**STRUCTURE OF PAGE TABLE**

Some of the common techniques that are used for structuring the Page table are as follows:

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

**Hierarchical Paging**

Another name for Hierarchical Paging is multilevel paging. As we knew when the CPU access a page of any process it has to be in the main memory. Along with the page, the page table of the same process must also be stored in the main memory that too in contiguous manner. Now, what if the size of the page table is larger than the frame size of the main memory. In that case, we have to breakdown the page table at multiple levels in order to fit in the frame of the main memory.

In this type of Paging the logical address space is broke up into Multiple page tables. Hierarchical Paging is one of the simplest techniques and for this purpose, a two-level page table and three-level page table can be used.

### **Two Level Page Table:**

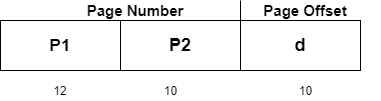
Consider a system having 32-bit logical address space and a page size of 1 KB and it is further divided into:

* Page Number consisting of 22 bits.
* Page Offset consisting of 10 bits.

As we page the Page table, the page number is further divided into :

* Page Number consisting of 12 bits.
* Page Offset consisting of 10 bits.

Thus the Logical address is as follows:



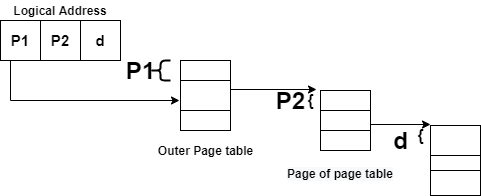
In the above diagram,

P1 is an index into the **Outer Page** table.

P2 indicates the displacement within the page of the **Inner page** Table.

As address translation works from outer page table inward so is known as **forward-mapped Page Table**.

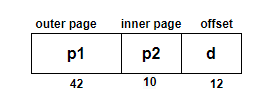
Below given figure below shows the Address Translation scheme for a two-level page table



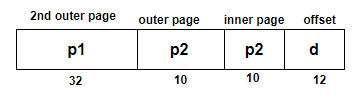
### 

### **Three Level Page Table**

For a system with 64-bit logical address space, a two-level paging scheme is not appropriate. Let us suppose that the page size, in this case, is 4KB.If in this case, we will use the two-page level scheme then the addresses will look like this:



Thus in order to avoid such a large table, there is a solution and that is to divide the outer page table, and then it will result in a **Three-level page table:**



**2)Hash page tables:**

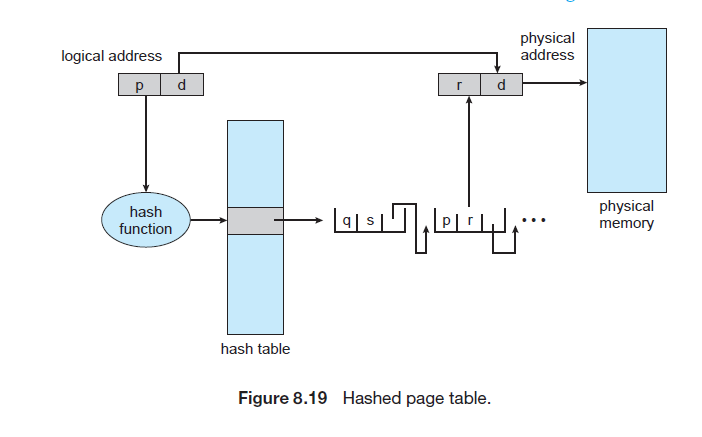
This approach is used to handle address spaces that are larger than 32 bits.

* In this virtual page, the number is hashed into a page table.
* This Page table mainly contains a chain of elements hashing to the same elements.

Each element mainly consists of :

1. The virtual page number
2. The value of the mapped page frame.
3. A pointer to the next element in the linked list.

Given below figure shows the address translation scheme of the Hashed Page Table:

****

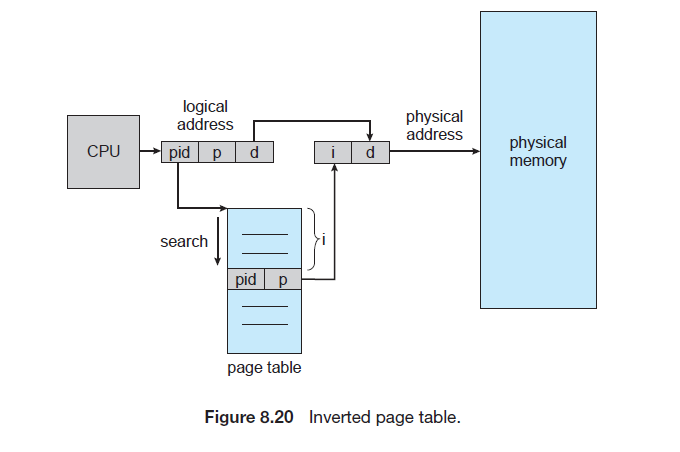
Also, many entries can map into the same index in the page table after going through the hash function. Thus chaining is used in order to handle this.

**3)Inverted page tables:**

The Inverted Page table basically combines a page table and a frame table into a single data structure.

* There is one entry for each virtual page number and a real page of memory
* And the entry mainly consists of the virtual address of the page stored in that real memory location along with the information about the process that owns the page.
* Though this technique decreases the memory that is needed to store each page table; but it also increases the time that is needed to search the table whenever a page reference occurs.

Given below figure shows the address translation scheme of the Inverted Page Table:

****

In this, we need to keep the track of process id of each entry, because many processes may have the same logical addresses.