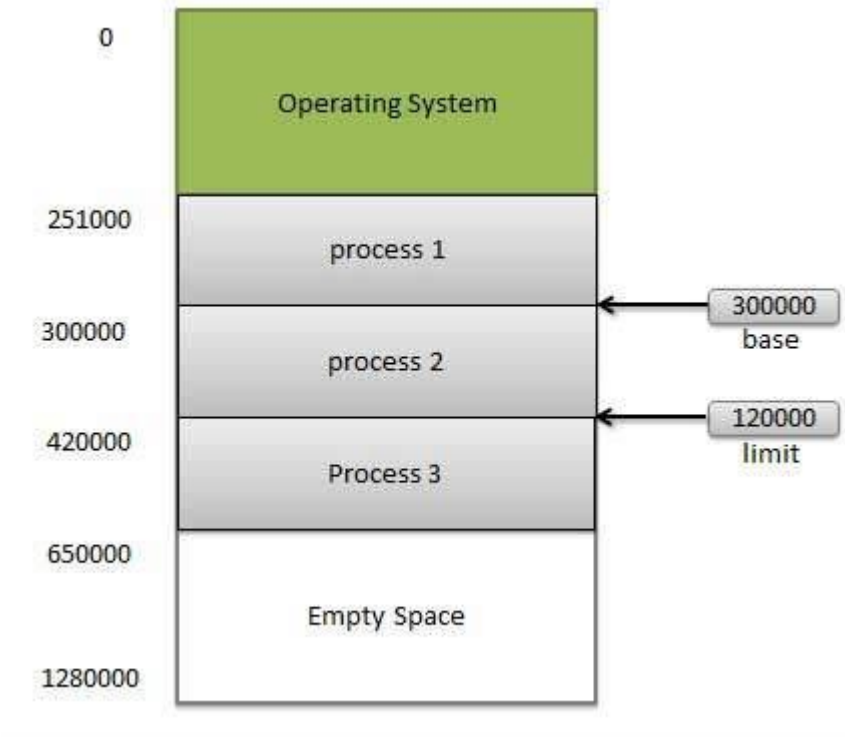
Memory Managment

Introduction to Memory Management

Memory management is the functionality of an operating system which handles or manages primary memory. Memory management keeps track of each and every memory location either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Memory management provides protection by using two registers, a base register and a limit register. The base register holds the smallest legal physical memory address and the limit register specifies the size of the range. For example, if the base register holds 300000 and the limit register is 120000, then the program can legally access all addresses from 300000 through 419999.



Instructions and data to memory addresses can be done in following ways

- **Compile time** -- When it is known at compile time where the process will reside, compile time binding is used to generate the absolute code.
- **Load time** -- When it is not known at compile time where the process will reside in memory, then the compiler generates re-locatable code.
- **Execution time** -- If the process can be moved during its execution from one memory segment to another, then binding must be delayed to be done at run time

Dynamic Loading In dynamic loading, a routine of a program is not loaded until it is called by the program. All routines are kept on disk in a re-locatable load format. The main program is loaded into memory and is executed. Other routines methods or modules are loaded on

request. Dynamic loading makes better memory space utilization and unused routines are never loaded.

Dynamic Linking

Linking is the process of collecting and combining various modules of code and data into a executable file that can be loaded into memory and executed. Operating system can link system level libraries to a program. When it combines the libraries at load time, the linking is called static linking and when this linking is done at the time of execution, it is called as dynamic linking.

In static linking, libraries linked at compile time, so program code size becomes bigger whereas in dynamic linking libraries linked at execution time so program code size remains smaller.

Logical versus Physical Address Space

An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address. Logical address is also known a Virtual address.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a logical address space. The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.

The run-time mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory to a backing store, and then brought back into memory for continued execution.

Backing store is a usually a hard disk drive or any other secondary storage which fast in access and large enough to accommodate copies of all memory images for all users. It must be capable of providing direct access to these memory images.

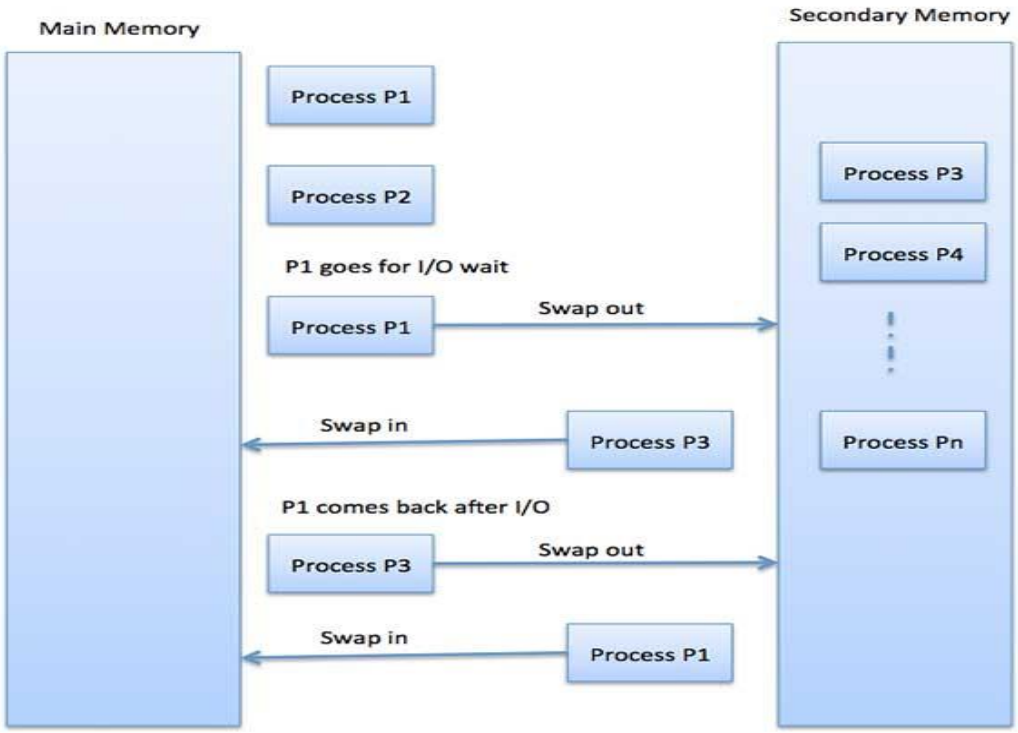
Major time consuming part of swapping is transfer time. Total transfer time is directly proportional to the amount of memory swapped. Let us assume that the user process is of size

100KB and the backing store is a standard hard disk with transfer rate of 1 MB per second. The actual transfer of the 100K process to or from memory will take

100KB / 1000KB per second

= 1/10 second

= 100 milliseconds



Contiguous Allocation

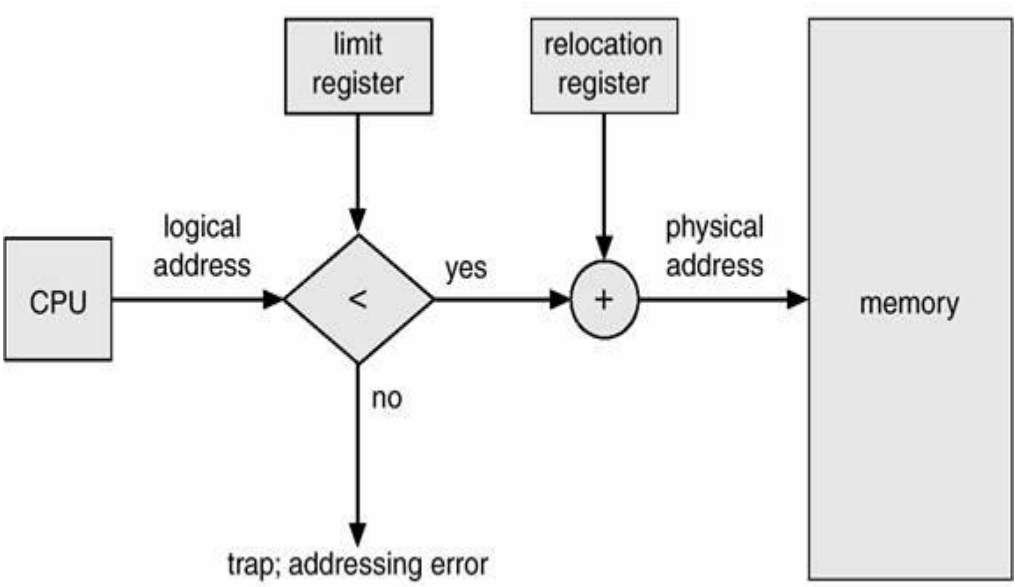
Main memory usually into two partitions:

✦ User processes then held in high memory. Single-partition allocation

✦ Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.

✦ Relocation register contains value of smallest physical address; limit registercontains range of logical addresses – each logical address must be less than the limit register.

Hardware Support for Relocation and Limit Registers



Multiple-partition allocation

✦*Hole* – block of available memory; holes of various size are scattered throughout memory.

✦ When a process arrives, it is allocated memory from a hole large enough to accommodate it.

✦ Operating system maintains information about:

a) Allocated partitions b) free partitions (hole)

First-fit: Allocate the *first* hole that is big enough.

Best-fit: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.

Worst-fit: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.

Fragmentation

External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous.

Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.Reduce external fragmentation by compaction

✦ Shuffle memory contents to place all free memory together in one large block.

✦ Compaction is possible *only* if relocation is dynamic, and is done at execution time.

✦ I/O problem

Paging

Logical address space of a process can be non-contiguous; process is allocated physical memory whenever the latter is available.

- 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.

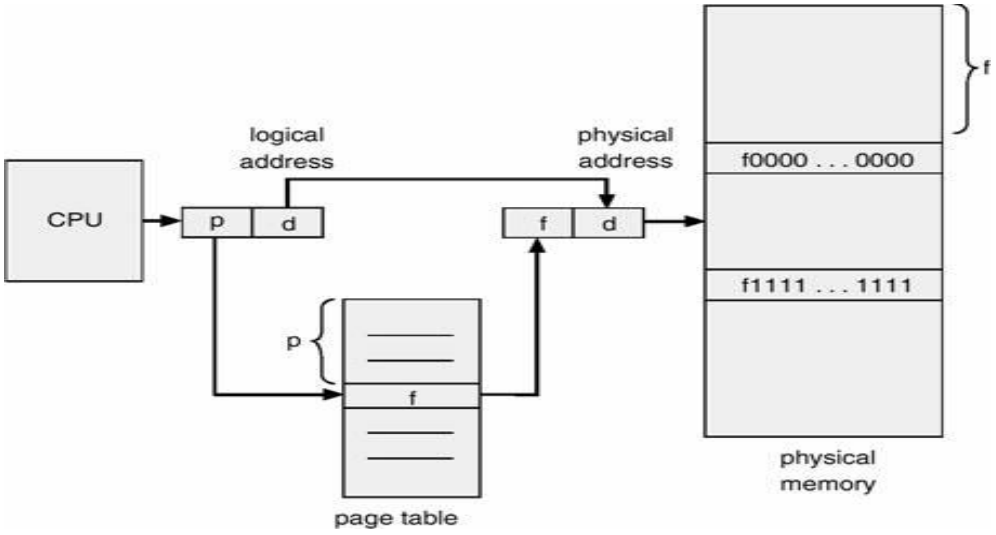
Address Translation Scheme

Address generated by CPU is divided into:

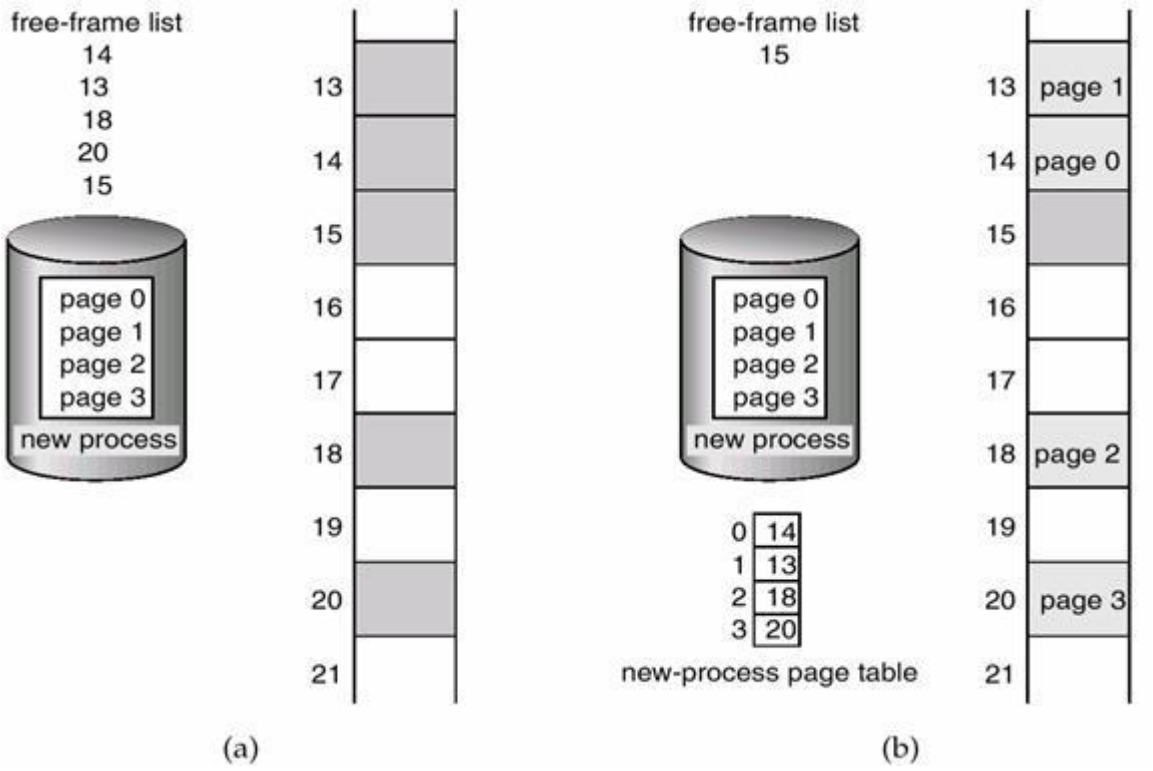
1. *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory.

2. *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit

Address Translation Architecture



Free Frames



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* (PRLR) 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

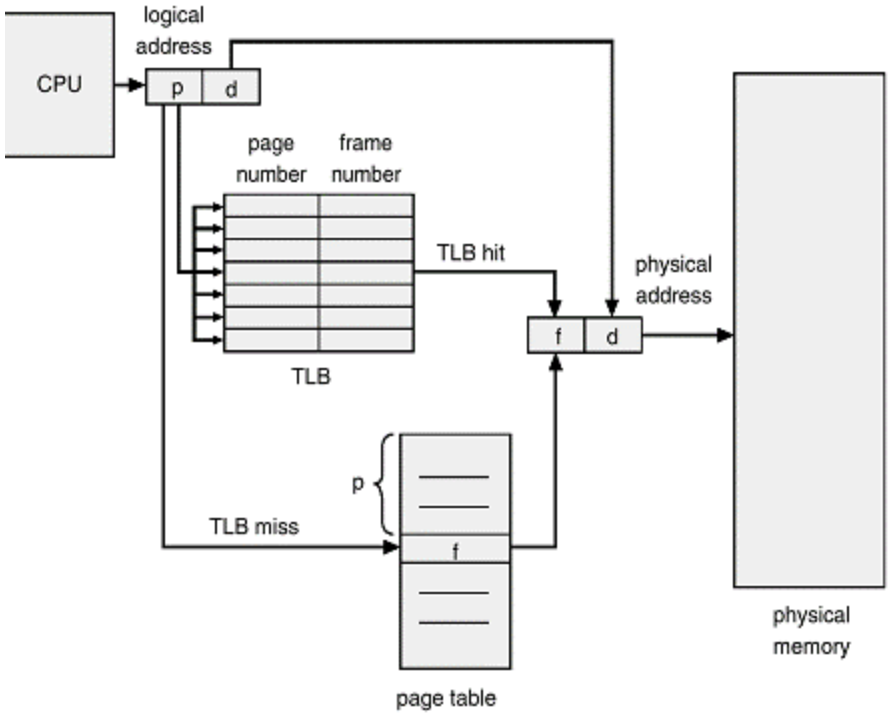
Associative Memory

Associative memory – parallel search Address translation (A´, A´´)

1. If A´ is in associative register, get frame # out.

2. Otherwise get frame # from page table in memory

Paging Hardware with TLB



Effective Access Time

Associative Lookup = å time unit

Assume memory cycle time is 1 microsecond

Hit ratio – percentage of times that a page number is found in the associative registers; ration related to

number of associative registers.

Hit ratio = a

Effective Access Time (EAT)

EAT = (1 + å) a + (2 + å)(1 – a)

= 2 + å – a

Memory Protection

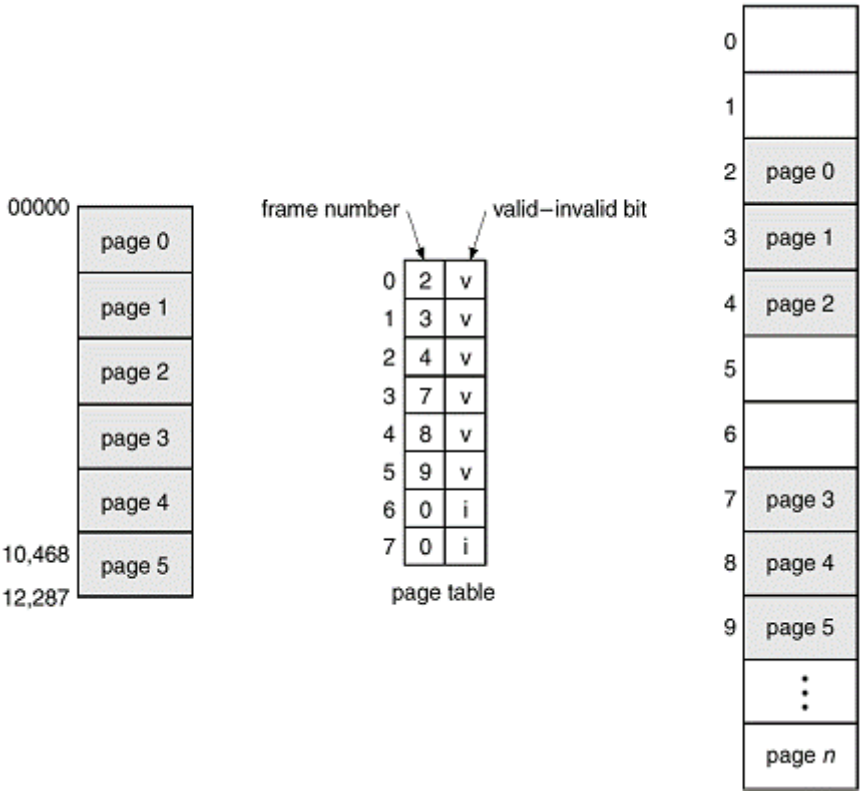
Memory protection implemented by associating protection bit with each frame.

***Valid-invalid* bit attached to each entry in the page table:**

1. "Valid" indicates that the associated page is in the process' logical address space, and is thus a legal page.

2. "invalid" indicates that the page is not in the process' logical address space.

Valid (v) or Invalid (i) Bit In A Page Table



Page Table Structure

1. Hierarchical Paging
2. Hashed Page Tables
3. Inverted Page Tables

Two-Level Paging Example

A logical address (on 32-bit machine with 4K page size) is divided into:

1. a page number consisting of 20 bits.
2. a page offset consisting of 12 bits.

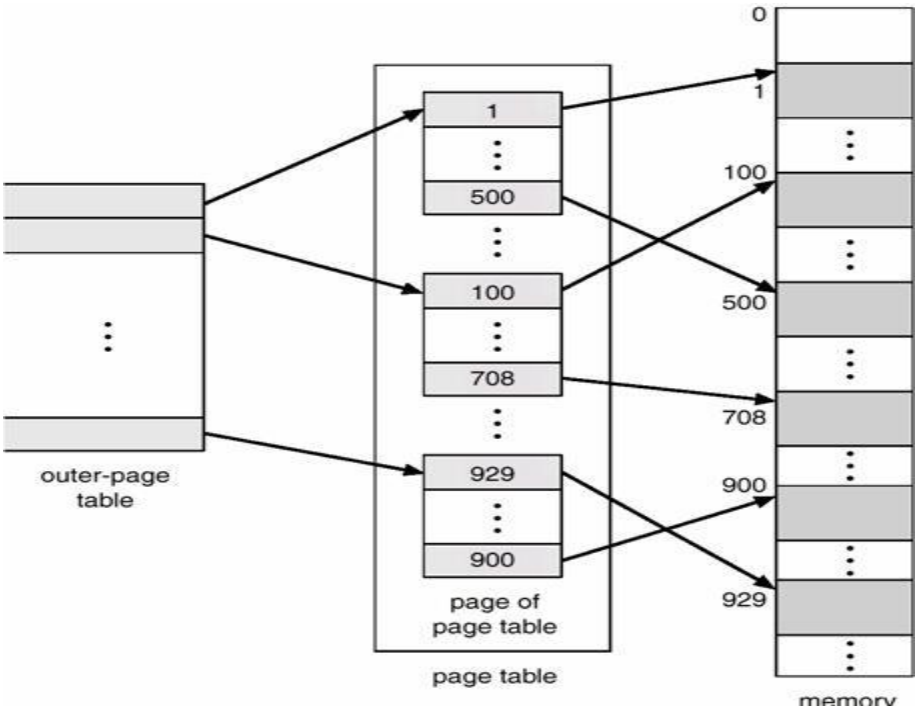
Since the page table is paged, the page number is further divided into:

3. a 10-bit page number.
4. a 10-bit page offset.
5. Thus, a logical address is as follows:

page number		page offset
*p*i	*p*2	*d*
10	10	12

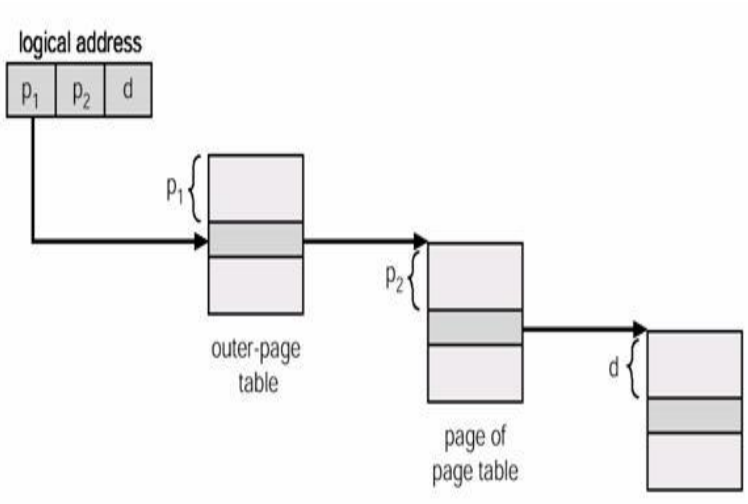
where*pi* is an index into the outer page table, and *p2* is the displacement within the page of the outer page table.

Two-Level Page-Table Scheme



Address-Translation Scheme

Address-translation scheme for a two-level 32-bit paging architecture



Inverted Page Table

- One entry for each real page of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- 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.

Shared Pages

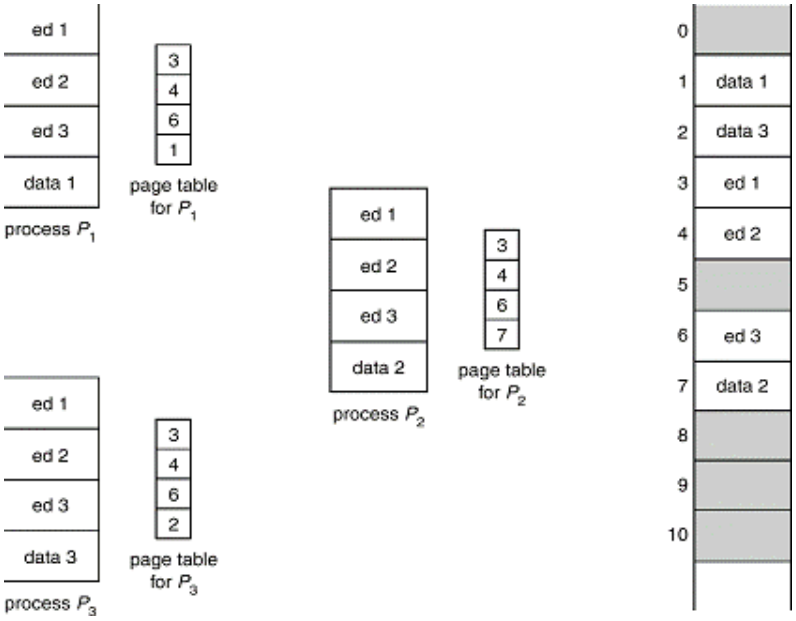
Shared code

1. One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).

2. Shared code must appear in same location in the logical address space of all processes.

3. Each process keeps a separate copy of the code and data.

4.The pages for the private code and data can appear anywhere in the logical address space.


ed 1
ed 2
ed 3
data 1
process P_1
3
4
6
1
page table for P_1
ed 1
ed 2
ed 3
data 2
process P_2
3
4
6
7
page table for P_2
ed 1
ed 2
ed 3
data 3
process P_3
3
4
6
2
page table for P_3
0
1
data 1
2
data 3
3
ed 1
4
ed 2
5
6
ed 3
7
data 2
8
9
10