Memory Management

Main memory

Virtual memory

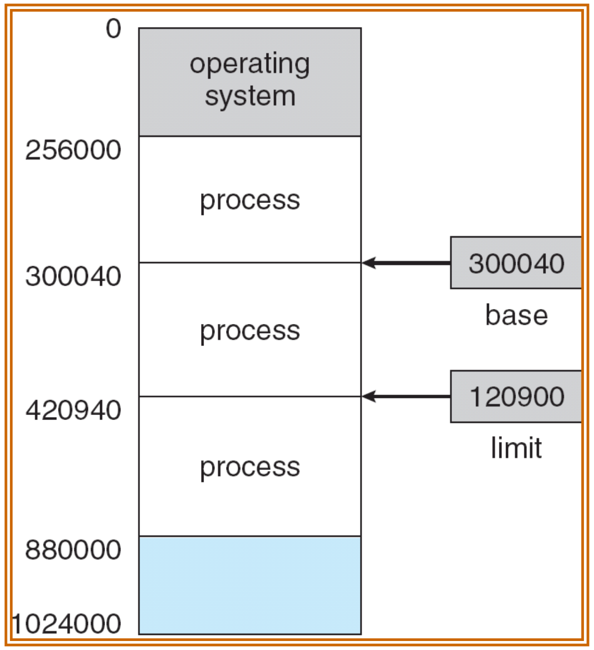
Main memory

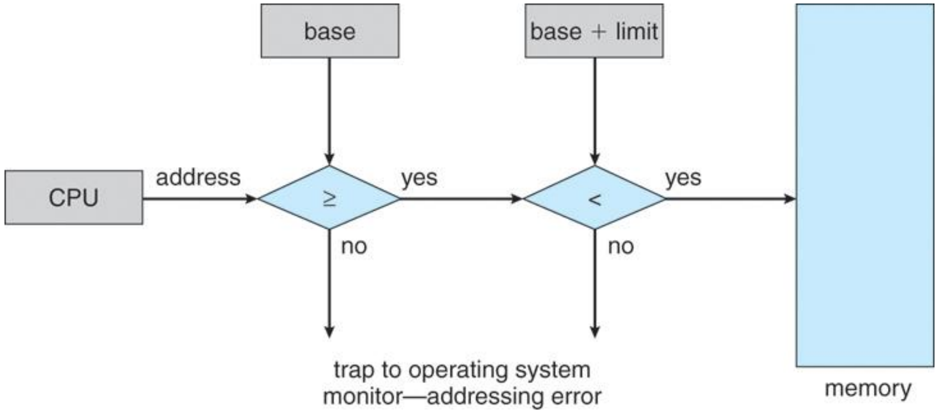
Background (1)

- Processes need to share memory
- Instruction execution cycle leads to a stream of memory addresses
- Basic hardware
  - CPU can only access data in registers and main memory
    - Accessing registers is fast
    - Accessing main memory takes a number of processor cycles
    - Cache: a fast memory between registers and main memory
  - Memory protection is needed for correct operation

Background (2)

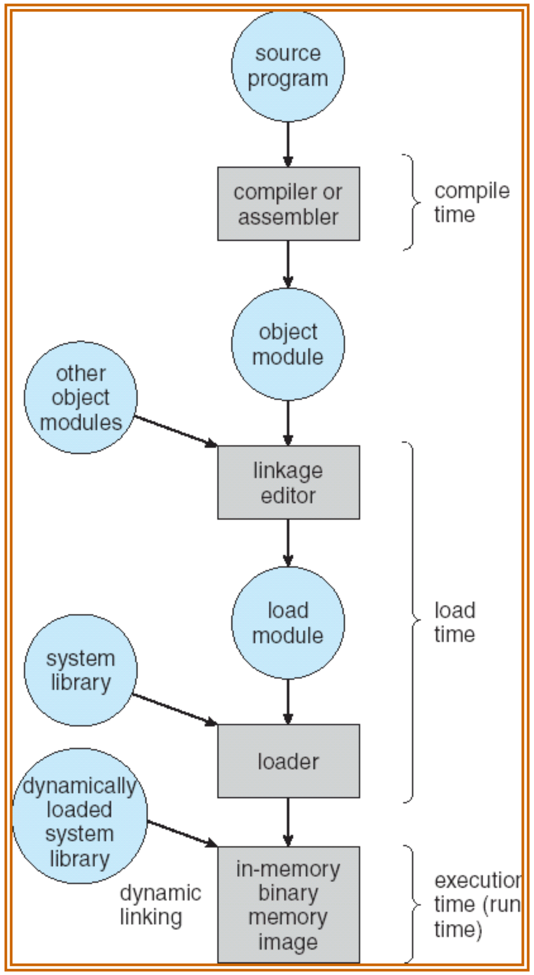
- Privileged instructions to load the base and limit registers





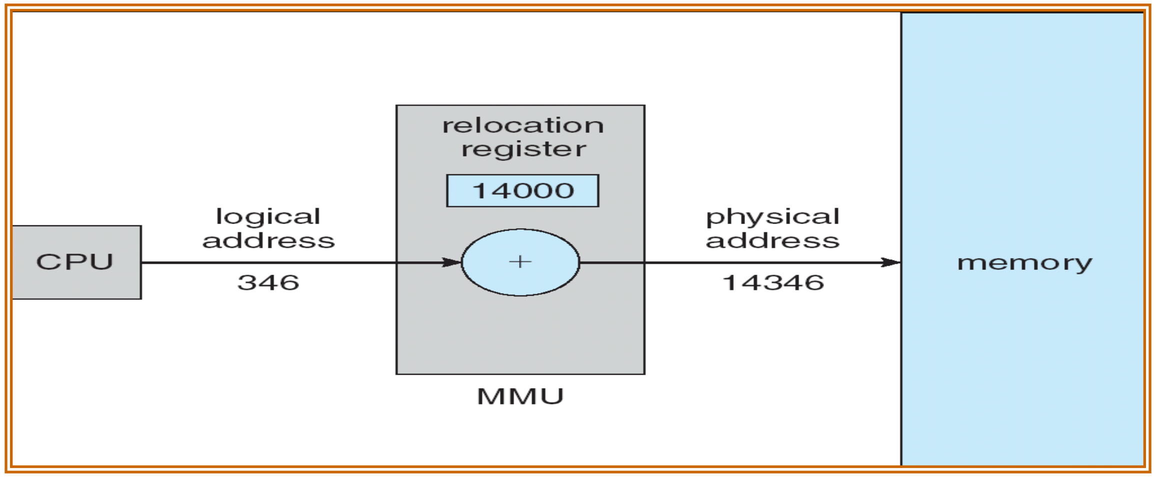
Background (3)

- Address binding
  - Compile time
    - Absolute code
    - Example: MS-DOS .COM programs
  - Load time
    - Relocatable code
  - Execution time
    - Most OS use this



Background (4)

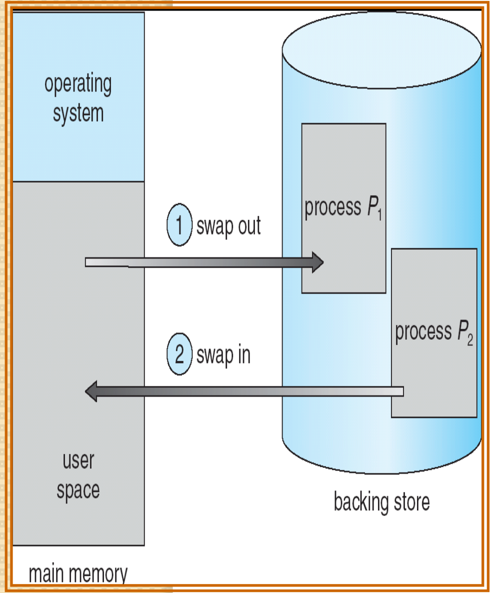
- Logical vs. physical address space
  - Logical address: CPU generated
  - Physical address: Memory unit (memory address register)
  - In compile and load time binding logical address = physical address
  - In execution time binding logical address ≠ physical – virtual address
    - Memory management unit (MMU): mapping from virtual to physical addresses
    - Programs only see virtual addresses



Background (5)

- Dynamic loading
  - Aim: obtain better memory space utilisation
  - Routines are not loaded until they are called
    - Routines kept in disk in relocatable load format
  - + unused routines are never loaded
  - Not the responsibility of the OS
- Dynamic linking and shared libraries
  - Similar to dynamic loading
    - Some OS only support static linking
  - Often used for system libraries – shared libraries
    - Include a stub in the image for each library routine reference
    - Can be extended to library updates (e.g. patches) – version information is included to ensure compatibility
  - Because of memory protection dynamic linking needs OS support

Swapping (1)



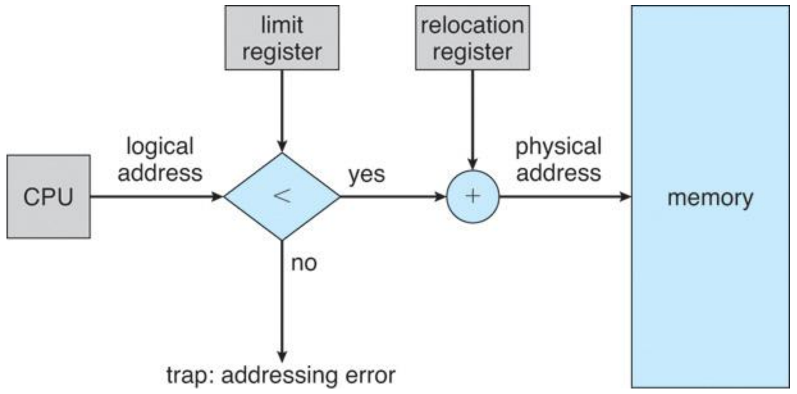
- Processes can be temporarily swapped out of memory to a backing store
  - roll out, roll in
- Type of binding determines memory location to be used
- Backing store is commonly a fast disk
  - Chunk of disk separate from the file system

Swapping (2)

- Processes in main memory or backing store are considered ready for execution
  - Swapping triggered by processor allocation – high context switch time
  - Main part of the swap time is transfer time – amount of memory to be swapped
  - It is essential for the OS to the exact amount of memory used – request and release memory system calls
- Care is needed for processes waiting on I/O – I/O buffers
  - Never swap or use OS I/O buffers
- Standard swapping is not very common, but modified versions are
  - UNIX – start when above a memory usage threshold
  - MS Windows 3.1 – user controlled swapping

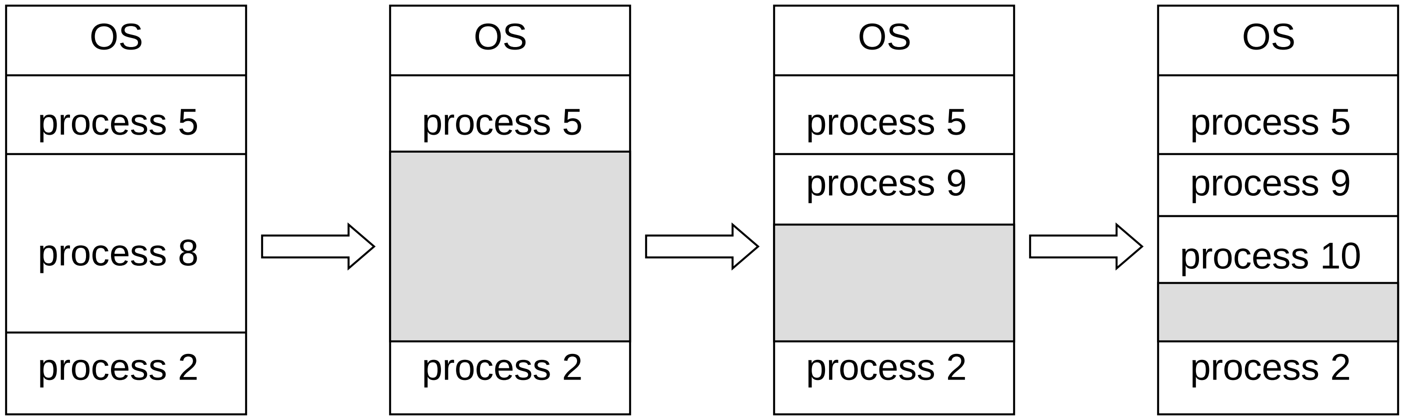
Contiguous Memory Allocation (1)

- Two memory partitions
  - Resident OS
    - Low or high memory ? – interrupt vector
  - User processes
    - Each process in a contiguous section of memory
- Memory mapping & protection
  - MMU maps logical addresses dynamically
  - Relocation and limit registers
    - Loading in context switch
  - Change dynamically OS size
    - I/O buffers & transient code



Contiguous Memory Allocation (2)

- Partitions: fixed-size chunks of memory
  - One partition for each process
- Variable partition scheme
  - Hole: a contiguous block of available memory
  - Dynamic storage allocation problem
    - First fit, Best fit, Worst fit
    - First and best better than worst fit
    - First is faster

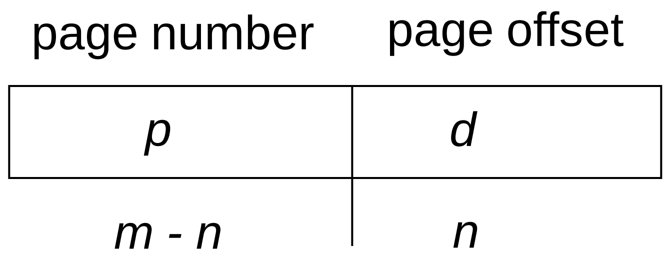


Contiguous Memory Allocation (3)

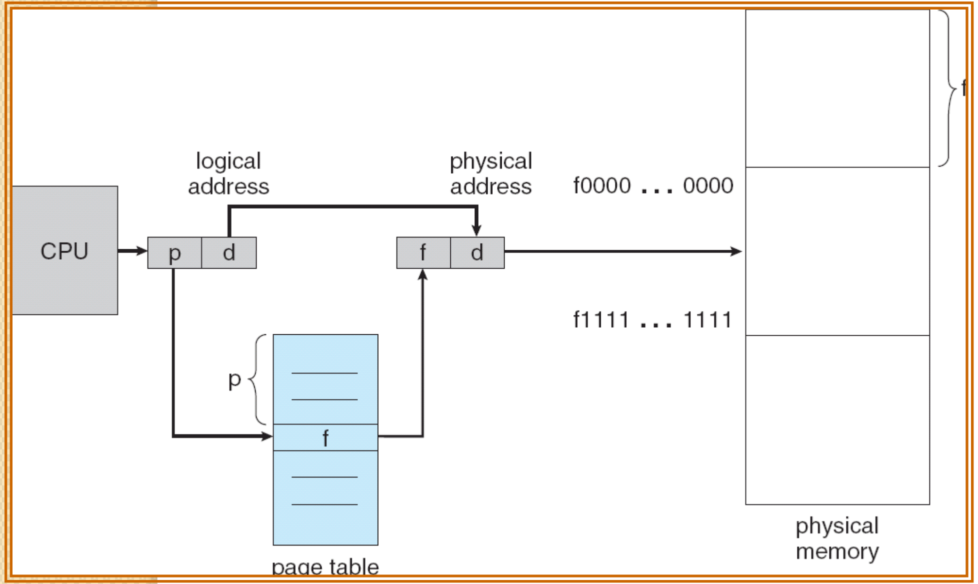
- First and Best fit suffer from external fragmentation
  - Enough total space to satisfy a request, but not contiguous
  - Where is the leftover piece?
  - 50 percent rule: if N allocated blocks, another 0.5N blocks lost to fragmentation
- Internal memory fragmentation (within block)
  - Fixed size blocks to avoid overheads of tracking
- Compaction – solution to external fragmentation
  - Depends on whether relocation is static or dynamic
  - Cost?

Paging (1)

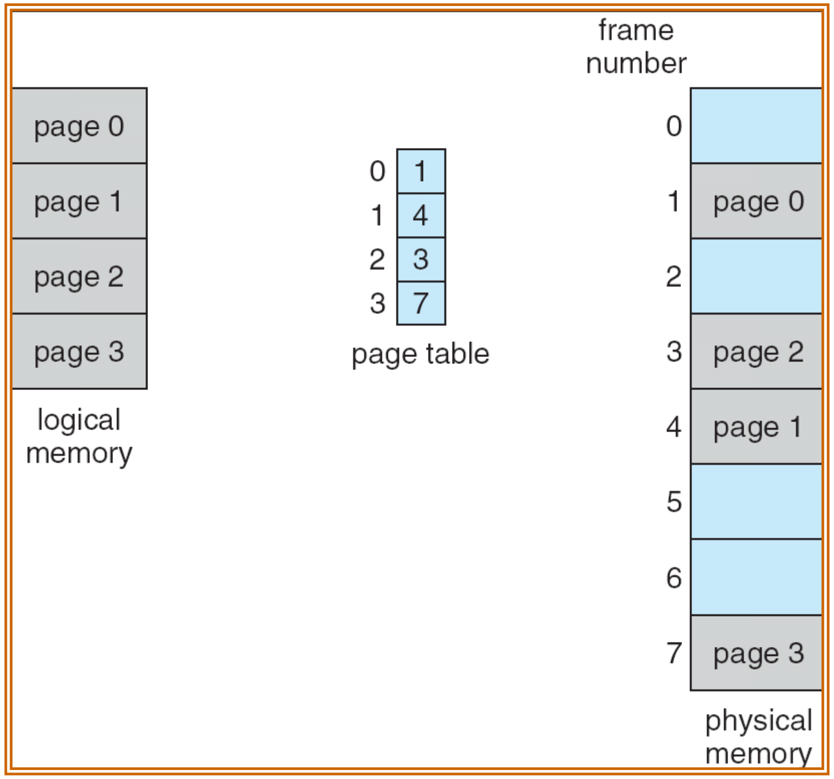
- Non-contiguous memory allocation
- Problem: how to fit variable size memory chunks onto the backing store?
  - Backing store fragmentation problem with compaction not an option
- Frames: fixed size physical memory blocks
- Pages: fixed size logical memory blocks
- Physical and logical memory and backing store blocks of same size
  - Power of 2 (512 bytes – 16Mb per page)
  - For given logical address space 2m and page size 2n



Paging (2)

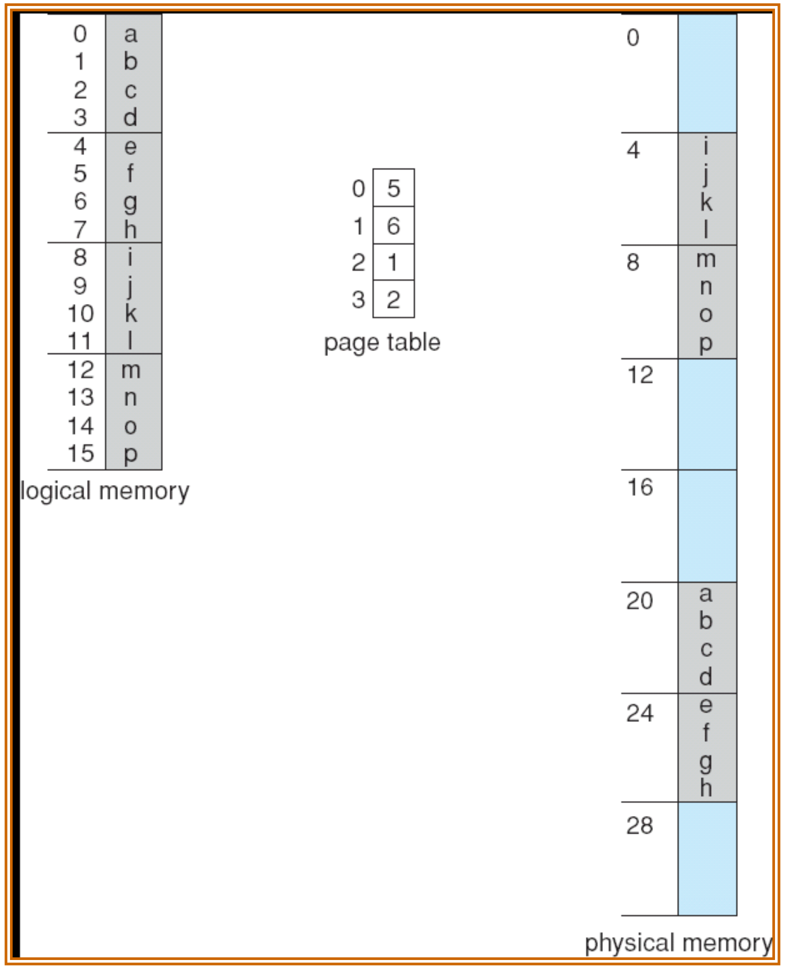


CPU
logical address
p
d
physical address
f
d
f0000 . . . 0000
f1111 . . . 1111
p
f
page table
physical memory
f



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

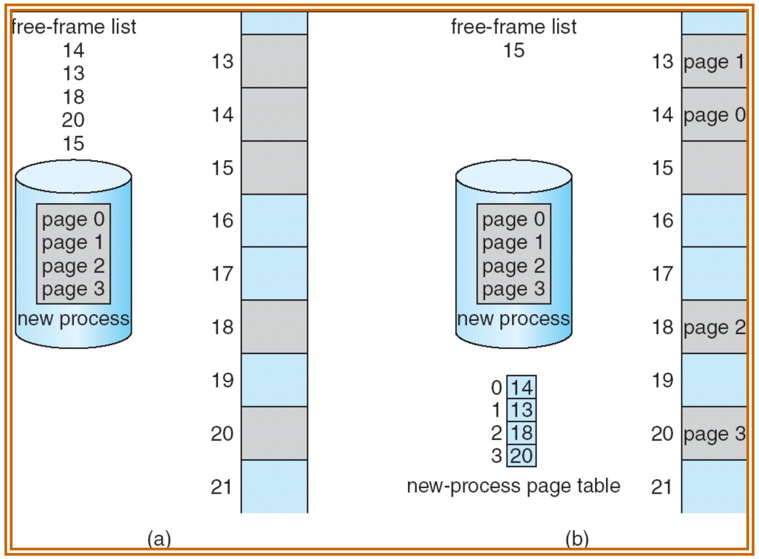
Paging (3)



32-byte memory and 4-byte pages

Paging (4)

- Paging is a form of dynamic relocation
- Paging does not suffer from external fragmentation, but does suffer from internal one
  - N+1 frames for the extra byte
  - Half a page on average
- What size of pages?
  - Fragmentation – small
  - Management overhead & efficiency – large
  - Variable on-the-fly page size support
- Separate user and actual view of memory
  - Address translation hardware
- Frame table
- OS maintain a process page table

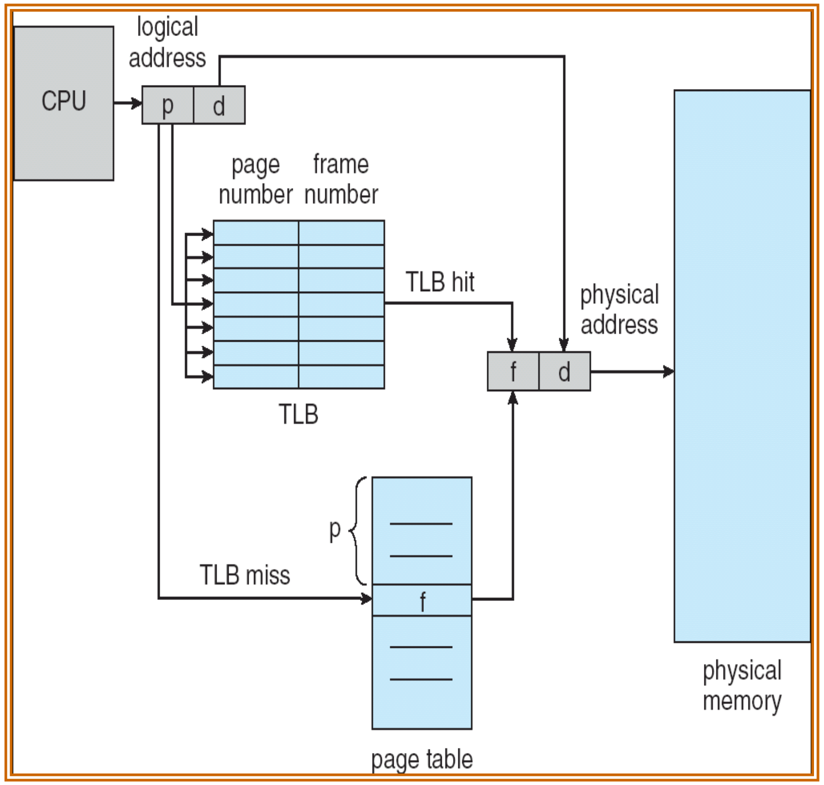


Before allocation After allocation

Paging (5)

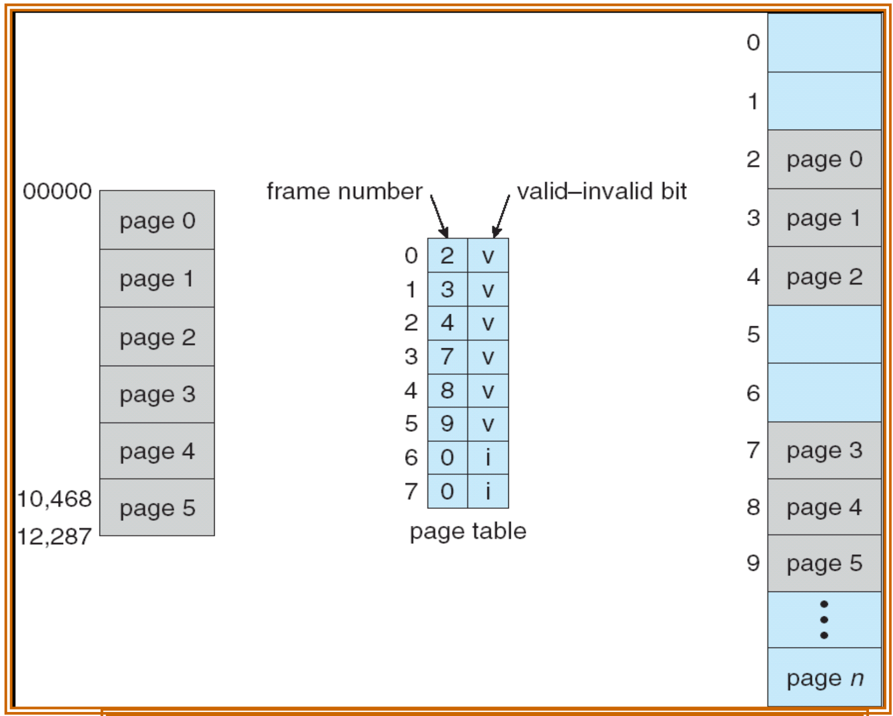
- Page table implementation
  - Set of dedicated registers
    - Efficiency is a major consideration
    - Dispatcher load page table with privileged instructions
  - In main memory with a page table base register (PTBR)
    - More efficient context switching, but less efficient memory access (2 accesses per time)
  - Translation look-aside buffer (TLB)
    - Key, value – entries
    - Fast access, but expensive hardware
    - Small size
    - TLB miss (replacement), wired down entries

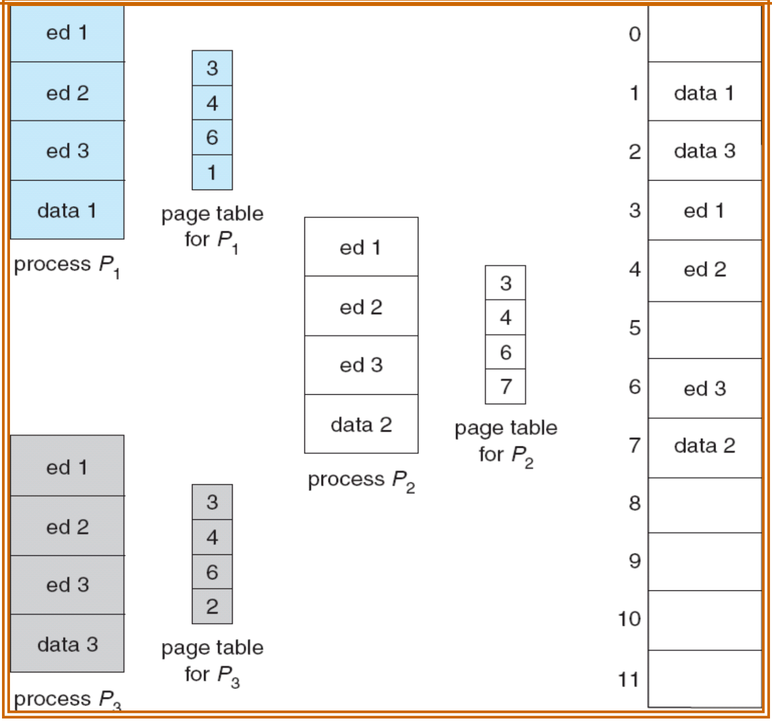
- Address space identifiers (ASIDs) to avoid flushing
- Hit ratio & effective-access time



Paging (6)

- Protection
  - Protection bit (read-write, read-only) for each frame kept in page table
  - Valid/invalid bit
  - Page table length register (PTLR)
- Shared pages
  - Reentrant code: non-self-modifying code

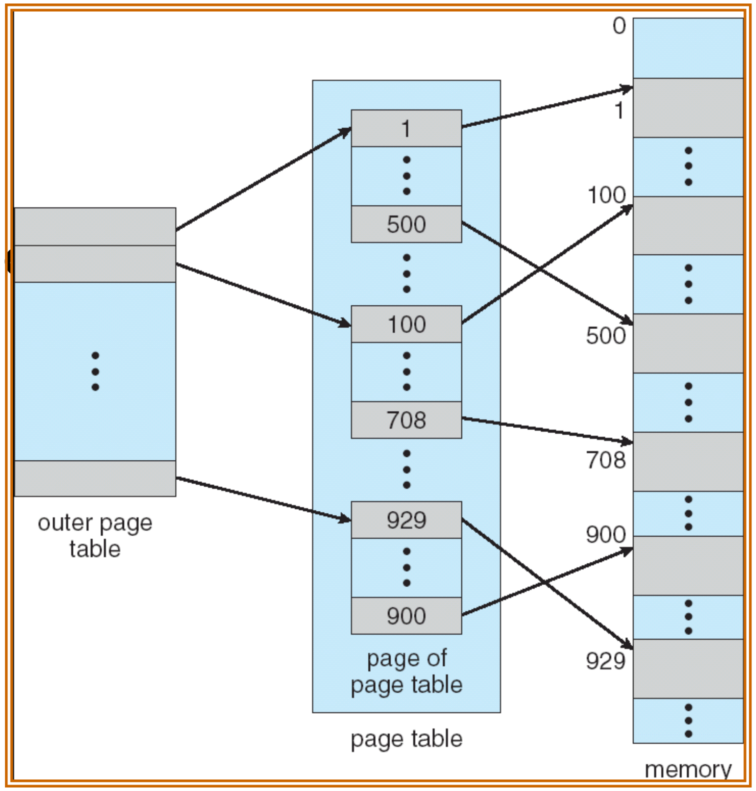


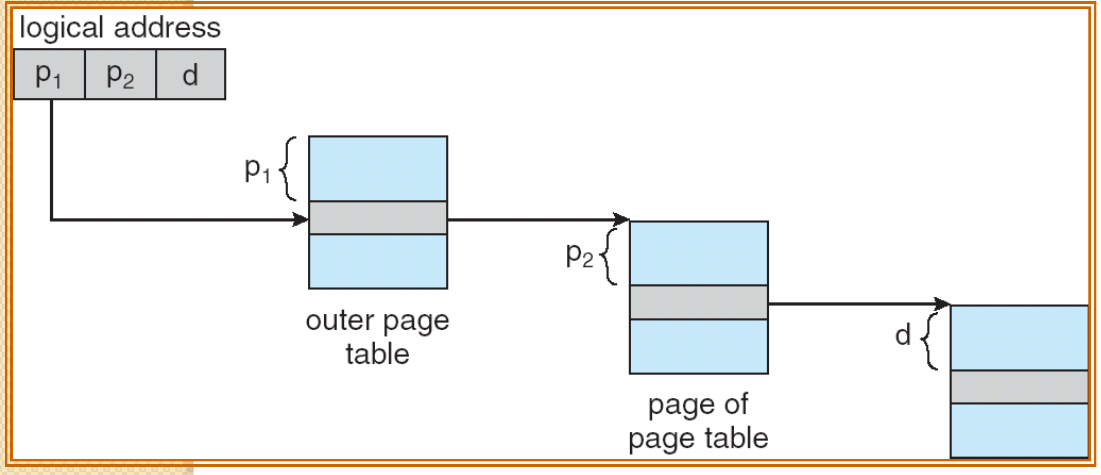


Structure of the Page Table (1)

- Hierarchical paging
  - Two-level page table
  - Forward-mapped page tabl

page number		page offset
p_i	p_2	d
12	10	10





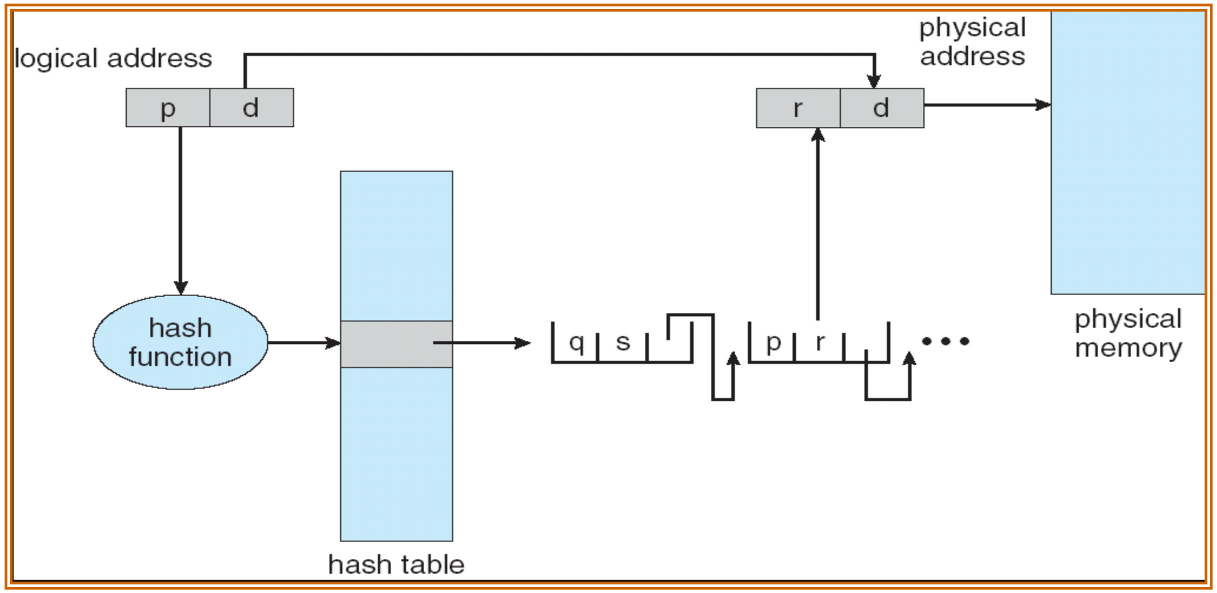
Structure of the Page Table (2)

outer page	inner page	offset
p_1	p_2	d
42	10	12

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

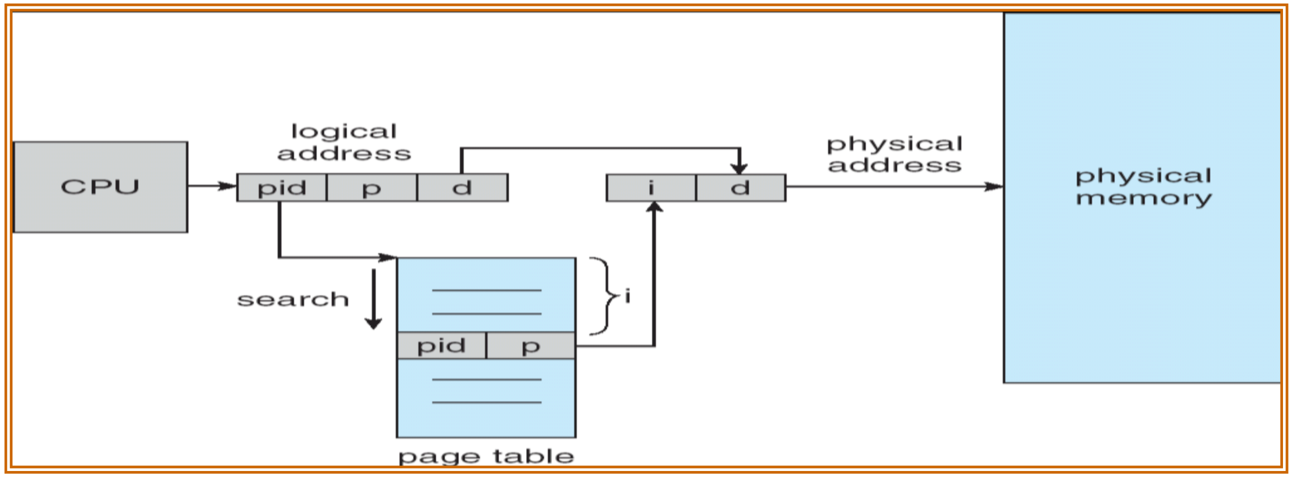
Structure of the Page Table (3)

- Hashed page table
  - Address space > 32 bits
  - Virtual page number, value of mapped page frame, pointer to next element in the linked list
  - Clustered page tables
    - Useful for sparse address spaces



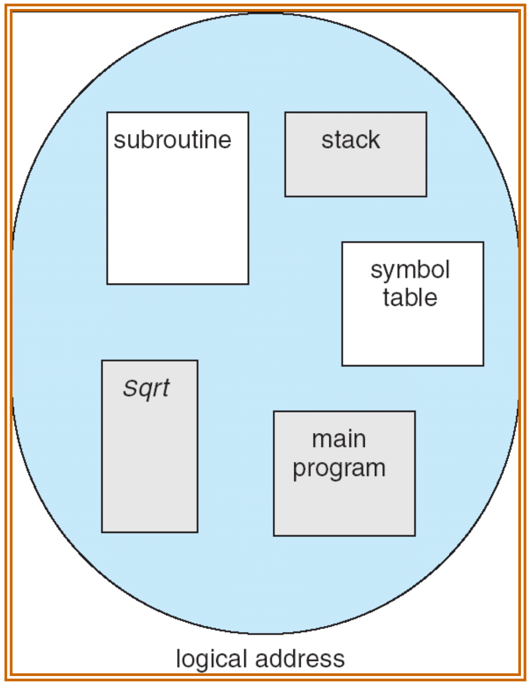
Structure of the Page Table (4)

- Inverted page tables
  - One entry for each real page of memory
  - Entry: virtual address of the page stored in that real memory location, with information about the process owning it
  - Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
  - Use hash table to limit the search to one — or at most a few — page-table entries

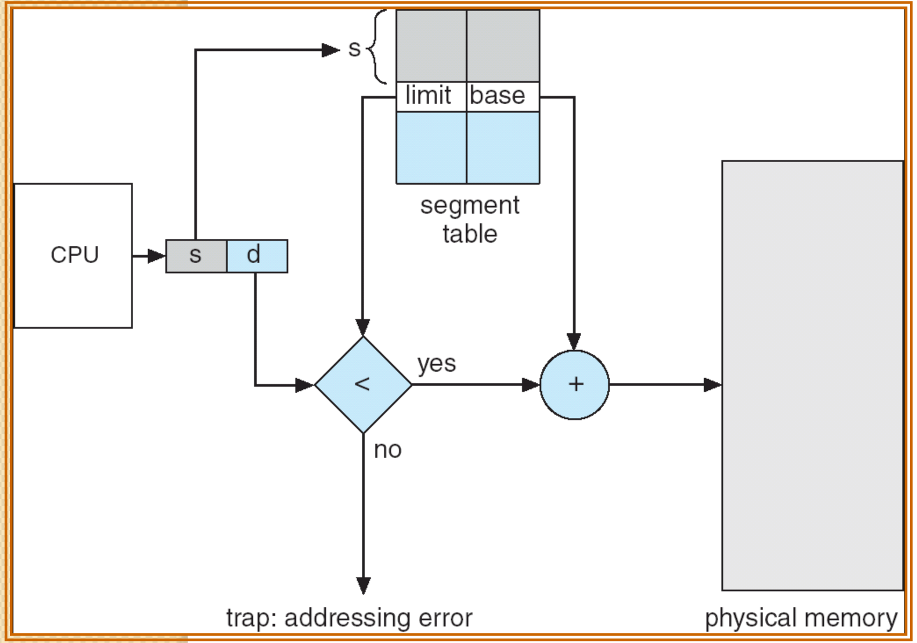


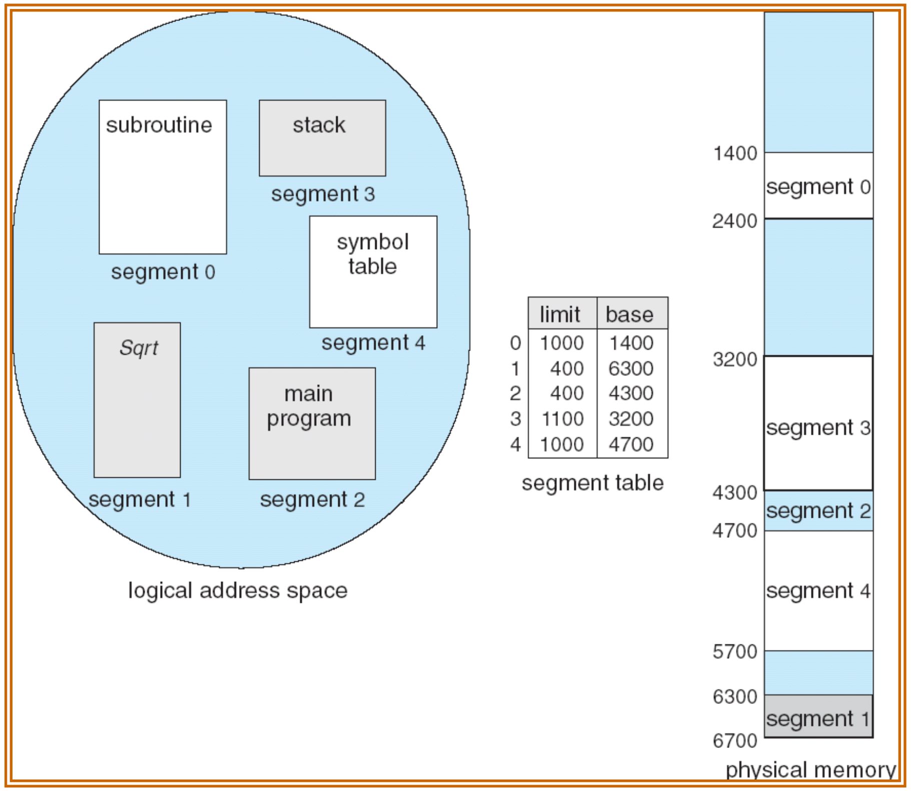
Segmentation (1)

- Memory-management scheme that supports user view of memory
  - Programs organised in segments of variable lengths
  - Address: segment no, offset
  - Compiler creates segments
- Segment table: mapping two dimensional user defined addresses into one dimensional physical addresses
  - Segment base and segment limit
  - Segment-table base register (STBR) points to the segment table's location in memory
  - Segment-table length register (STLR) indicates number of segments used by a program
- Protection and sharing at segment level



Segmentation (2)





	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

For contemplation (1)

- Explain the difference between internal and external fragmentation.
- Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?
- Most systems allow programs to allocate more memory to its address space during execution. Data allocated in the heap segments of programs are an example of such allocated memory. What is required to support dynamic memory allocation in the following schemes:
    - contiguous-memory allocation
    - pure segmentation
    - pure paging
- Compare the main memory organization schemes of contiguous-memory allocation, pure segmentation, and pure paging with respect to the following issues:
    - external fragmentation
    - internal fragmentation
    - ability to share code across processes

For contemplation (2)

- Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual addresses to physical addresses.
- Consider a paging system with the page table stored in memory.
  - If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?
  - If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time if the entry is there.)
- Why are segmentation and paging sometimes combined into one scheme?
- Compare the segmented paging scheme with the hashed page tables scheme for handling large address spaces. Under what circumstances is one scheme preferable to the other?
- Consider a system in which a program can be separated into two parts: code and data. The CPU knows whether it wants an instruction (instruction fetch) or data (data fetch or store). Therefore, two base–limit register pairs are provided: one for instructions and one for data. The instruction base–limit register pair is automatically read-only, so programs can be shared among different users. Discuss the advantages and disadvantages of this scheme.

For contemplation (3)

- Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

- What are the physical addresses for the following logical addresses?
    - 0,430
    - 1,10
    - 2,500
    - 3,400
    - 4,112

For contemplation (4)

- Assuming a 1KB page size, what are the page numbers and offsets of the following address references (provided as decimal numbers)?
  - 2375
  - 19366
  - 30000
  - 256
  - 16385
- Consider a logical space of 32 pages with 1,024 words per page, mapped onto a physical memory of 16 frames.
  - How many bits are required in the logical address?
  - How many bits are required in the physical address?
- Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512MB of physical memory. How many entries are there in each of the following?
  - A conventional single-level page table
  - An inverted page table