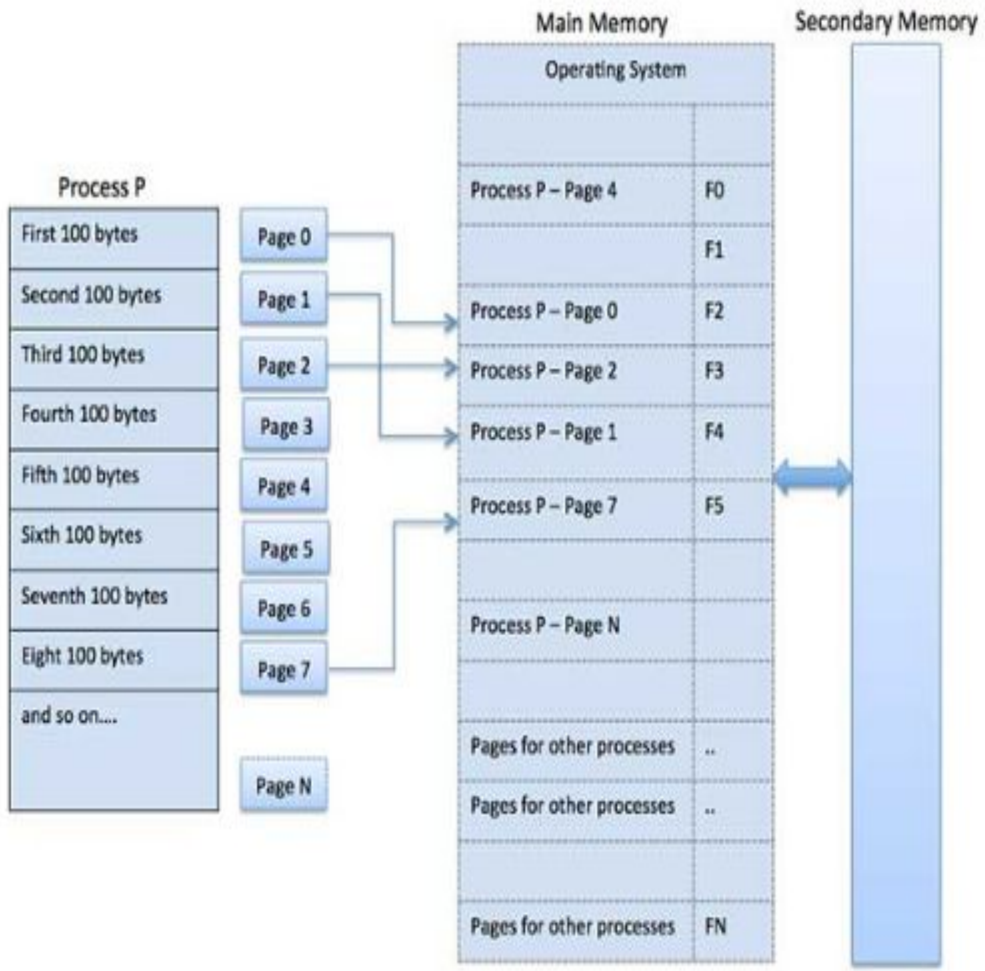


For given logical address space 2 and page size 2

- **Address Translation Scheme**
  - Logical Address = Page number + page offset
  - Physical Address = Frame number + page offset

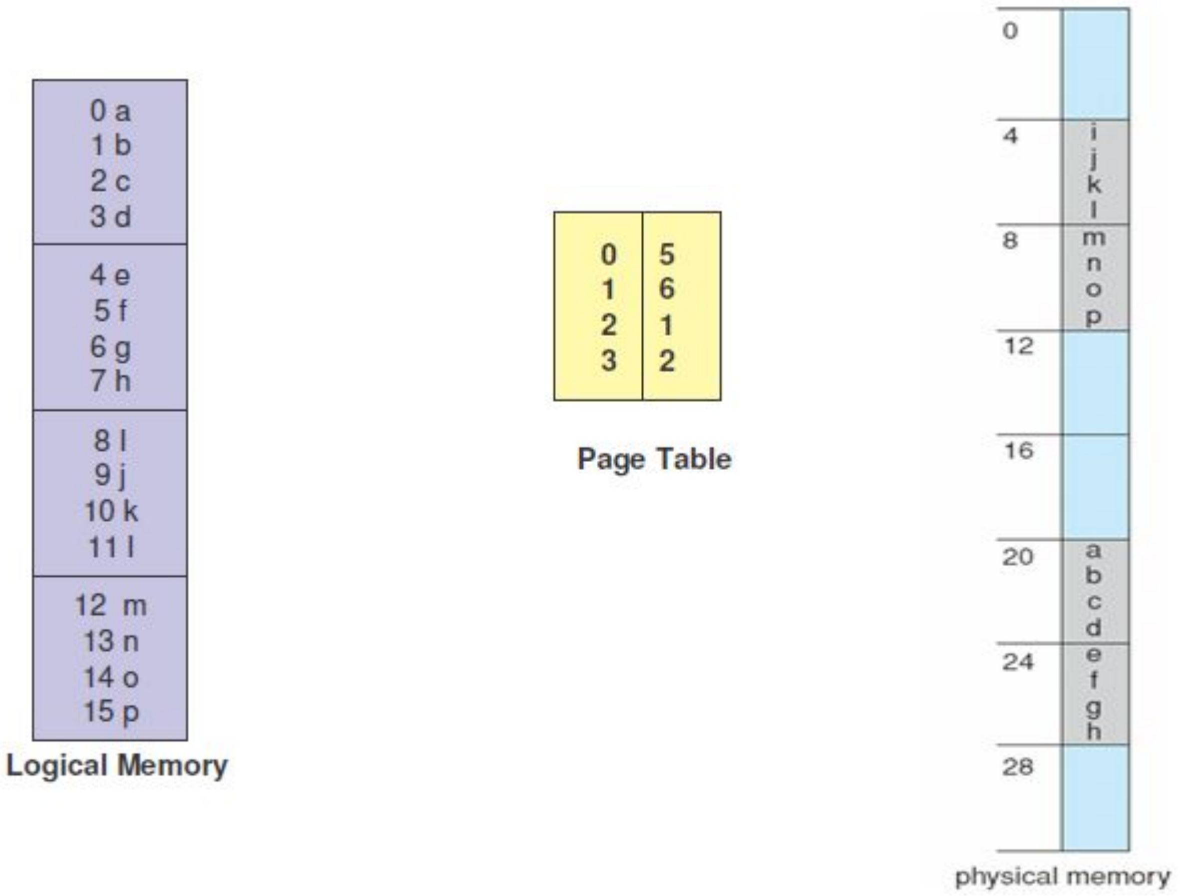
PAGING


frame number
page 0
page 1
page 2
page 3
logical memory
0 1
1 4
2 3
3 7
page table
0
1 page 0
2
3 page 2
4 page 1
5
6
7 page 3
physical memory


Main Memory
Secondary Memory
Operating System
Process P
First 100 bytes
Second 100 bytes
Third 100 bytes
Fourth 100 bytes
Fifth 100 bytes
Sixth 100 bytes
Seventh 100 bytes
Eight 100 bytes
and so on....
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7
Page N
Process P – Page 4 F0
F1
Process P – Page 0 F2
Process P – Page 2 F3
Process P – Page 1 F4
Process P – Page 7 F5
Process P – Page N
Pages for other processes ..
Pages for other processes ..
Pages for other processes FN

PAGING Example

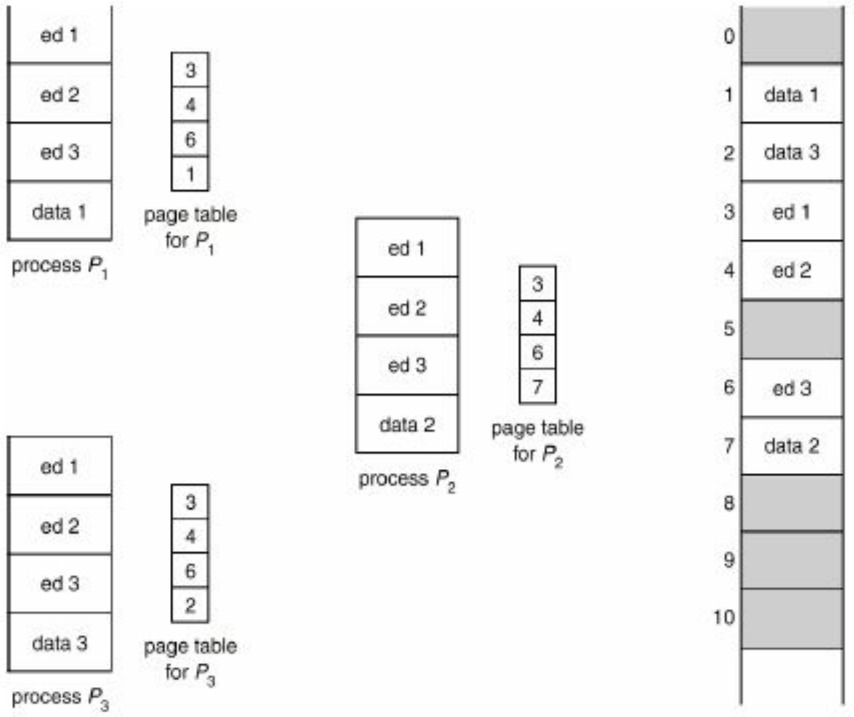
32-byte memory with 4-byte pages



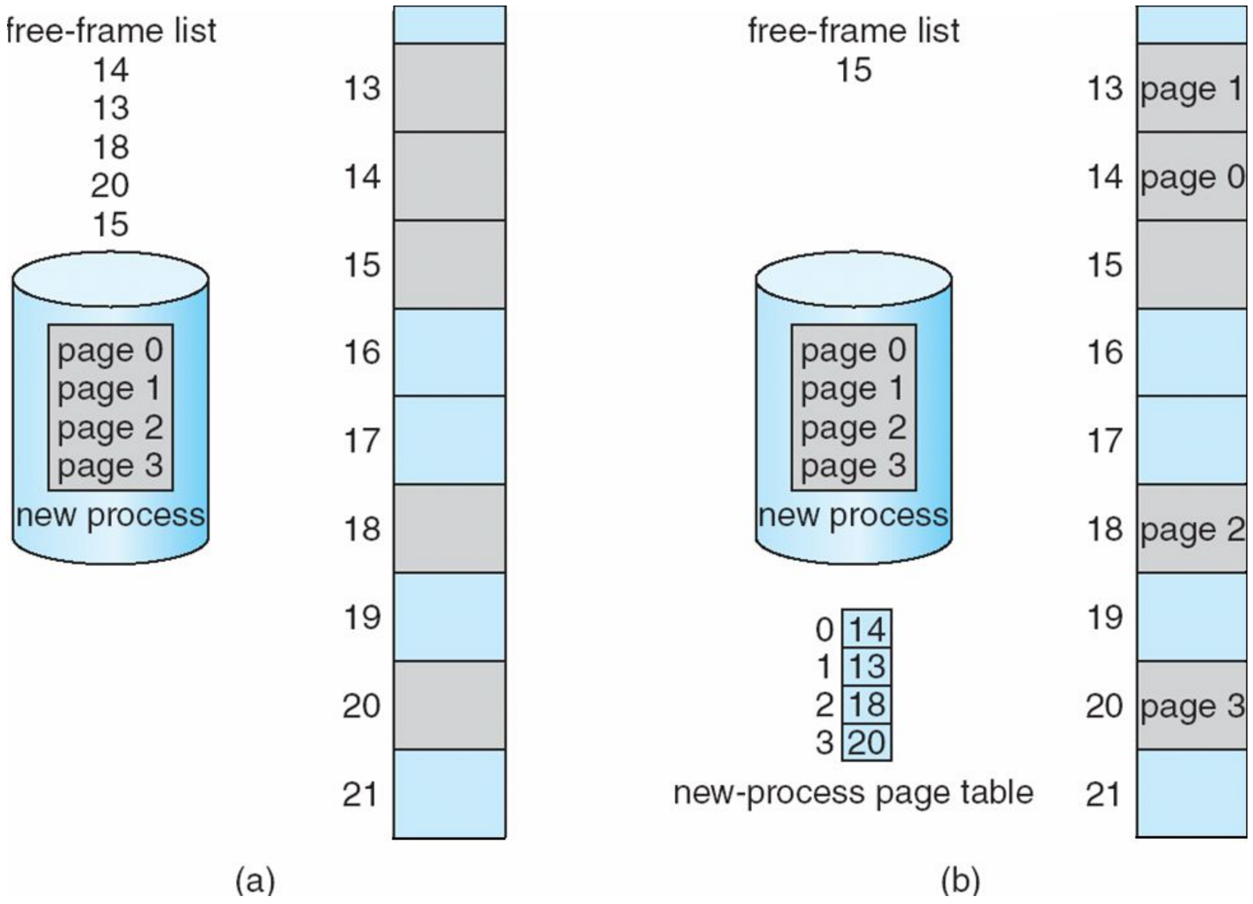
SHARED PAGES

SHARED PAGES

- Data occupying one physical page, but pointed to by multiple logical pages.
- Useful for common code -must be write protected. (NO write-able data mixed with code.)
- Extremely useful for read/write communication between processes.



Free Frames



Before allocation

After allocation

PAGING

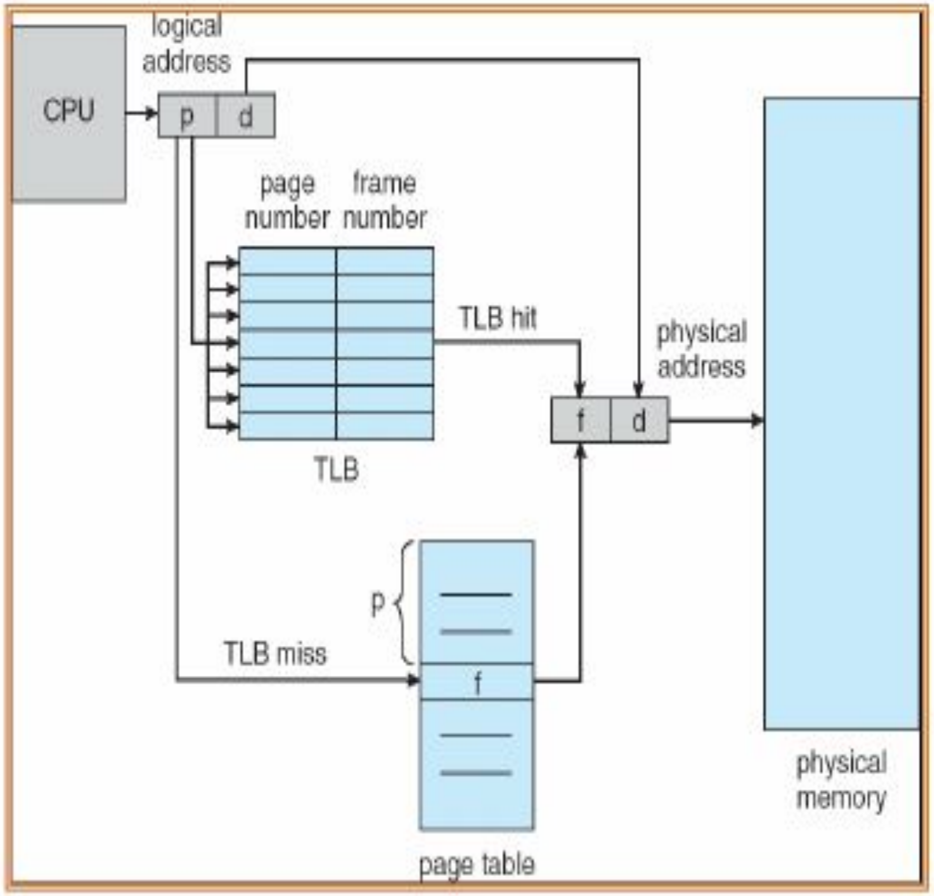
- **Calculating internal fragmentation**
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - Worst case fragmentation = 1 frame – 1 byte
  - On average fragmentation = 1 / 2 frame size
  - So small frame sizes desirable?
  - But each page table entry takes memory to track
  - Page sizes growing over time
    - Solaris supports two page sizes – 8 KB and 4 MB
- Process view and physical memory now very different
- By implementation process can only access its own memory

Implementation of Page Table

- Page table is kept in main memory

Page-table base register (PTBR) points to the page table
Page-table length register (PTLR) indicates size of the page table

- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative memory or **translation look-aside buffers (TLBs)**



Implementation of Page Table

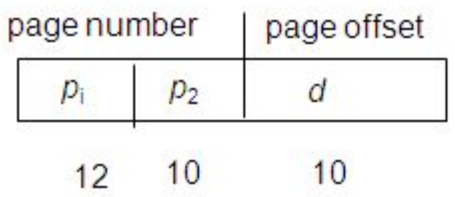
- Some TLBs store **address-space identifiers** (**ASIDs**) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
  - Replacement policies must be considered
  - Some entries can be **wired down** for permanent fast access

Structure of the Page Table

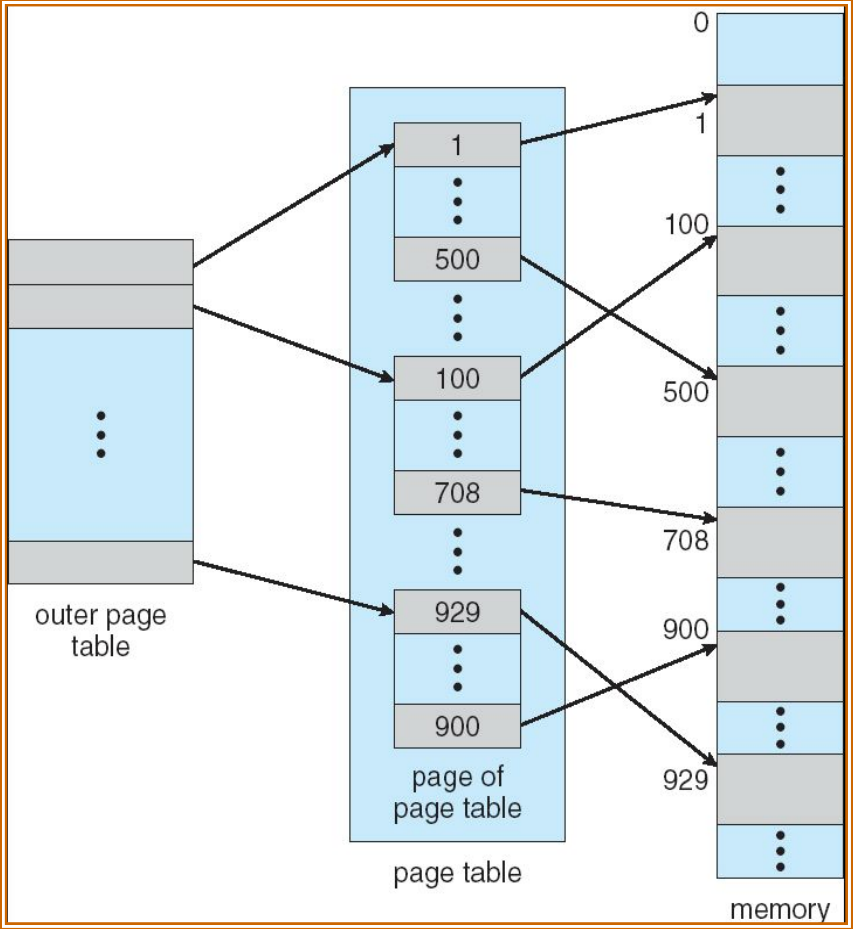
- Often a page table can become large
  - 32-bit address space with 4 KB (2^{12}) page
  - 2^{20} paging entries (1M), each entry 4B = 4MB
  - How about 64-bit address space?
  - It may not be allocated contiguously in physical memory
  - It may not fit into physical memory
- How to solve these issues?
  - Hierarchical Paging
  - Hashed Page Tables
  - Inverted Page Tables

Hierarchical Page Tables

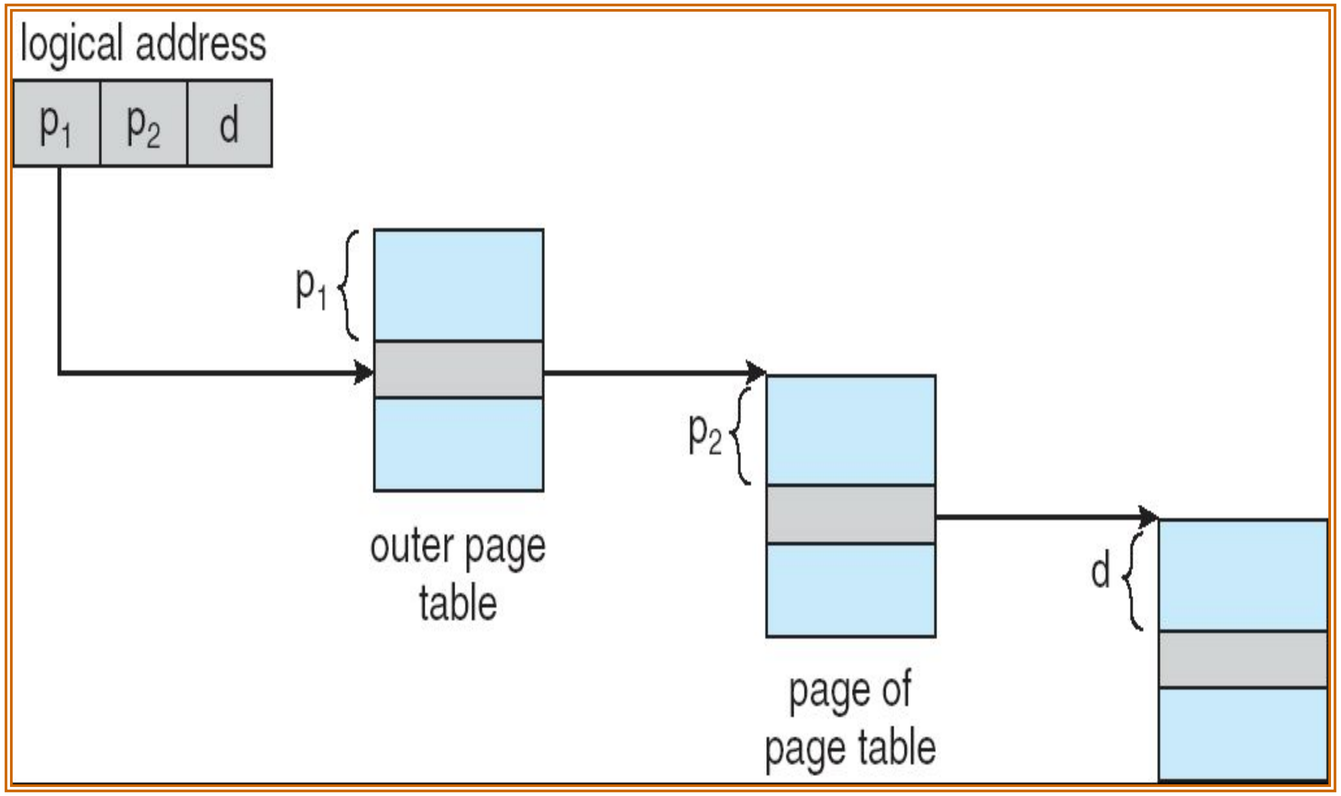
- Page the page table
  - Two or multi-level page table
- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:



- where pi is an index into the outer page table, and p2 is the displacement within the page of the outer page table



Address-Translation Scheme



Hashed Page Tables

- Common in address spaces > 32 bits
  - Hierarchical paging is too slow (in case of a TLB miss)
- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
- Know what is a hash table?
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

Brief Introduction on Hash table

- Why hash table?
- Suppose that we want to store 10,000 students records (each with a 5-digit ID) in a given container.
  - A linked list implementation would take O(n) time.
  - A height balanced tree would give O(log n) access time.
  - Using an array of size 100,000 would give O(1) access time but will lead to a lot of space wastage.
- Is there some way that we could get O(1) access without wasting a lot of space?
  - Consider the case of large 64-bit address space with sparse entries (limited by physical memory & hard disk)

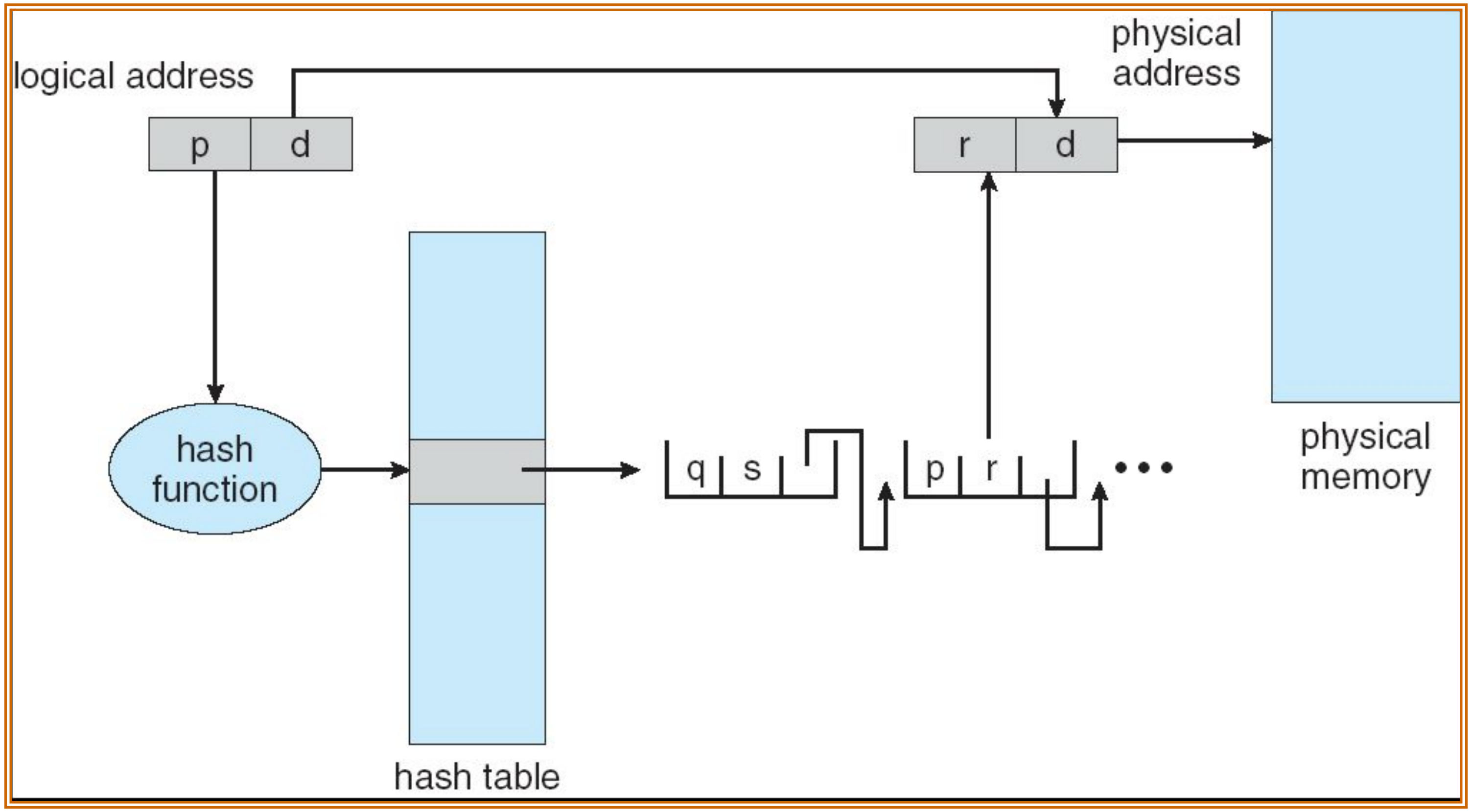
Types of Hashing

- There are two types of hashing :

  1. Static hashing: In static hashing, the hash function maps search-key values to a fixed set of locations.
  - In case of collision -> add overflow entries

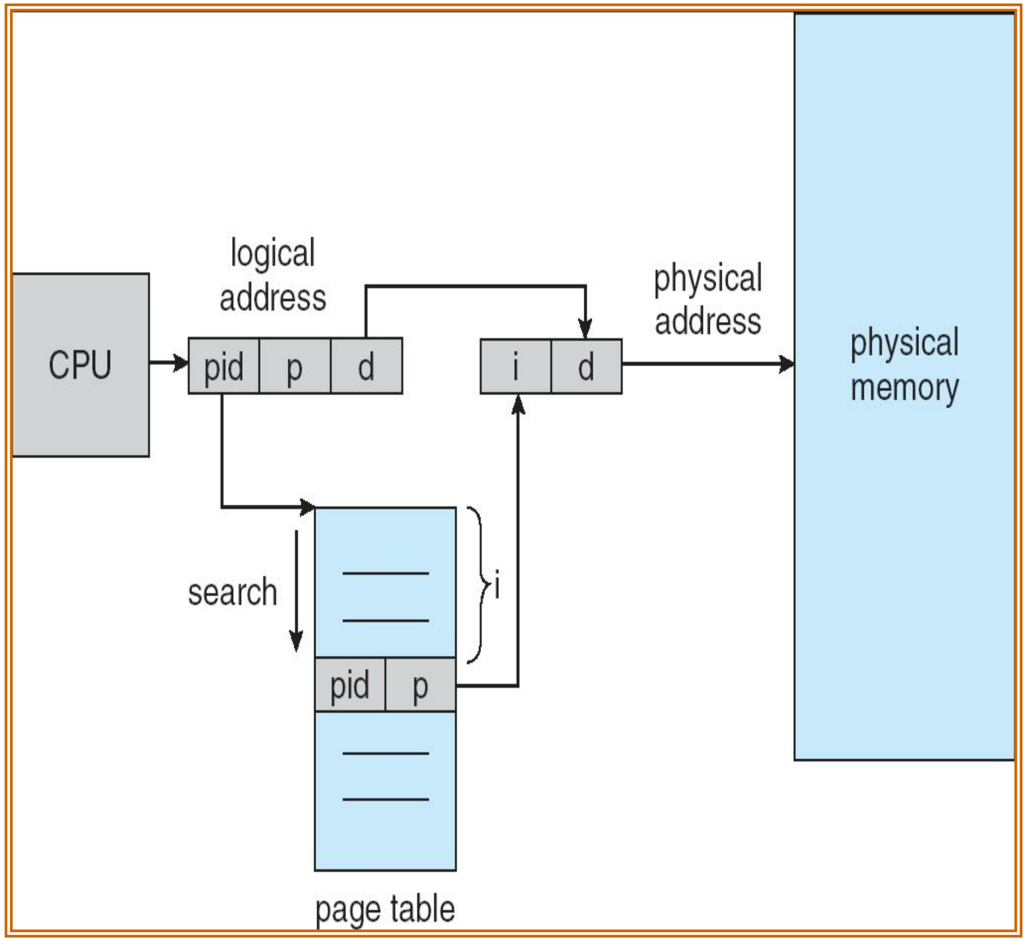
  2. Dynamic hashing: In dynamic hashing a hash table can grow to handle more items. The associated hash function must change as the table grows.
  - In case of collision -> grow the hash table

Hashed Page Table



Inverted Page Table

- To keep the page table small
  - One entry for each frame of the physical memory
- Each entry has (pid, page-number)
- Given (pid, page-number), how to find the physical frame?
  - Very slow: scan entries in the table
  - Fast: build a hash table with the hash key = (pid, page-number)



Conclusion

This lecture enables the students to understand Paging, shared pages, implementation of page tables and page table structures.

Video Link

https://www.youtube.com/watch?v=pJ6qrCB8pDw

https://www.youtube.com/watch?v=2pgI46Q72fA

https://www.youtube.com/watch?v=SqYigYLFvcI

References

https://www.geeksforgeeks.org/partition-allocation-methods-in-memory-management/

https://www.javatpoint.com/os-memory-management-introduction

http://www2.latech.edu/~box/os/ch08.pdf

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/8_MainMemory.html#:~:text=8.3%20Contiguous%20Memory%20Allocation,allocated%20to%20processes%20as%20needed.

http://www.csdl.tamu.edu/~furuta/courses/99a_410/slides/chap08

https://www.tutorialspoint.com/operating_system/os_memory_management.htm

https://www.studytonight.com/operating-system/memory-management

https://www.guru99.com/os-memory-management.html