1906003052015

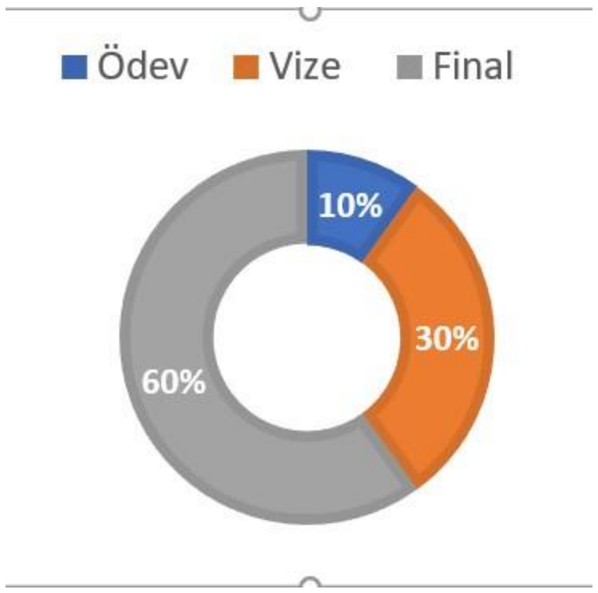
İşletim Sistemleri

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

Giriş

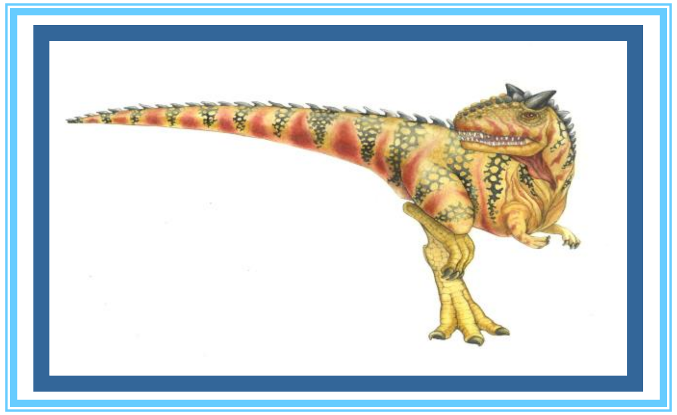
Ders Günü ve Saati:
Çarşamba: 13:00-16:00

- Uygulama Unix (Linux) İşletim sistemi
- Devam zorunluluğu %70
- Uygulamalar C programlama dili üzerinde gerçekleştirilecektir. Öğrencilerden programlama bilgisi beklenmektedir.



HAFTA	KONULAR
Hafta 1	: İşletim sistemlerine giriş, İşletim sistemi stratejileri
Hafta 2	: Sistem çağrıları
Hafta 3	: Görev, görev yönetimi
Hafta 4	: İplikler
Hafta 5	: İş sıralama algoritmaları
Hafta 6	: Görevler arası iletişim ve senkronizasyon
Hafta 7	: Semaforlar, Monitörler ve uygulamaları
Hafta 8	: Vize
Hafta 9	: Kritik Bölge Problemleri
Hafta 10	: Kilitlenme Problemleri
Hafta 11	: Bellek Yönetimi
Hafta 12	: Sayfalama, Segmentasyon
Hafta 13	: Sanal Bellek
Hafta 14	: Dosya sistemi, erişim ve koruma mekanizmaları, Disk planlaması ve Yönetimi
Hafta 15	: Final

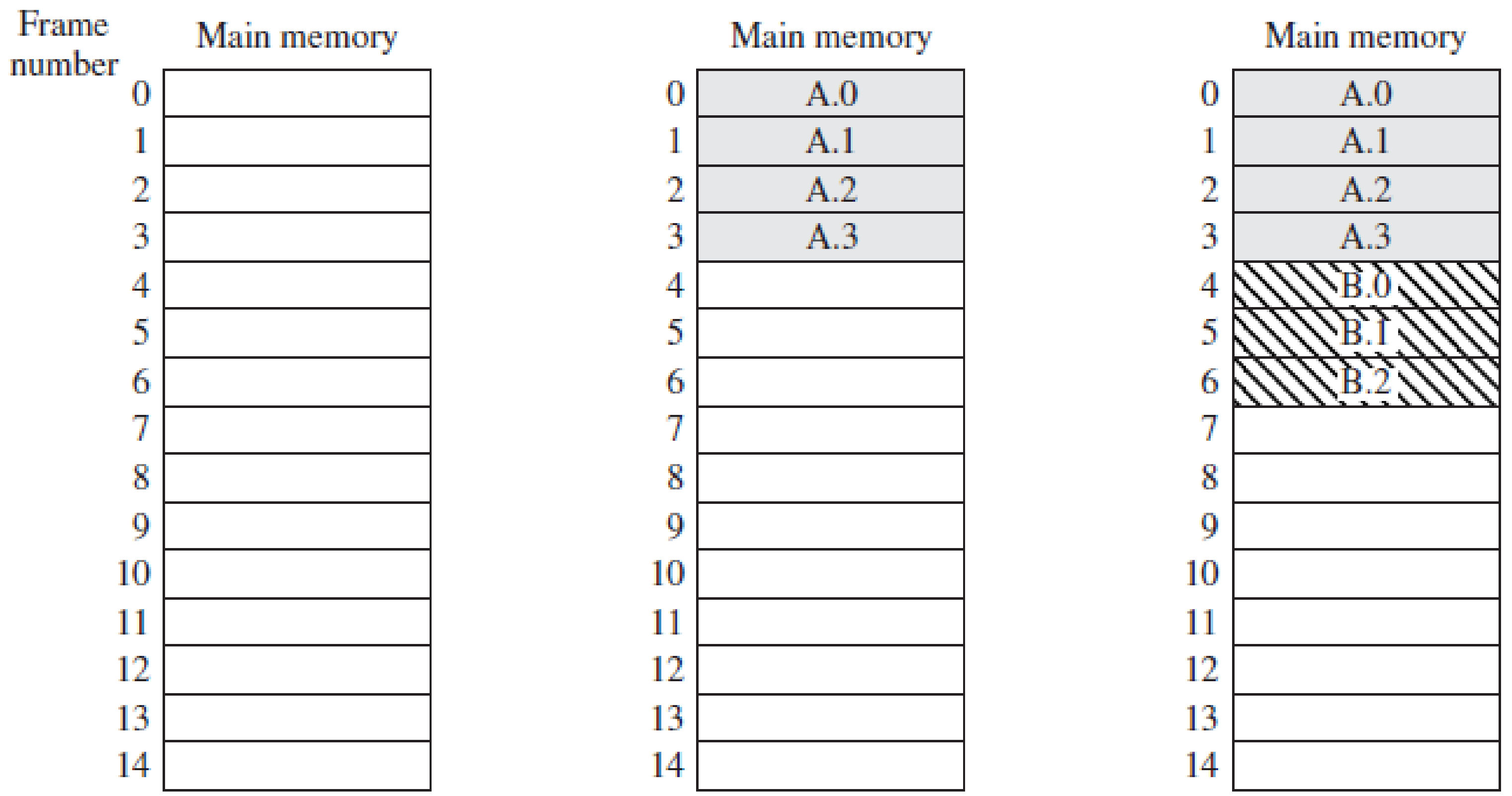
Main Memory



Paging

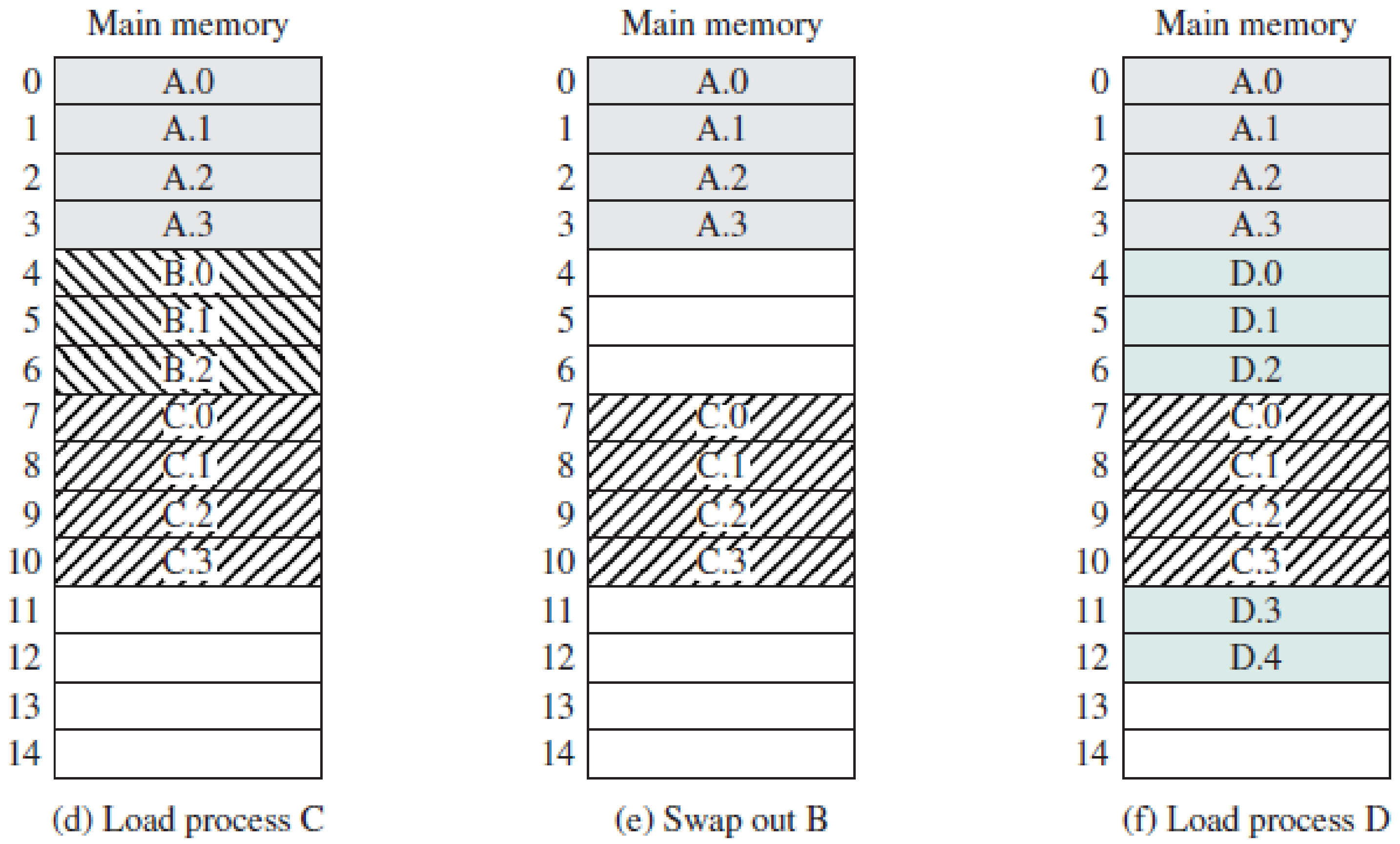
- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- 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
- Backing store likewise split into pages
- Still have Internal fragmentation

Paging



(a) Fifteen available frames (b) Load process A (c) Load process B

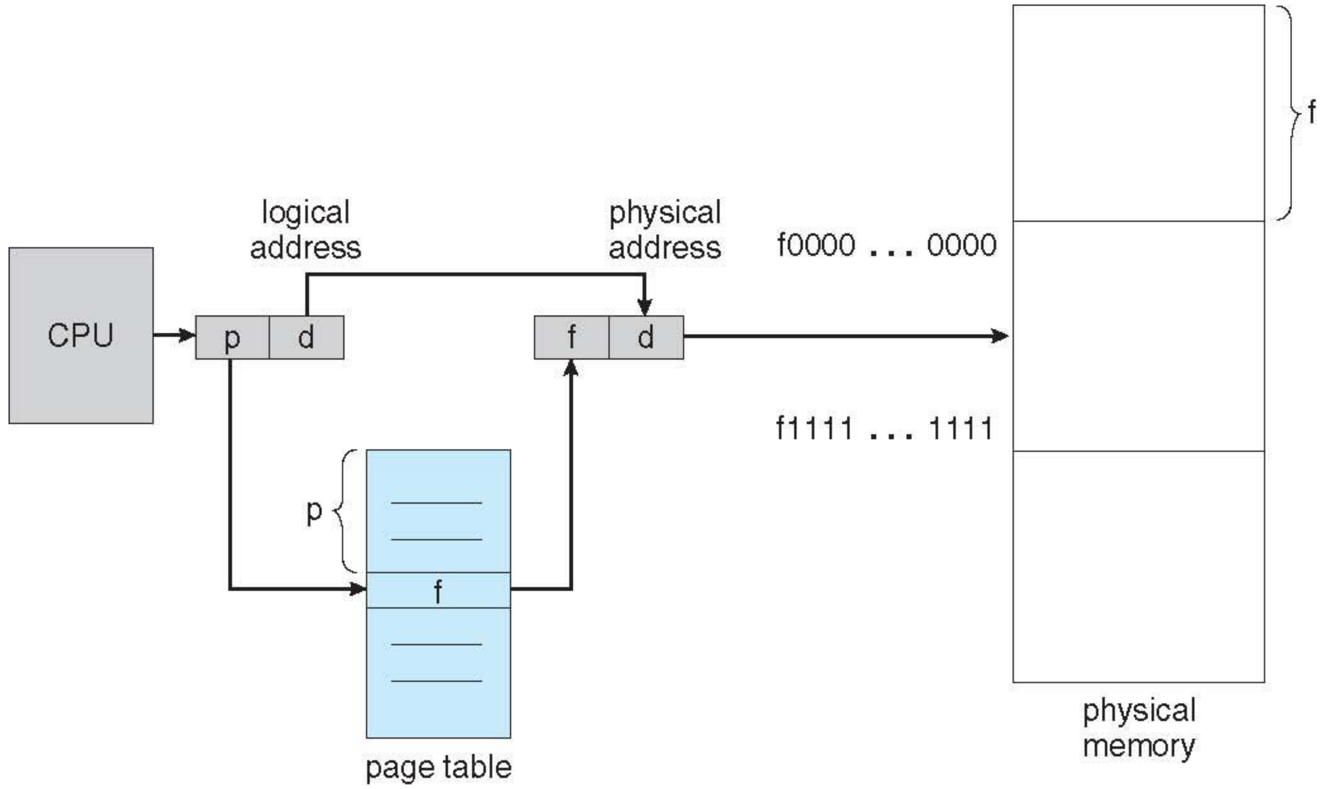
Paging



Main memory
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
A.0
A.1
A.2
A.3
B.0
B.1
B.2
C.0
C.1
C.2
C.3
(d) Load process C
Main memory
A.0
A.1
A.2
A.3
C.0
C.1
C.2
C.3
(e) Swap out B
Main memory
A.0
A.1
A.2
A.3
D.0
D.1
D.2
C.0
C.1
C.2
C.3
D.3
D.4
(f) Load process D

Paging

A simple base address register will no longer suffice.
Rather, the operating system maintains a page table for each process.
The page table shows the frame location for each page of the process.



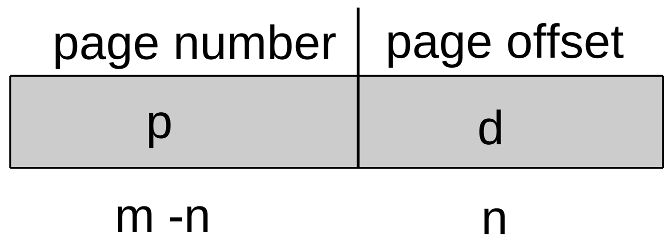
Address Translation Scheme

- Within the program, each logical address consists of a page number and an offset within the page.
- Recall that in the case of simple partition, a logical address is the location of a Word* relative to the beginning of the program; the processor translates that into a physical address.
- With paging, the logical-to-physical address translation is still done by processor hardware. Now the processor must know how to access the page table of the current process.
- Presented with a logical address (page number, offset), the processor uses the page table to produce a physical address (frame number, offset).

*The Word is the smallest unit of the memory. It is the collection of bytes. Every operating system defines different word sizes after analyzing the n-bit address that is inputted to the decoder and the 2 ^ n memory locations that are produced from the decoder.

Address Translation Scheme

- 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



  - For given logical address space 2^m and page size 2^n

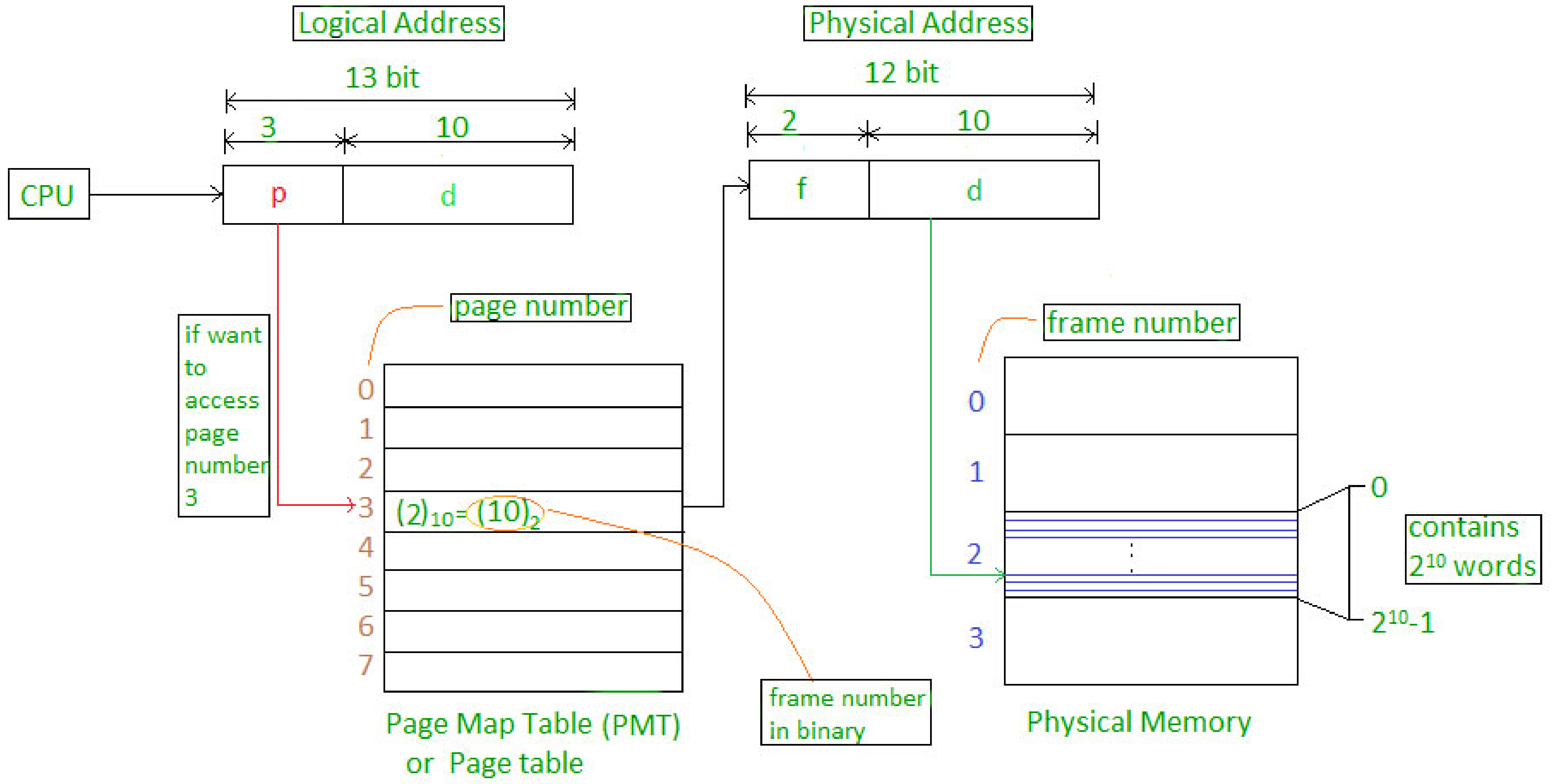
Address Translation Scheme

- Physical Address is divided into
  - **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
  - **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

Paging Hardware

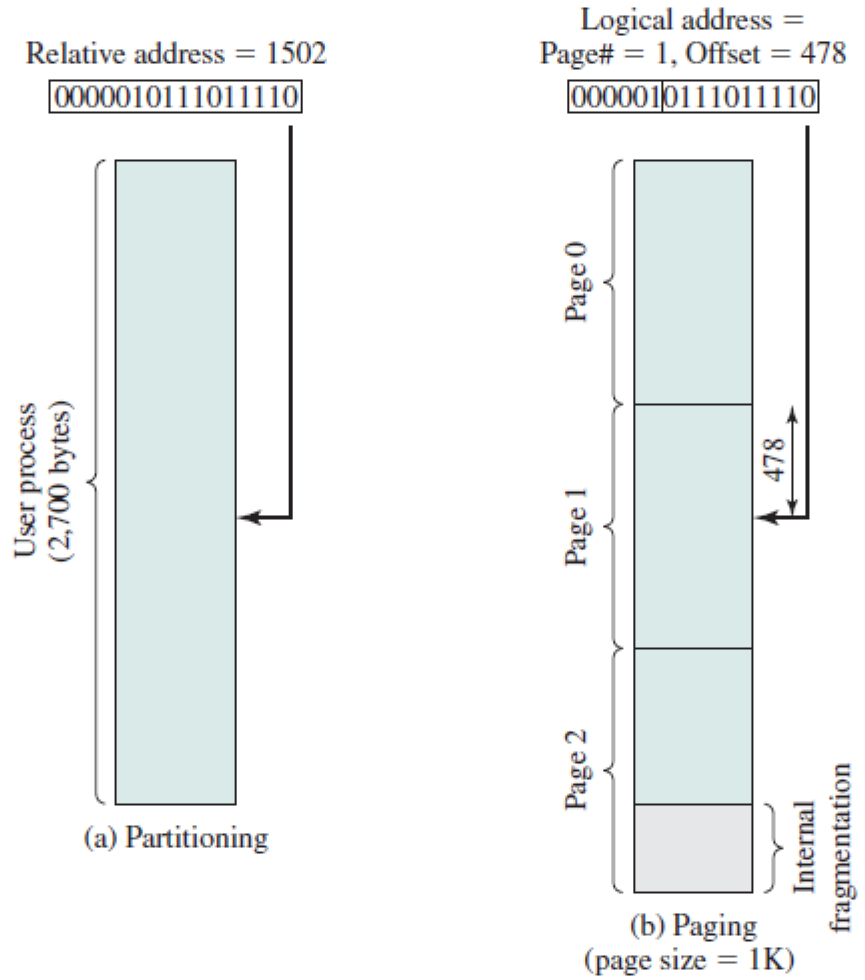
Number of frames = Physical Address Space / Frame size = 4 K / 1 K = 4 = 2^2

Number of pages = Logical Address Space / Page size = 8 K / 1 K = 8 = 2^3



Paging Example

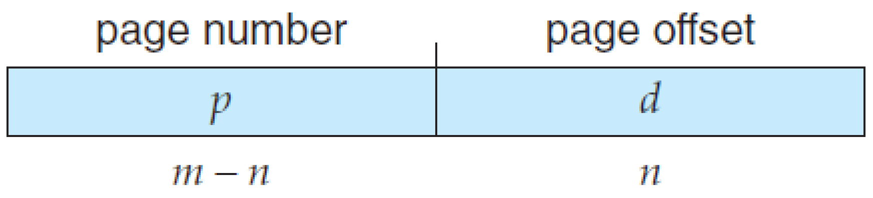
- 16-bit addresses are used, and the page size is 1K 1,024 bytes. The relative address 1502, in binary form, is 0000010111011110.
- With a page size of 1K, an offset field of 10 bits is needed, leaving 6 bits for the page number.
- Thus a program can consist of a maximum of 2^6=64 pages of 1K bytes each.



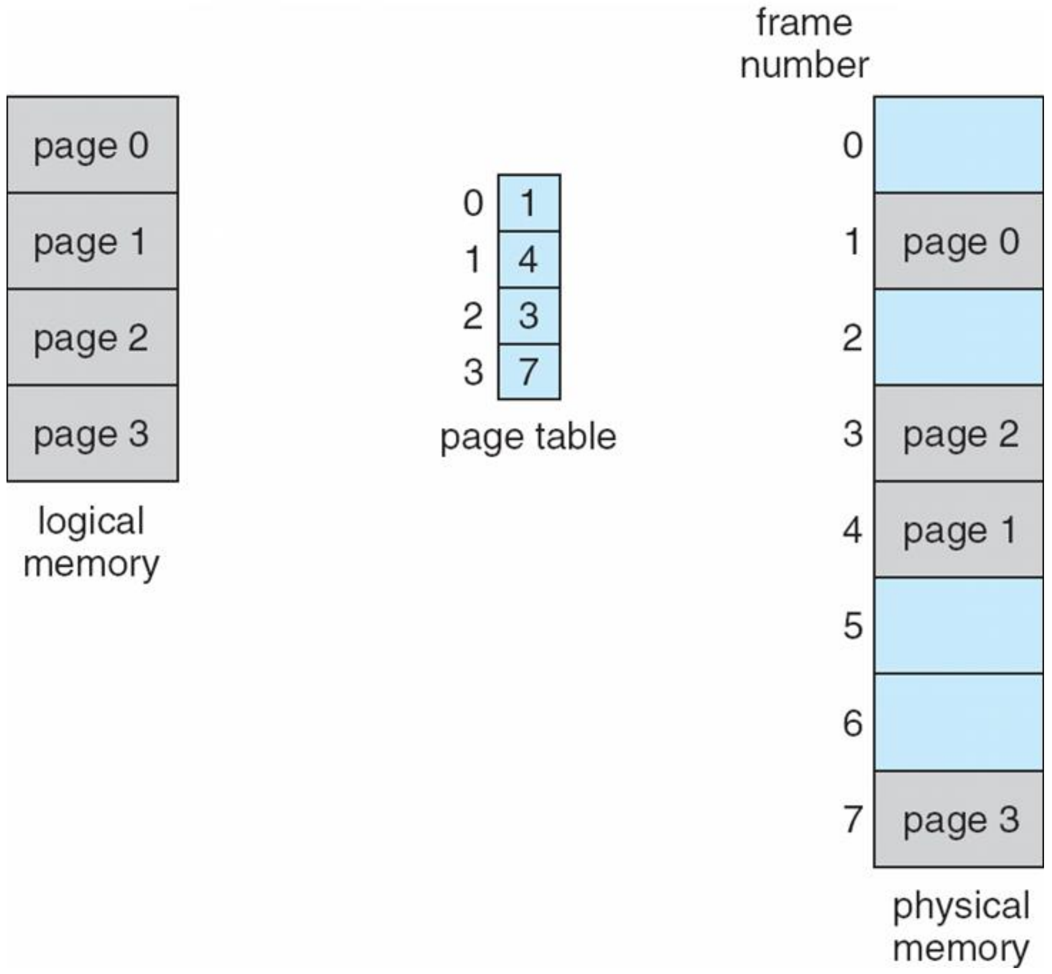
Paging

- The page table contains the base address of each frame in physical memory, and the offset is the location in the frame being referenced.
- Thus, the base address of the frame is combined with the page offset to define the physical memory address
- The following outlines the steps taken by the MMU to translate a logical address generated by the CPU to a physical address:
    1. Extract the page number ***p*** and use it as an index into the page table.
    2. Extract the corresponding frame number ***f*** from the page table.
    3. Replace the page number ***p*** in the logical address with the frame number ***f*** .

- As the offset ***d*** does not change, it is not replaced, and the frame number and offset now comprise the physical address



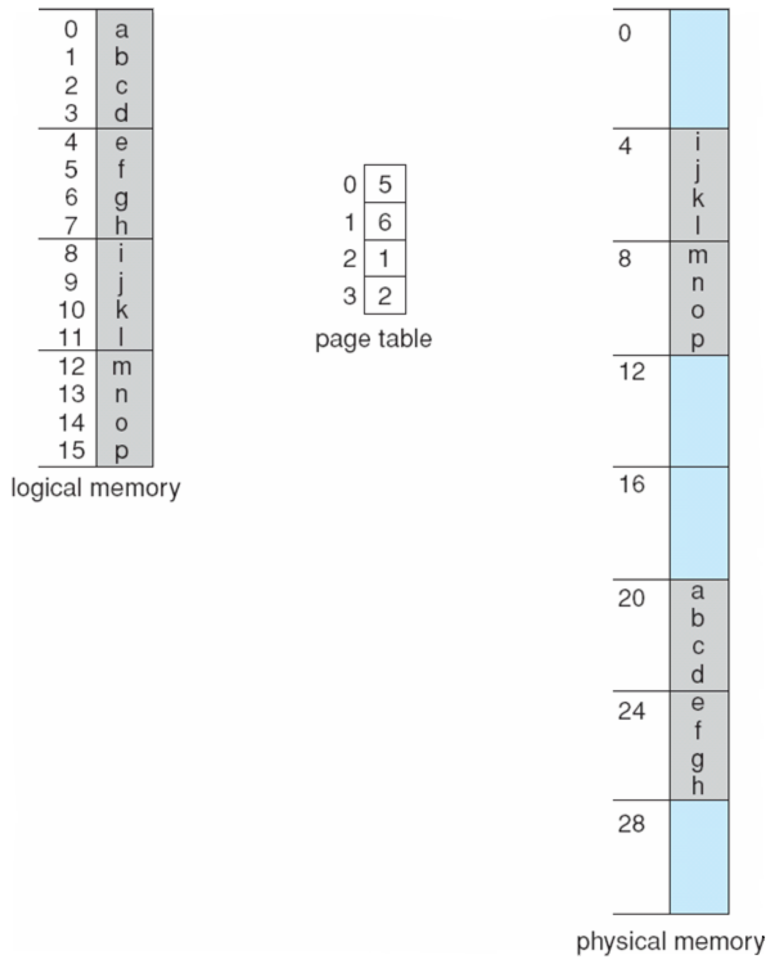
Paging Model of Logical and Physical Memory



Paging (Cont.)

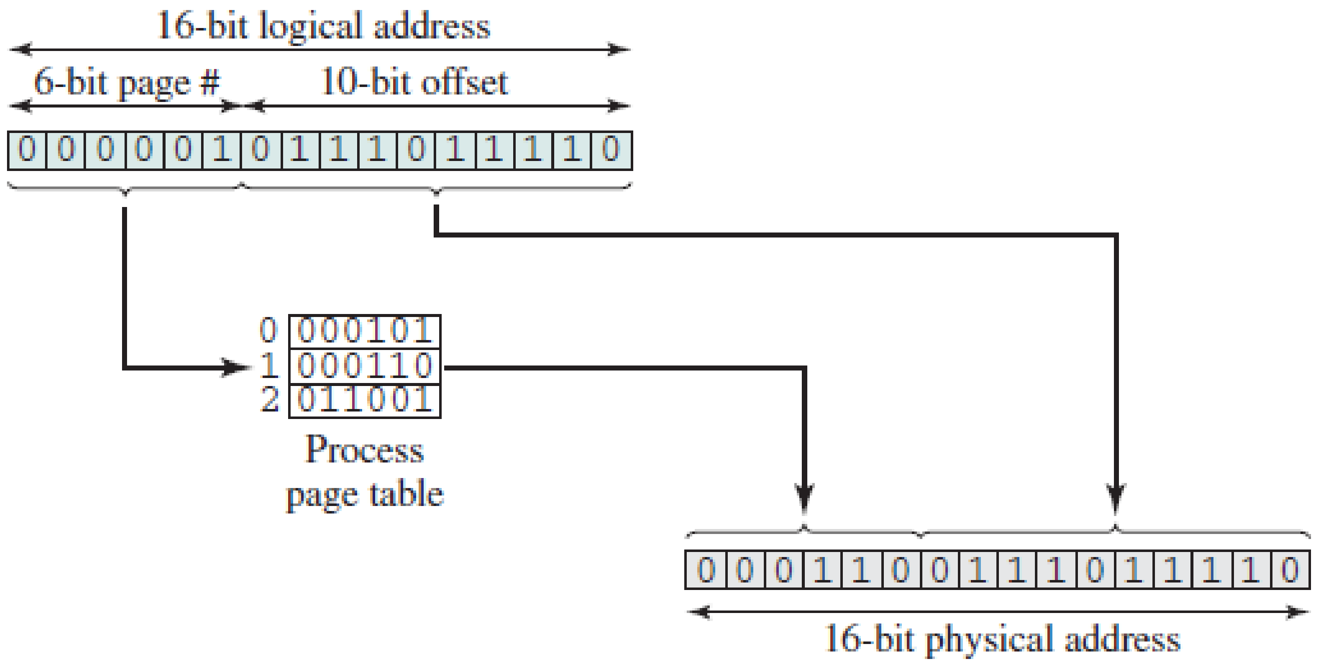
- When we use a paging scheme, we have no external fragmentation: any free frame can be allocated to a process that needs it.
- However, we may have some internal fragmentation.
- Notice that frames are allocated as units. If the memory requirements of a process do not happen to coincidewith page boundaries, the last frame allocated may not be completely full.
- For example, if page size is 2,048 bytes, a process of 72,766 bytes will need 35 pages plus 1,086 bytes.
- It will be allocated 36 frames, resulting in internal fragmentation of 2,048 − 1,086 = 962 bytes. In the worst case, a process would need n pages plus 1 byte. It would be allocated n + 1 frames, resulting in internal fragmentation of almost an entire frame.

Paging Example



n=2 and m=4   32-byte memory and 4-byte pages

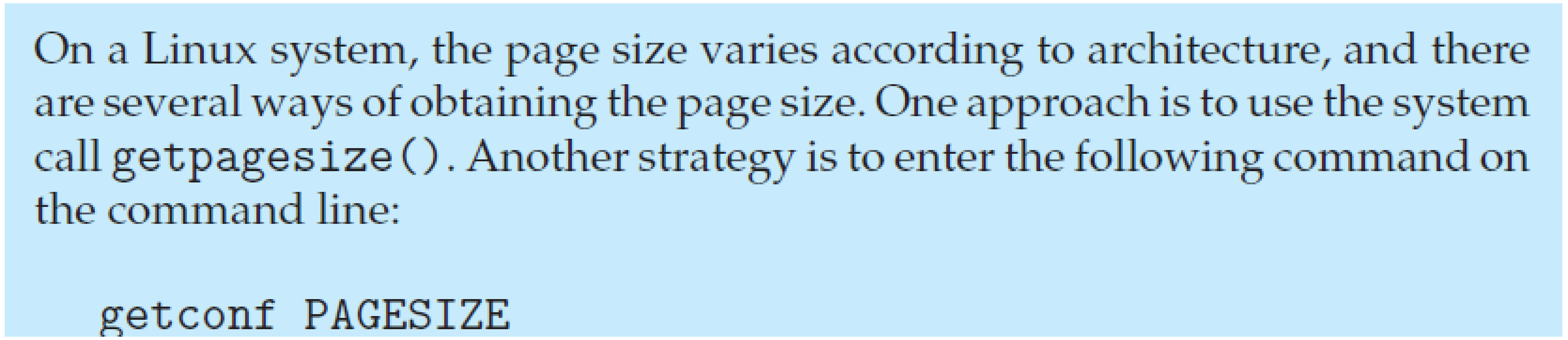
Paging Example



(a) Paging

Free Frames

- The pages are typically either 4 KB or 8 KB in size, and some systems support even larger page sizes.
- Some CPUs and operating systems even support multiple page sizes.
- For instance, on x86-64 systems,Windows 10 supports page sizes of 4 KB and 2 MB.
- Linux also supports twopage sizes: a default page size (typically 4 KB) and an architecturedependent larger page size called huge pages.



On a Linux system, the page size varies according to architecture, and there are several ways of obtaining the page size. One approach is to use the system call `getpagesize()`. Another strategy is to enter the following command on the command line:

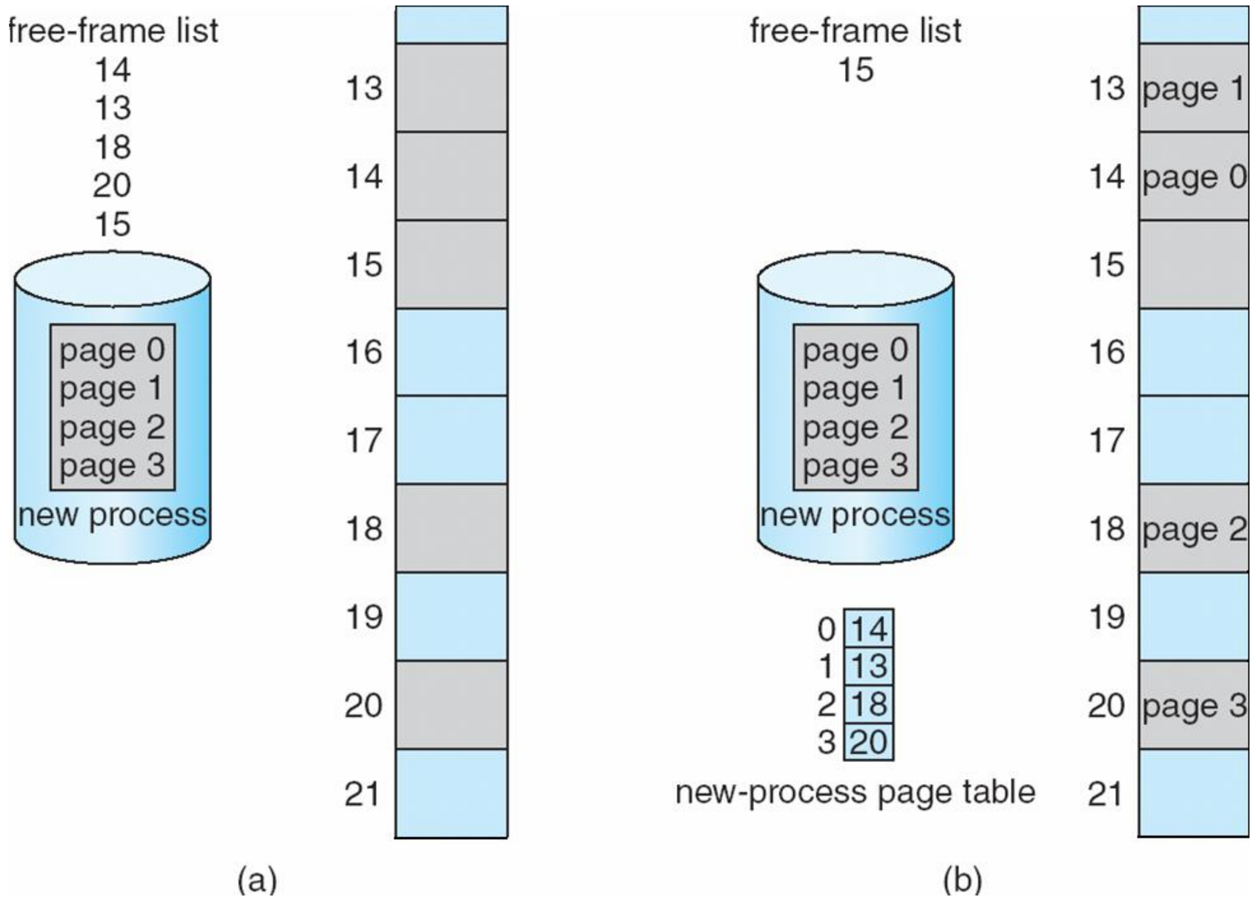
```
    getconf PAGESIZE
```

Free Frames

- Frequently, on a 32-bit CPU, each page-table entry is 4 bytes long, but that size can vary as well.
- A 32-bit entry can point to one of 232 physical page frames. If the frame size is 4 KB (212), then a system with 4-byte entries can address 244 bytes (or 16 TB) of physical memory.

Free Frames

- When a process arrives in the system to be executed, its size, expressed in pages, is examined.
- Each page of the process needs one frame. Thus, if the process requires ***n*** pages, at least ***n*** frames must be available in memory.
- If ***n*** frames are available, they are allocated to this arriving process. The first page of the process is loaded into one of the allocated frames, and the frame number is put in the page table for this process.
- The next page is loaded into another frame, its frame number is put into the page table, and so on



Before allocation

After allocation

Implementation of Page Table

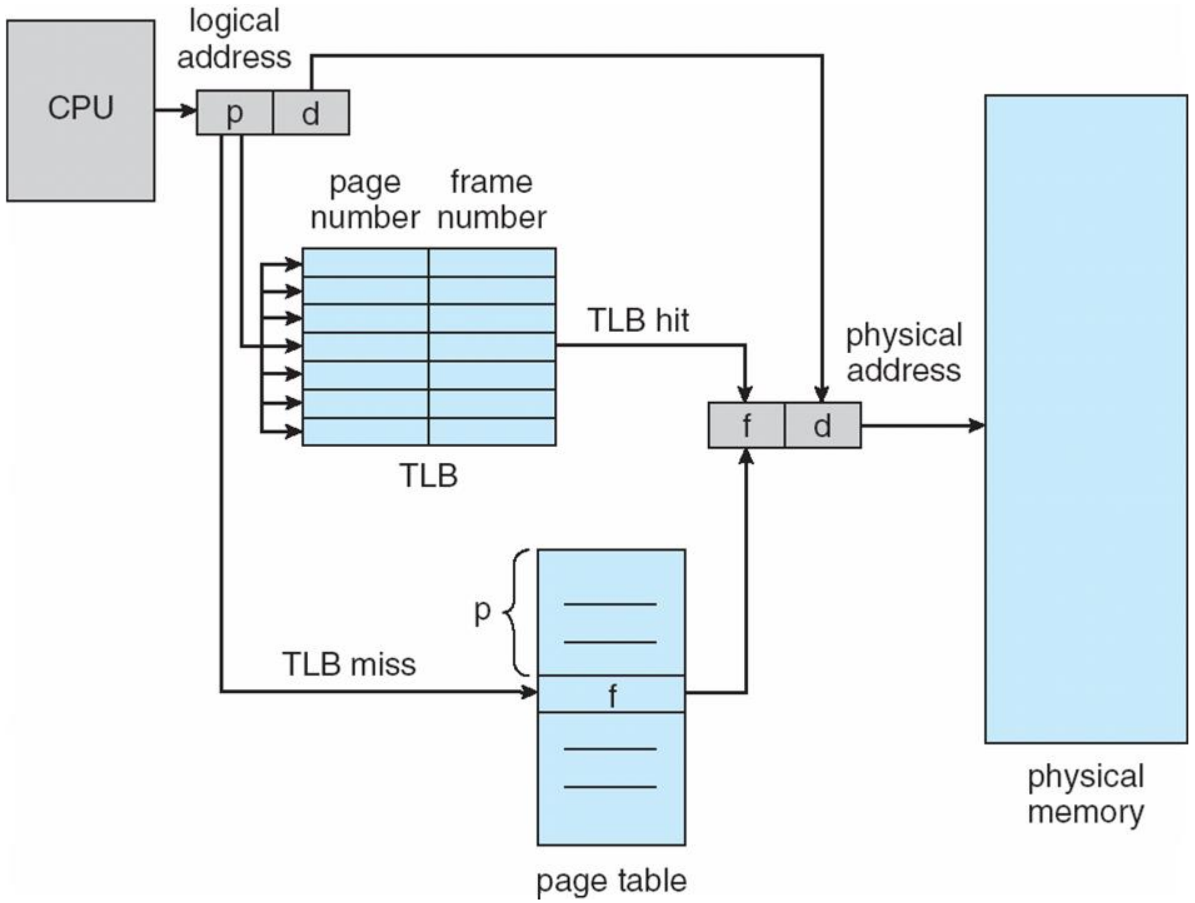
- The hardware implementation of the page table can be done in several ways.
- Page table is kept in main memory
- **Page-table base register** (**PTBR**) points to the page table
- **Page-table length register** (**PTLR**) indicates size of the page table
- In this scheme every data/instruction access 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 fast-lookup hardware cache called **associative memory** or **translation look-aside buffers** (**TLBs**)

Implementation of Page Table (Cont.)

- Some TLBs store **address-space identifiers** (**ASIDs**) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
  - Replacement policies must be considered
  - Some entries can be **wired down** for permanent fast access

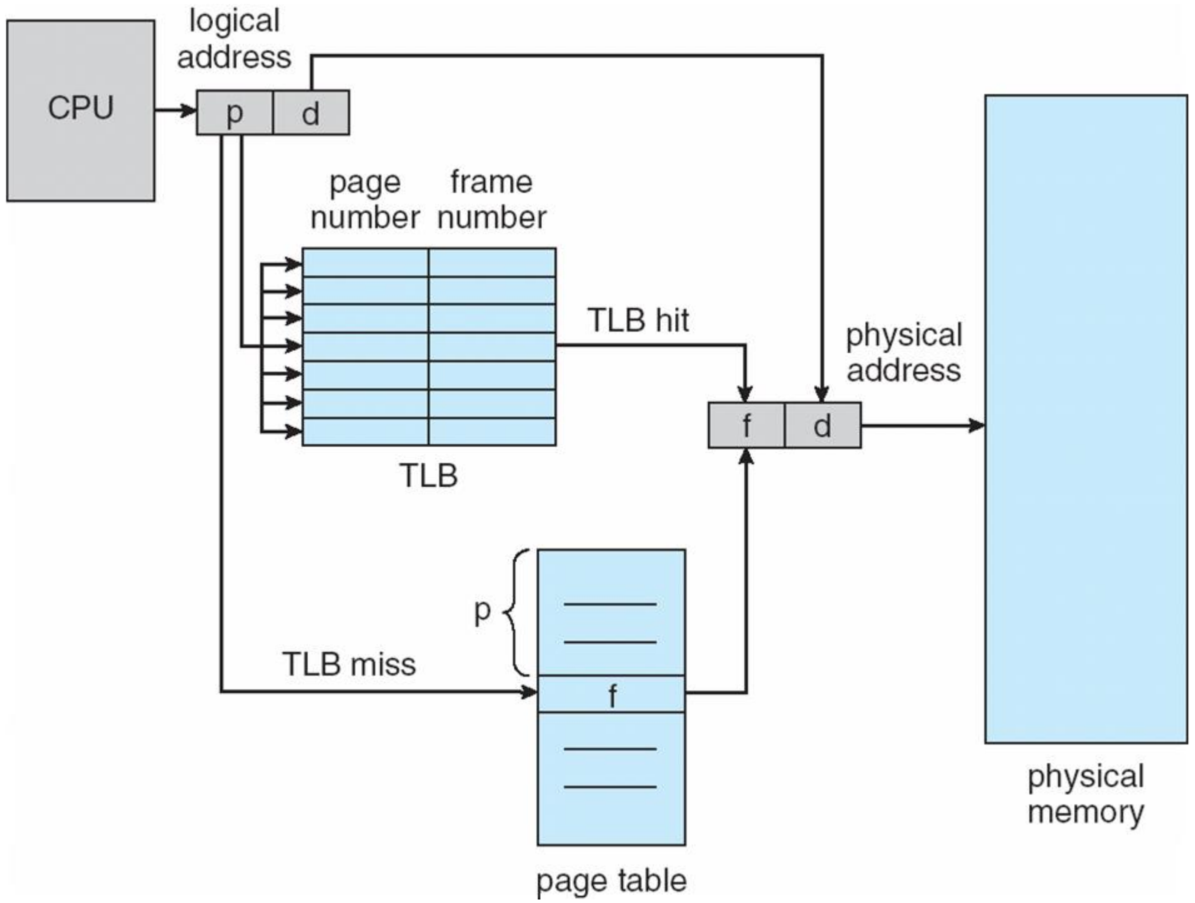
Paging Hardware With TLB

Although storing the page table in main memory can yield faster context switches, it may also result in slower memory access times. Suppose we want to access location i. We must first index into the page table, using the value in the PTBR offset by the page number for i. This task requires one memory access. It provides us with the frame number, which is combined with the page offset to produce the actual address.We can then access the desired place in memory. With this scheme, two memory accesses are needed to access data (one for the page-table entry and one for the actual data). Thus, memory access is slowed by a factor of 2, a delay that is considered intolerable under most circumstances.



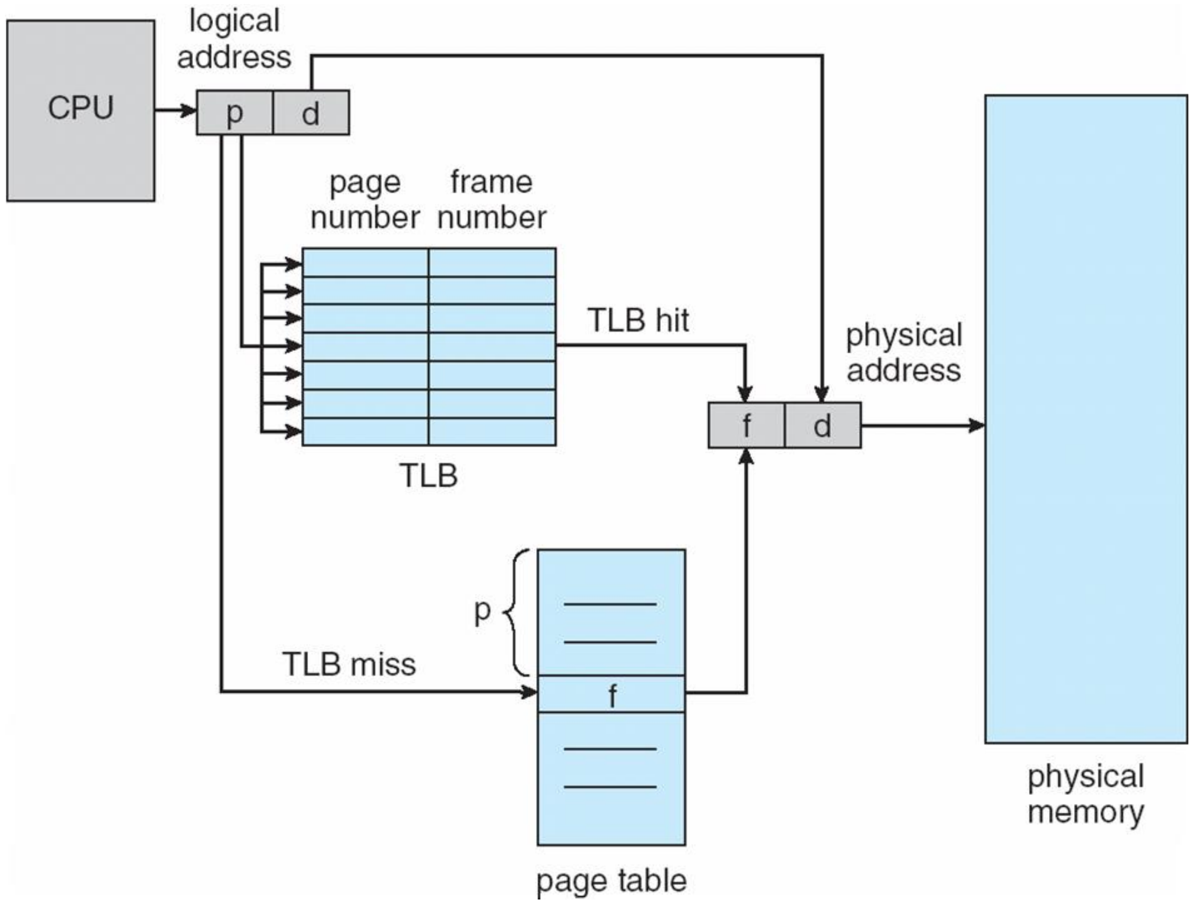
Paging Hardware With TLB

- The standard solution to this problem is to use a special, small, fast-lookup hardware cache called a translation look-aside buffer (TLB). The TLB is associative, high speed memory.
- Each entry in the TLB consists of two parts: a key (or tag) and a value.
- When the associative memory is presented with an item, the item is compared with all keys simultaneously. If the item is found, the corresponding value field is returned.
- To be able to execute the search within a pipeline step, however, the TLB must be kept small.



Paging Hardware With TLB

- The TLB is used with page tables in the following way. The TLB contains only a few of the page-table entries.
- When a logical address is generated by the CPU, the MMU first checks if its page number is present in the TLB.
- If the page number is found, its frame number is immediately available and is used to access memory. As just mentioned, these steps are executed as part of the instruction pipeline within the CPU,.



Associative Memory

- Associative memory – parallel search

Page #	Frame #

- Address translation (p, d)
  - If p is in associative register, get frame # out
  - Otherwise get frame # from page table in memory