
1878
Western
UNIVERSITY · CANADA

Chapter 9B – Memory Management (Paging)

Spring 2023

Overview

- Basic Method
- Hardware Support
- Protection
- Shared Pages
- Structure
- Swapping

Basic Method

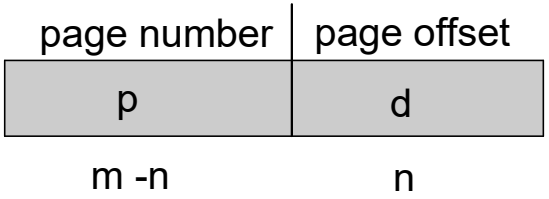
- With paging, process space is no longer contiguous
- Used by most modern operating systems
- Partition the physical memory into fixed-size blocks called **frames**
- Partition the logical memory into same-sized fixed-size blocks called **pages**
  - There can be more pages than there are frames. This is virtual memory and will be covered in the next chapter.
- Use a **Page Table** for each process to translate logical addresses to physical addresses

Basic Method

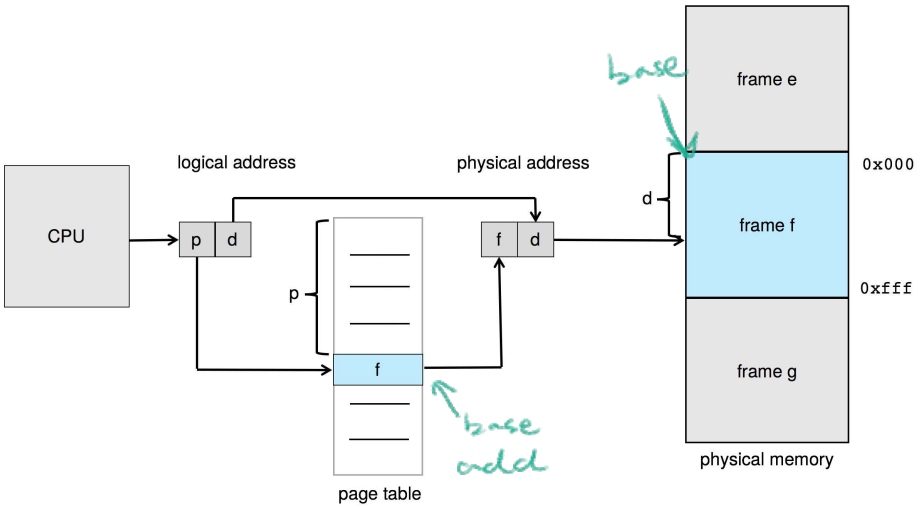
- Processes use pages 0 to N so it *thinks* the memory is contiguous
- However, the physical location of each frame in memory is non-contiguous
- Internal fragmentation is still possible <= not all frames are used but at least all processes could find a place to go.
- External fragmentation is avoided <=

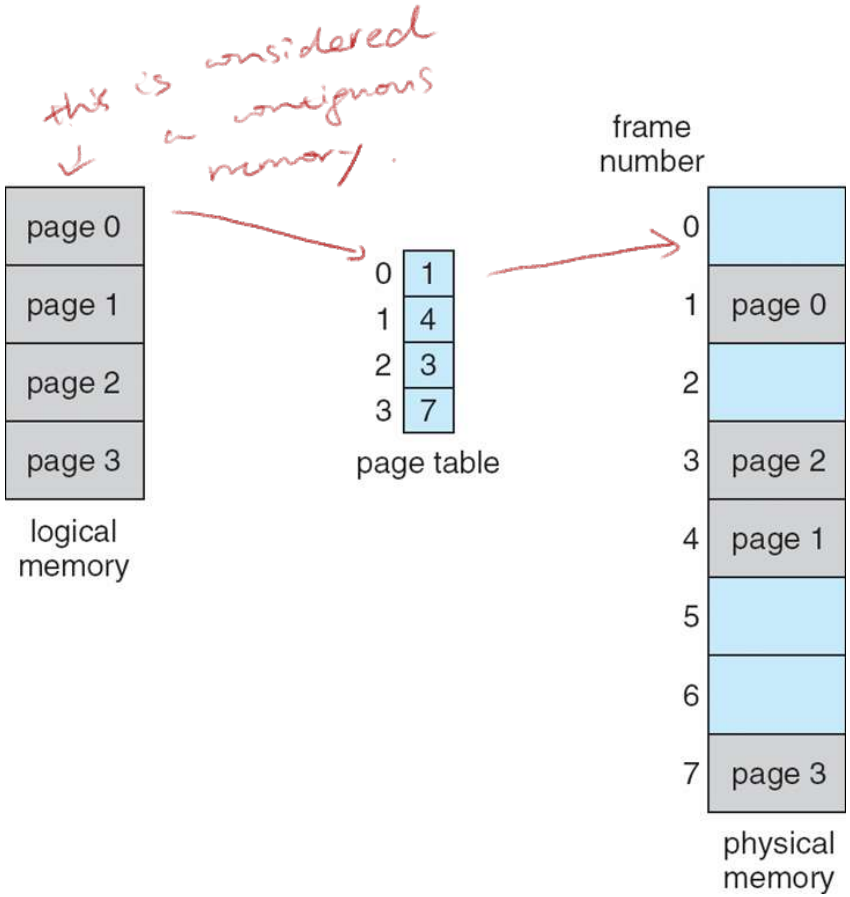
Basic Method

- The logical address generated by the CPU is divided into a bitmap of length m. That's 2^m logical addresses with:
  - **Page number (p)** - used as an index into a page table which contains base address of each page in physical memory. It is of length m-n or 2^{m-n} bits.
  - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit. It is length n or 2^n bits. This is the size of each page.



Basic Method





Basic Method

- Example page numbers and page offsets
  - 3 bits – 2^1 | 2^2 (2 pages, 4 page offsets each)

000	Page 0	Location 0
001	Page 0	Location 1
010	Page 0	Location 2
011	Page 0	Location 3
100	Page 1	Location 0
101	Page 1	Location 1
110	Page 1	Location 2
111	Page 1	Location 3

Basic Method

- Example page numbers and page offsets
  - 4 bits – 2^3 | 2^1 (8 pages, 2 page offsets each)

0000	Page 0	Location 0
0001	Page 0	Location 1
0010	Page 1	Location 0
0011	Page 1	Location 1
0100	Page 2	Location 0
0101	Page 2	Location 1
0110	Page 3	Location 0
0111	Page 3	Location 1

1000	Page 4	Location 0
1001	Page 4	Location 1
1010	Page 5	Location 0
1011	Page 5	Location 1
1100	Page 6	Location 0
1101	Page 6	Location 1
1110	Page 7	Location 0
1111	Page 7	Location 1

Basic Method

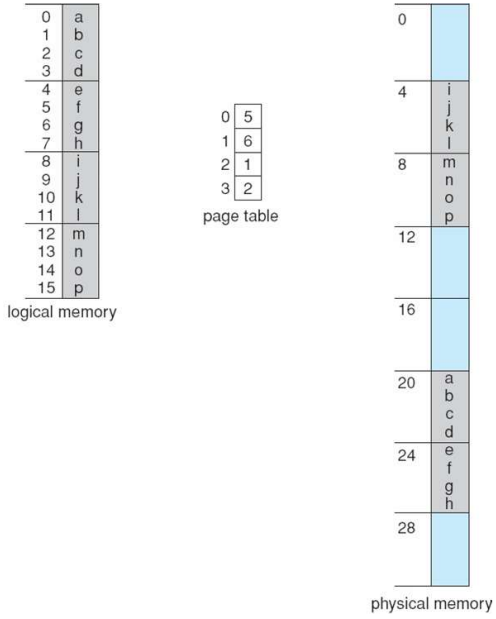
- Example page numbers and page offsets
  - 4 bits – 2^2 | 2^2 (4 pages, 4 page offsets each)

0000	Page 0	Location 0
0001	Page 0	Location 1
0010	Page 0	Location 2
0011	Page 0	Location 3
0100	Page 1	Location 0
0101	Page 1	Location 1
0110	Page 1	Location 2
0111	Page 1	Location 3

1000	Page 2	Location 0
1001	Page 2	Location 1
1010	Page 2	Location 2
1011	Page 2	Location 3
1100	Page 3	Location 0
1101	Page 3	Location 1
1110	Page 3	Location 2
1111	Page 3	Location 3

Basic Method

- Mapping pages to frames
  - 5 bits – 2^3 | 2^2 (8 pages, 4 page offsets each)
    - Page 0, offset 0 (**a**) is at physical address 20
    - Page 0, offset 3 (**d**) is at physical address 23
    - Page 1, offset 0 (**e**) is at physical address 24
    - Page 3, offset 2 (**n**) is at physical address 9
- (Note: In this diagram, if a physical address is handled in 1-byte increments, there are 32 bytes of memory, a page is 4 bytes, and there are 8 pages/frames)



Basic Method

- Internal fragmentation
  - Consider a page size of 2048 bytes and process size is 72,766 bytes
  - 35 pages + 1,086 bytes
  - The last page is using 1,086 bytes with 962 bytes left over
  - A process could use n pages + 1 byte so n+1 frames in the worst case. Almost the entire frame is useless. On average, ½ of the last frame is unused
  - Smaller page sizes minimizes the waste but adds overhead to the operating system
  - Larger page sizes make disk I/O to memory more efficient but adds more internal fragmentation

Basic Method

- Internal fragmentation
  - Most operating systems support two page sizes
    - 4KB or 8KB for regular processes
    - 2MB or 4MB or higher for processes that use a lot of memory

Basic Method

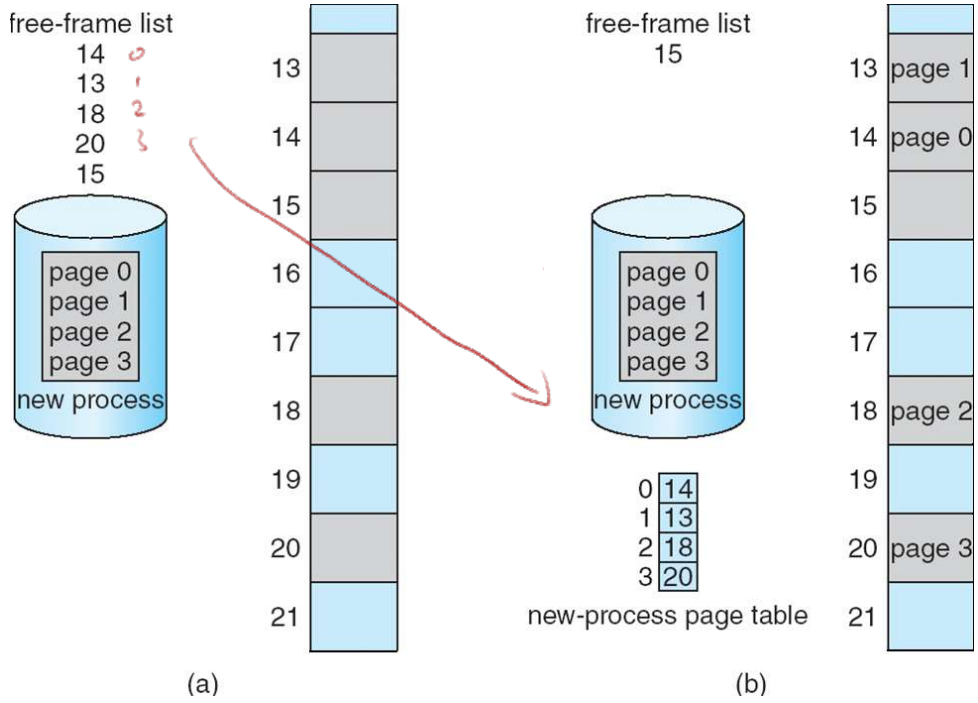
- External fragmentation
  - We have none. Any free frame can be allocated to a process as needed.

Basic Method

- Free frames
  - The operating system maintains a system-wide frame table
    - One entry per frame
    - Each entry has a flag indicating whether the frame is free or not
      - If it is not free, which process is using it

Basic Method

- Free frames



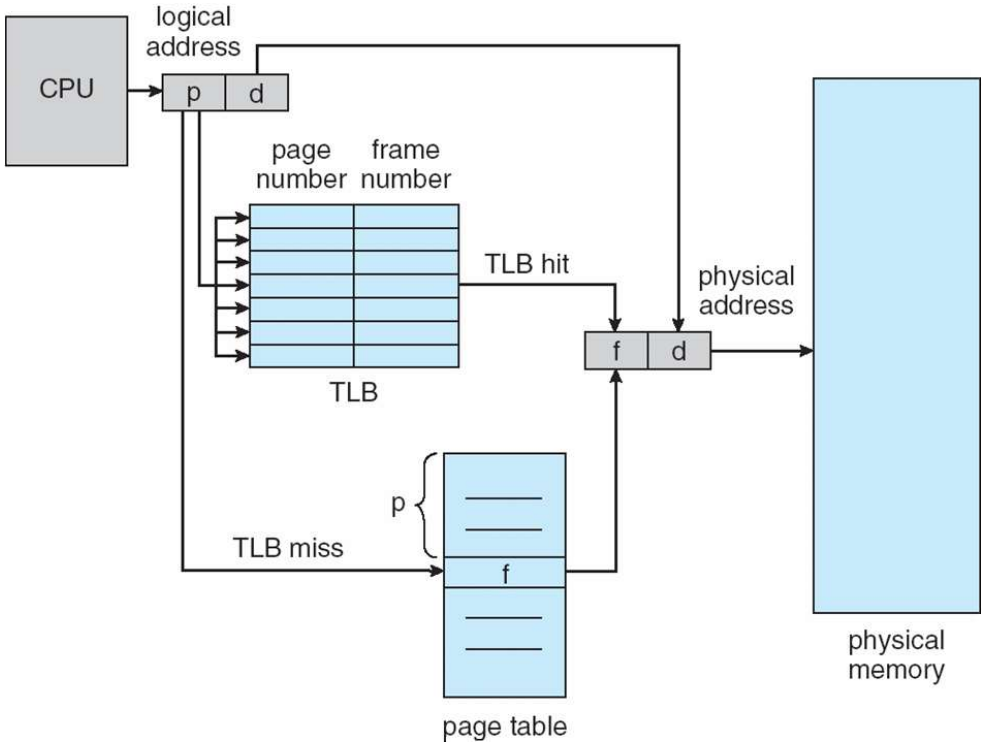
Hardware Support

- Due to the size of a process' page table, the table is kept in memory
  - The address of the page table is stored in the **Page-table base register (PTBR)**
  - The size of the page table is stored in the **Page-table length register (PTLR)**
  - Every instruction that accesses memory will trigger two memory accesses:
    - One to reference the page table itself and find the physical address
    - One to access the found address in memory
  - Solution: Cache the page table (Most systems have multiple levels)
    - This is called the **Translation Look-aside Buffer (TLB)**

Hardware Support

- Some TLBs hold address-space identifiers (ASIDs) so processes can only use their own entries
  - Otherwise, the TLB will need to be reloaded at each context switch
- Typically small (e.g. 64 – 1024 entries). Some entries can be made permanent.
- If the page is found in the TLB (a **hit**), use the frame
- If the page is not found in the TLB (a **miss**)
  - Consult the page table in memory instead
  - Load it into the TLB so we can avoid a miss in the future (Use an algorithm like Least-recently used, round-robin, or random to replace existing entries)

Hardware Support



Hardware Support

- Effective memory-access time
  - The percentage of times the page number of interest is found in the TLB is the **hit ratio**
    - If it is not found in the TLB, we need to consult the page table (1 access)
    - Then we can use the address (1 access)

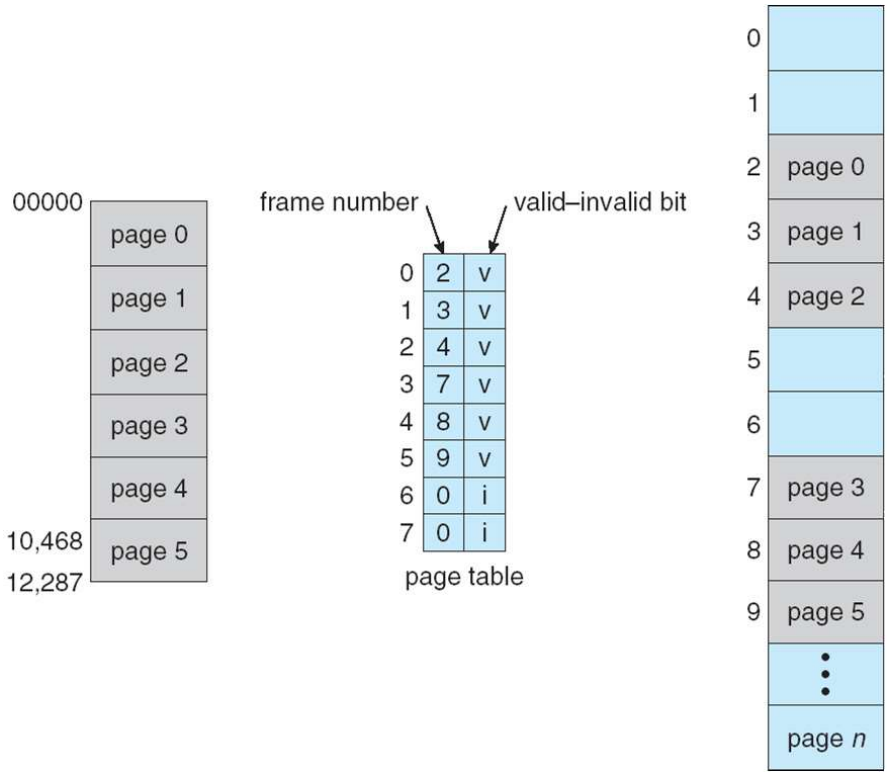
$$\text{Effective Access Time} = (Hit) \bullet AccessTime + (1 - Hit) \bullet 2(AccessTime)$$

  - Suppose a hit ratio of 80% and a memory access time of 10 nanoseconds, then effective access time is 0.80x10 + 0.2x20 = 12 nanoseconds
  - A hit ratio of 99% is 10.1 nanoseconds

Protection

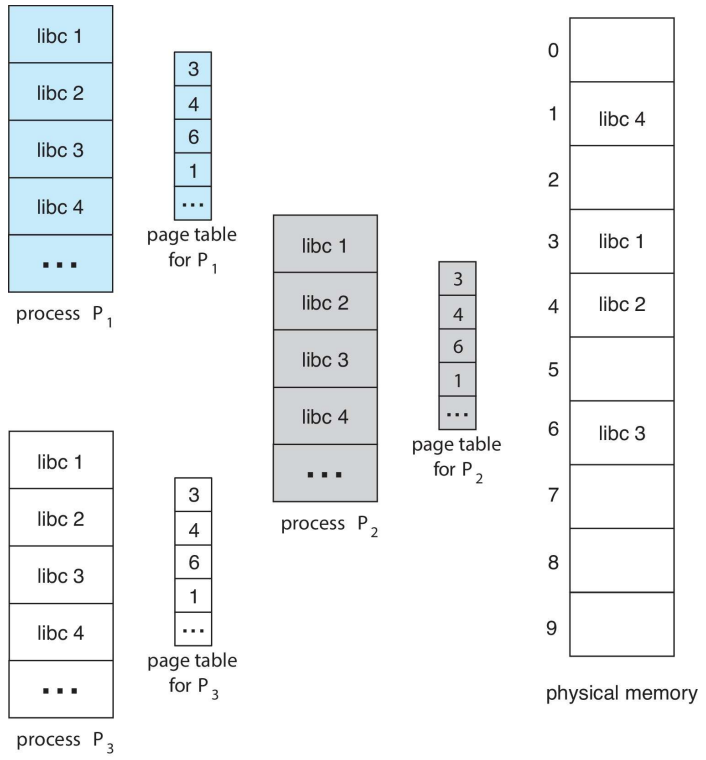
- Each entry in the page table can hold a protection bit to indicate whether the frame is read-only or read-write.
- To determine whether the frame is in the process' logical address space
  - Use a page-table length register
  - Each entry in the page table can hold a valid-invalid bit for all frames in the address space

Protection


00000
page 0
page 1
page 2
page 3
page 4
10,468
page 5
12,287
frame number
valid–invalid bit
0 2 v
1 3 v
2 4 v
3 7 v
4 8 v
5 9 v
6 0 i
7 0 i
page table
0
1
2 page 0
3 page 1
4 page 2
5
6
7 page 3
8 page 4
9 page 5
page n

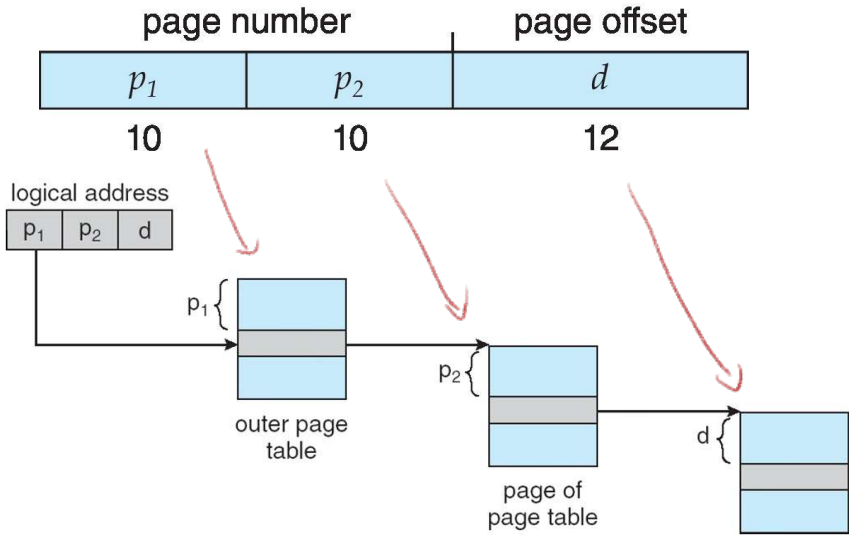
Shared Pages

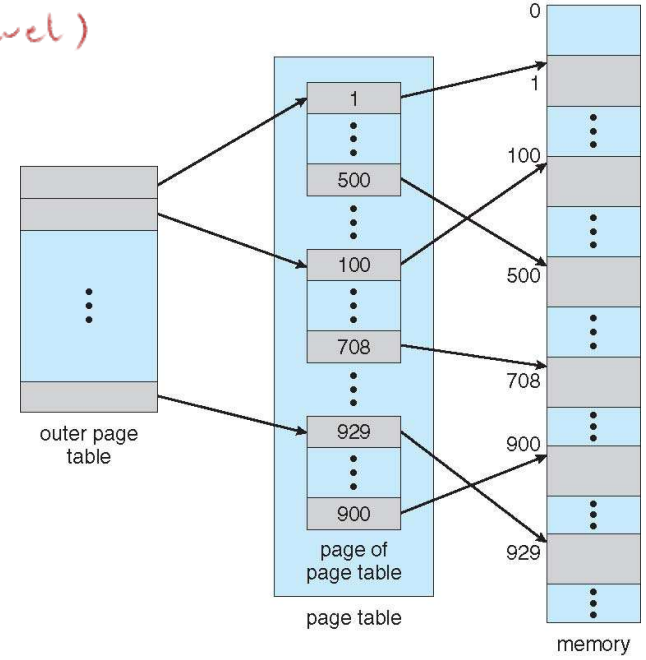
- Paging makes sharing address space easy
- These pages should be read-only
- Commonly used for DLLs



Structure

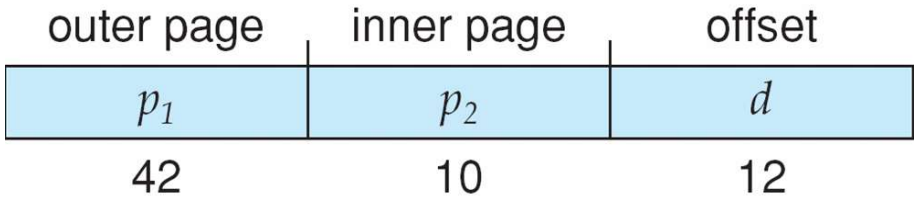
- Page tables can be very large. We don't want to put this in one contiguous block
  - Solution: **Hierarchical Page Tables** (multilevel)

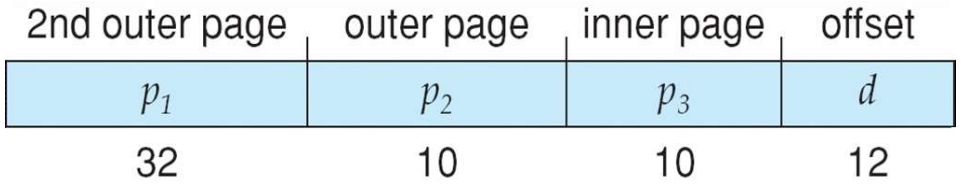




Structure

- Here are some examples using 64 bits
- More page tables means more memory accesses, so this strategy only goes so far



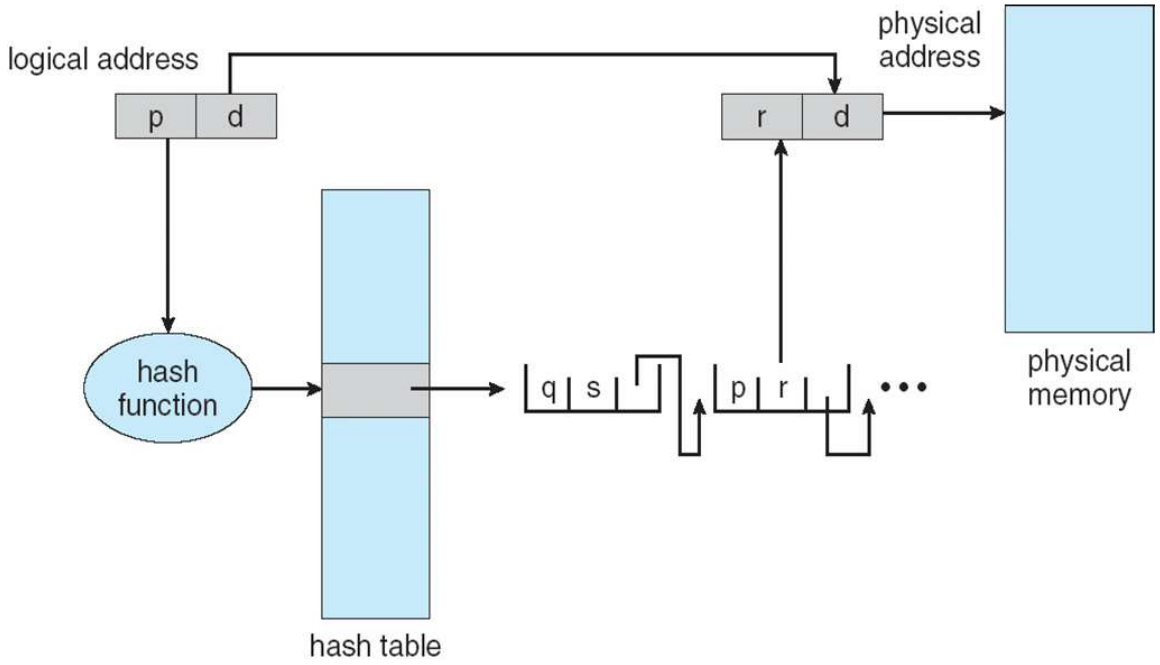


Structure

- **Hashed Page Tables**
  - Common in address spaces larger than 32 bits
  - Each entry in the page table holds
    - A hash of the page number
    - The frame
    - A pointer to the next hash
  - A logical address is hashed and then the matching hash is searched in the list

Structure

- Hashed Page Tables

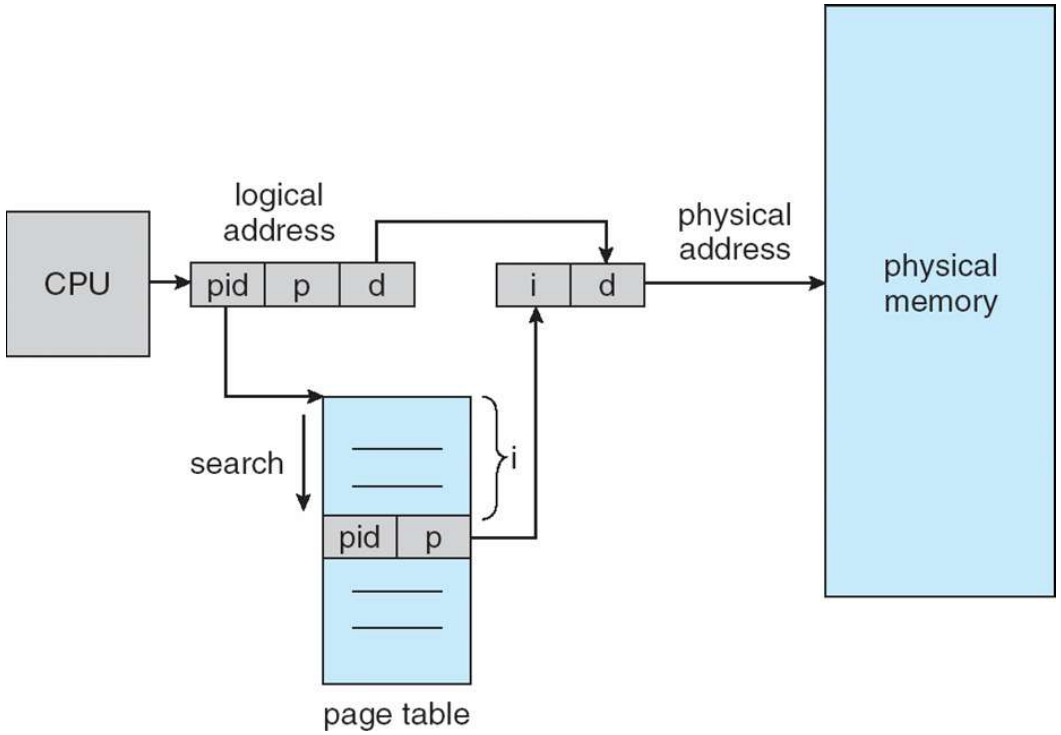


Structure

- **Inverted Page Tables**
  - Instead of one page table per process, one page table for all processes
  - One entry in the table for each frame in the system
  - This removes the need to maintain many page tables
  - This adds to the amount of time it takes to search the page table
  - Since pages to frames is one-to-one, shared memory will be regularly mapped and re-mapped

Structure

- Inverted Page Tables

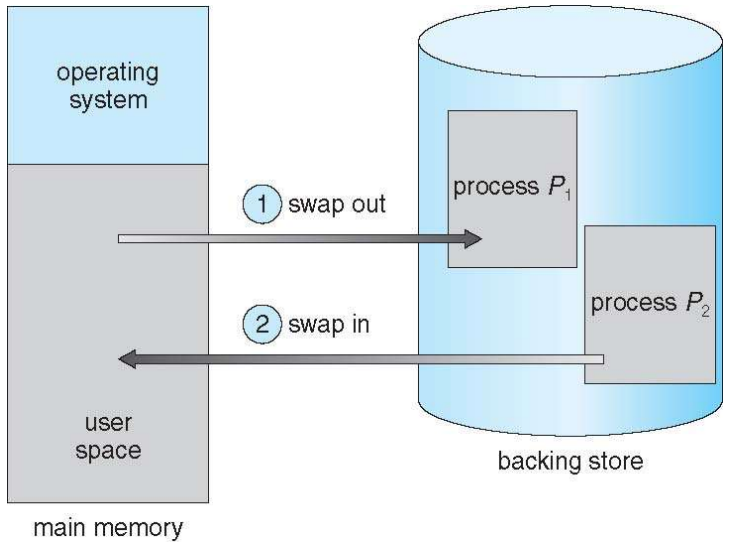


Structure

- In reality, modern operating systems will employ a mixture of these structures

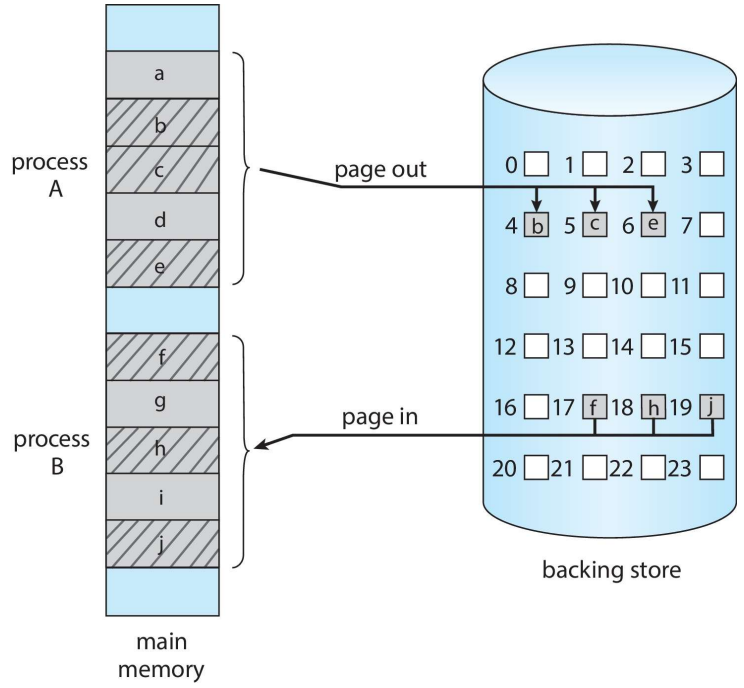
Swapping

- A process can be temporarily **swapped** to a backing store (usually disk) and then brought back in
- Total memory space for processes can exceed the physical memory (oversubscribed)



Swapping

- The backing store needs to be reasonably fast
- Swapping entire processes is generally no longer used
- Modern operating systems may still swap pages
- Swapping pages due to low memory is an "emergency" situation and should be avoided
  - Terminate unnecessary processes
  - Get more memory




1878
Western
UNIVERSITY · CANADA