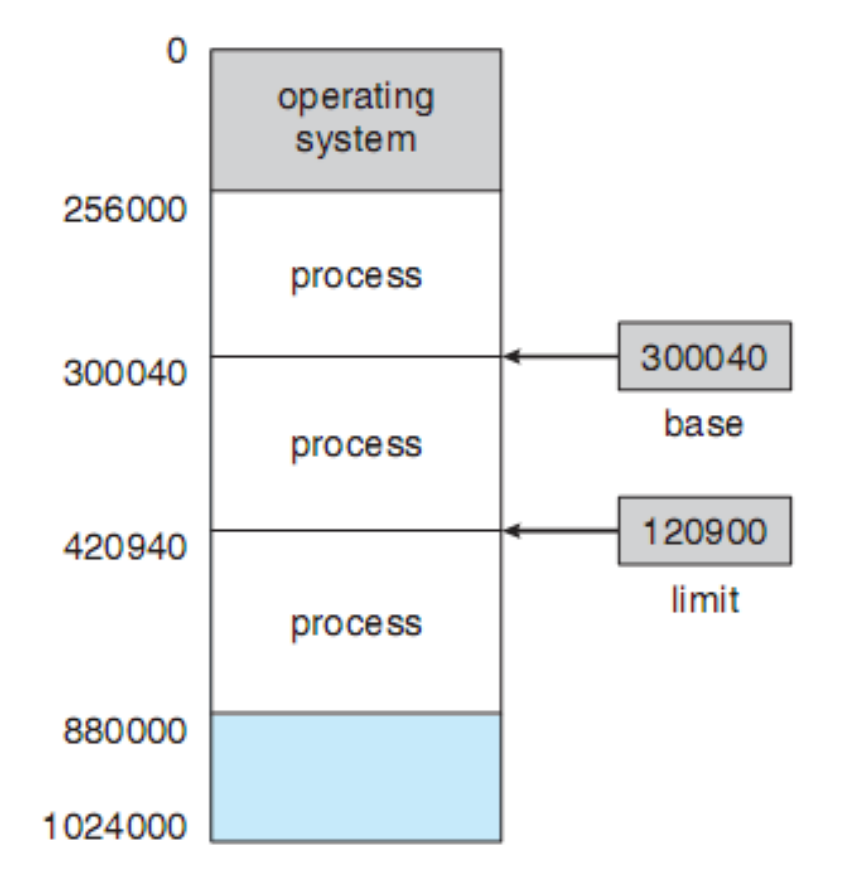
Memory Management (Main Memory)

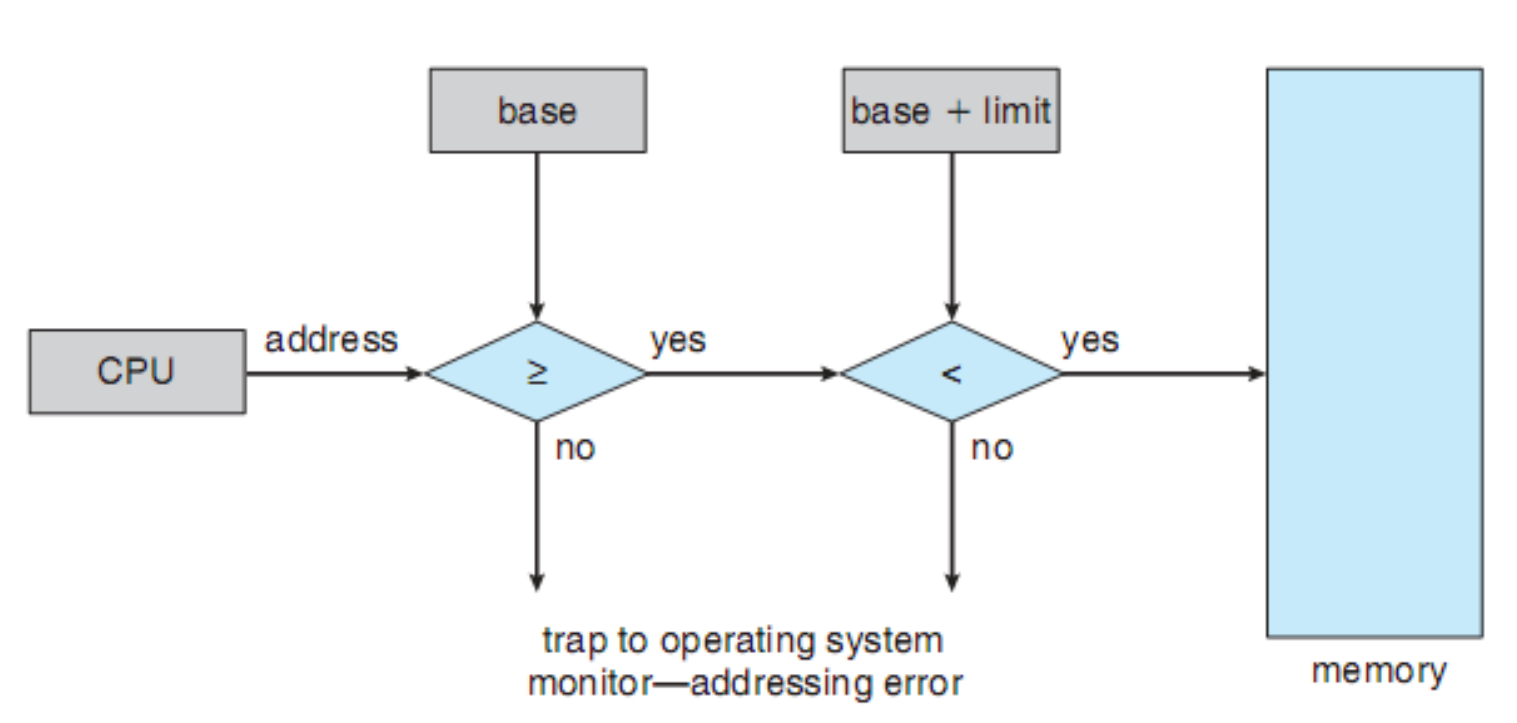
Mehdi Kargahi
School of ECE
University of Tehran
Summer 2016

Hardware Address Protection



Hardware Address Protection

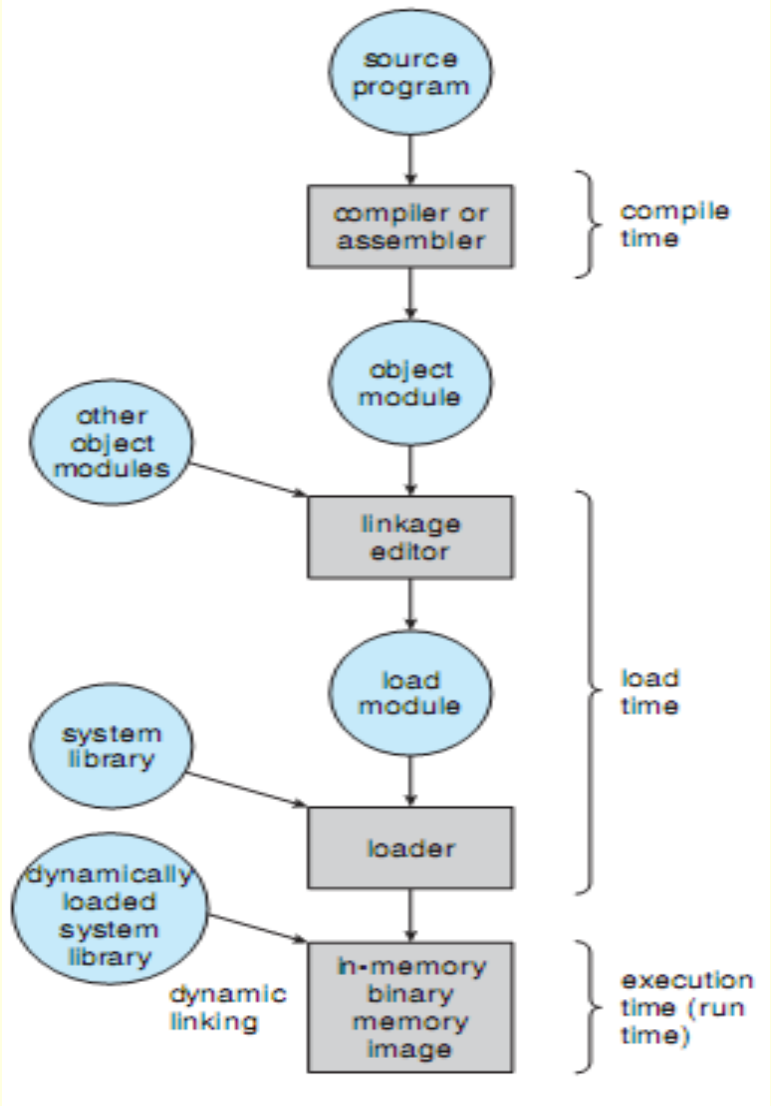
- Base and Limit Registers



Address Binding

- Binding: a mapping from one address space to another
  - Compiler time (absolute code)
  - Load time (relocatable code)
  - Execution time (process can be moved during its execution)
    - Code may partially be loaded
    - Available in most general-purpose OS

Address Binding



Logical versus Physical Address Space

- Logical address: the address generated by the CPU
- Physical address: the address seen by the memory unit (loaded into memory address register)
- Virtual address: logical address in execution time binding which is different from the respective physical address
  - Mapping is done using MMU (memory-management unit)

Dynamic Loading

- A routine is not loaded until it is called
- For better memory space utilization
- Process size is not limited to the size of physical memory
- An almost general rule: 10% main program and 90% for exception handling
- Dynamic loading does not require special support from the OS, but OS may help the programmer by providing special library routines

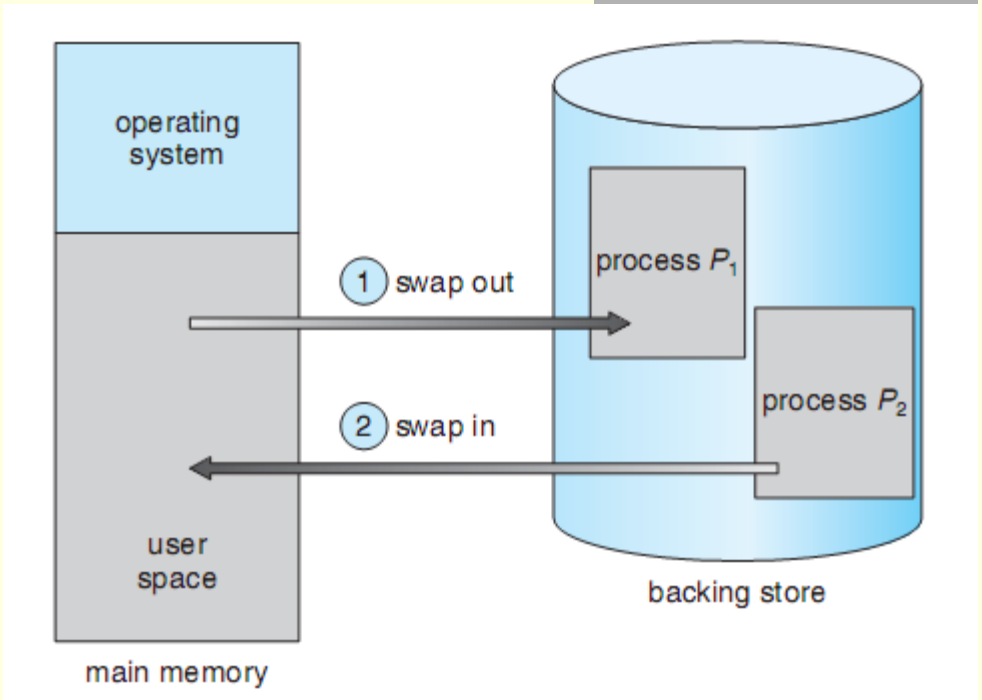
Overlay Technique

- A weak memory management technique without using the OS
- Ex.
  - An assembler
    - Pass1 (70K) & symbol table (20K)
    - Pass2 (80K) & common routines (30K)
    - 200K is needed, while 150K is available
    - ST (20K) + CR (30K) +OV (10K) + Pass1 & Pass2 (80K) = 140K

Dynamic Linking and Shared Libraries

- Static linking versus dynamic linking
- Here, linking rather than loading is postponed until execution time
- To save both disk space and memory
- Stub: a small piece of code indicating how to locate the appropriate library routine
- Only one copy of the library routine is loaded
- Library bug fixes are much more simpler
  - Shared libraries: related to different library versions for different processes→ previously compiled programs are not affected by the new versions
- Needs support from the OS for usage of a library by multiple processes

Swapping



- Ex.
  - Average latency and seek time: 8ms
  - Transfer rate to HDD: 40 MB/s
  - User process 10 MB
  - 10/40 + 0.008 = 258 ms ➔ 2*258 = 516 ms

Swapping

- What if any pending I/O
  - Solution 1: Never swap a process with pending I/O
  - Solution 2: Execute I/O operations only in the OS buffers

Swapping on Mobile Systems

- Mobile devices generally use flash memory rather than more spacious hard disks as their persistent storage.
- Why mobile operating-system designers avoid swapping:
  - Space constraints
  - The limited number of writes that flash memory can tolerate before it becomes unreliable
  - The poor throughput between main memory and flash memory in these devices.

Contiguous Memory Allocation

- OS usually resides in low memory space rather than high memory space because of the location of the interrupt vector
- *Memory manager* should be aware of current *holes* and *used pieces* of memory and do the allocation
- Fixed-size partitions
- *First-fit*, *best-fit*, *worst-fit*, and *next-fit* strategies

Fragmentation

- External fragmentation
  - Ex.: Memory size: 2560K , RR (q=1)
  - OS (0-400K)
  - P1 (600K, 10ms), P2(1000K, 5ms), P3(300K, 20ms), P4(700K, 8ms), P5 (500K, 15ms)
- Internal fragmentation
  - Ex.: 18464 bytes are free, but 18462 bytes are needed ➔ a good allocation policy may use all the free memory block to prevent some overheads

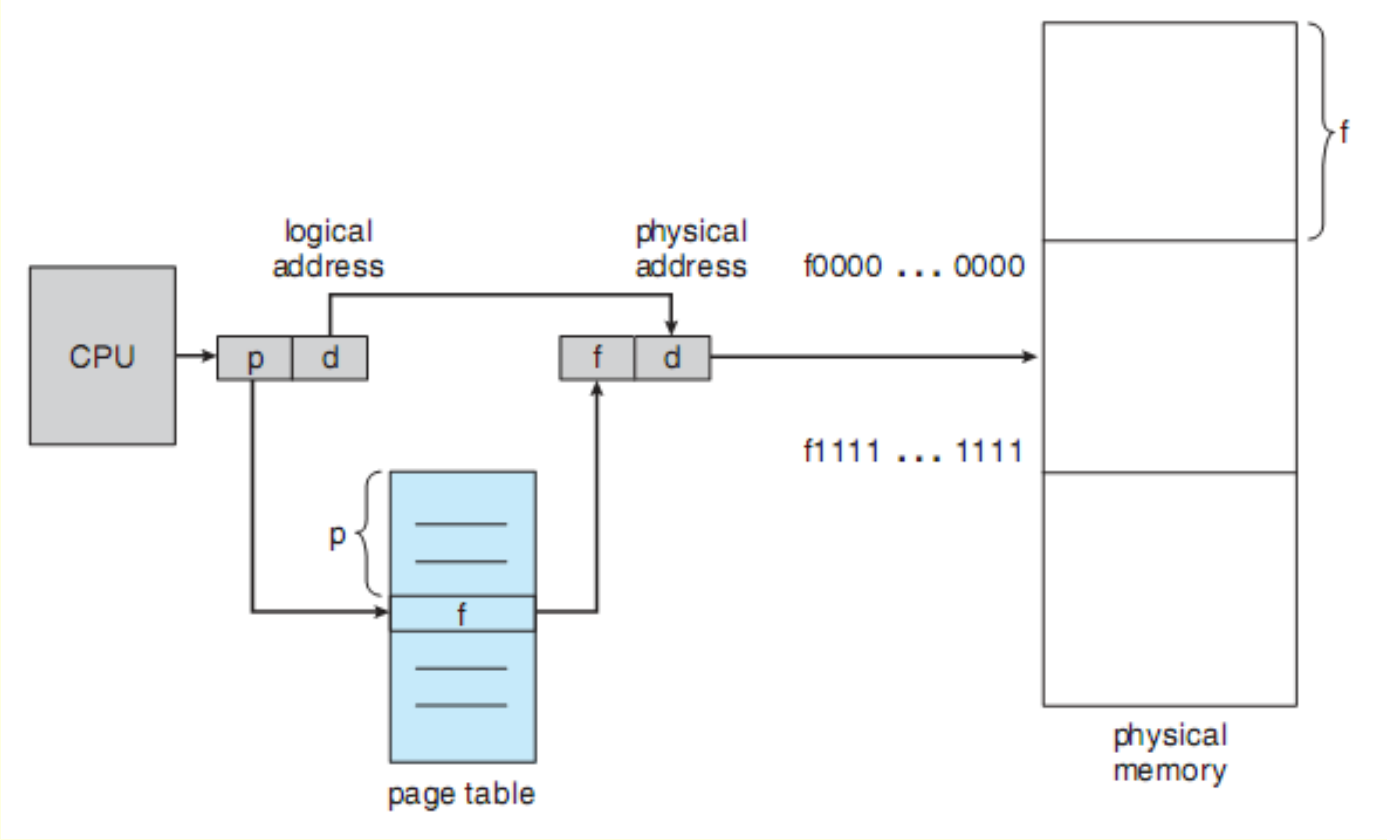
Solutions to External Fragmentation

- Compaction
  - Only applicable if relocation is dynamic and is done at execution time
  - Trying on the previous example!
- Using non-contiguous logical address space
  - Paging
  - Segmentation

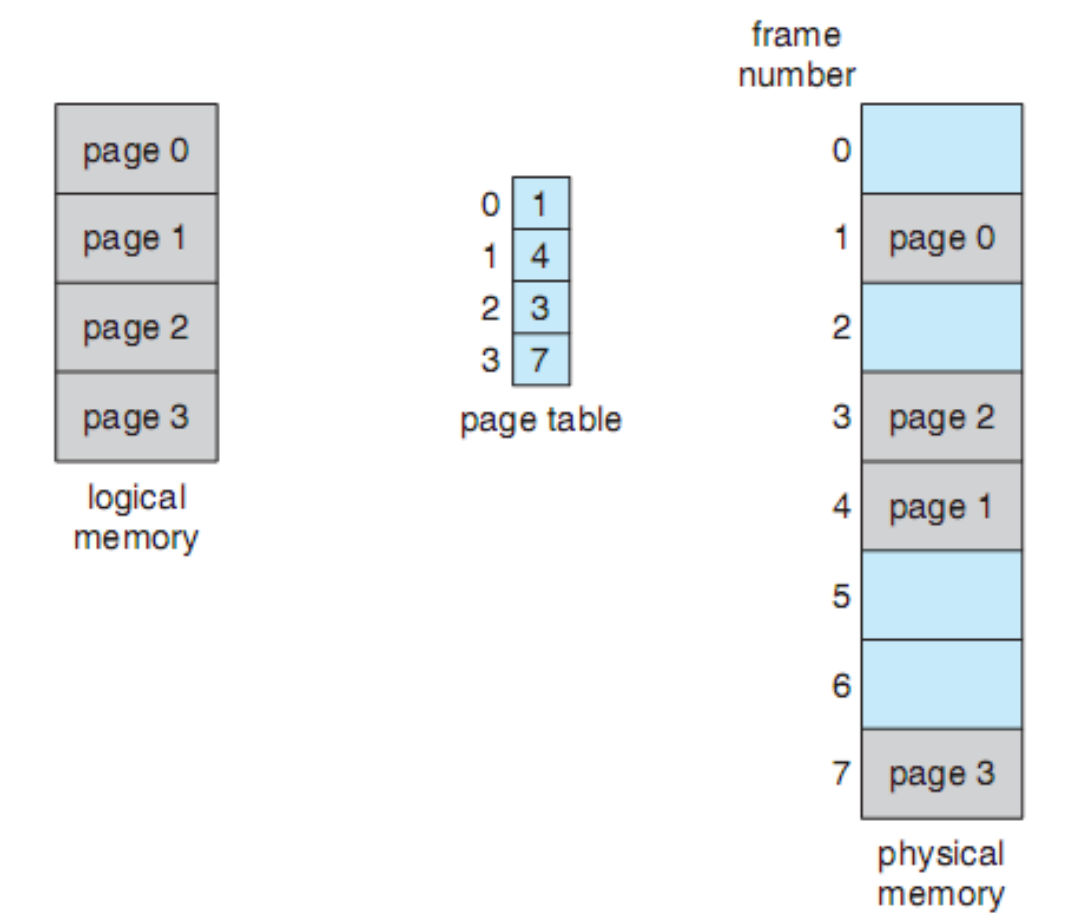
Paging

- Basic method
  - Fixed size frames and pages
- Advantages
  - Contiguous space is not required
  - Fitting memory chunks onto backing store is not problematic
  - Fitting pages onto frames is straightforward

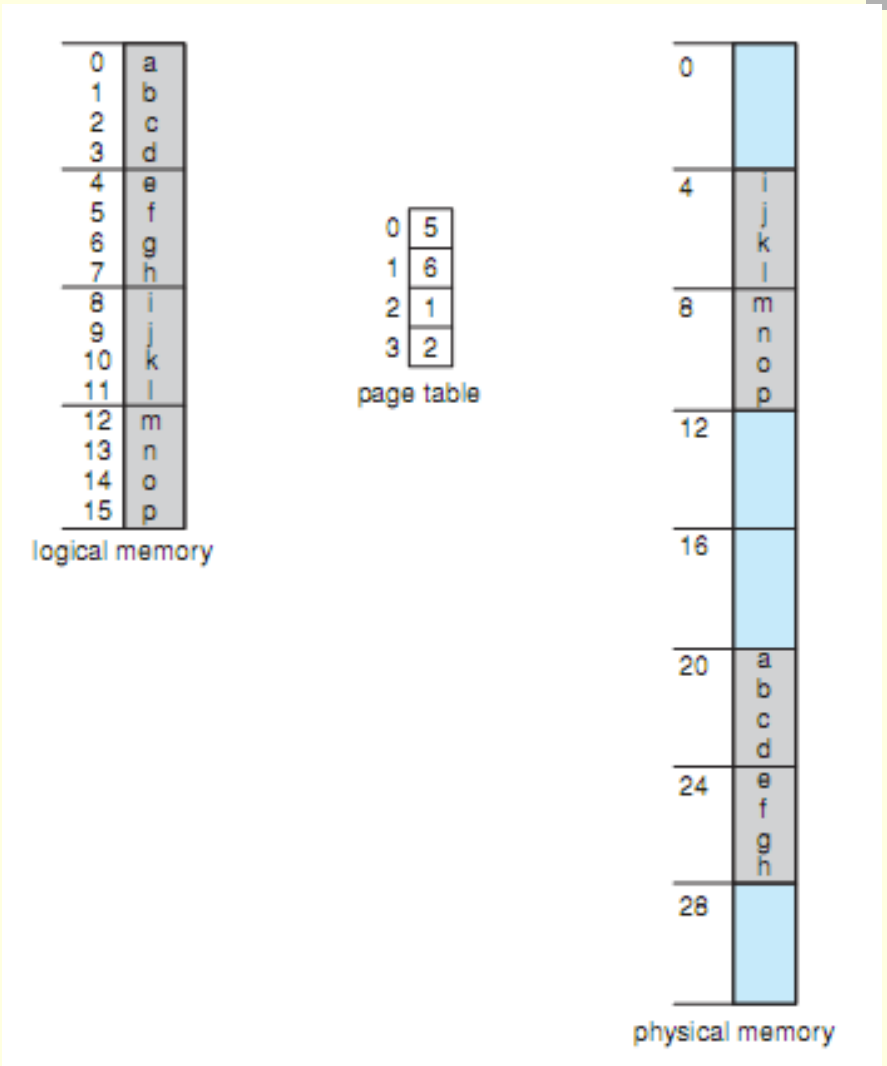
Hardware Support for Paging



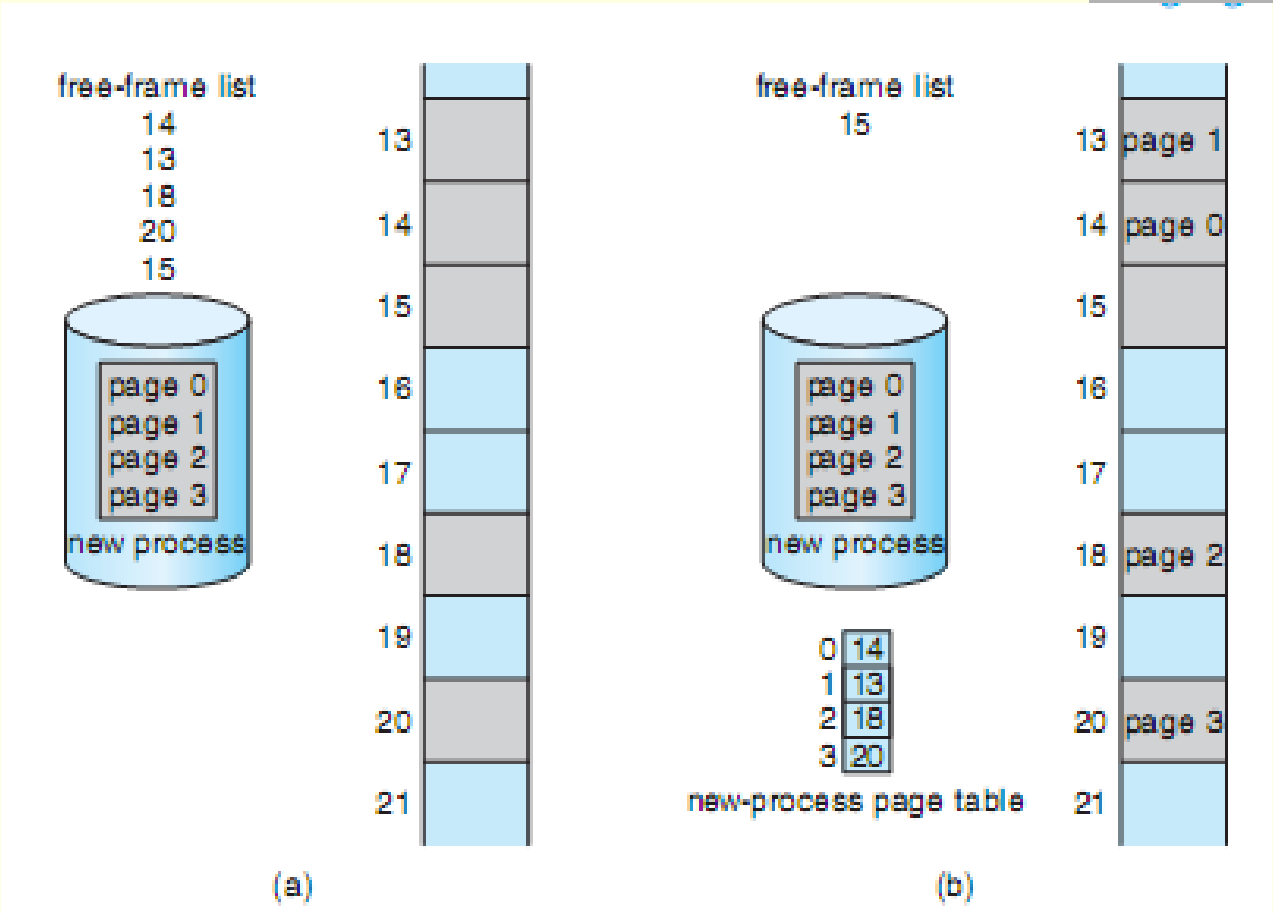
Example


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

Calculating Physical Address

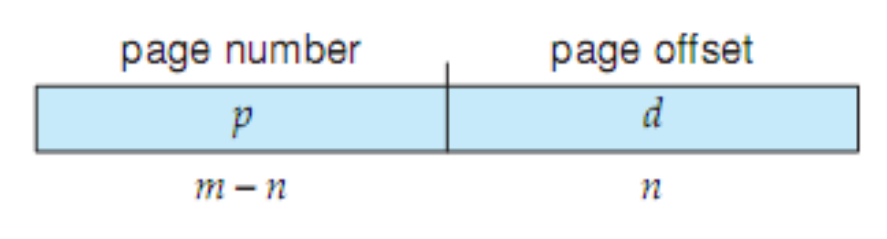


Frame Table


free-frame list
14
13
18
20
15
page 0
page 1
page 2
page 3
new process
13
14
15
16
17
18
19
20
21
(a)
free-frame list
15
page 0
page 1
page 2
page 3
new process
0 14
1 13
2 18
3 20
new-process page table
13 page 1
14 page 0
15
16
17
18 page 2
19
20 page 3
21
(b)

Logical Address

- Page size and frame size are defined by hardware
- Some operating systems support variable page sizes
- Ex.
  - Size of logical address space: 2^m
  - Page size: 2^n words



Where the Page Table is Stored?

- OS maintains a copy of the page table
  - Whenever a process makes a system call and passes a pointer to a buffer as a parameter, binding should be done properly
  - CPU dispatcher also uses this copy to define the hardware page table when a process is to be allocated the CPU

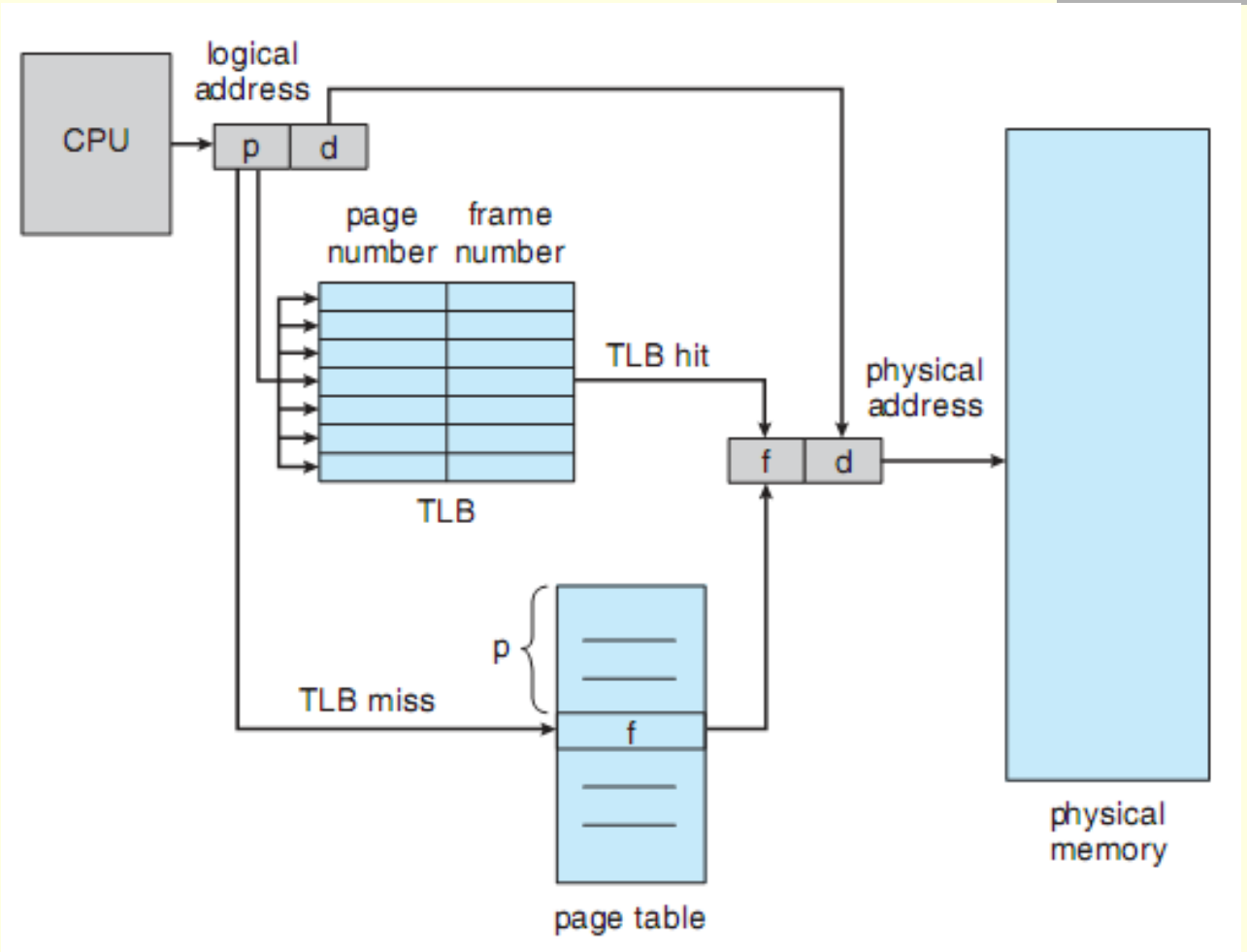
Hardware Support

- Where the page table is stored?
  - In special CPU registers
    - Modification is privileged
    - Limitation in size
    - Relatively high context-switch time
  - In memory (having a PTBR)
    - Modifying PTBR is privileged
    - Lower context-switch time
    - Memory access time is larger

Hardware Support

- Using Translation Look-aside Buffer (TLB)
  - TLB: a special small fast lookup hardware cache (associative high speed memory)
  - Number of entries in TLB are typically between 64 and 1024
  - What should be done if a TLB miss occurs?
  - TLB full ➔ Using a replacement policy
  - **Wired down** TLBs (non-removable) are used for kernel code

Paging Hardware with TLB

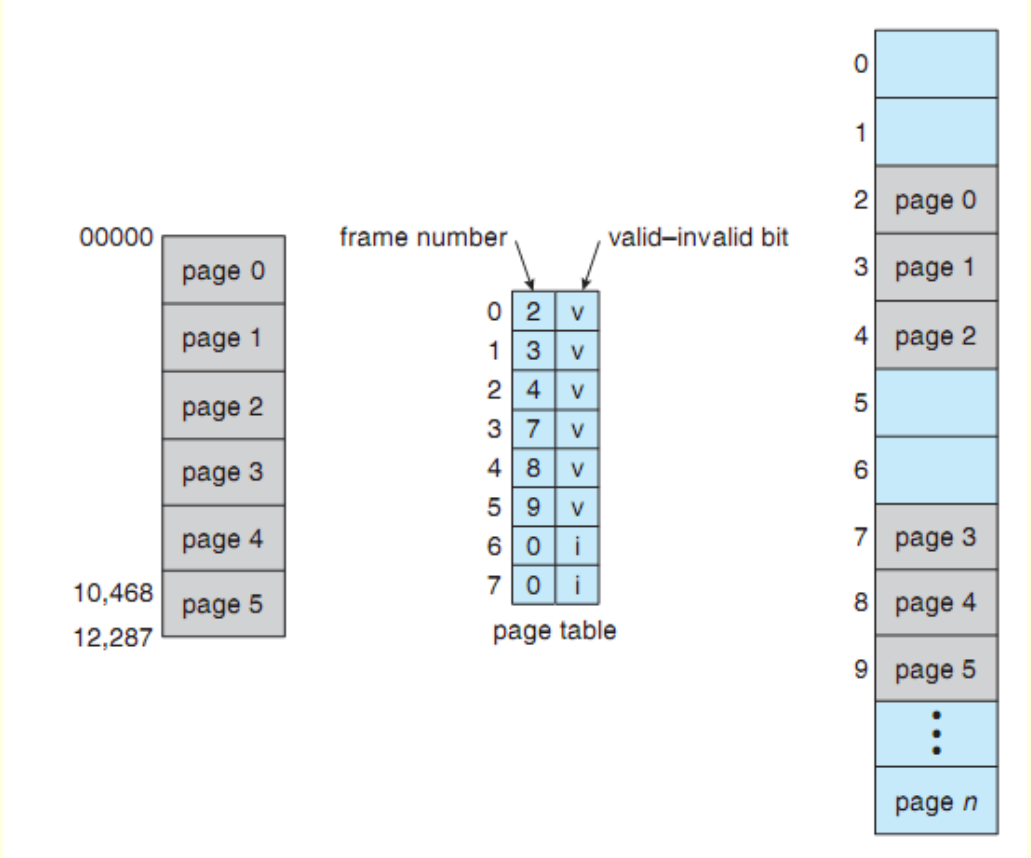


Hardware Support

- Some TLBs store address-space identifiers (ASIDs)
- ASID: uniquely identifies a process
- By checking the ASID for each virtual page number, address space protection is done. Other wise the TLB should be flushed on each CS.
- Effective memory access time
  - Ex.: Hit ratio: 80% (80386 with 1 TLB), TLB access: 20^{ns}, memory access: 100^{ns}
  - Effective access time: 80%*120+ 20%*220 = 140^{ns}
  - hr= 98% (80486 with 32 TLBs) ➔ EAT: 122^{ns}

Protection

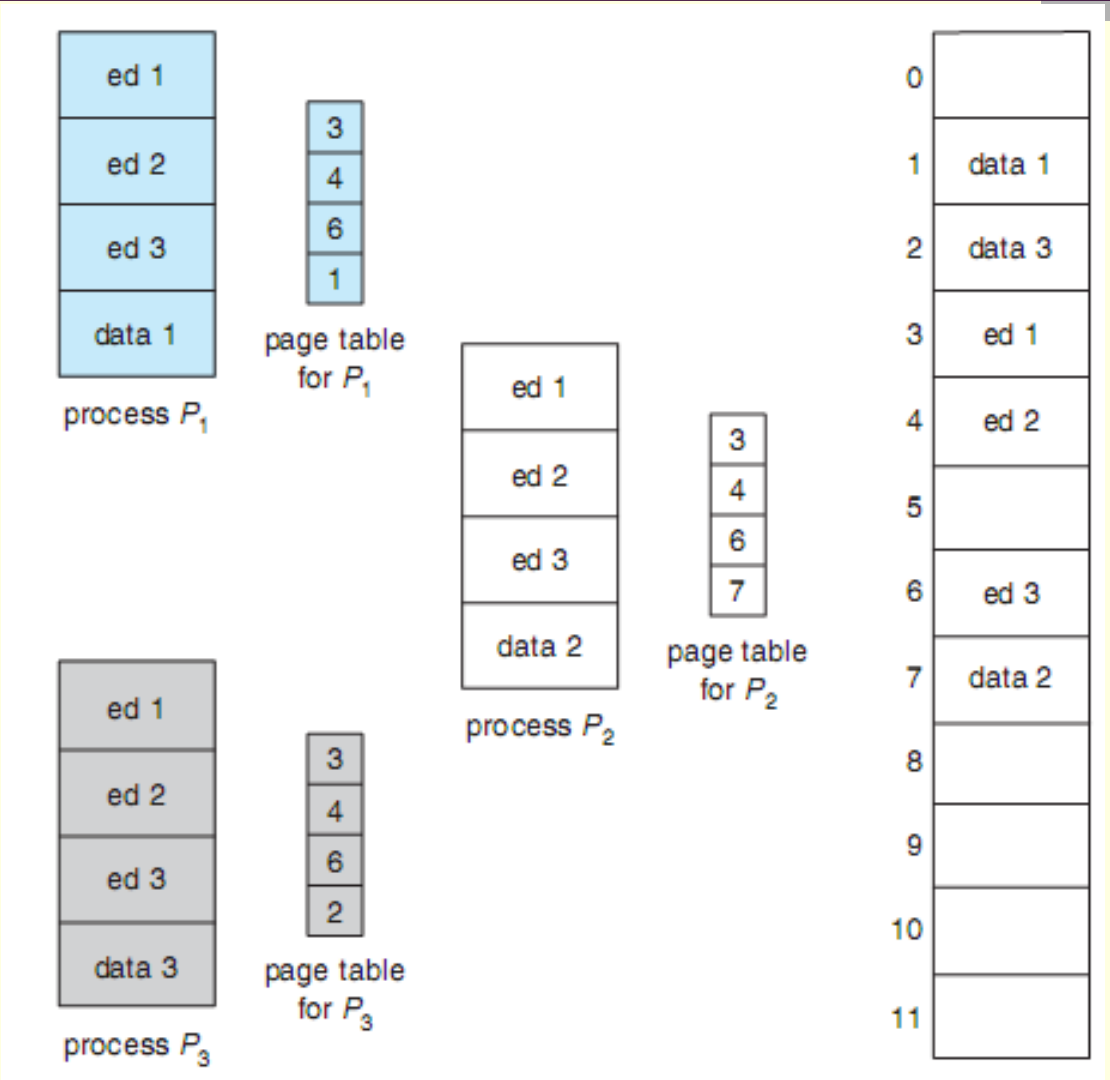
- An example of internal fragmentation!
- Extension: read-only, read-write, execute-only
- PTLR to have efficient page table size



Shared Pages

- Ex.: An editor of 150K size
  - Page size: 50K
  - User data: 50K
  - 40 users ➔ 8000K
  - Shared pages ➔ 2150K
- Only reentrant codes can be considered as shared pages

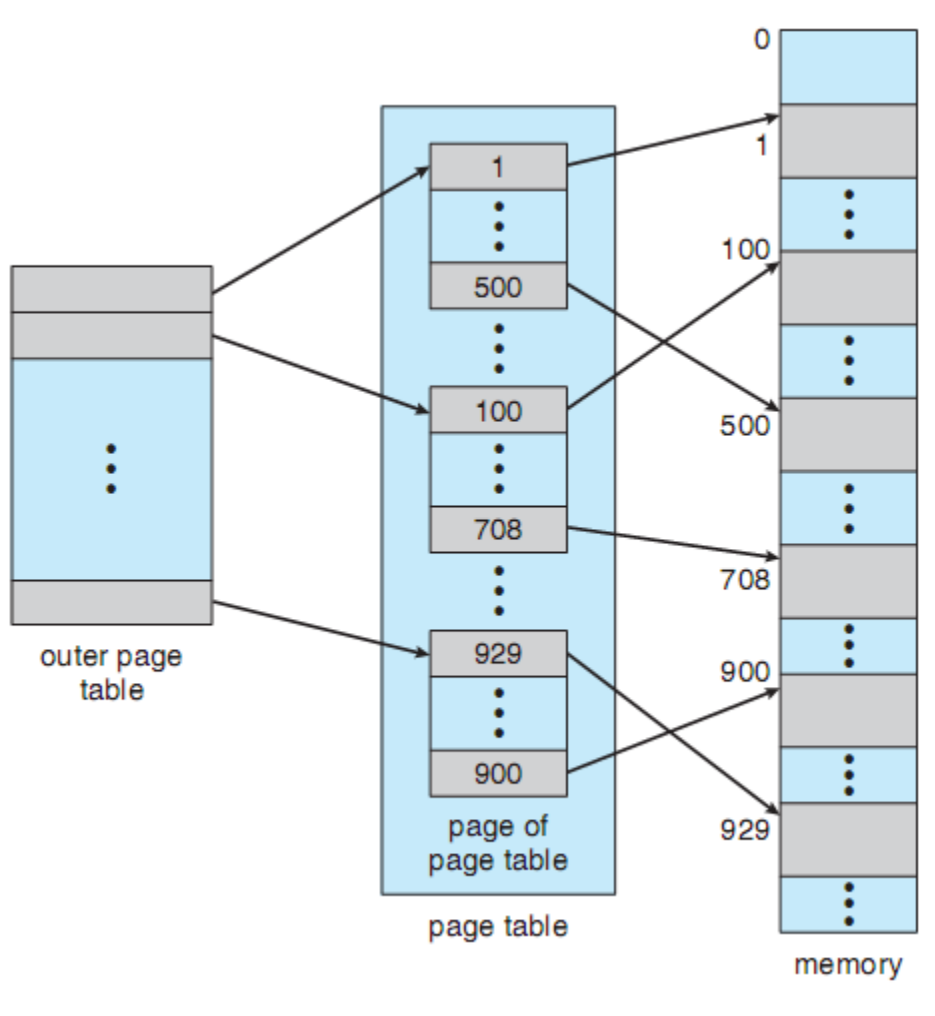
Shared Pages



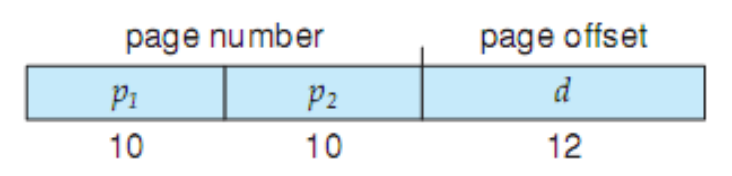
Hierarchical Paging

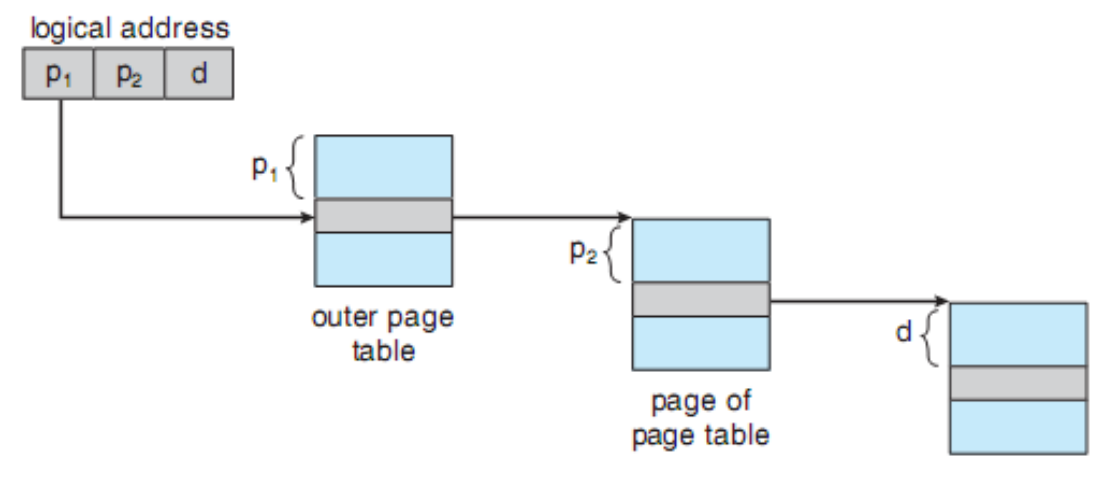
- Assume a system with 32 bit logical address space
- Each page is considered to be 4KB
- Page table size: 2^{20} entries
- Each entry is 4 bytes
- Page table size: 4MB
- Is it stored in contiguous memory?
- *One solution*: paging the page table

Hierarchical Paging



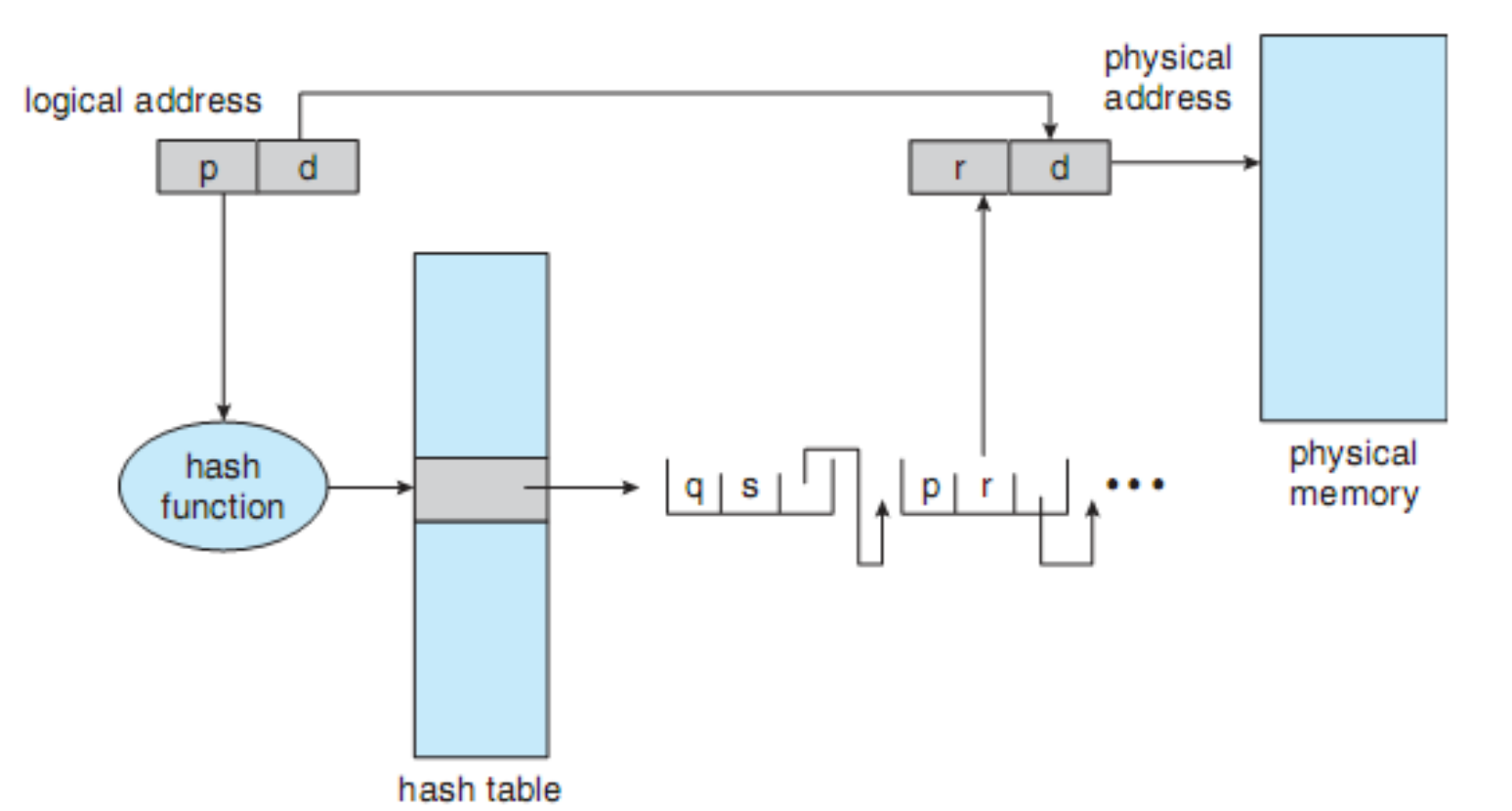
Hierarchical Paging





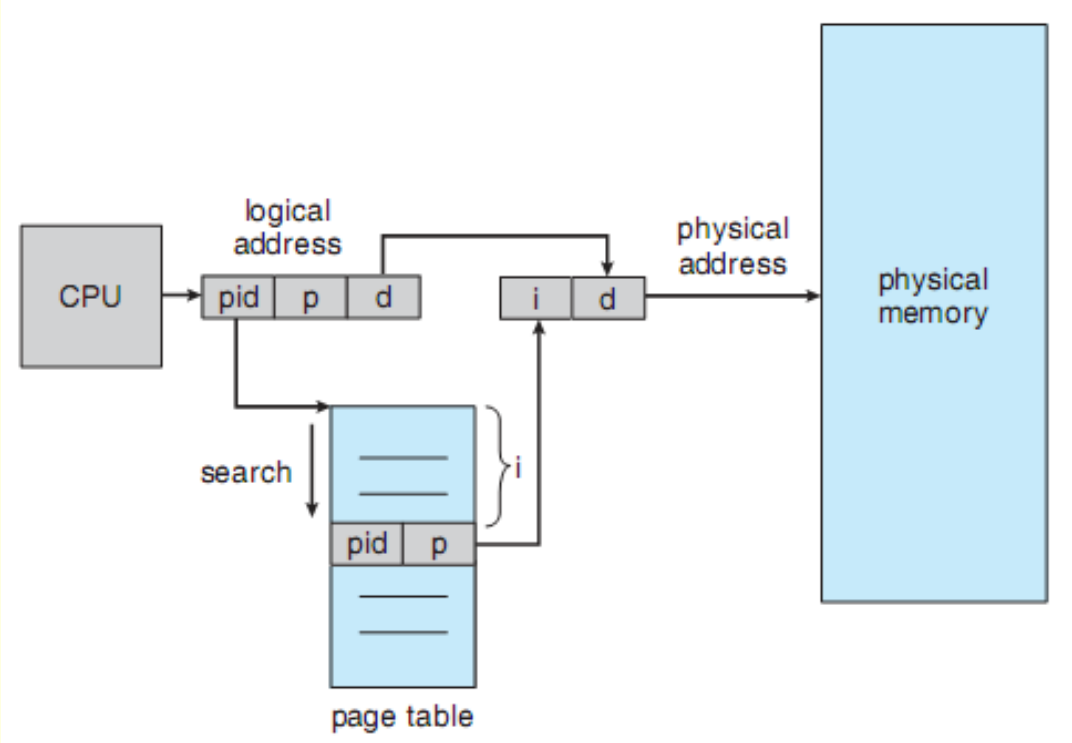
- How many level of paging is required for a 64 bit computer with 4KB pages?
  - 7 levels of paging

Hashed Page Tables



Inverted Page Tables

- Standard page tables use very large amounts of memory

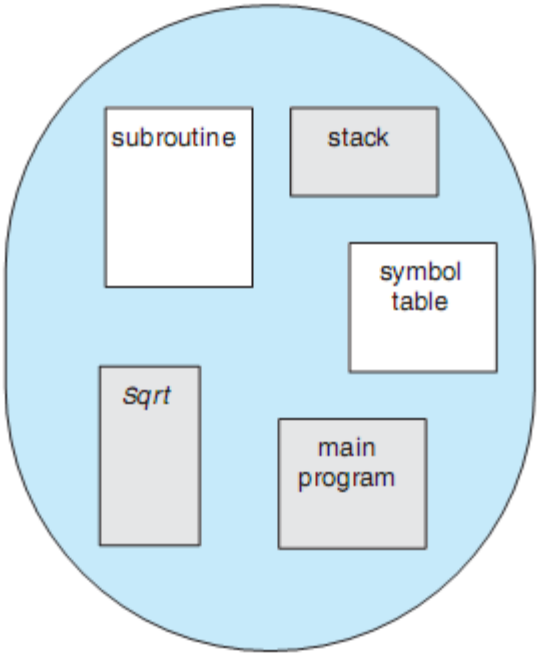


- Disadvantage: Time overhead, difficulty with shared pages.

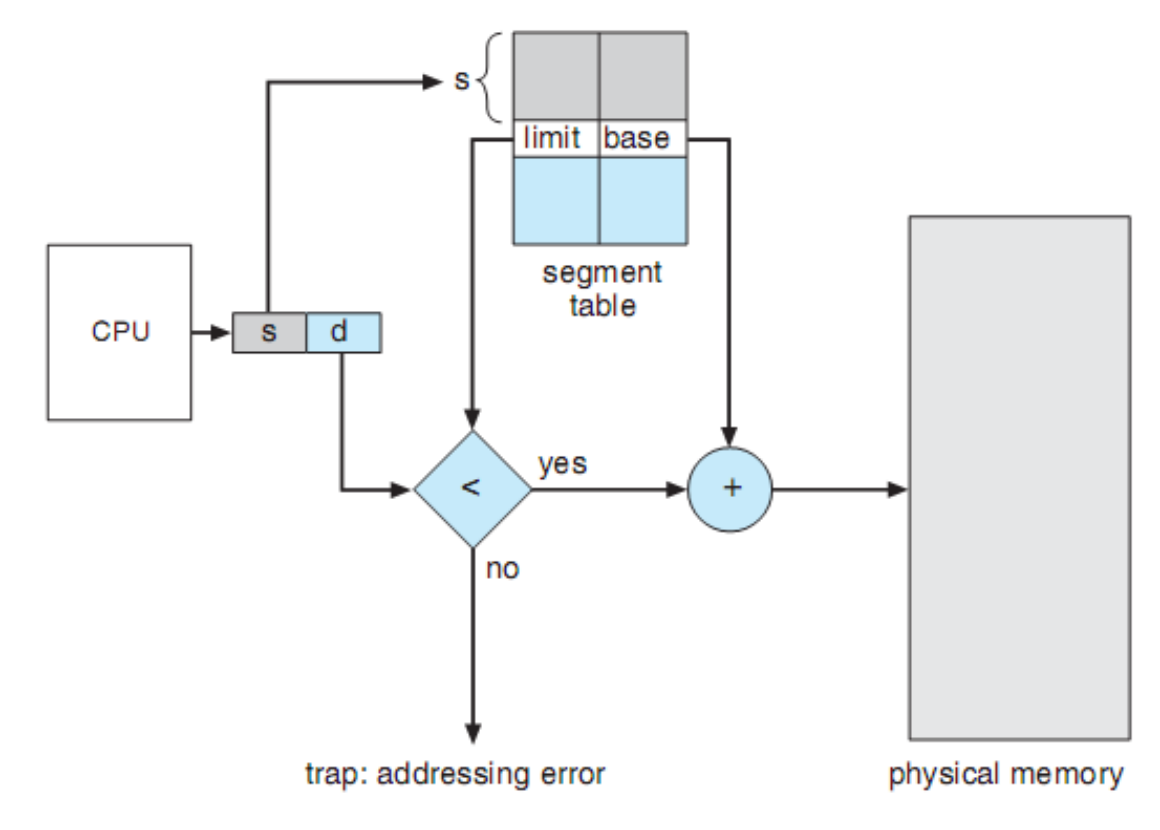
Segmentation

- User's view of memory vs. actual physical memory
- Paging: one virtual address
- Segmentation: a two-partition address
  - A segment-name (or segment-id)
  - An offset
- Some segments generated by a C compiler

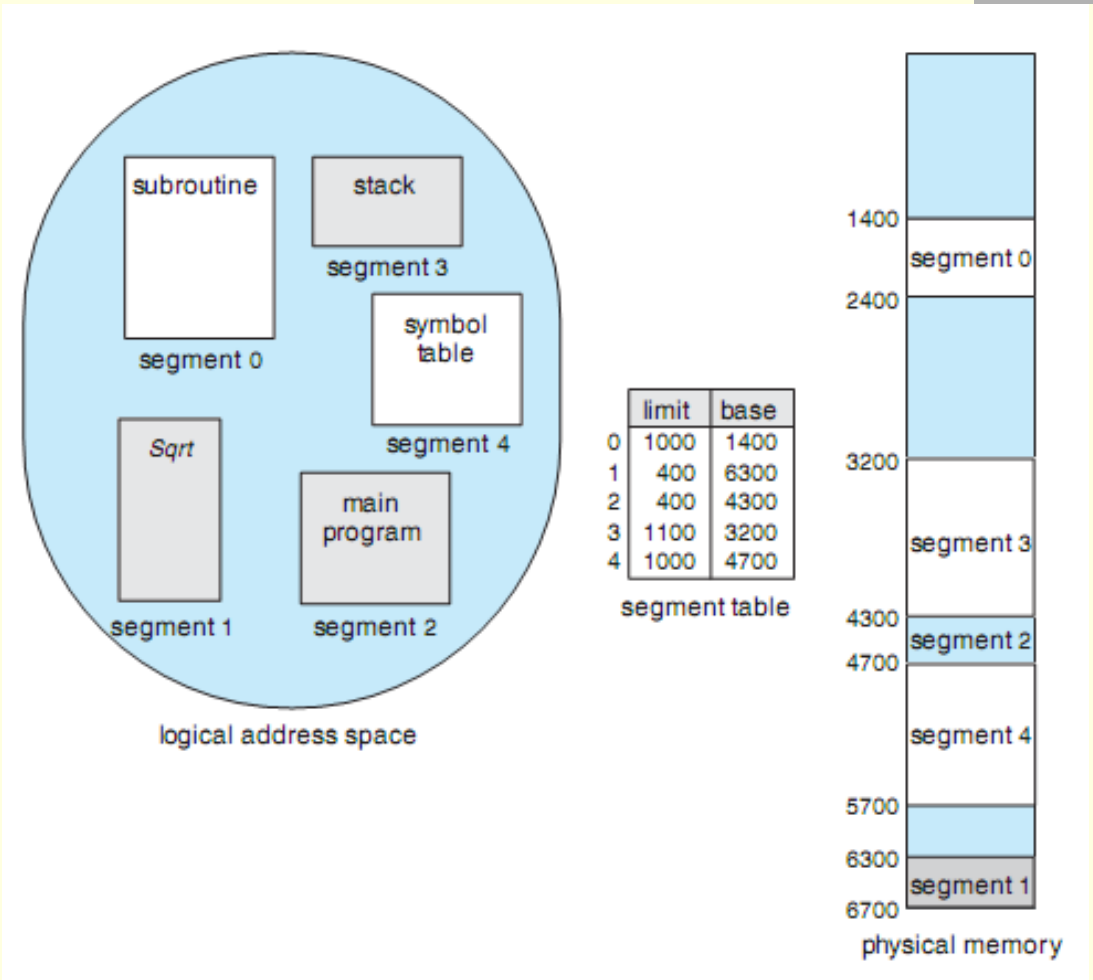
1. The code
2. Global variables
3. The heap, from which memory is allocated
4. The stacks used by each thread
5. The standard C library



Segmentation Hardware



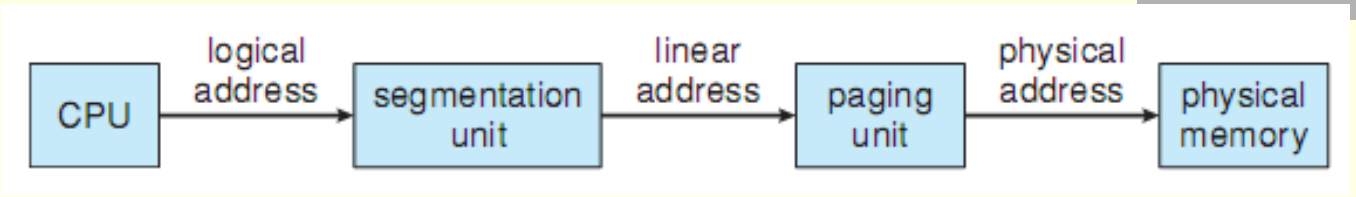
Example


subroutine
stack
segment 3
segment 0
symbol table
segment 4
Sqrt
main program
segment 1
segment 2
logical address space
limit base
0 1000 1400
1 400 6300
2 400 4300
3 1100 3200
4 1000 4700
segment table
1400
segment 0
2400
3200
segment 3
4300
segment 2
4700
segment 4
5700
6300
segment 1
6700
physical memory

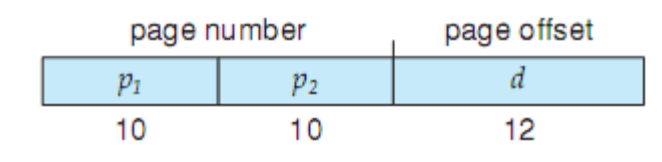
Properties

- Two memory references
- External fragmentation

Example: The Intel Pentium (Including segmentation with paging)



- Segmentation unit+ paging unit replaces MMU
- Pentium paging
  - 4KB and 4MB



- Study Linux on

Pentium Systems!!!