Virtual Memory

Reading: Section 4.3 of Textbook (Tanenbaum)

Virtual memory is a concept that is associated with ability to address a memory space much larger than that the available physical memory.

The basic idea behind the virtual memory is that the combined size of the of the program, data, and stack may exceed the amount of physical memory available for it. The OS keeps those part of the program currently in use in main memory, and the rest on the disk.

Virtual Memory

Virtual storage is not a new concept, this concept was devised by Fotheringham, 1961 and used in Atlas computer system.

But the common use in OS is the recent concept, all microprocessor now support virtual memory.
Virtual memory can be implemented by two most commonly used methods : *Paging* and *Segmentation or mix of both*.

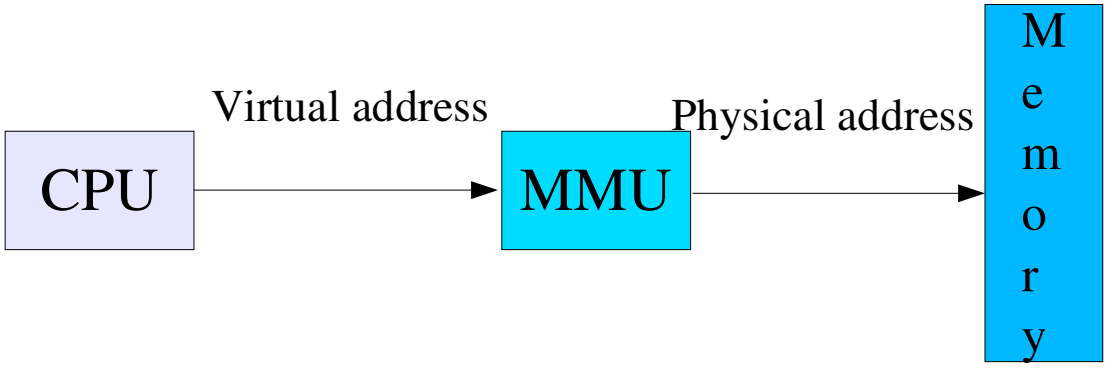
Virtual Memory

Virtual address space vs. Physical address space

The set of all virtual (logical) addresses generated by a program is a *virtual address space*; the set of all physical addresses corresponding to these virtual addresses is a *physical address space*.

MMU

The run time mapping from virtual address to physical address is done by hardware devices called *memory-management-unit (MMU)*.



Paging

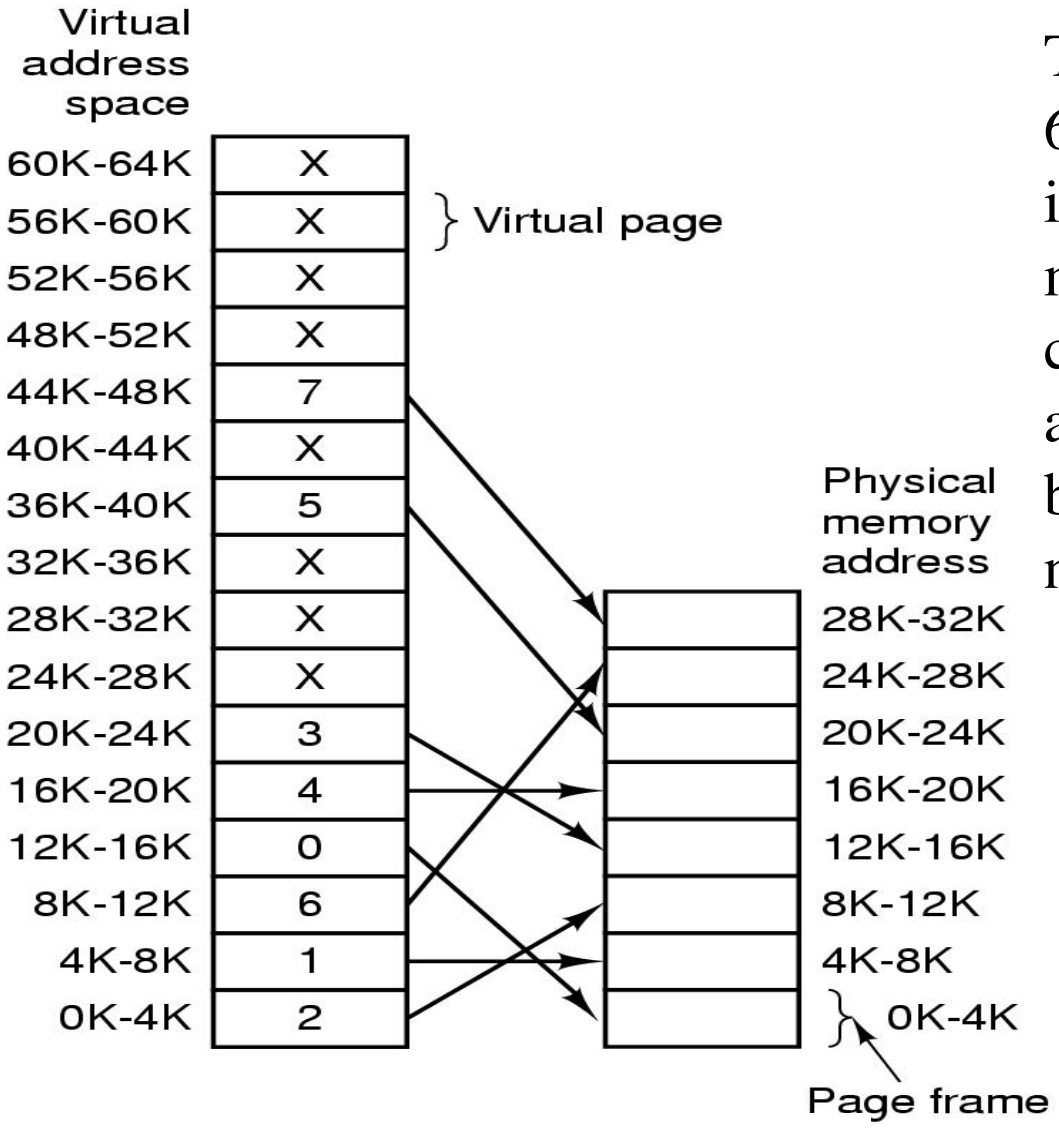
The virtual address space (process) is divided up into fixed sized blocks called pages and the corresponding same size block in main memory is called frames. When a process is to be executed, its pages are loaded into any available memory frames from the backing store.

The size of the pages is determined by the hardware, normally from 512 bytes to 64KB (in power of 2).

Paging permits the physical address space of process to be noncontiguous.

Traditionally, support for paging has been handled by hardware, but the recent design have implemented by closely integrating the hardware and OS.

Paging



This example shows 64KB program can run in 32KB physical memory. The complete copy is stored in the disk and the pieces can be brought into memory as needed.

Paging

With 64KB of virtual address space and 32KB of physical memory and 4KB page size, we get 16 virtual pages and 8 frames.
What happen in following instruction?
MOV REG, 0
This virtual address, 0, is sent to the MMU. The MMU sees that this virtual address falls in page 0 (0-4095), which is mapping to frame 2 (8192 -12287). Thus the address 0 is transformed to 81912 and output address is 8192.

Address Translation

Address generated by CPU is divided into:

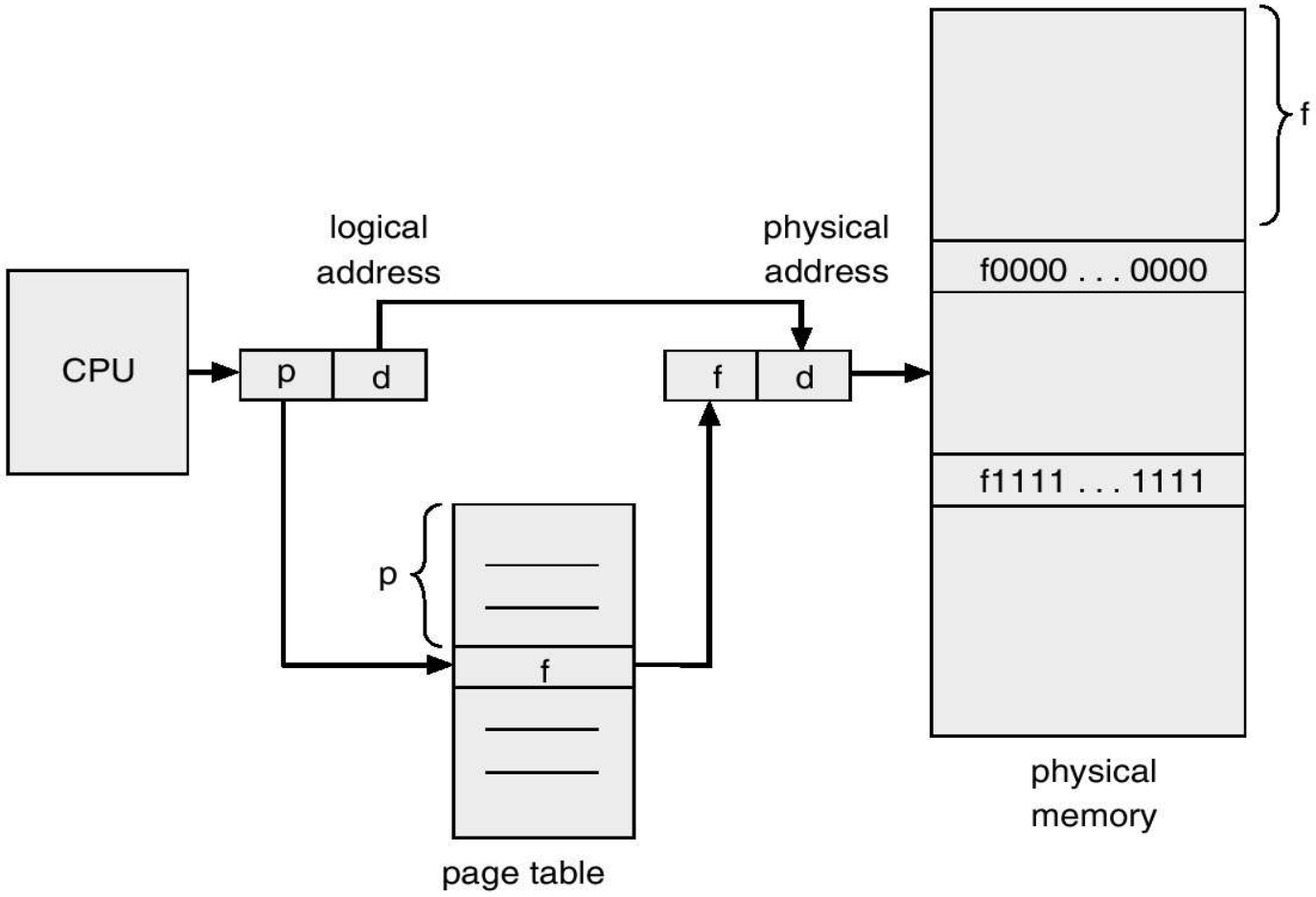
Page number (p) – used as an index into a *page table* which contains base address of each page in physical memory.

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

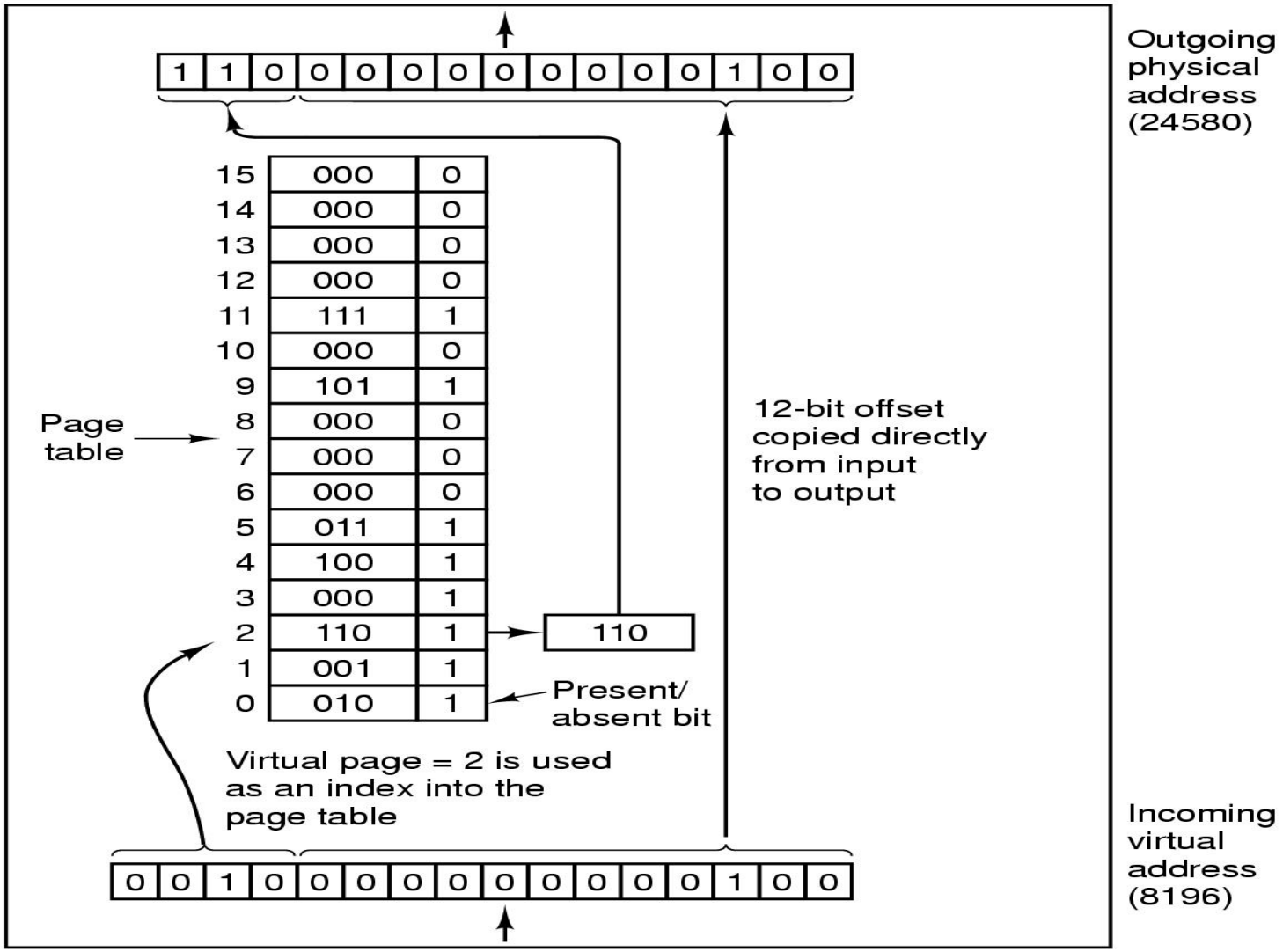
If the size of the logical address space is 2^m and page size is 2^n, then the high-order m-n bits of logical address designate page number, and n lower order bits designate the page offset.

Present/absent bit keeps the track of which pages are physically present in memory.

Address Translation



Address Translation Example



Page Tables

For each process, page table stores the number of frame, allocated for each page.

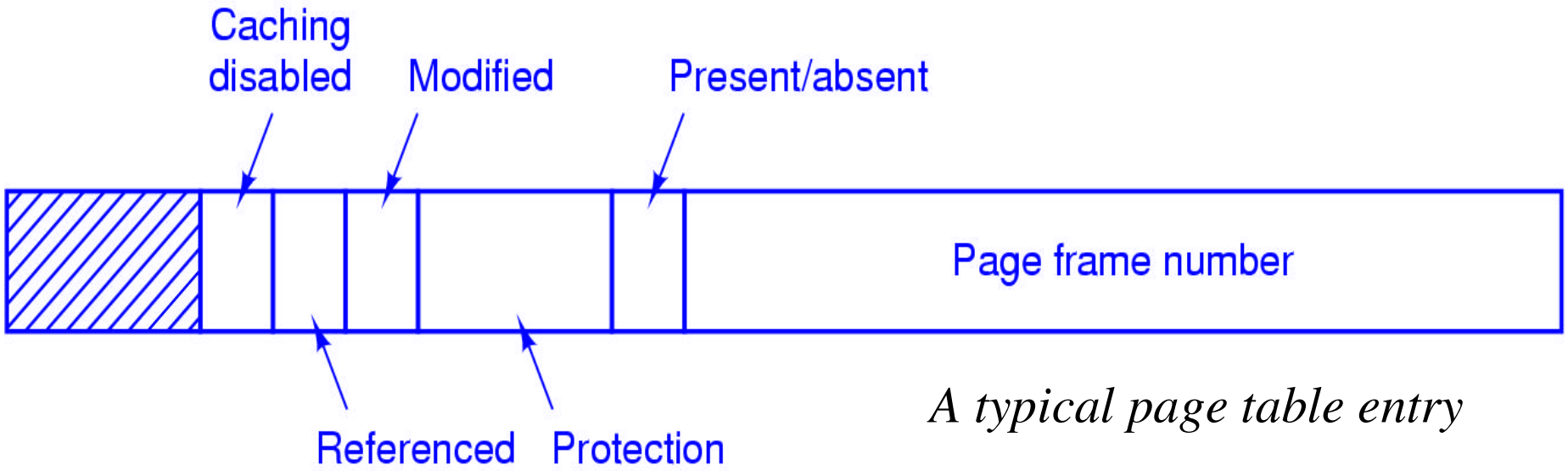
The purpose of the page table is to map virtual pages into pages frames. This function of page table can be represented in mathematical notation as:

$$page\_frame = page\_table(page\_number)$$

The virtual page number is used as an index into the page table to find the corresponding page frame.

Page Table Structure

The exact layout of page table entry is highly machine dependent, but more common structure for 32-bit system is as:



A typical page table entry

Frame number: The goal is to locate this value.

Page Table Structure

Present/absent bit: If present/absent bit is present, the virtual addresses is mapped to the corresponding physical address. If present/absent is absent the trap is occur called page fault.

Protection bit: Tells what kinds of access are permitted read, write or read only.

Modified bit (dirty bit): Identifies the changed status of the page since last access; if it is modified then it must be rewritten back to the disk.

Referenced bit: set whenever a page is referenced; used in page replacement.

Caching disabled: used for that system where the mapping into device register rather than memory.

Page Tables Issues

Size of page table.

Most modern computers support a large virtual-address space (2^32 to 2^64). If the page size is 4KB, a 32-bit address space has 1 million pages. With 1 million pages, the page table must have 1 million entries.

think about 64-bit address space???

Efficiency of mapping.

If a particular instruction is being mapped, the table lookup time should be very small than its total mapping time to avoid becoming CPU idle.

What would be the performance, if such a large table have to load at every mapping.

How to handle these issues?

Multilevel Page Tables

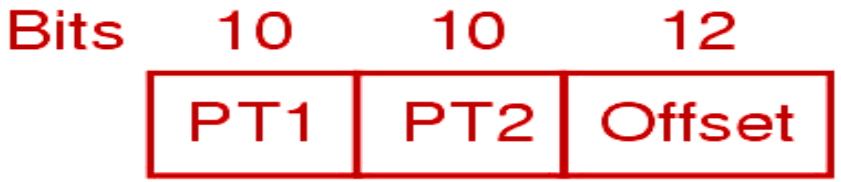
To get around the problem of having to store huge page tables in memory all the time, many computers use the multilevel page table in which the page table itself is also paged.

Pentium II -2 level, 32-bit Motorola -4 level, 32 -bit SPARC-3 level etc.
For 64-bit architectures, multilevel page table are generally considered inappropriate. For example, the 64-bit UltraSPARC would require seven level of paging – increase accessing complexity.

Multilevel Page Tables

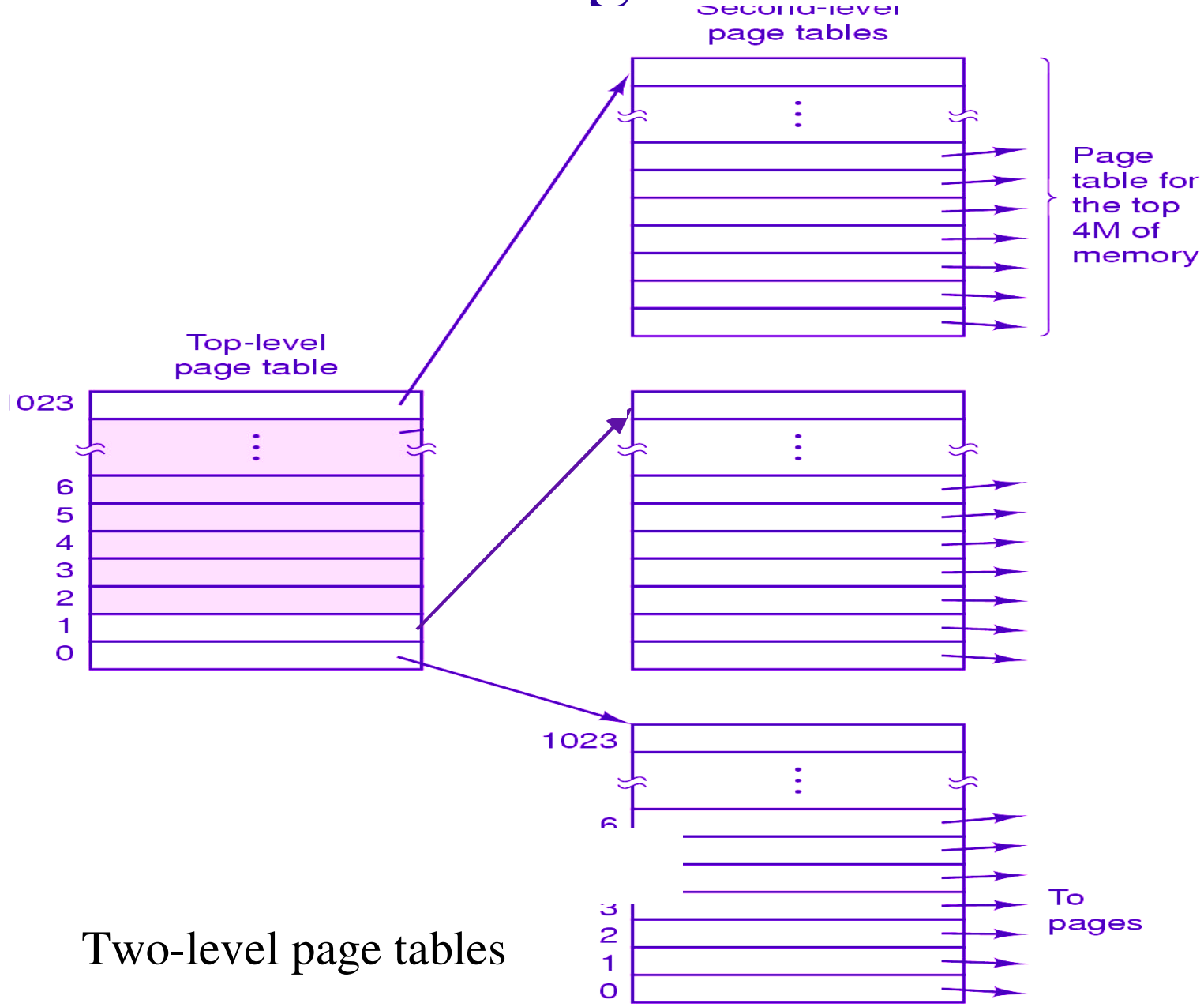
Example: Two-Level Page Tables

A 32-bit virtual address space with a page size of 4 KB, the virtual address space is partitioned into a 10-bit PT1 field, a 10-bit PT2 field, and a 12-bit offset field.



The top level have 1024 entries, corresponding to PT1. At mapping, it first extracts the PT1 and uses this value as an index into the top level page table. Each of these entries have again 1024 entries, the resulting address of top-level yields the address or page frame number of second-level page table.

Multilevel Page Tables



Two-level page tables

Hashed Page Tables

A common approach for handling address space larger than 32-bit.

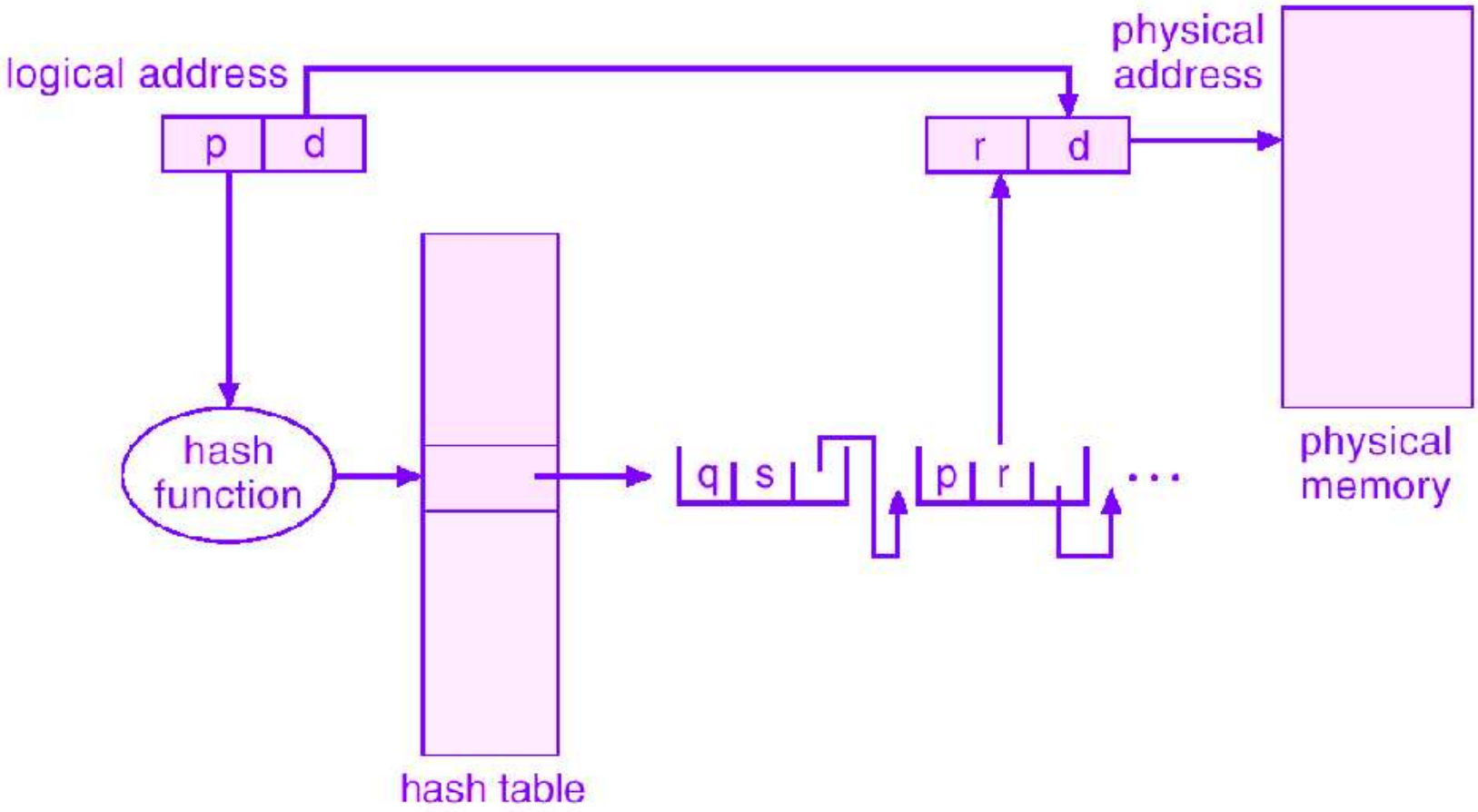
The hash value is the virtual-page number.

Each entry in the hash table contains a linked list of elements that hash to the same location.

Each element consists of three fields: virtual-page-number, value of mapped page frame, and a pointer to the next element.

The virtual address is hashed into the hash table, if there is match the corresponding page frame is used, if not, subsequent entries in the linked list are searched.

Hashed Page Tables

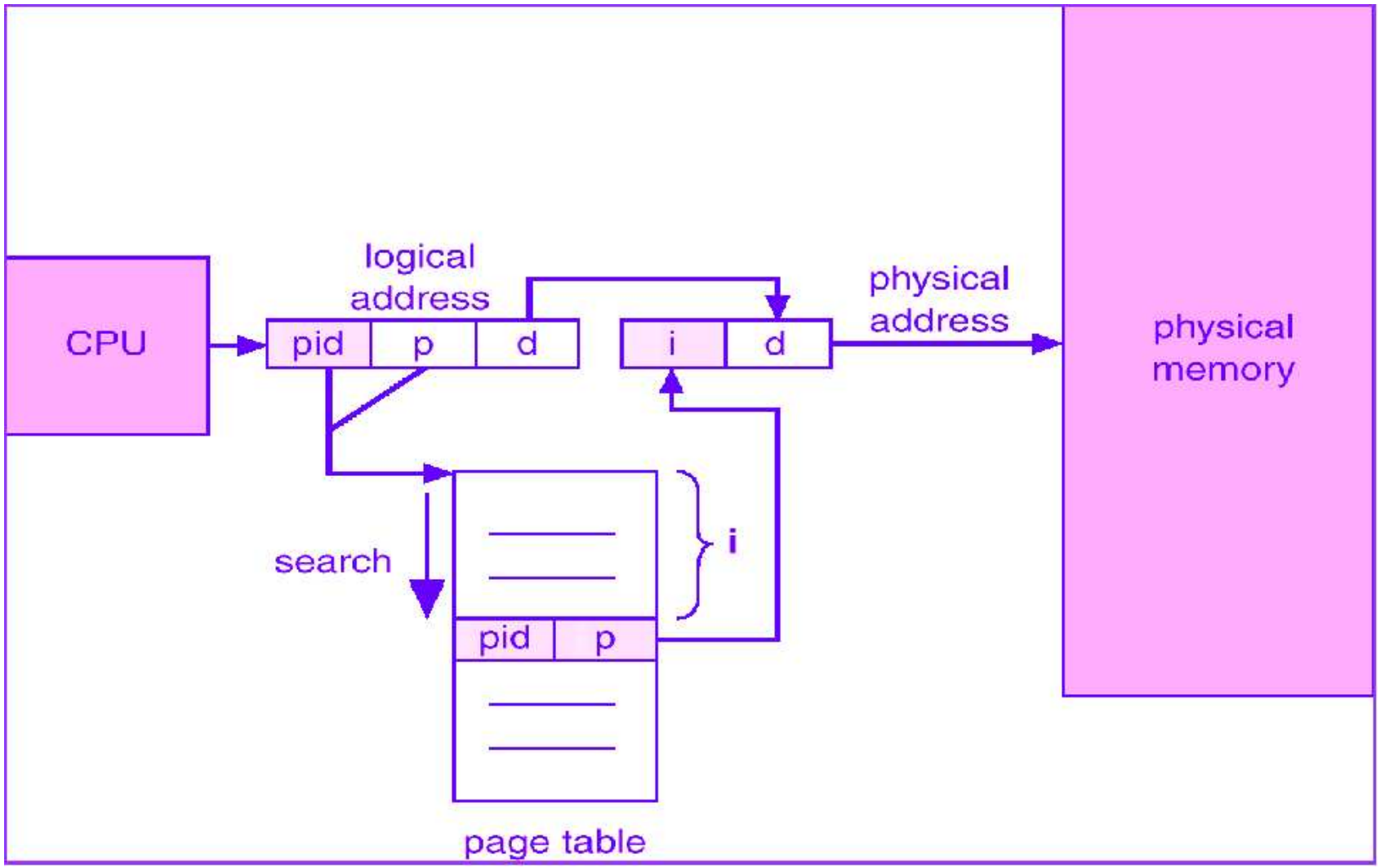


Hashed Page Table

Inverted Page Tables

A common approach for handling address space larger than 32-bit.
One entry per page frame, rather than one entry per page in earlier tables. (Ex: 256MB RAM with 4KB page requires only 65,536 entries in table)
Entry consists of the virtual address of the page stored in that physical memory location, with information about the process that owns that page.
Virtual address consists three fields: [process-id, page-number, offset]. The inverted page table entry is determined by [process-id, page-number]. The page table is search for the match, say at entry i the match is found, then the physical address [i, offset] is generated.

Inverted Page Tables



Inverted Page Tables

Advantage: Decreases the memory size to store the page table.

Problem: It must search entire table in every memory reference, not just on page faults.

Searching a 64K table in every reference is not a way to make system fast!

Solutions:

- Use of Hashed tables.
- Use of TLBs.

Translation Lookaside Buffers (TLBs)

How to speed up address translation?

Locality: Most processes use large number of reference to small number of pages.

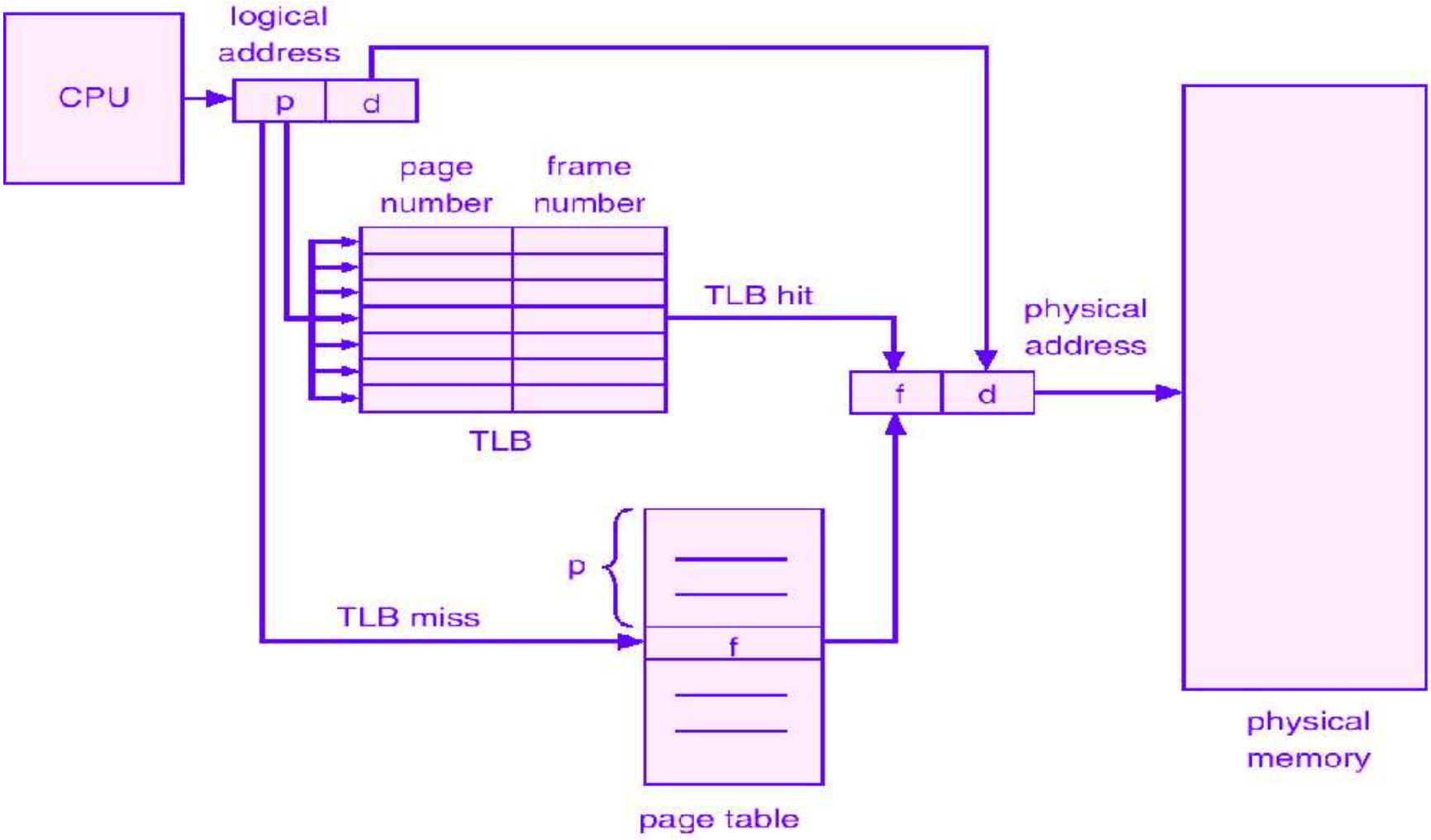
The small, fast, lookup hardware cache, called TLB is used to overcome this problem, by mapping logical address to physical address without page tables.

The TLB is associative, high-speed, memory; each entry consists information about virtual page-number and physical page frame number.

Typical sizes: only 64 to 1024 entries.

Key of improvement: Parallel search for all entries.

Translation Lookaside Buffers (TLBs)



MMU with TLB

Translation Lookaside Buffers (TLBs)

When logical address is generated by CPU, its page number is presented to the TLB; if page number is found (*TLB hit*), its frame number is immediately available, the whole task would be very fast because it compare all TLB entries simultaneously.
If the page number is not in TLB (*TLB miss*), a memory reference to the page table must be made. It then replace one entry of TLB with the page table entry just referenced. Thus in next time, for that page, TLB hit will found.

Advantages/Disadvantages of Paging

Advantages:

Fast to allocate and free:

- Alloc: keep free list of free pages, grab first page in the list.
- Free: Add pages to free list.

Easy to swap-out memory to disk.

- Frame size matches disk page size.
- Swap-out only necessary pages.
- Easy to swap-in back from disk.

Disadvantages:

Additional memory reference.

- Page table are kept in memory.

Internal fragmentation: process size does not match allocation size.

Home Works

HW #8

1. 6,7, 8, 10, 11, 12, 13,17 & 20 from text book (Tanenbaum), Ch. 4.
2. Why are page sizes always a power of 2?
3. On a simple paging system with 2^24 bytes of physical memory, 256 pages of logical address space, and a page size of 2^10 bytes, how many bits are in a logical address?
4. Describe, how TLB increase performance in paging.

Read:

Page Replacement (Section 4.4 of Tanenbaum)