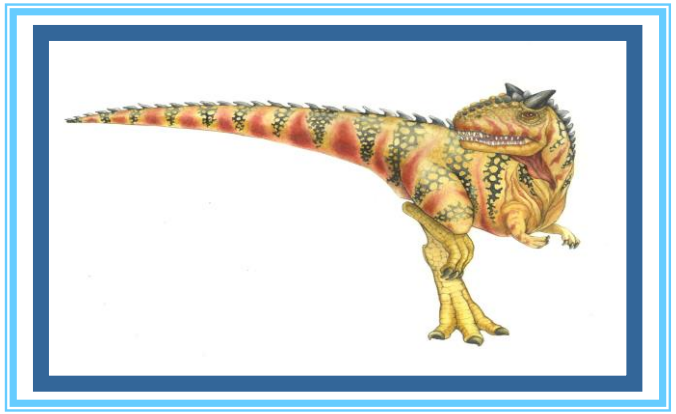
Xi'an Jiaotong-Liverpool University
西交利物浦大學

CPT104 - Operating Systems Concepts

MEMORY MANAGEMENT

Main Memory

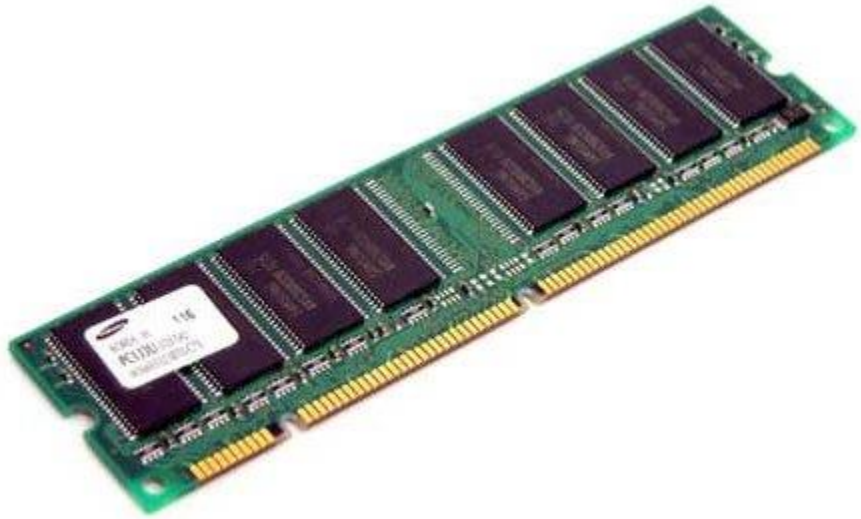


Memory Management: Main Memory

- Memory Management Unit (MMU)
- Swapping
- Contiguous Memory Allocation
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging
  - Structure of the Page Table

Random Access Memory RAM – Main Memory

The **memory** is where the programs are kept when they are running; it also contains the data needed by the running programs.



Random Access Memory RAM consists of a number of integrated circuits mounted on one or more circuit boards (*memory modules*) that plug into specialised slots on the mainboard.

Goals and Tools of Memory management

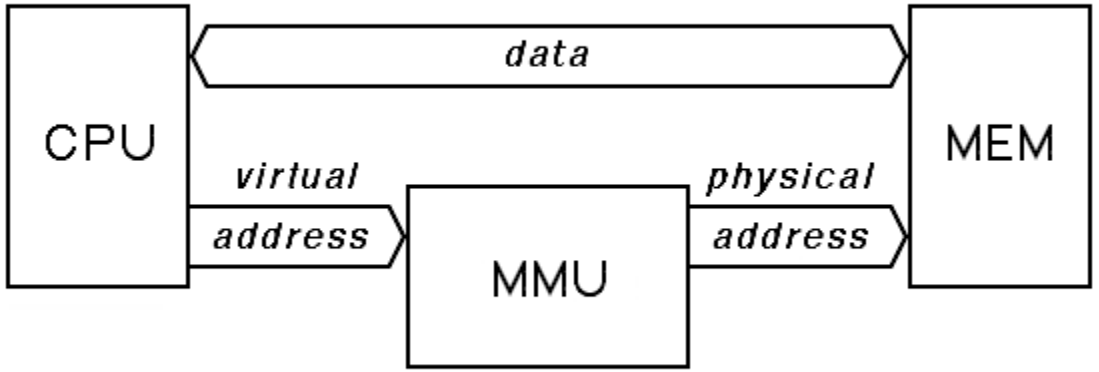
- **Allocate memory resources among competing processes,**
  - maximizing memory utilization and system throughput
- **Protection** - provides isolation between processes.
- **Relocation** - the ability to move process around in memory without affecting its execution.
- **Sharing** - OS has to allow sharing, while at the same time ensure protection.
- **Logical** and **Physical Organization** of memory - main memory in a computer system is organized as a linear, or one-dimensional, address space, consisting of a sequence of bytes or words.

Tools

- Base and limit registers
- Swapping
- Segmentation
- Paging, page tables and translation lookaside buffer (TLB)
- Virtual memory

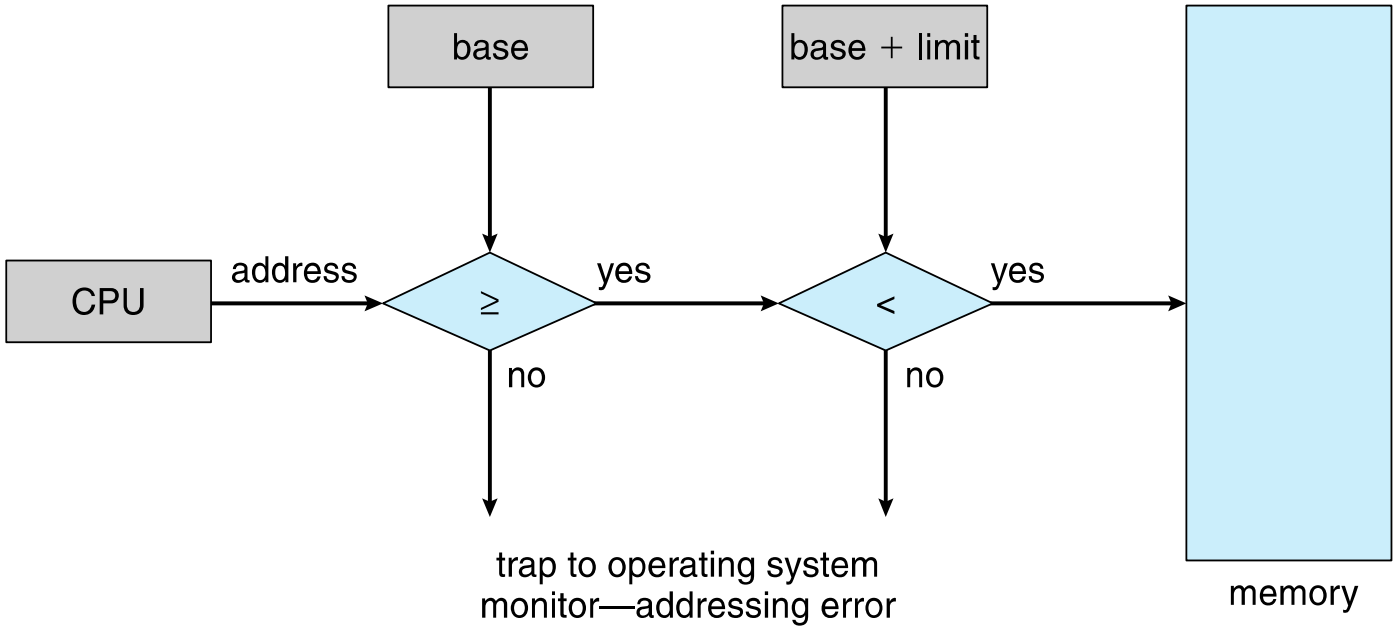
Memory Management Unit (MMU)

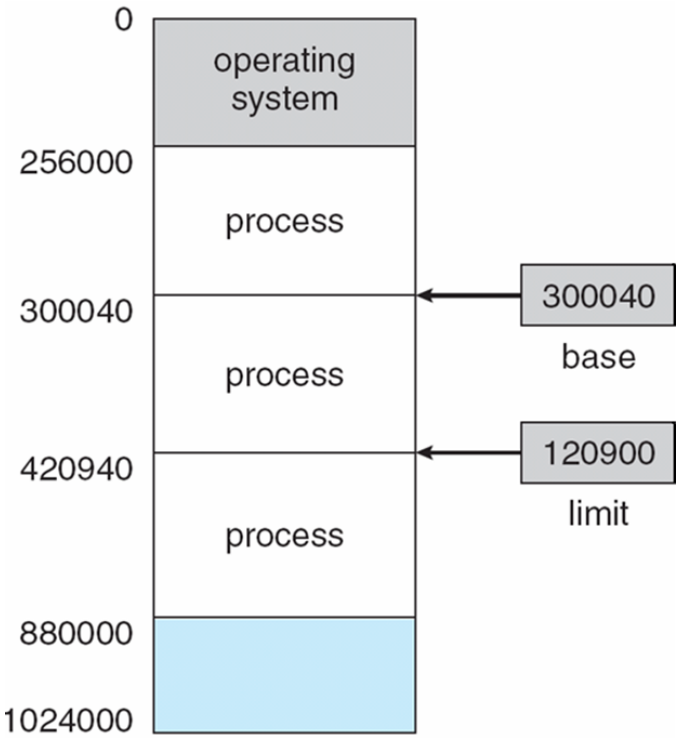
- The memory addresses generated by the CPU when accessing the code and data of a running process are called **virtual/logical addresses**. The virtual address space of a process ranges from 0 to a maximum value that depends on the CPU architecture (4GB for 32-bit addresses).
- The addresses used by the actual memory hardware to locate information are called **physical addresses**. The physical address space of a system spans from 0 to a maximum value (determined by how much RAM the machine has).
- The **MMU** performs two primary functions:
  - translates virtual addresses into physical addresses,
  - controls memory access permissions.



Virtual/logical address space

- A pair of **base** and **limit registers** define the *logical address space.*
  - *Limit is checked on all memory references*
  - *Base is automatically added to all addresses*
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



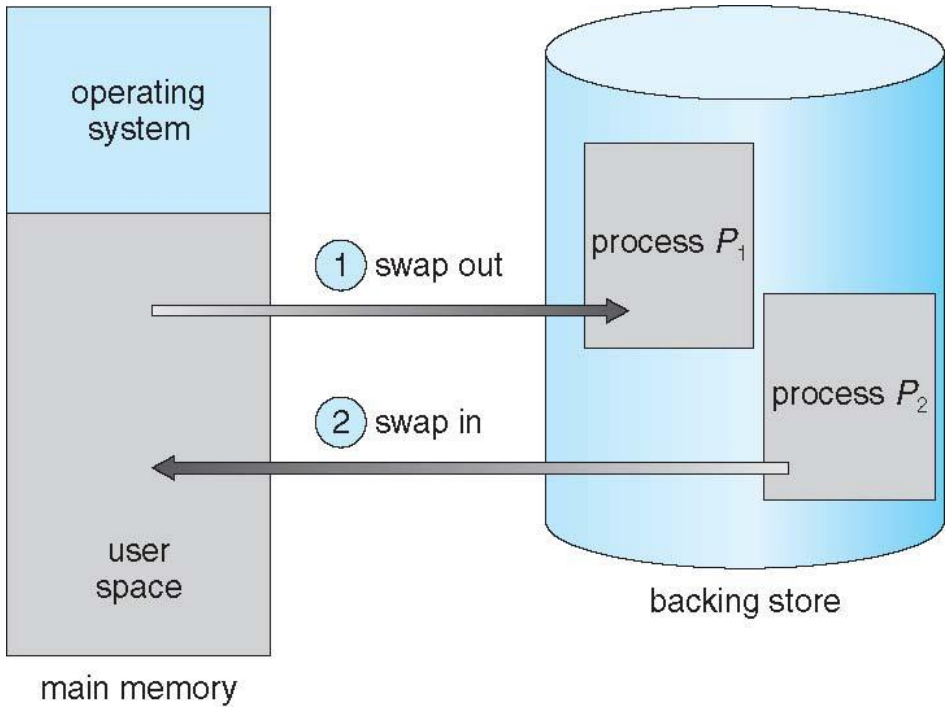


Dynamic relocation using a relocation register

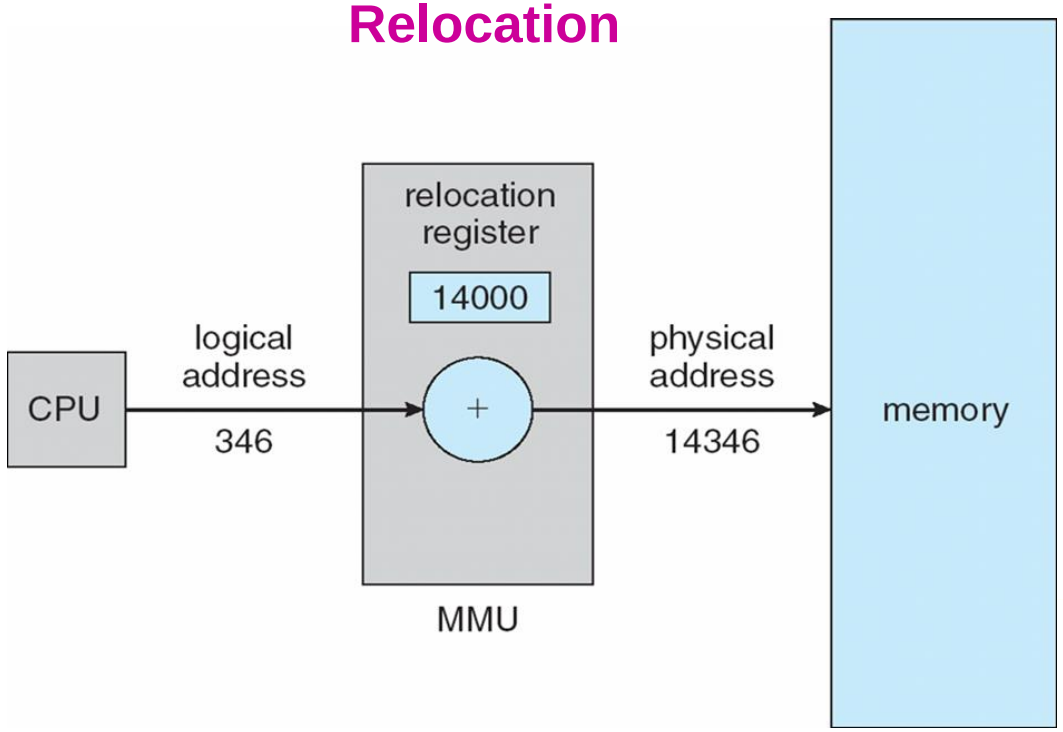
What if not enough memory to hold all processes?

- **Relocation:** moving a program from one area of memory to another area. In MMU scheme, the value in the **relocation register** is **added to every address generated by a user process** at the time it is sent to memory.
- The *base register* is now termed a **relocation register.**

Swapping



Relocation



Logical Organization of Memory:

Allocation

The memory allocation can be classified into two methods:

- **contiguous memory allocation** - assigns consecutive memory blocks to a process.
- **non-contiguous memory allocation** - assigns different blocks of memory in a nonconsecutive manner to a process.

CONTIGUOUS MEMORY ALLOCATION

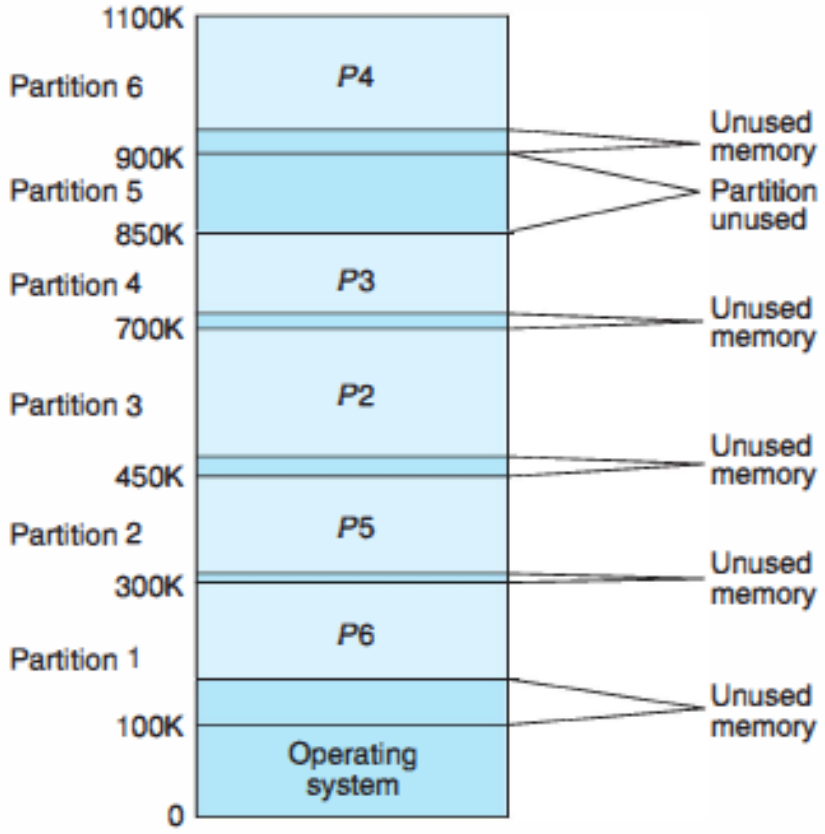
Early computers

Relevant: PDAs, smartcards

CONTIGUOUS ALLOCATION

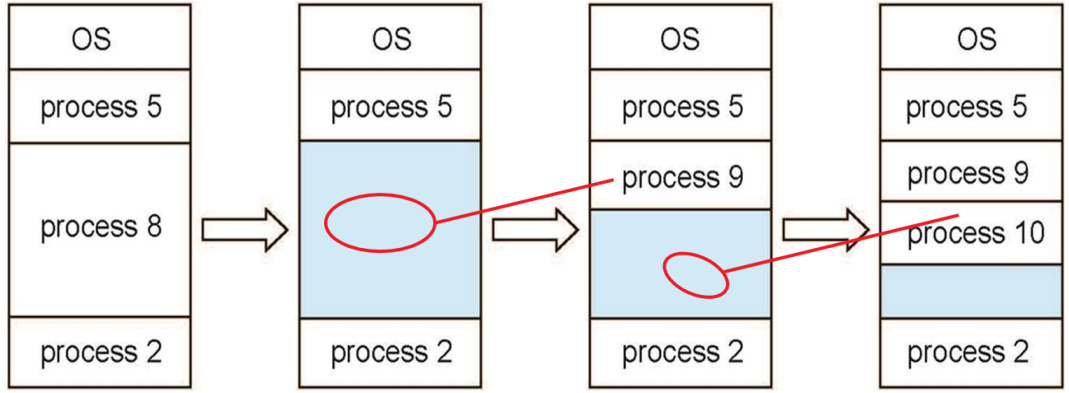
Fixed/Static Partitioning

- fixed partitions can be of *equal* or *unequal* sizes.
- assign each process to their own partition.



Variable/Dynamic Partitioning

- the operating system keeps a table indicating which parts of memory are available and which are occupied.
- list of free memory blocks **(holes)** to find a hole of a suitable size whenever a process needs to be loaded into memory.



PLACEMENT ALGORITHMS:

First-fit: Allocate the first hole that's big enough

Best-fit: Allocate the smallest hole that's big enough

Worst-fit: Allocate the largest hole

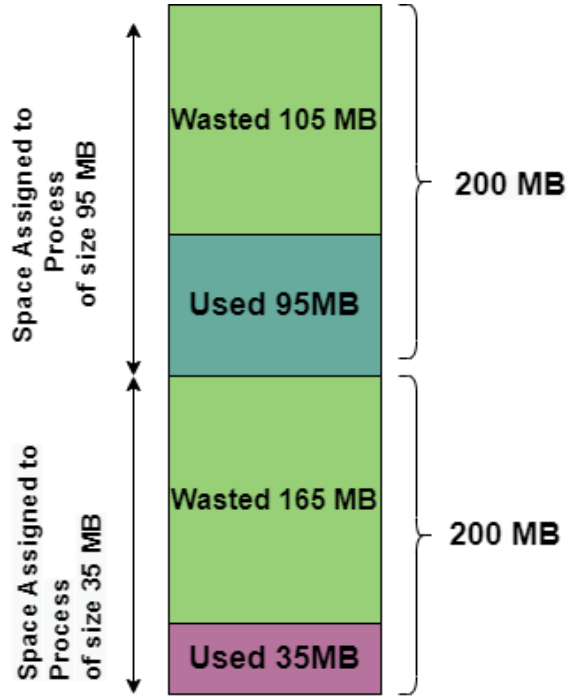
Fragmentation

Internal fragmentation

- occurs in fixed size blocks, because the last allocated process is not completely filled.

Solution:

- can be reduced by using variable sized memory blocks rather than fixed sized.



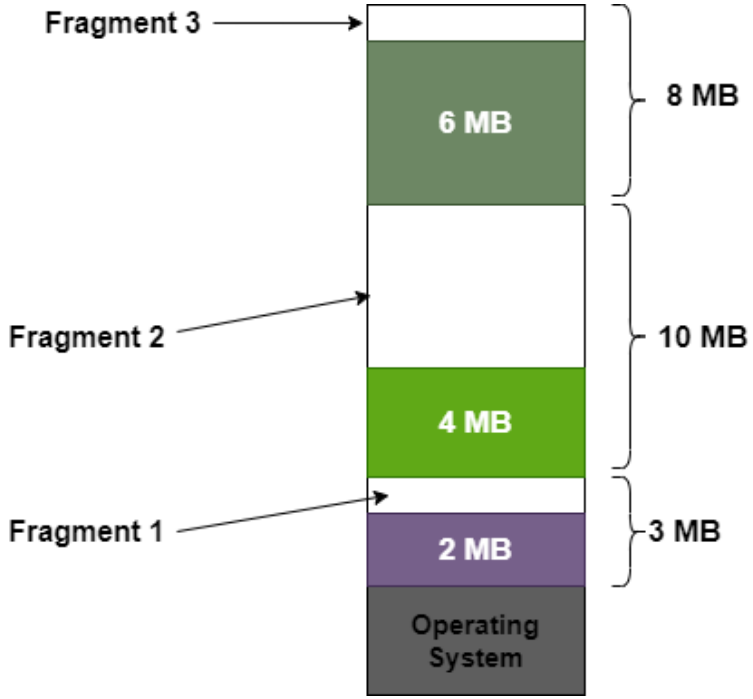
External fragmentation

- occurs with variable size segments, because some holes in memory will be too small to use.

Solutions:

Compaction - moving all occupied areas of storage to one end of memory. This leaves one big hole.

Non-contiguous memory allocation: **Segmentation** and **Paging**.



NON-CONTIGUOUS MEMORY ALLOCATION

Segmentation and Paging

Modern PCs

NON-CONTIGUOUS MEMORY ALLOCATION

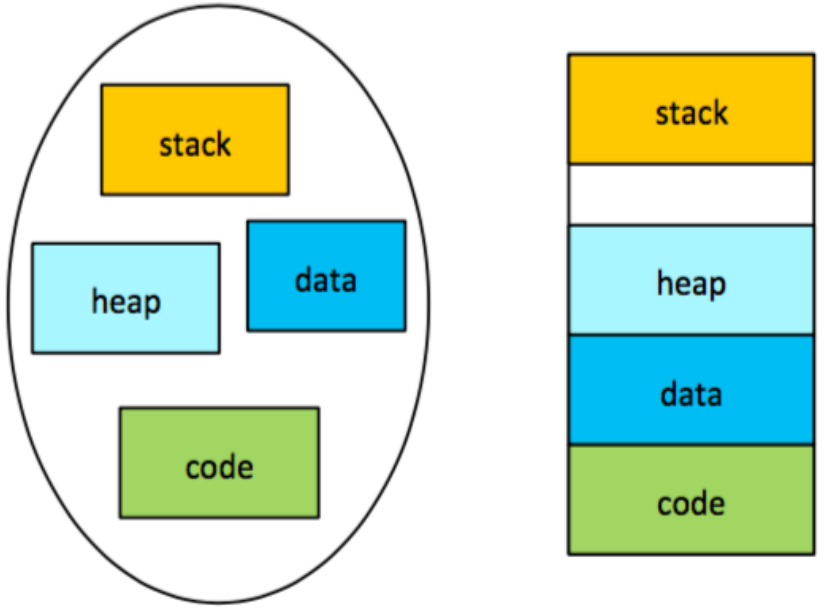
Segmentation

Modern PCs

SEGMENTATION

The segments are logical divisions of a program, and they may be of different sizes.

Each segment has a *name* and a *length*.



- *Logical address* consists of a two tuple:

  <segment-number, offset> = <s, d>

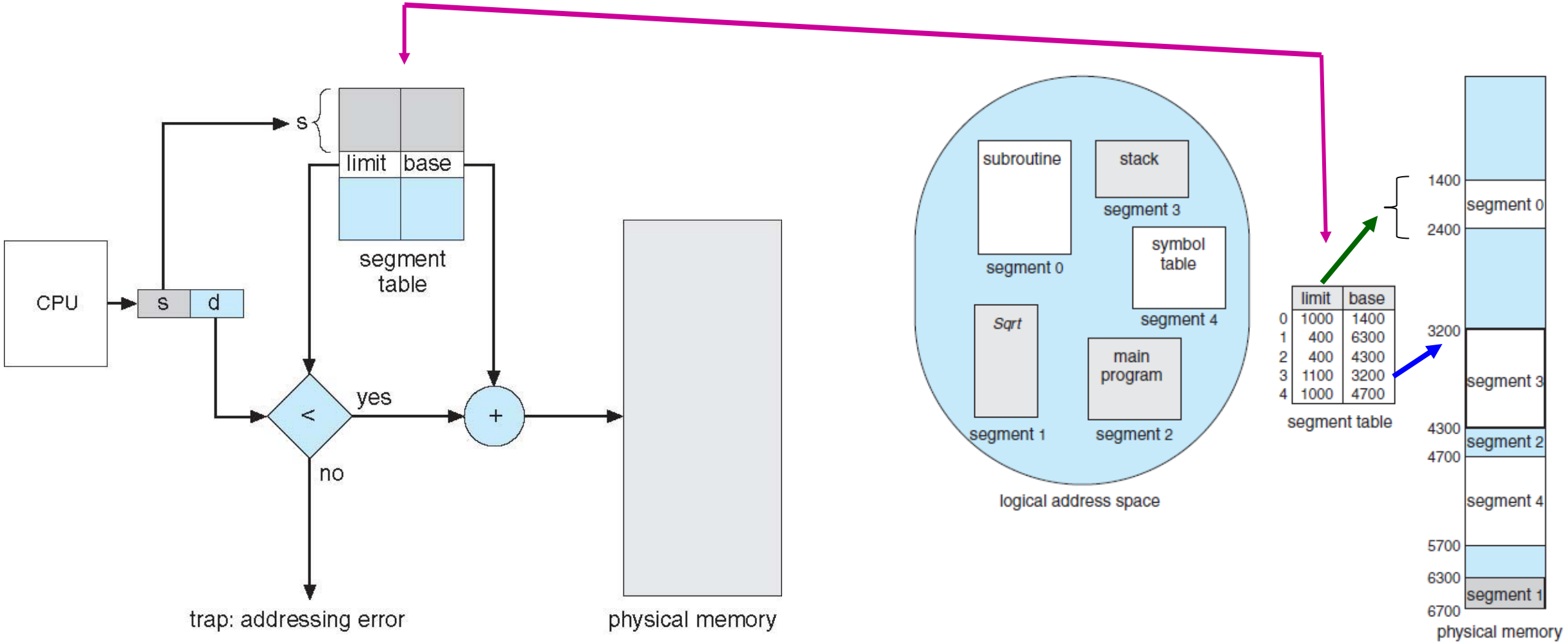
- A logical address space is a collection of segments.

- **Segment table** – maps two-dimensional logical address to one-dimensional physical address;
- Each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – the length of the segment

SEGMENTATION

The starting (base) addresses of all the segments are stored in a table, known as **segment table**

A **segment table** is a data structure used to store the base addresses of each segment in the process, along with their limits.

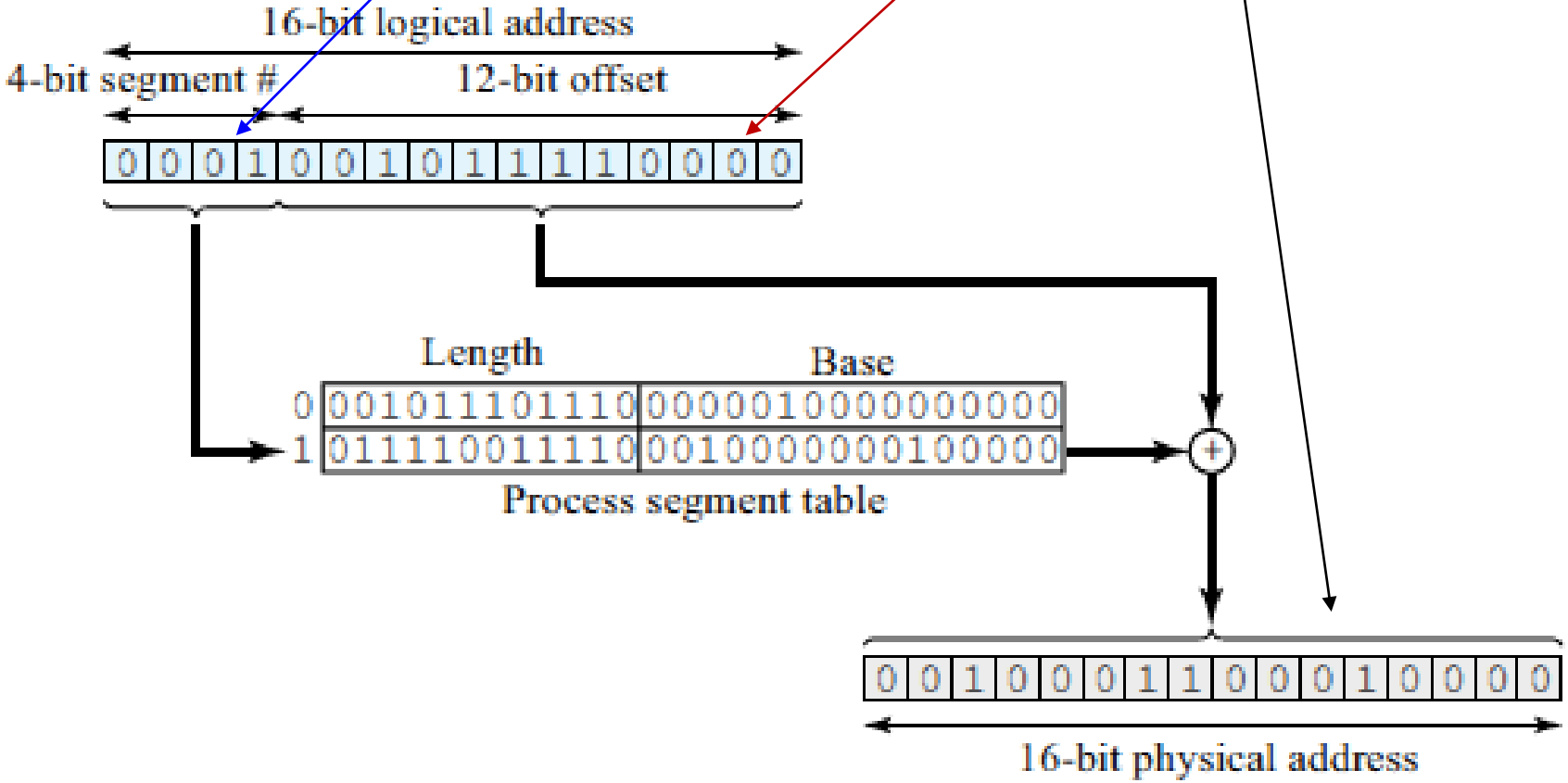


Segmentation - *Example*

Suppose the **logical address 0001001011110000**, which is **segment number 1**, **offset 752**.

Suppose this segment is residing in **main memory starting at physical address 0010000000100000**.

Then the physical address is **0010000000100000 + 001011110000 = 0010001100010000**



NON-CONTIGUOUS MEMORY ALLOCATION

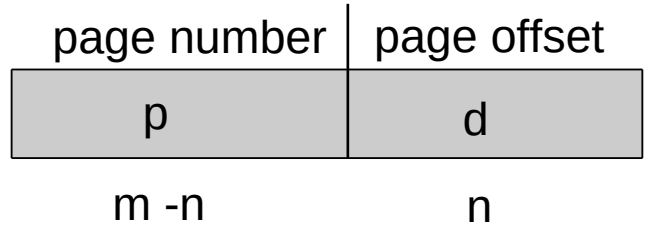
Paging

Modern PCs

PAGING

- Memory is divided into equal-size partitions called **frames**.
- **Logical address (space)** is divided into blocks of same size as frames called **pages** of a process
- All the pages of the process to be executed are loaded into any available frame in the memory.
- **Size of a page is power of 2** (512 bytes to 1GB per page).
- The processor generates a **two-dimensional logical address**, (p, d) :

<page-number, offset>



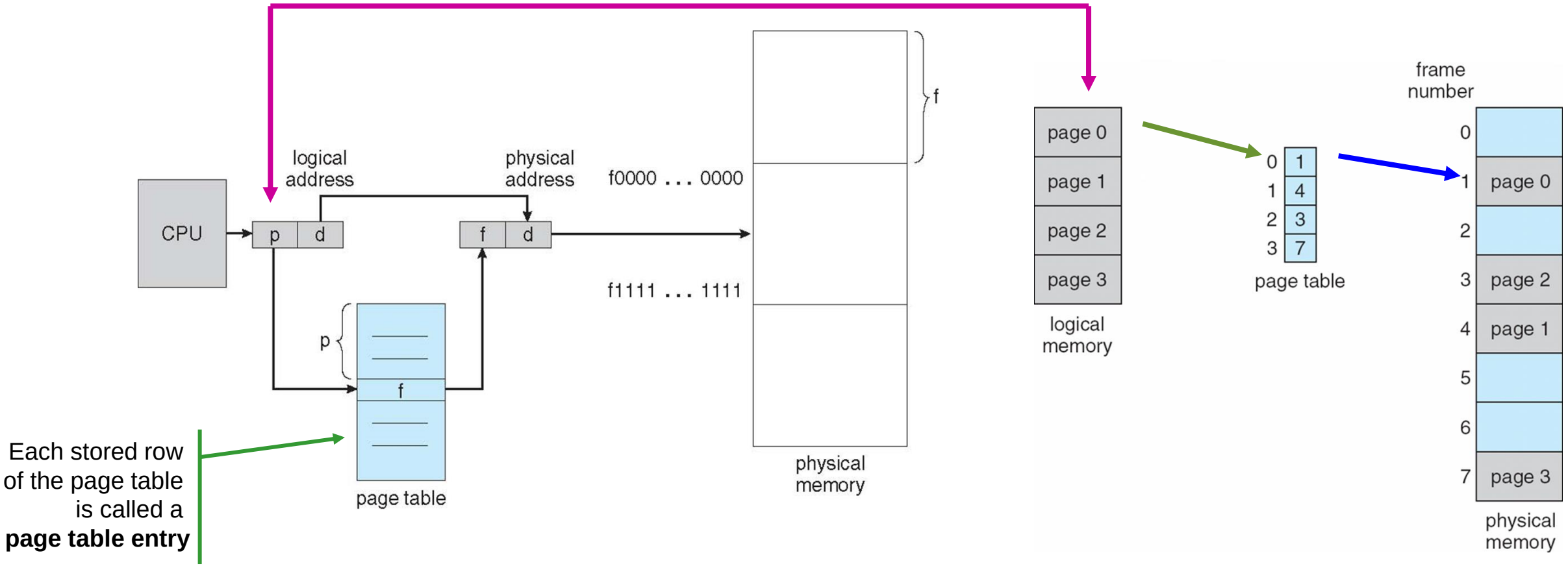
- The *start addresses of pages* are stored in the form of a table, known as a **page table**.

A **page table** is a data structure used to store the base addresses of each page in the process, that **maps** the **page number** referenced by the CPU to the **frame number** where that page is stored.

A **page table** enables the memory management unit (MMU) to translate logical addresses into physical addresses

PAGING

- **Page number** (*p*) – is extracted from the logical address and is used as an index in the **page table**
- The **base address *f***, corresponding to the page number, is retrieved.
- The base address f is added to the **page offset (d)** to get the corresponding physical address.



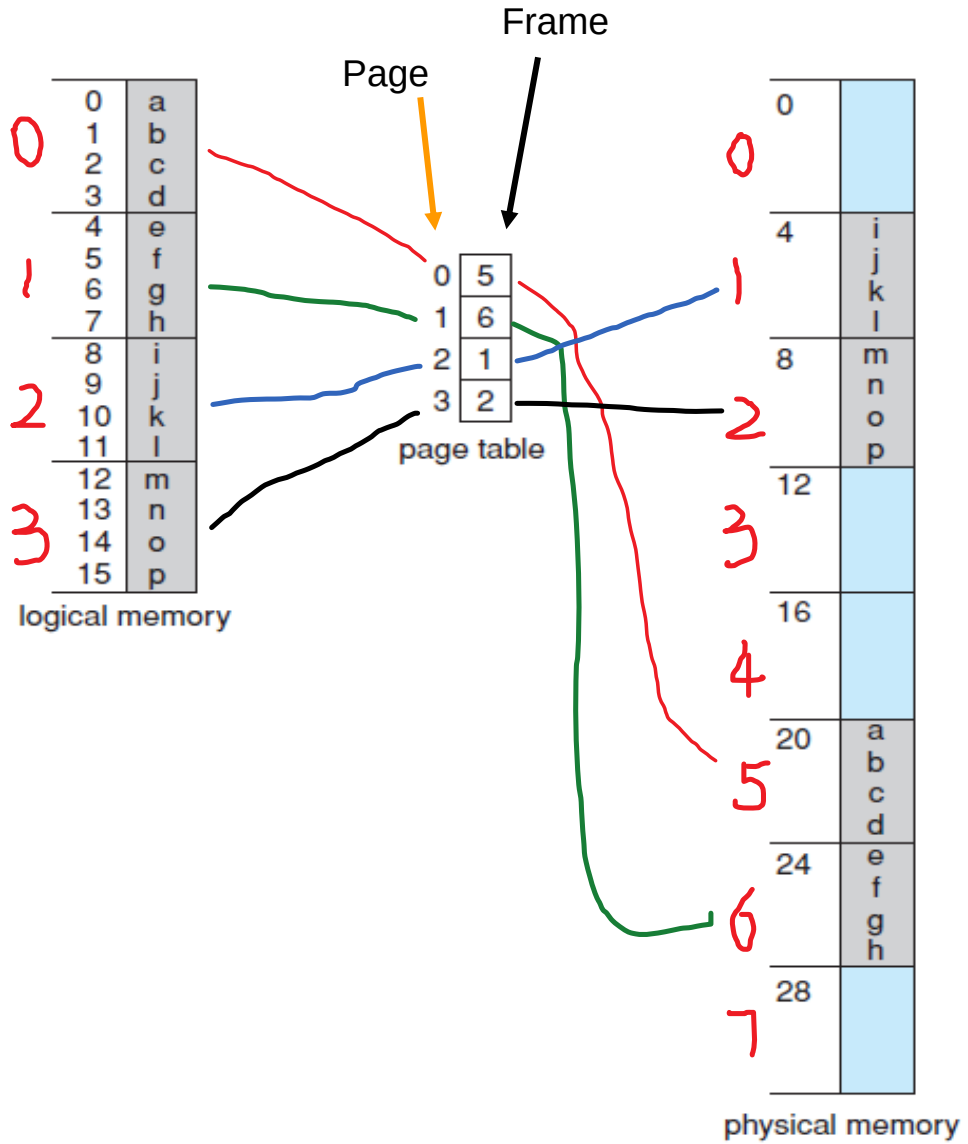
Paging *Example* for a 32-byte memory with 4-byte pages

- the size of the logical address space is **2^m = the process size**
- the **page size** is **2^n bytes**

page number	page offset
p	d
m -n	n

<page-number, offset>

<m - n, n>



Here, in the logical address: ***m* = 4 and *n* = 2;**

m - n = page number = 2

no. of pages = 2^4 / 2^2 = 2^2 = **4 pages** – **0, 1, 2, 3**

Number of pages the process is divided = Process size / Page size

page size = frame size = 4 bytes

32-byte physical memory

physical memory / frame size = 32 / 4 = 8 frames – 0, …, 7

Paging - *Example*

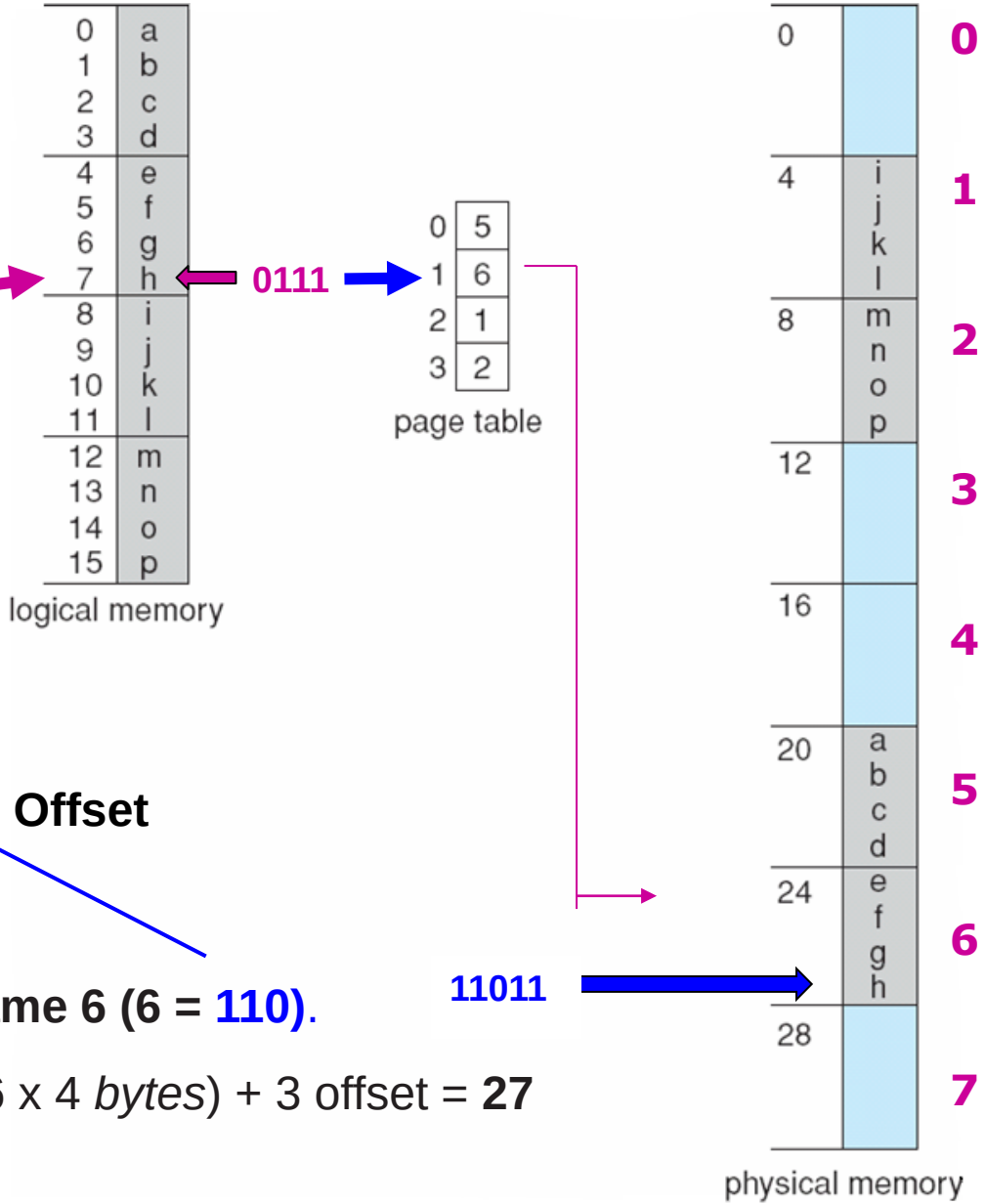
Page size = 4 bytes

Logical address space = 4 pages

Physical address space = 8 frames

Logical address 7: < 1, 3 > = **0111**

Physical address 27: < 6, 3 > = **11011**



Physical memory address is: **(Frame # * Page size) + Offset**

Logical address **7** (page 1 offset 3)

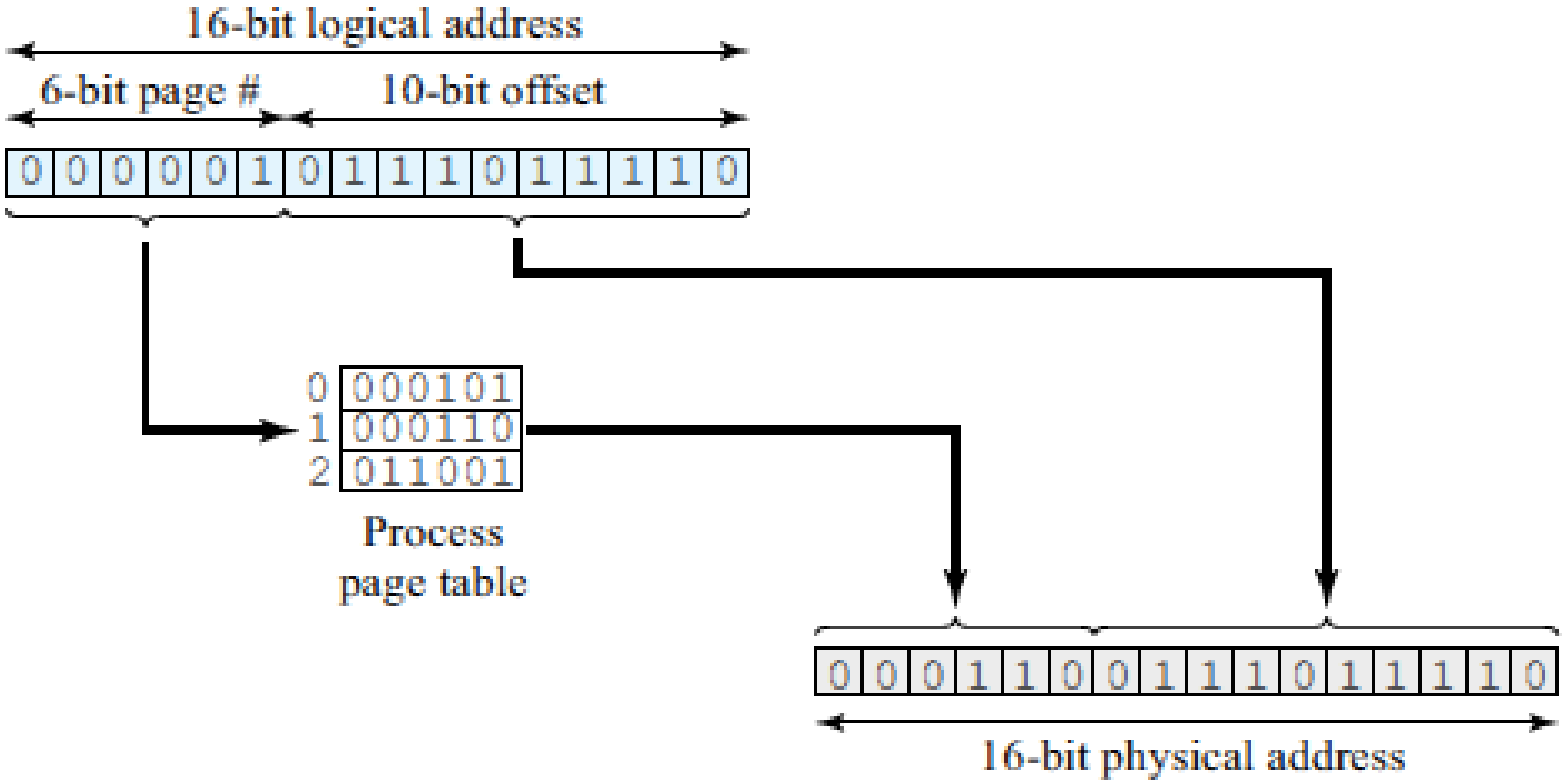
- according to the page table, **page 1** is mapped to **frame 6 (6 = 110)**.
- Physical memory address for logical address 7 is : (6 x 4 *bytes*) + 3 offset = **27**

Paging - *Example*

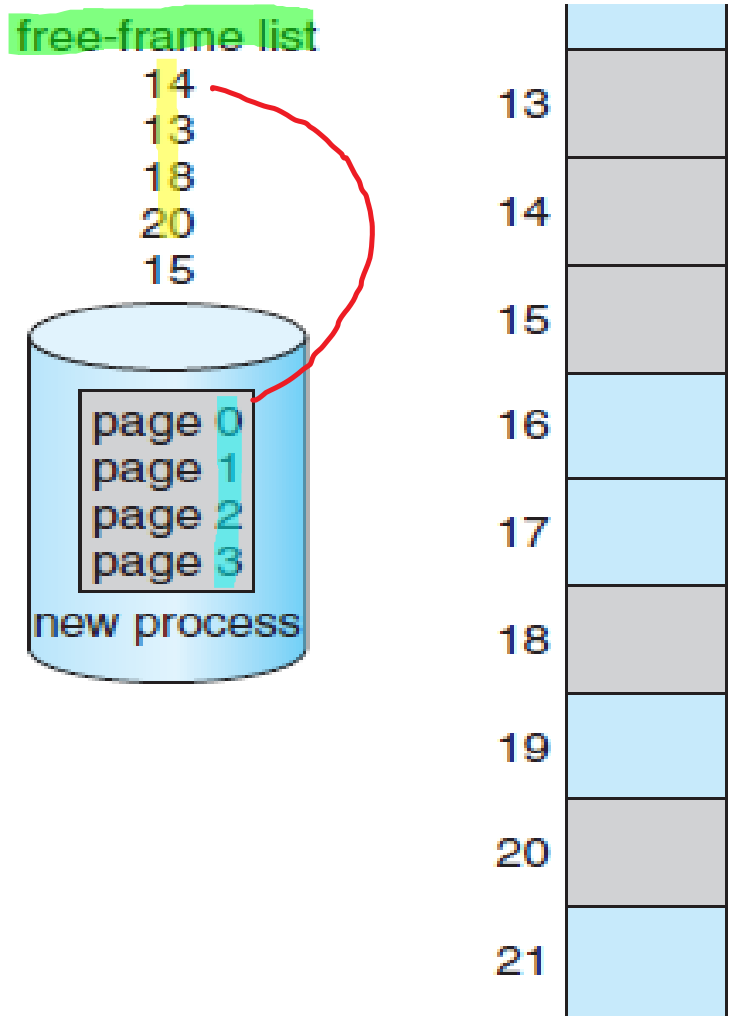
Suppose the **logical address 0000010111011110**, which is **page number 1**, **offset 478**.

Suppose this page is residing in main **memory frame 6 = binary 000110**.

Then the physical address is frame number 6, offset 478 = **0001100111011110**

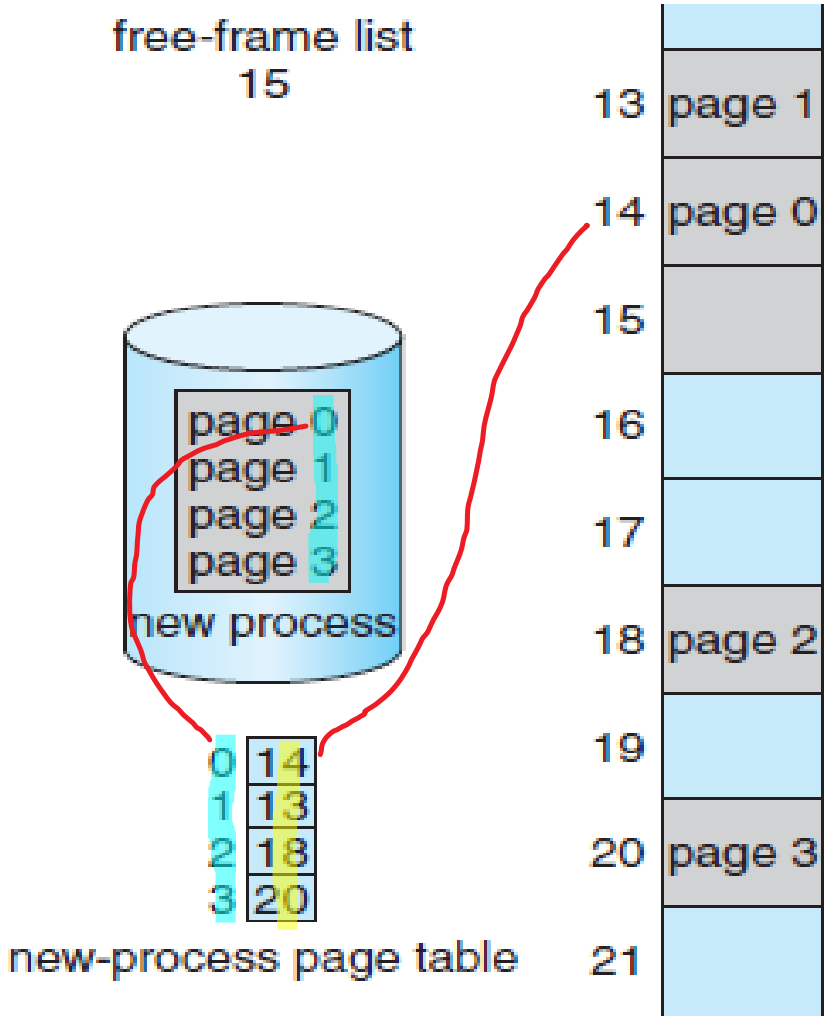


Free Frames Allocation



(a)

Before allocation



(b)

After allocation

Implementation of Page Table

Page table is kept in *main memory*

- **Page-table base register** (**PTBR**) points to the page table. hold the physical address of the base of the translation table maintained in main memory.
- **Page-table length register** (**PTLR**) indicates size of the page table
- Whenever a process is scheduled to be executed, the page table address from its PCB is loaded into PTBR, and the corresponding page table is accessed in the memory.
- When the current process is suspended or terminated, and another process is scheduled to execute, then the PTBR entry is replaced with the page table address of a new process… **context switch**

  TOTAL MEMORY ACCESS TIME = TIME TO ACCESS PAGE TABLE + TIME TO ACCESS MEMORY LOCATION

- The two-memory access problem can be solved by the use of a special **fast-lookup hardware cache** called **Translation Look-aside Buffers** (**TLBs**)

Paging Hardware With TLB

A **translation look-aside buffer (TLB)** is a *cache memory* that stores *(*temporary) the recent translations of virtual memory to physical memory.

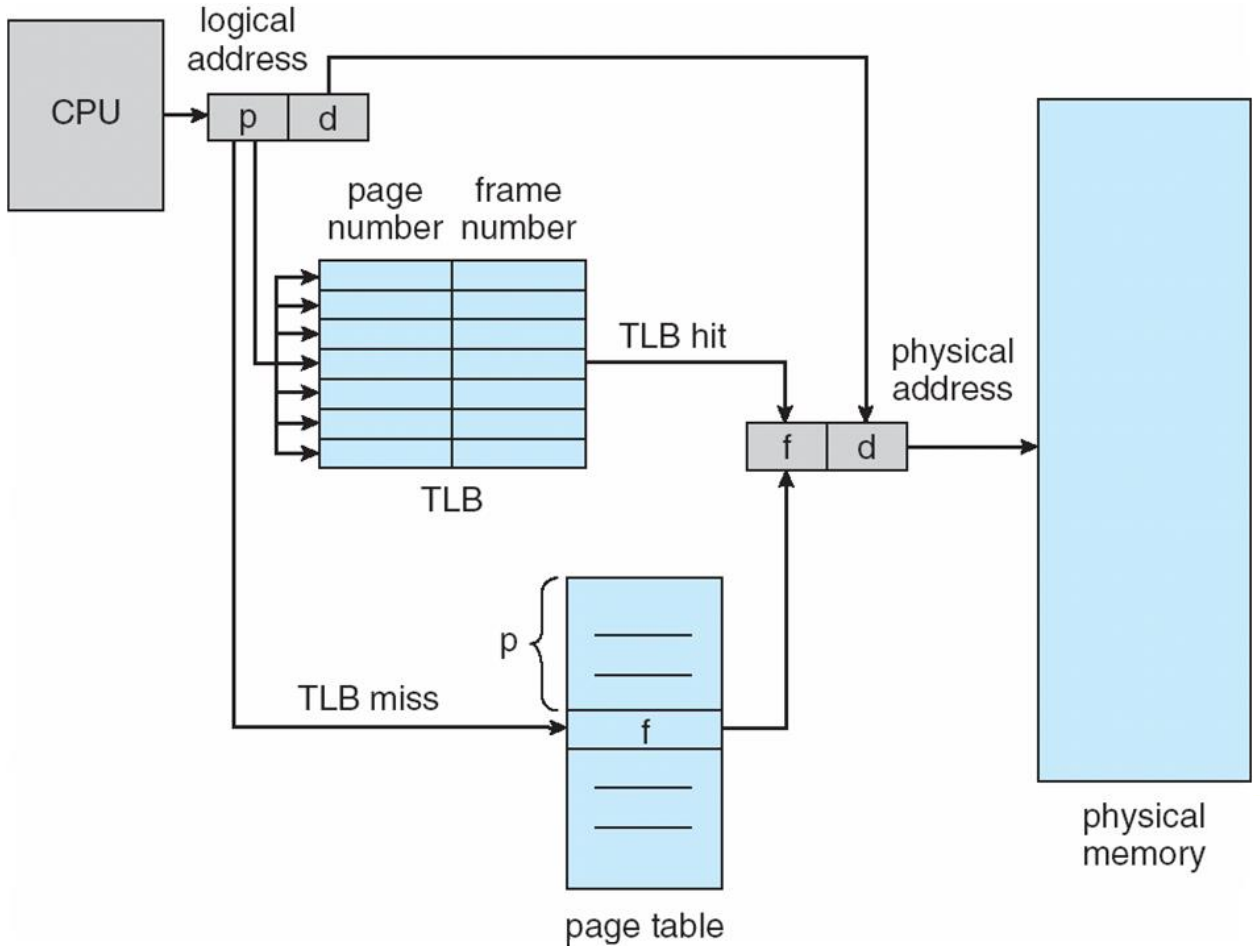
The percentage of times that the page number of interest is found in the TLB is called the **hit ratio**.

- **TLB lookup time (E)**,
- **memory access time (M)** and
- **hit ratio (A)**

EFFECTIVE MEMORY-ACCESS TIME (EAT) - estimation of the impact of the TLB on the execution speed of the computer

EAT = A (E + M) + (1 - A) (E + 2 x M)

1 - A = TLB **miss ratio**



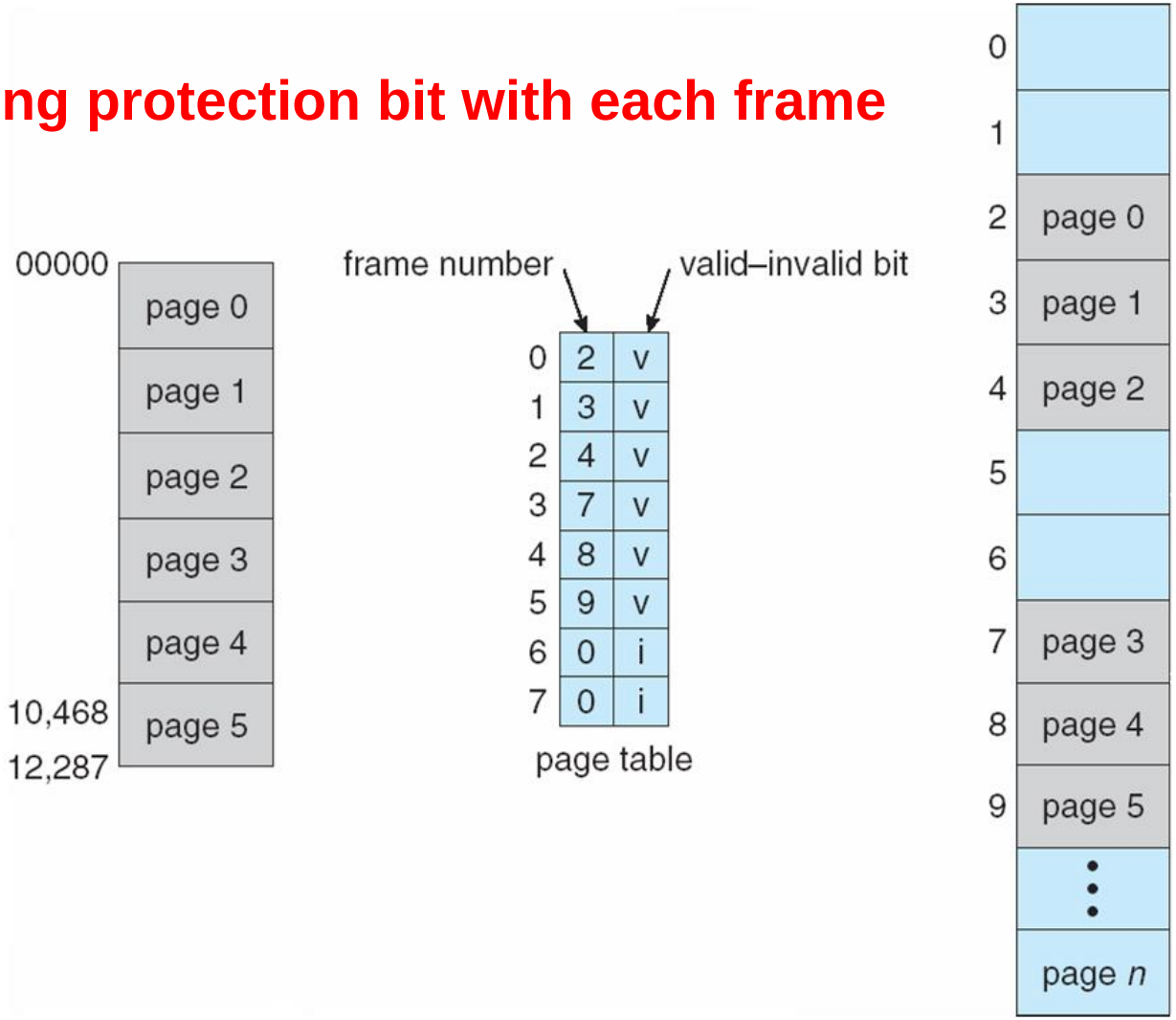
Memory Protection

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

Memory protection implemented by **associating protection bit with each frame**

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

- *"valid"* indicates that the associated page is in the process logical address space, and is thus a legal page/ indicates **page is in the main memory**
- "*invalid*" indicates that the page is not in the process' logical address space



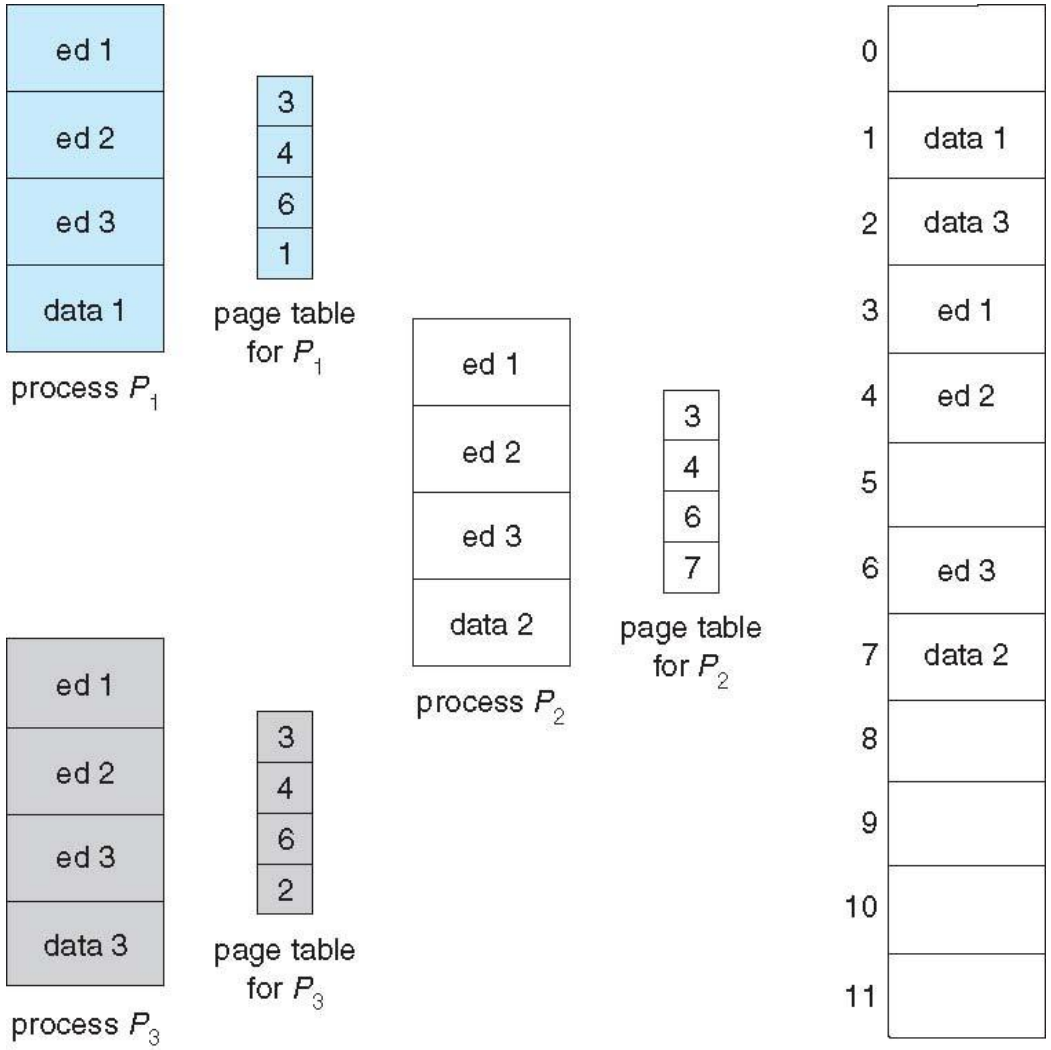
Shared Pages

Motivation for page sharing

- **Efficient communication.** Processes communicate by write to shared pages
- **Memory efficiency.** One copy of read-only code/data shared among processes

- Only one copy of the editor need be kept in physical memory.
- Each user's page table maps onto the same physical copy of the editor,
- But data pages are mapped onto different frames.

Three processes sharing a three-page editor



Structure of the Page Table

Techniques for **Large page table**:

- **Hierarchical Paging -** break up virtual address space into multiple page tables at different levels. *Two-Level Paging* Example
- **Hashed Page Tables** - the virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location
- **Inverted Page Tables** - stores a process ID of each process to identify its address space uniquely.

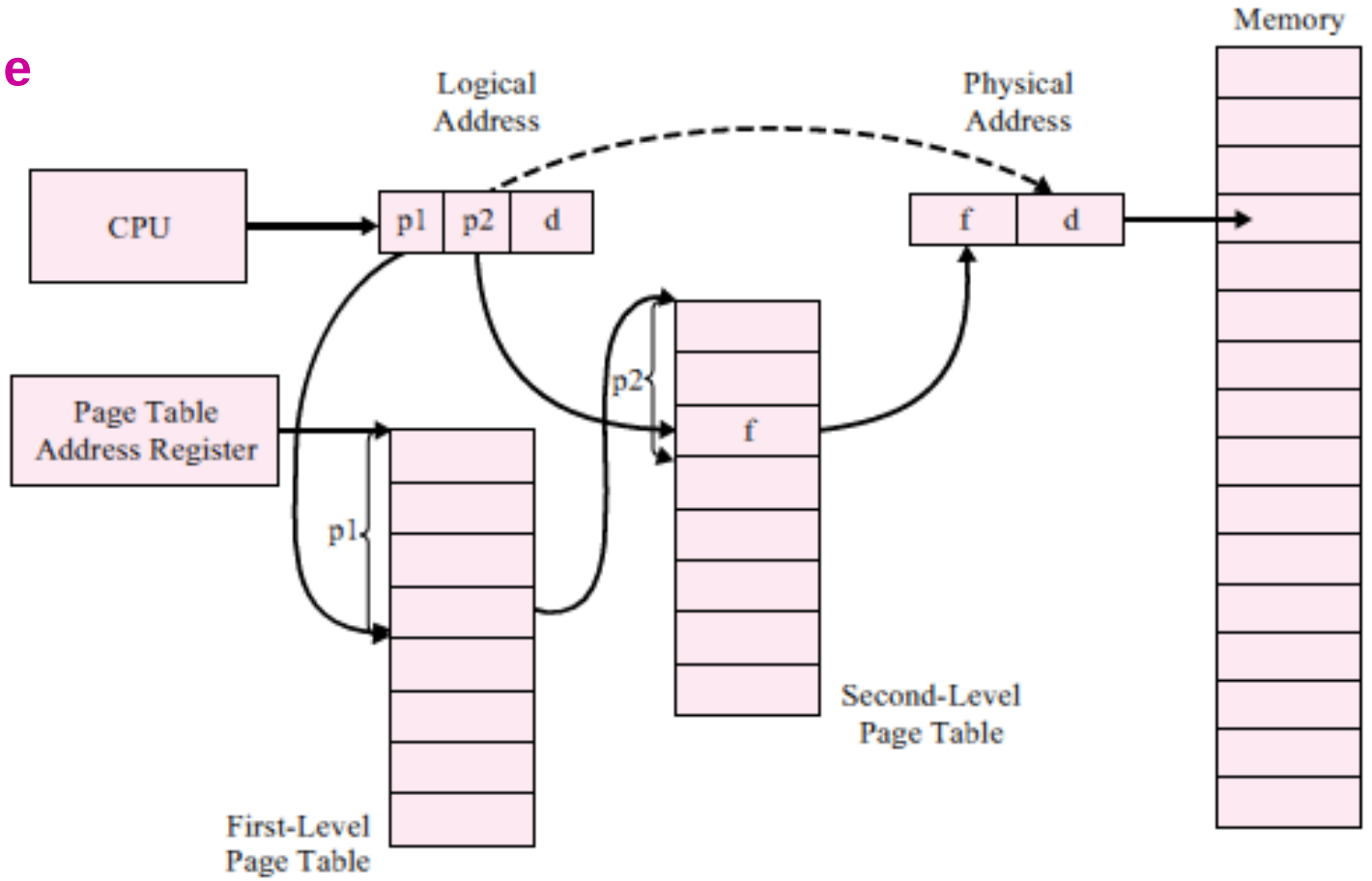
Hierarchical Paging

A two-level page table

A simple technique is a **two-level page table**

The page number as being made up of two parts: **p1** and **p2**.

- **p1** will be used by the hardware to access into the **first-level page table**
- **p2**, will be used to access within the **second-level page table**. This entry will be the frame number for the page number (the original address by p1 and p2 together).



Example Two-Level Paging

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

Logical Address Space: 2^{32} bytes;

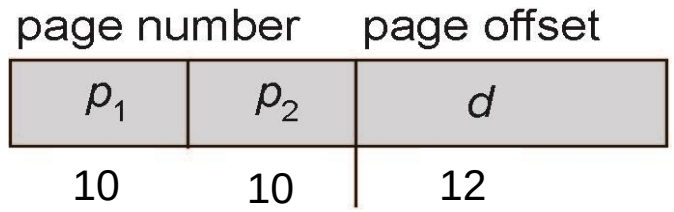
Page Size: 2^{12} = 4KB;

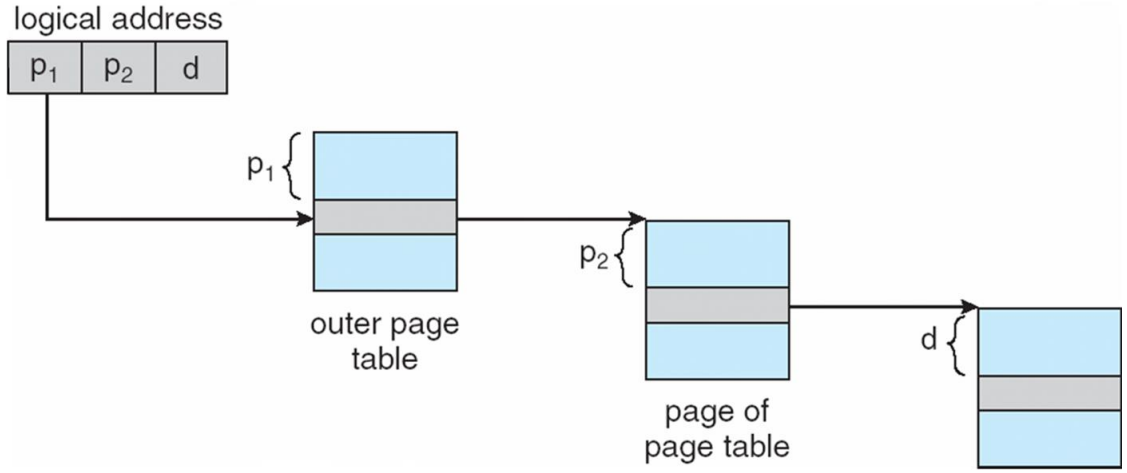
Number of Virtual Pages: $2^{32}/2^{12} = 2^{20}$

- a page number consists of 20 bits;
- a page offset consists of 12 bits

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

- a 10-bit page number;
- a 10-bit page offset





The logical address is as follows:

- ***p_1* is an index into the outer page table**, and

- *p_2* is the displacement (offset) within the page of the outer page table

- Known as **forward-mapped page table**

Hashed Page Tables

- Common in address spaces > 32 bits

The page number is hashed into a page table.

  - This page table contains a chain of elements hashing to the same location.

The page numbers are compared in this chain searching for a match.

  - If a match is found, the corresponding physical frame is extracted.

Each entry in the hash table contains a linked list of elements

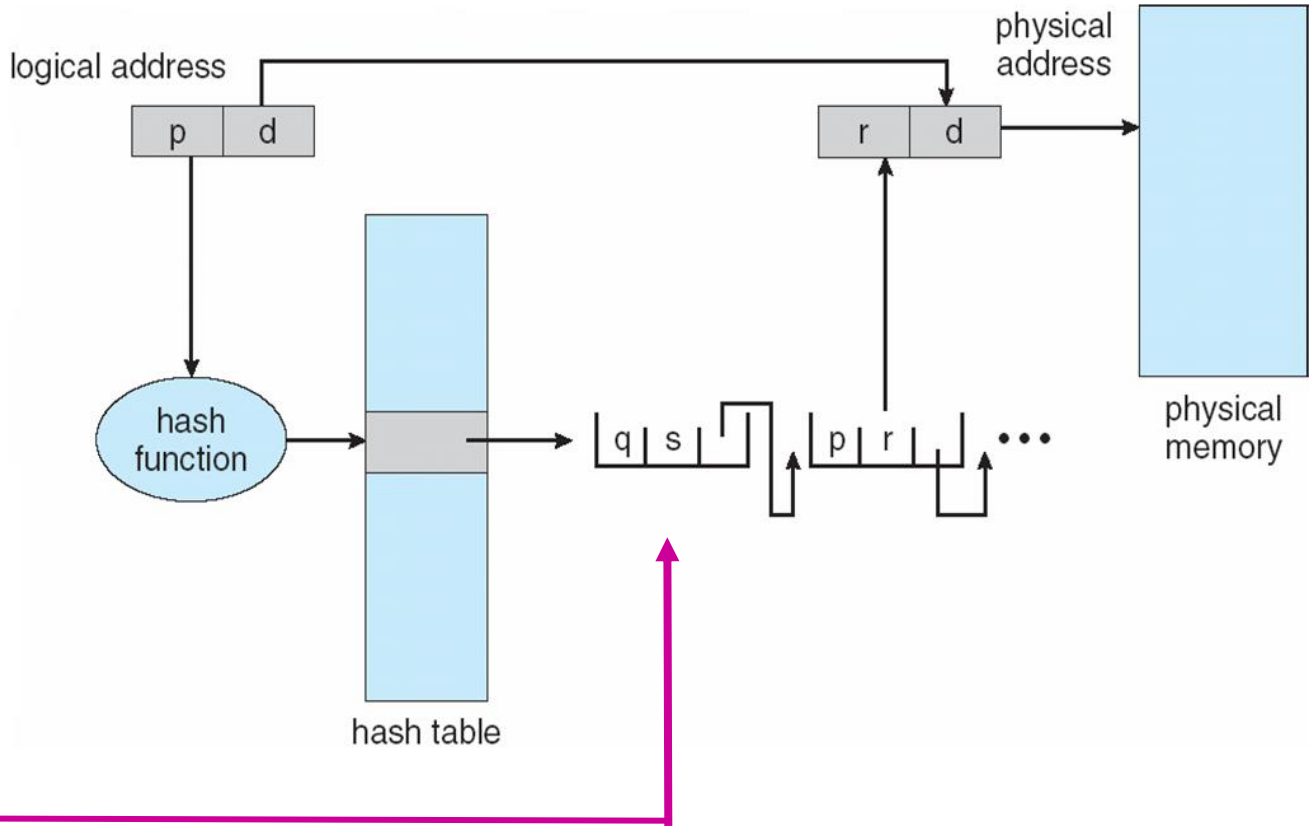
- **Each element** consists of three fields:

  (1) the page number,

  (2) the value of the mapped page frame,

  (3) a pointer to the next element in the linked list.

Page Number (PN): p, q
Page Frame Number (PFN): r
Offset: d
Hash Function: h(x)

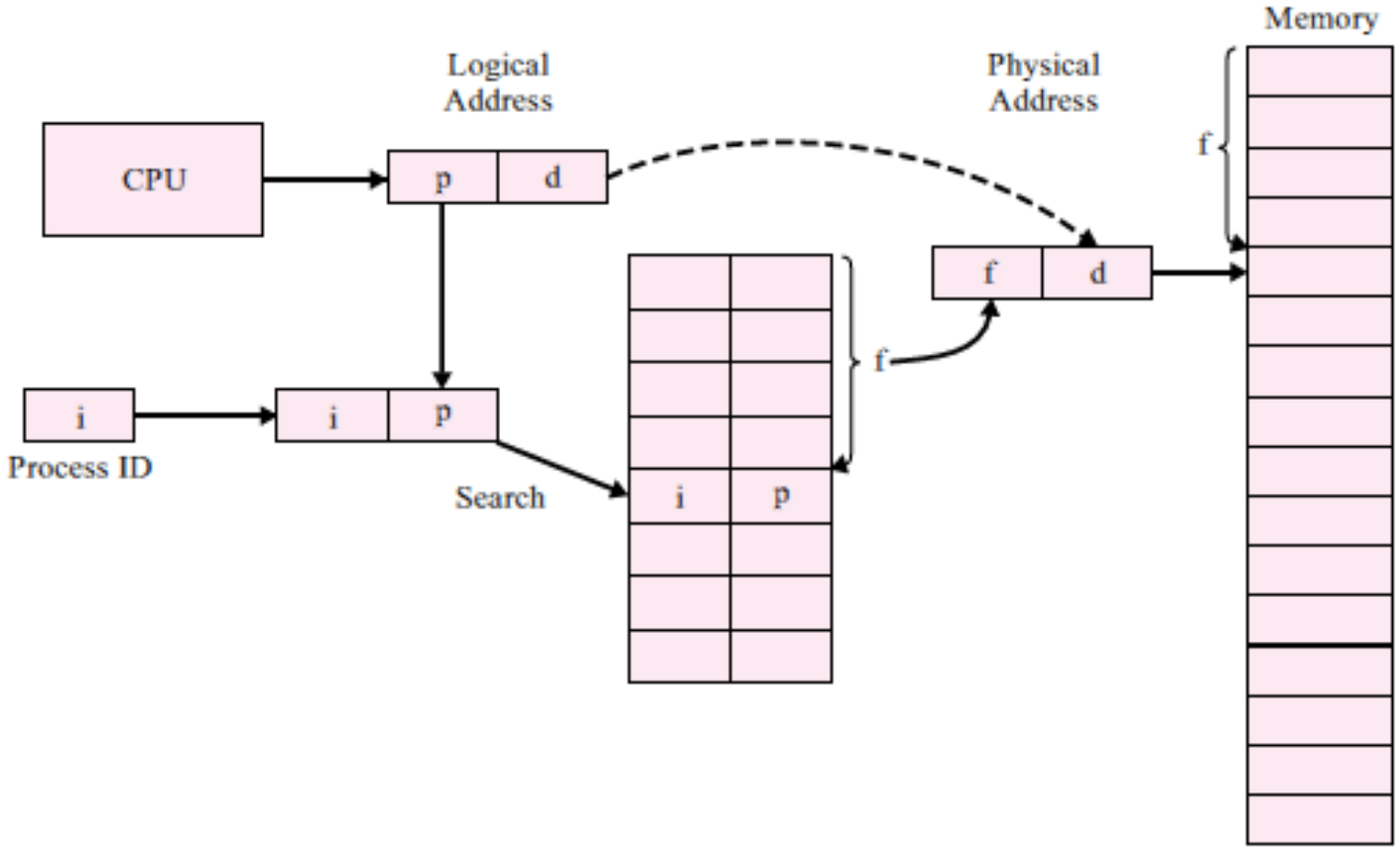


Inverted Page Table

Each process having a page table and keeping track of all possible logical pages, track all physical pages

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
  - TLB can accelerate access



32-bit vs 64-bit systems

A **32-bit system** can access 2^{32} different memory addresses, i.e 4 GB (4,294,967,296 bytes) of RAM or physical memory ideally, it can access more than 4 GB of RAM also.

A **64-bit system** can access 2^{64} different memory addresses (18,446,744,073,709,551,616 bytes), i.e actually 18-Quintillion bytes of RAM.

A computer with a 64-bit processor can have a 64-bit or 32-bit version of an operating system installed. However, with a 32-bit operating system, the 64-bit processor would not run at its full capability.

A major difference between **32-bit processors** and **64-bit processors** is the number of calculations per second they can perform, which affects the speed at which they can complete tasks.

64-bit processors can come in **dual**-core, **quad-core**, **six-core**, and **eight-core** versions for home computing.

End of Lecture

- **Summary**
  - Memory Management Unit (MMU)
  - Swapping
  - Contiguous Memory Allocation
  - Non-contiguous Memory Allocation
    - Segmentation
    - Paging
    - Structure of the Page Table
- **Reading**
  - Textbook 9th edition, **chapter 8 of the module textbook**