Information Technology Institute

Operating Systems

Lecture 5

Memory Management

- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging

Swapping

Swapping

- **What happens if the main memory is full ?**
- **Memory swapping is a method wherein memory contents not currently in use are swapped to a disk to make the memory available for other applications or processes.**

- A process can be *swapped* temporarily out of memory to a *disk*, and then brought back into memory for continued execution.

- Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.

- Modified versions of swapping are found on many systems, i.e., UNIX, Linux, and Windows.

- Memory swapping is a computer technology that enables an operating system to provide **more memory** to a running application or process than is available in physical (RAM). When the physical system memory is exhausted, the operating system can use the memory swapping techniques to get additional memory.

- With memory swapping, the operating system makes use of storage disk space to provide the functional equivalent of memory storage execution space. The space on the storage device is referred to as “**swap space**” and is used to run processes that have been swapped out of main physical memory.

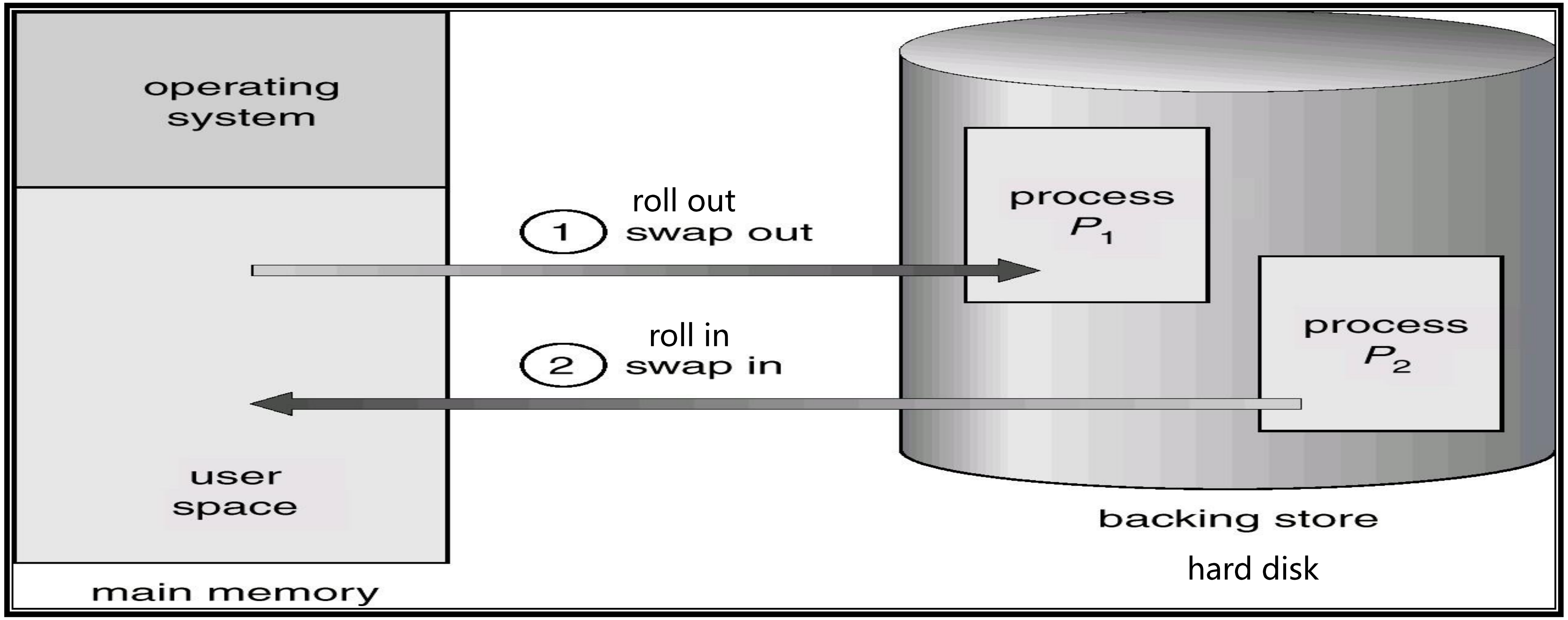
hard disk

swap space

==> the space of storage device (hard disk)

==> used to run the process that have been swapped out

Schematic View of Swapping



Roll out, roll in – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.

Advantages of Memory Swapping

- **More Memory**

Memory swapping is a critical component of memory management, enabling an operating system to handle requests that would otherwise overwhelm a system.

- **Continuous Operations**

Swap file memory can be written to disk in a continuous manner, enabling faster lookup times for operations.

- **System Optimization**

Application processes of lesser importance and demand can be relegated to swap space, saving the higher performance physical memory for higher value operations.

Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process

- **Context switch time can then be very high**

Limitations of Memory Swapping

- **Performance.** Disk storage space, when called up by memory swapping, does not offer the same performance as physical RAM for process execution.

- **Disk Limitations**. Swap files are reliant on the stability and availability of storage media, which might not be as stable as system memory.

The standard swapping **not used in modern operating systems**, **we Swap only when free memory extremely low**

Contiguous Allocation

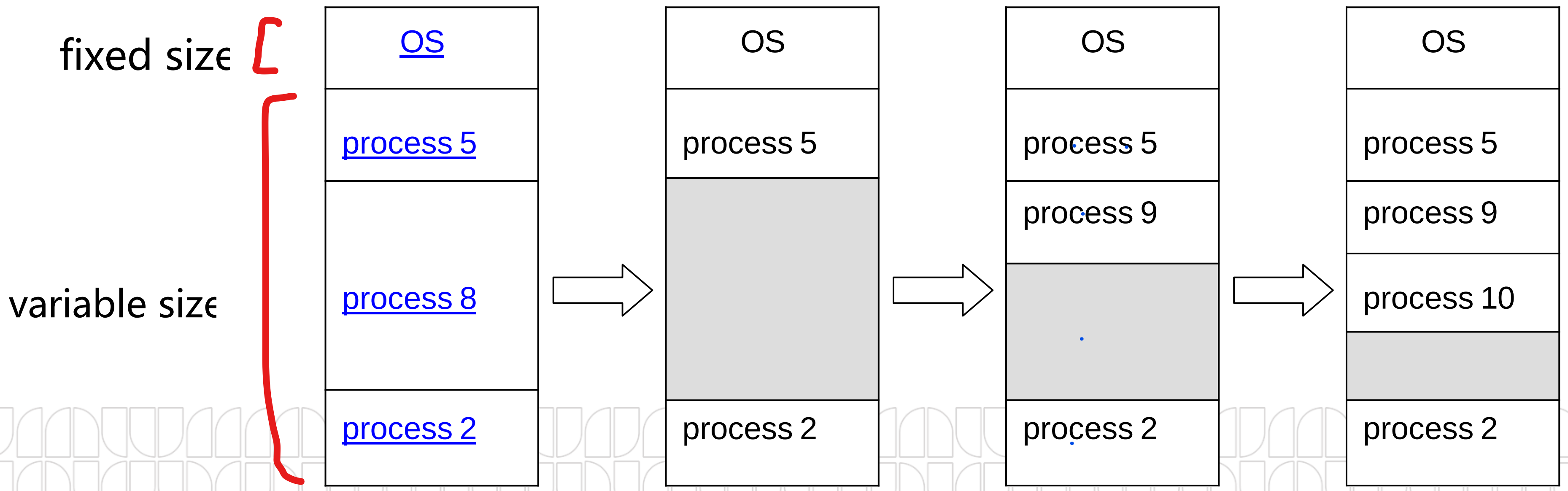
Contiguous Allocation

- One of the early methods to divide the memory to **two parts**

- Main memory usually into two partitions:
  - Resident **operating system**, usually held in **low memory** ROM
  - **User processes** then held in **high memory**. RAM

Multiple-partition allocation

- **Memory is divided into nuber of contiguous regions "partitions" (fixed size or variable size)**
- **Variable-partition** sizes for efficiency (sized to a given process' needs)
- **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)



Dynamic Storage-Allocation Problem

- **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**. $\leftarrow$

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

First-fit and best-fit **better than** worst-fit in terms of speed and storage utilization.

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous. (generated from **Variable-size partition**). No solution for it

- **Internal Fragmentation** – waste of memory within a partition cased by deference between the size of partition and the size of process loaded into it. this size difference is memory internal to a partition, but not being used. (generated from **fixed-size partition**) has solutions

- Reduce **external fragmentation** by compaction

Compaction

- Compaction is a technique to **collect all the free memory present in form of fragments into one large chunk of free memory**, which can be used to run other processes.
- It does that by moving all the processes towards one end of the memory and all the available free space towards the other end of the memory so that it becomes contiguous.
- Compaction is possible *only* if relocation is dynamic, and is done at execution time

Compaction (cont.)

- Before compaction, the main memory has some free space between occupied space. This condition is known as external fragmentation. Due to less free space between occupied spaces, large processes cannot be loaded into them.
- After compaction, all the occupied space has been moved up and the free space at the bottom. This makes the space contiguous and removes external fragmentation. Processes with large memory requirements can be now loaded into the main memory.

Compaction (cont.)

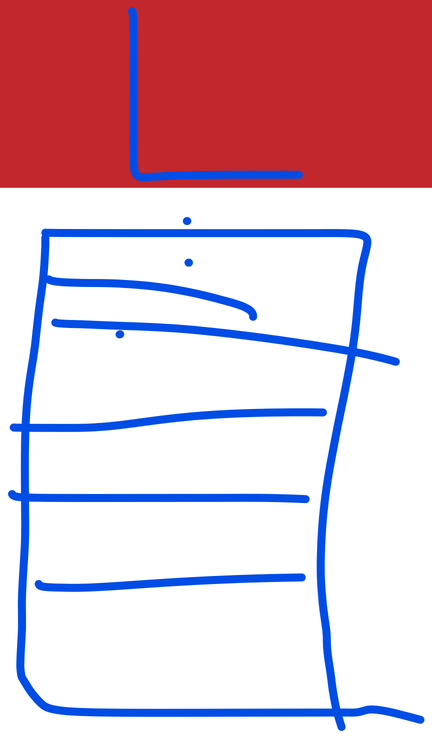
Advantages of Compaction

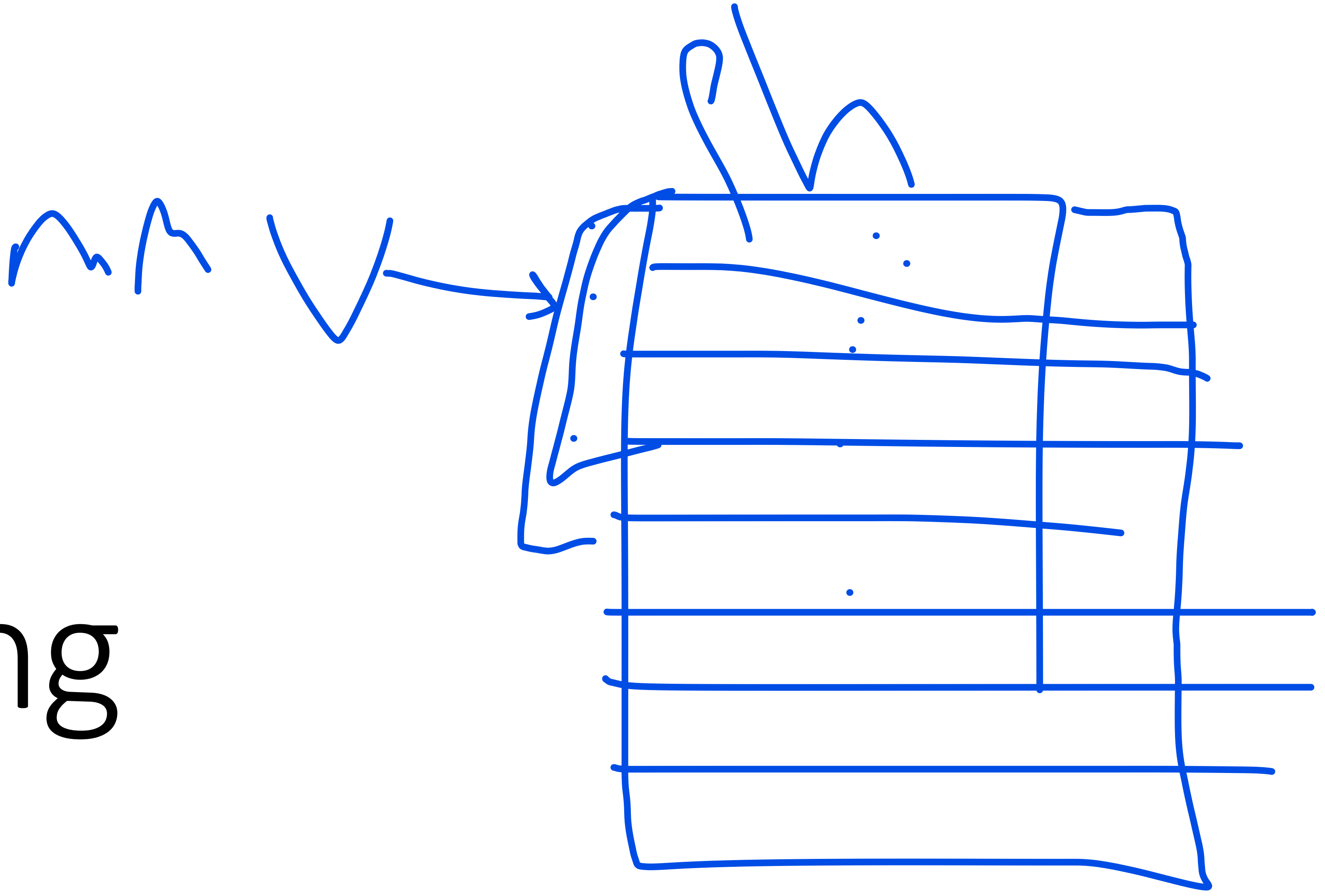
- Reduces external fragmentation.
- Make memory usage efficient.
- Memory becomes contiguous.
- Since memory becomes contiguous more processes can be loaded to memory.

Disadvantages of Compaction

- A huge amount of time is wasted in performing compaction.
- CPU sits idle for a long time.
- Not always easy to perform compaction.

In the following two topics (Paging and segmentation) we talk about how the MMU link between Logical addresses and physical addresses





Paging

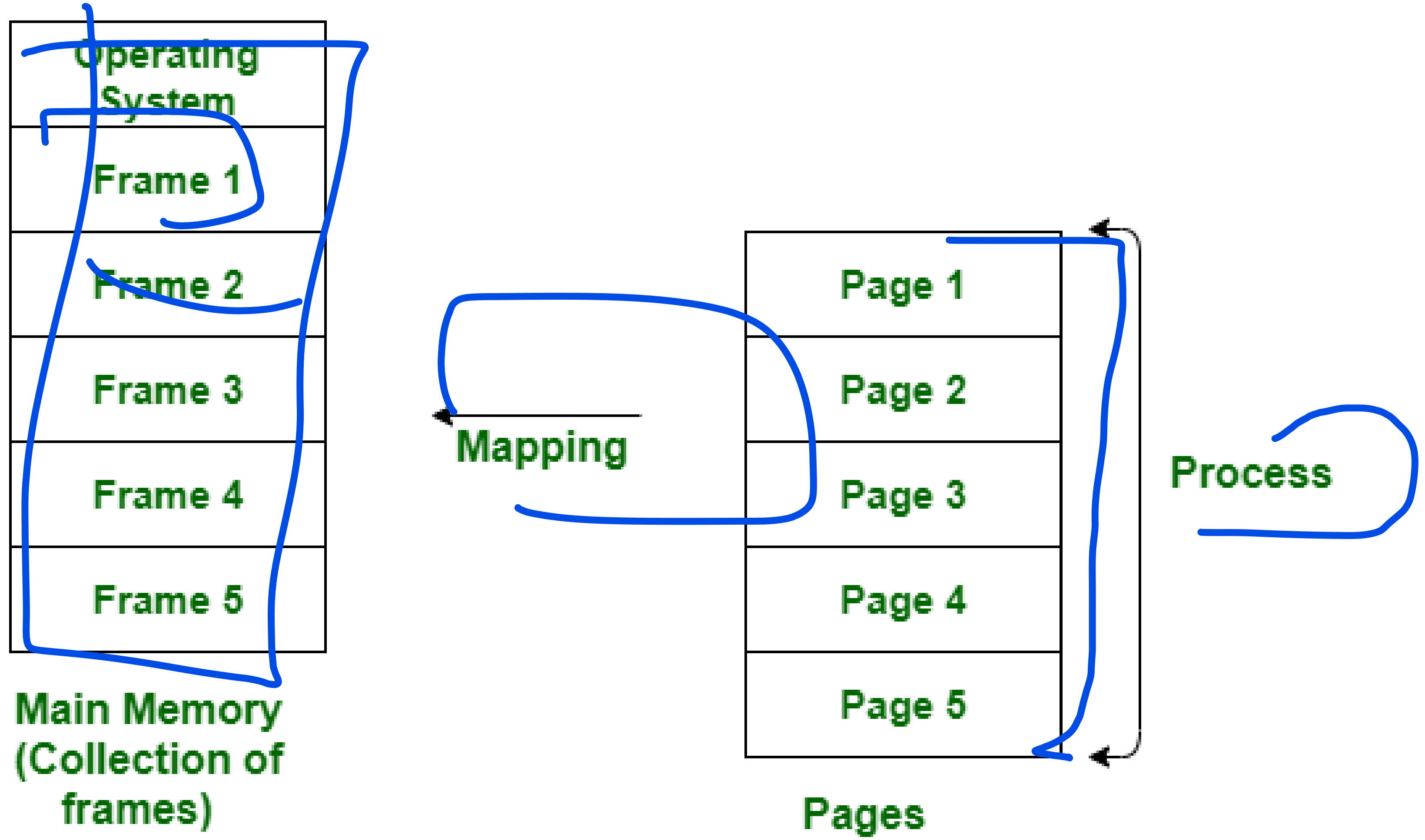
Paging

- Paging is a **memory management method.**
- Paging is a method or technique which is **used for non-contiguous memory allocation**.

- It is a fixed-size partitioning scheme. In paging, both **logical memory** and **physical memory** are divided into equal fixed-size partitions known as **pages** and **frames** respectively.

- In paging, each process is split into parts wherever the size of every part is the same (page size). The pages of the process area unit hold on within the frames of main memory.

Paging



Operating System
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Main Memory (Collection of frames)
Mapping
Page 1
Page 2
Page 3
Page 4
Page 5
Pages
Process

Paging

- Divide **physical memory** into fixed-sized blocks called $\boxed{\textbf{frames}}$ (size is power of 2, between 512 bytes and 8192 bytes).

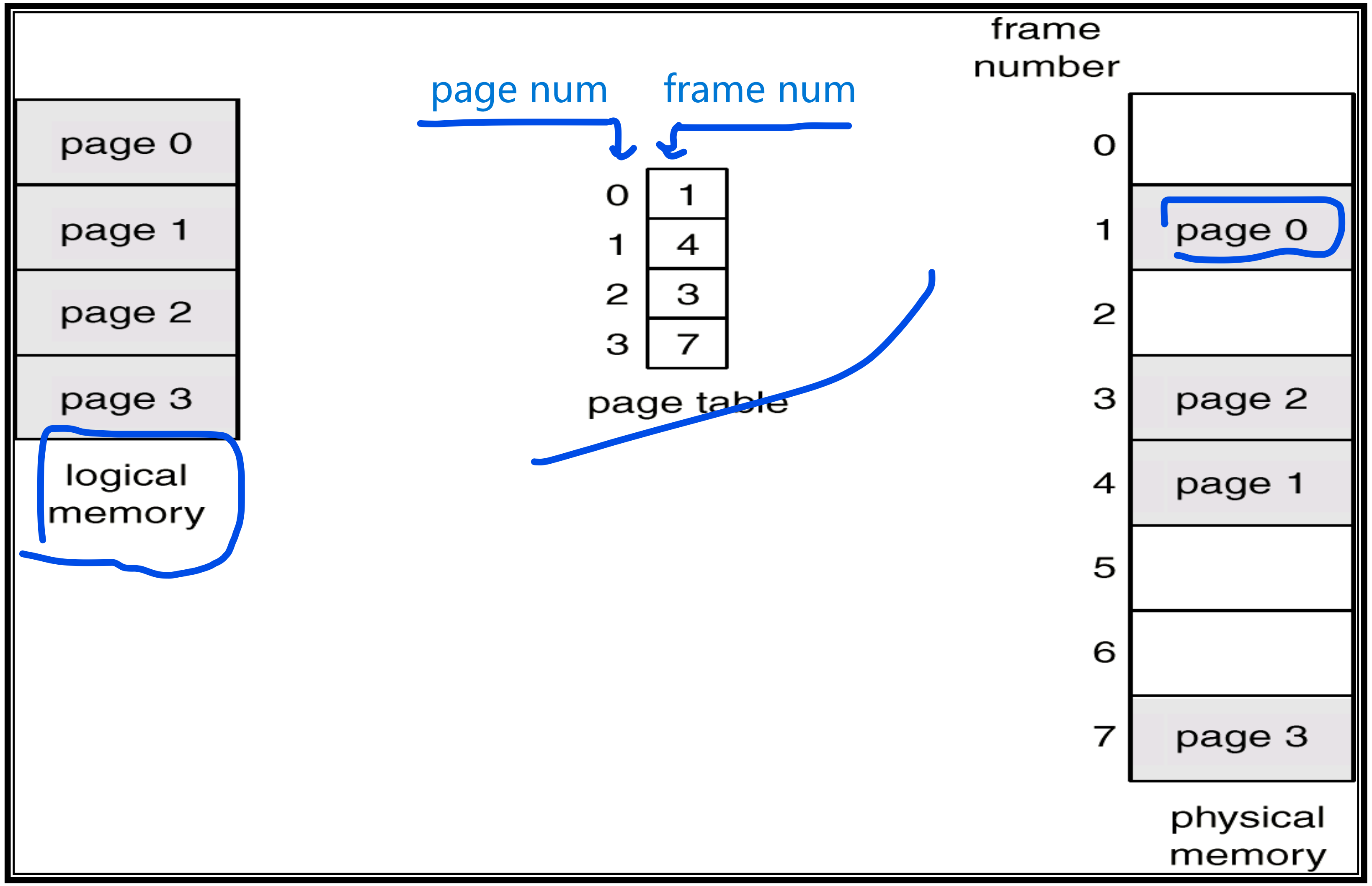
  $\boxed{2^{size}}$

- Divide **logical memory** into blocks of same size called $\boxed{\textbf{pages}}$.

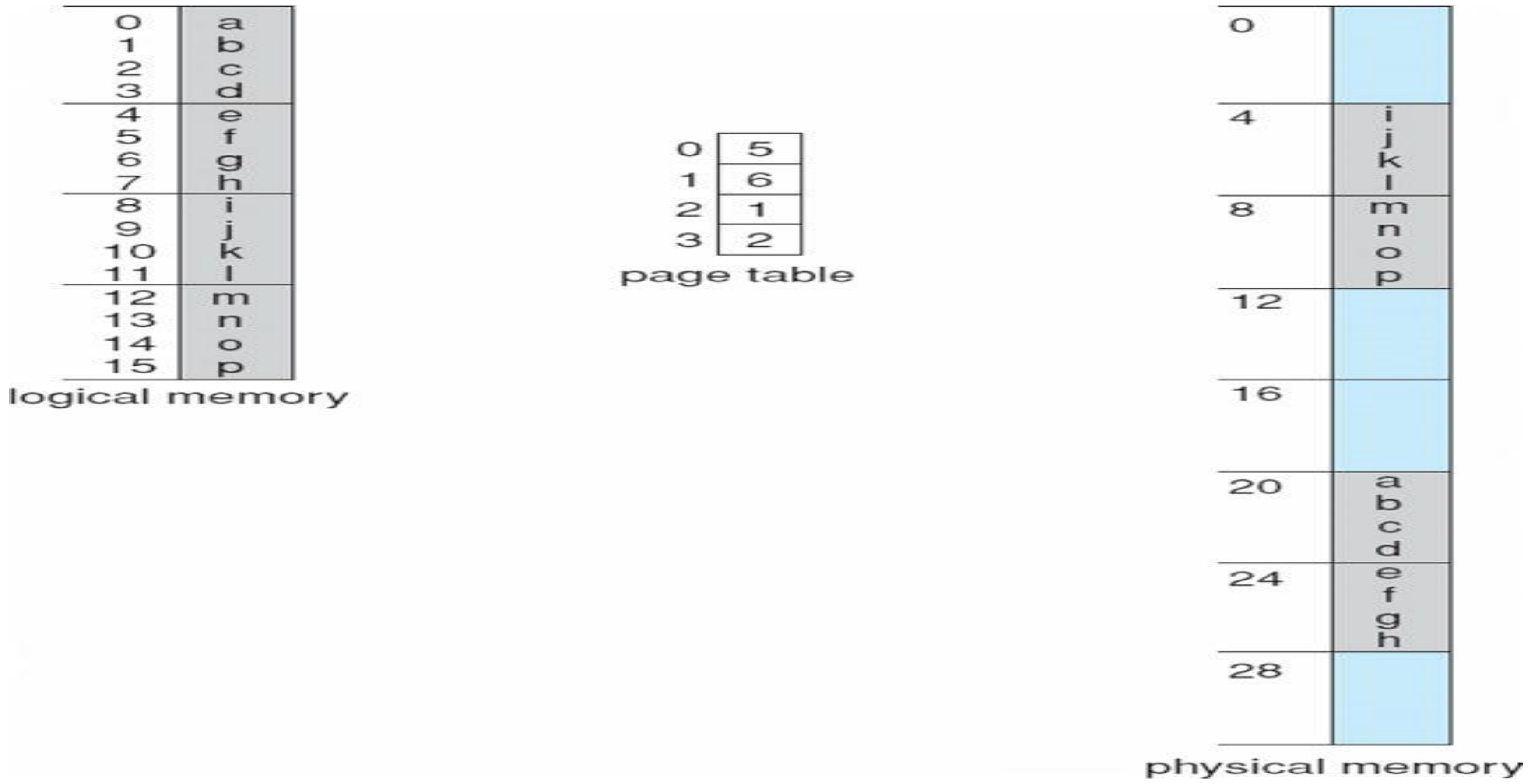
- To run a program of size *n* pages, need to find *n* free frames and load program.

- **Page table** to translate logical to physical addresses (**stores the Frame numbers corresponding to the page numbers**). Page table **is kept in main memory**

Paging Example



Paging Example



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

Paging

Paging still suffers from Internal fragmentation

Example to calculate the internal fragmentation

Page size = 2,048 bytes

Process size = 72,766 bytes

(72,766 / 2048) = 35 pages + 1,086 bytes

So the **Internal fragmentation** is = 2,048 (Page size) - 1,086 = **962 bytes**

To reduce the internal fragmentation ↓ → page size reduced ↓ → page table increased ↑ (page table stored in main memory)

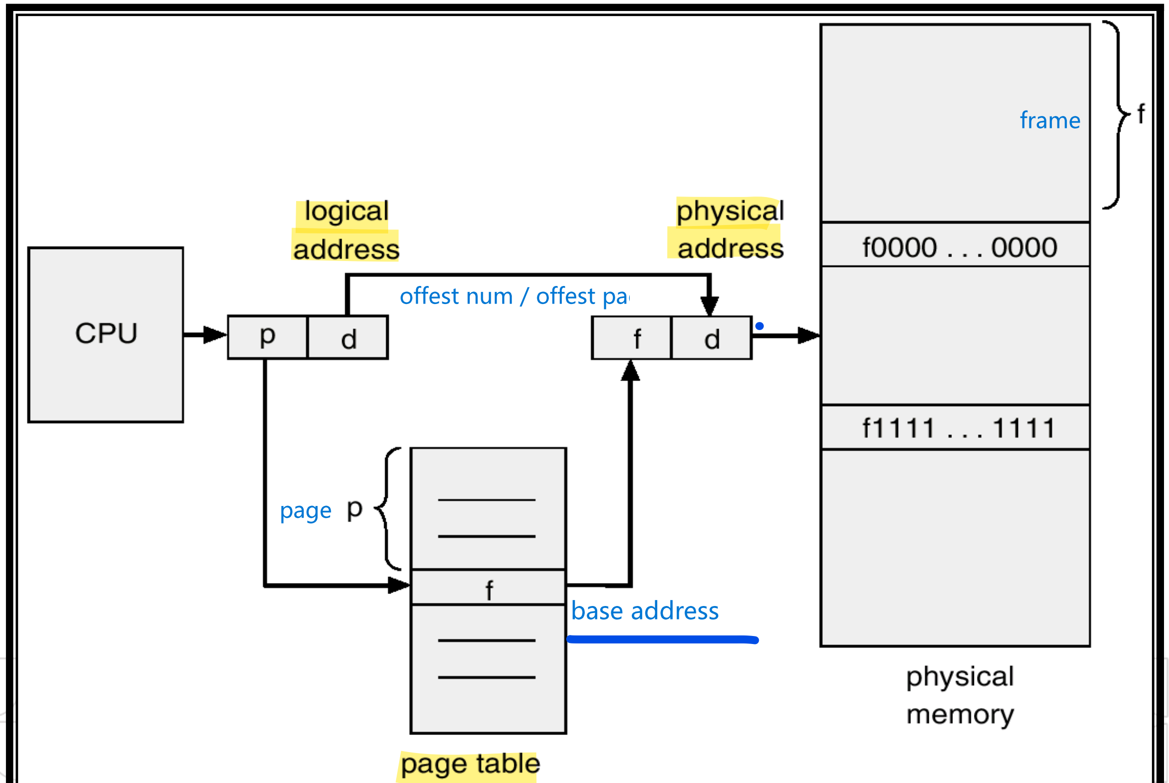
Address Translation

- Address generated by CPU is divided into: logical address
  - *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.

To access one location in RAM, unfortunately we need do **two memory operation** (two access to RAM)

1 access the page table in RAM
2 And then to access the actual location in the RAM.

The solution to reduce this time is using **cache memory**

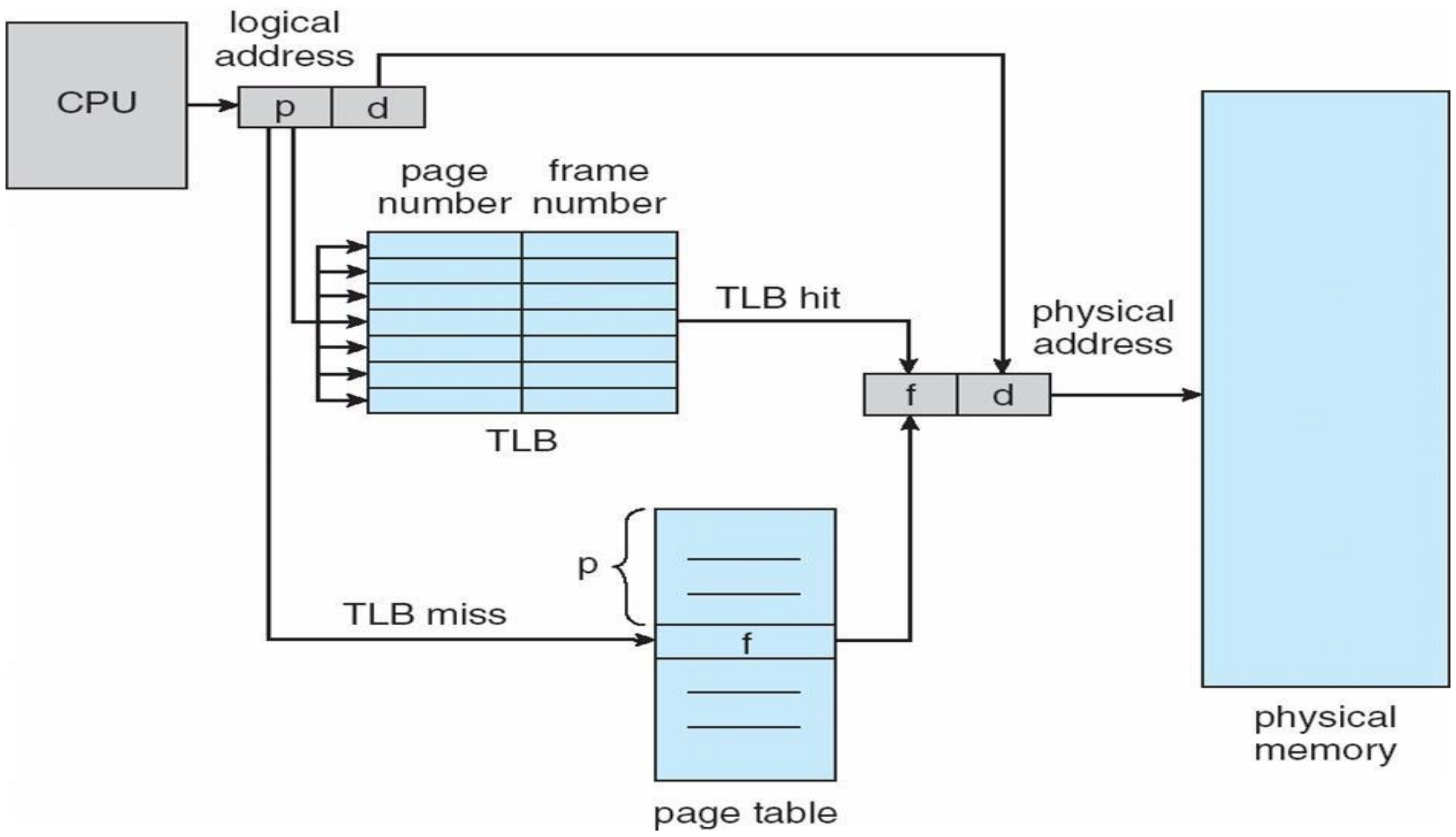


Address Translation (using cache memory)

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 called **translation look-aside buffers** (**TLBs**) **saved in cache memory**

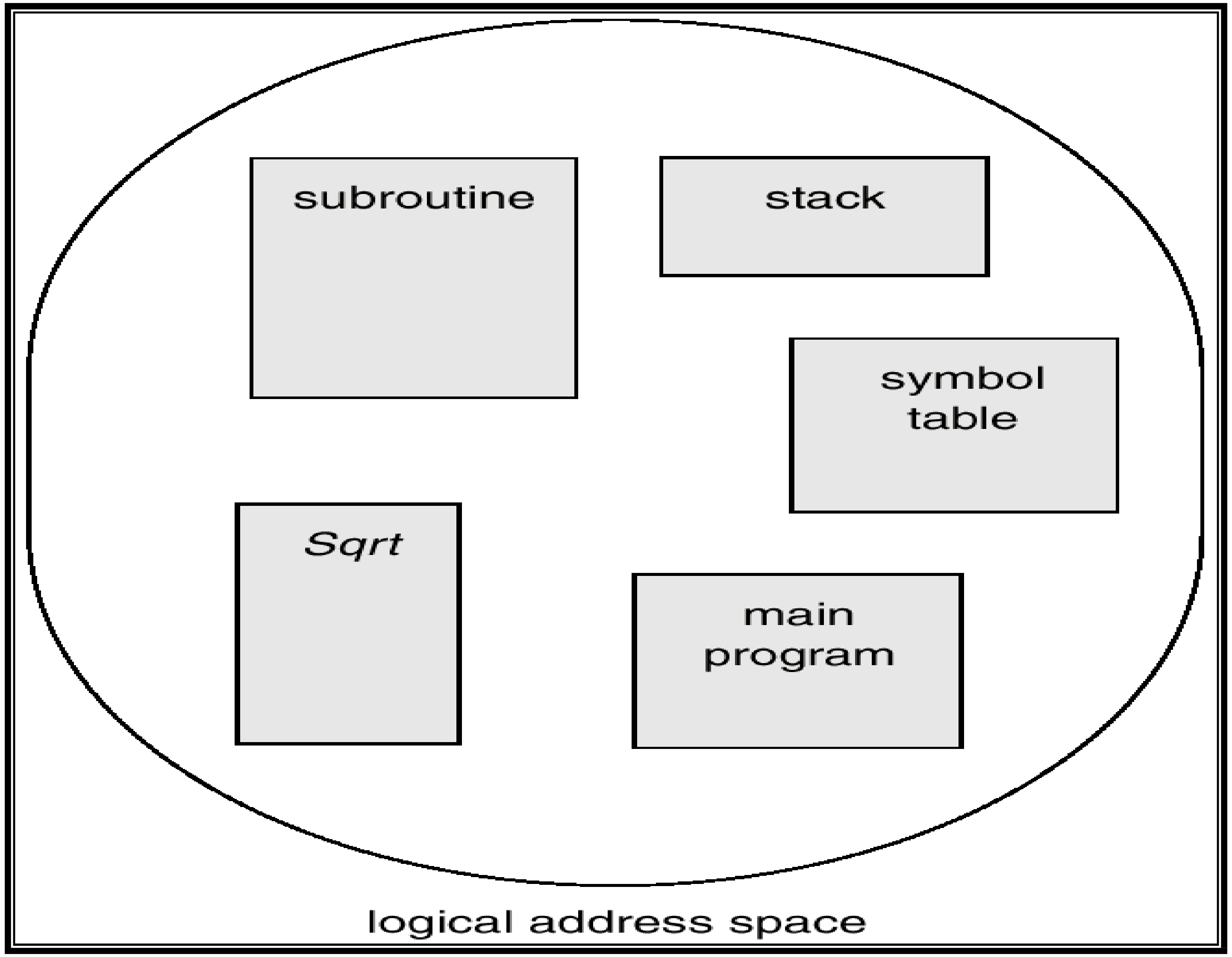


Segmentation

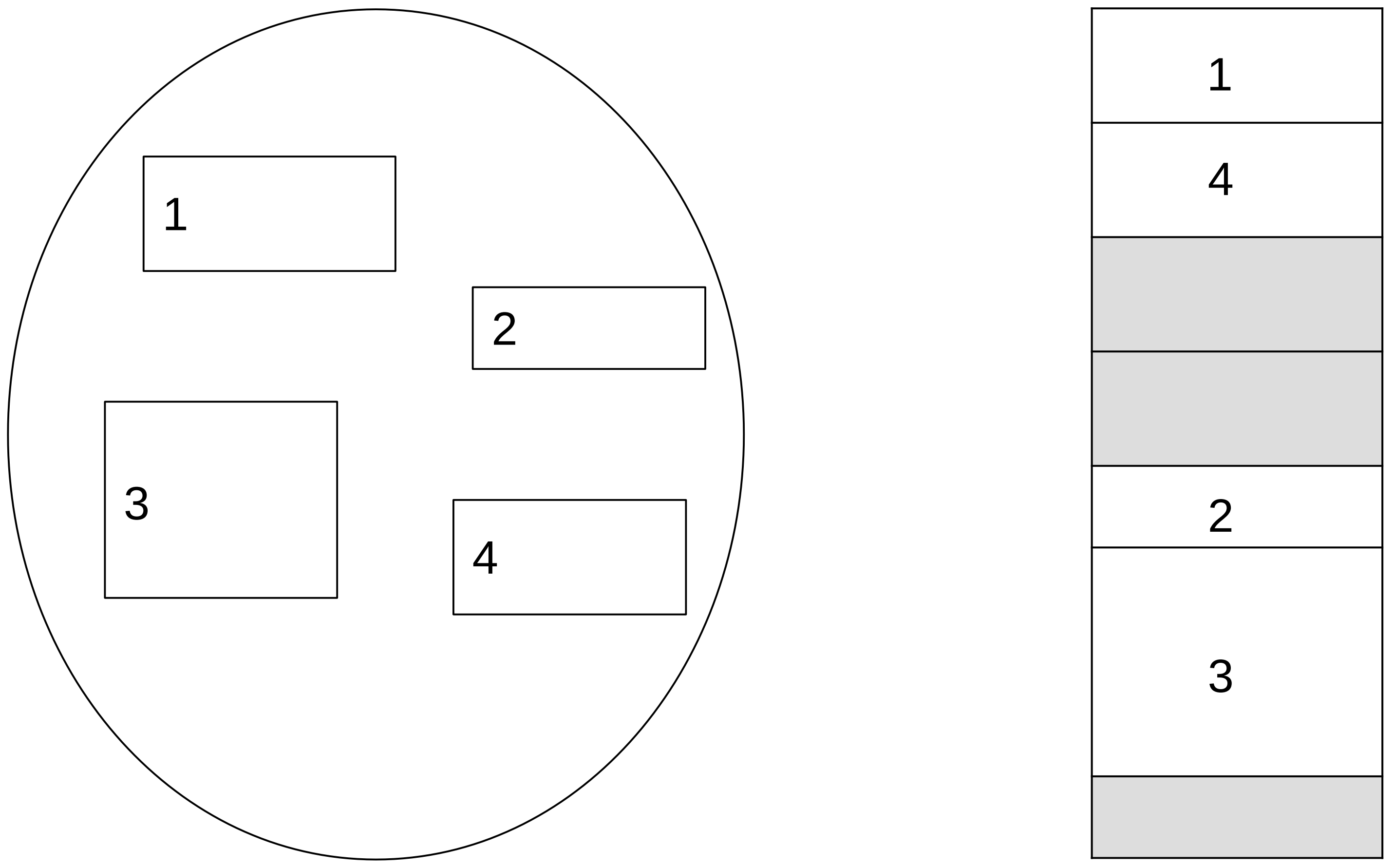
Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:
  - main program,
  - function,
  - Local variable
  - Global variable
  - stack,
  - arrays
- When a process executes, segmentation assigns related data into segments for faster processing.
- The segmentation function maintains a segment table that includes physical addresses of the segment, size, and other data.

User’s View of a Program



Logical View of Segmentation



user space

physical memory space

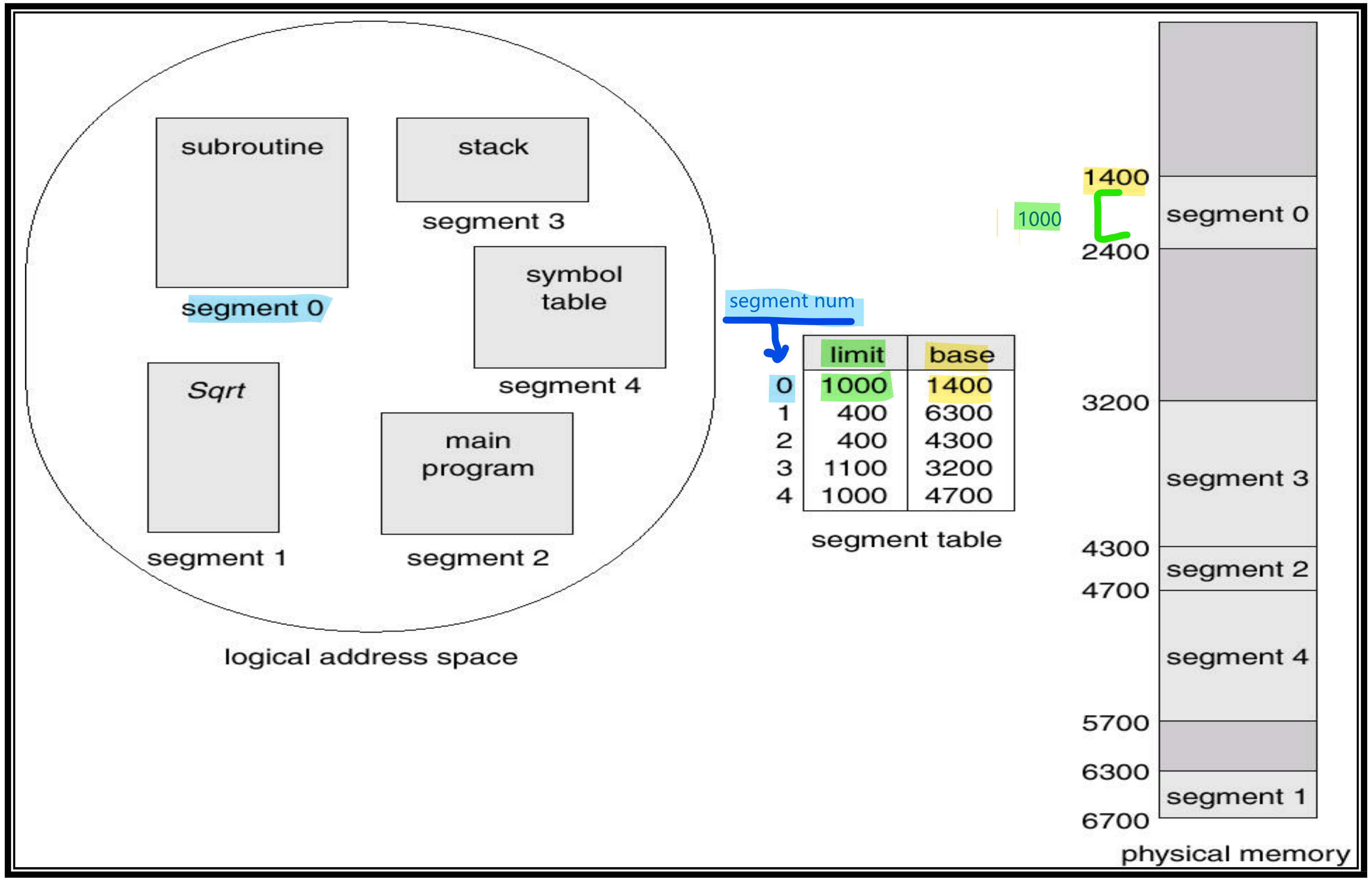
Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

- *Segment table* – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory.
  - *limit* – specifies the length of the segment.

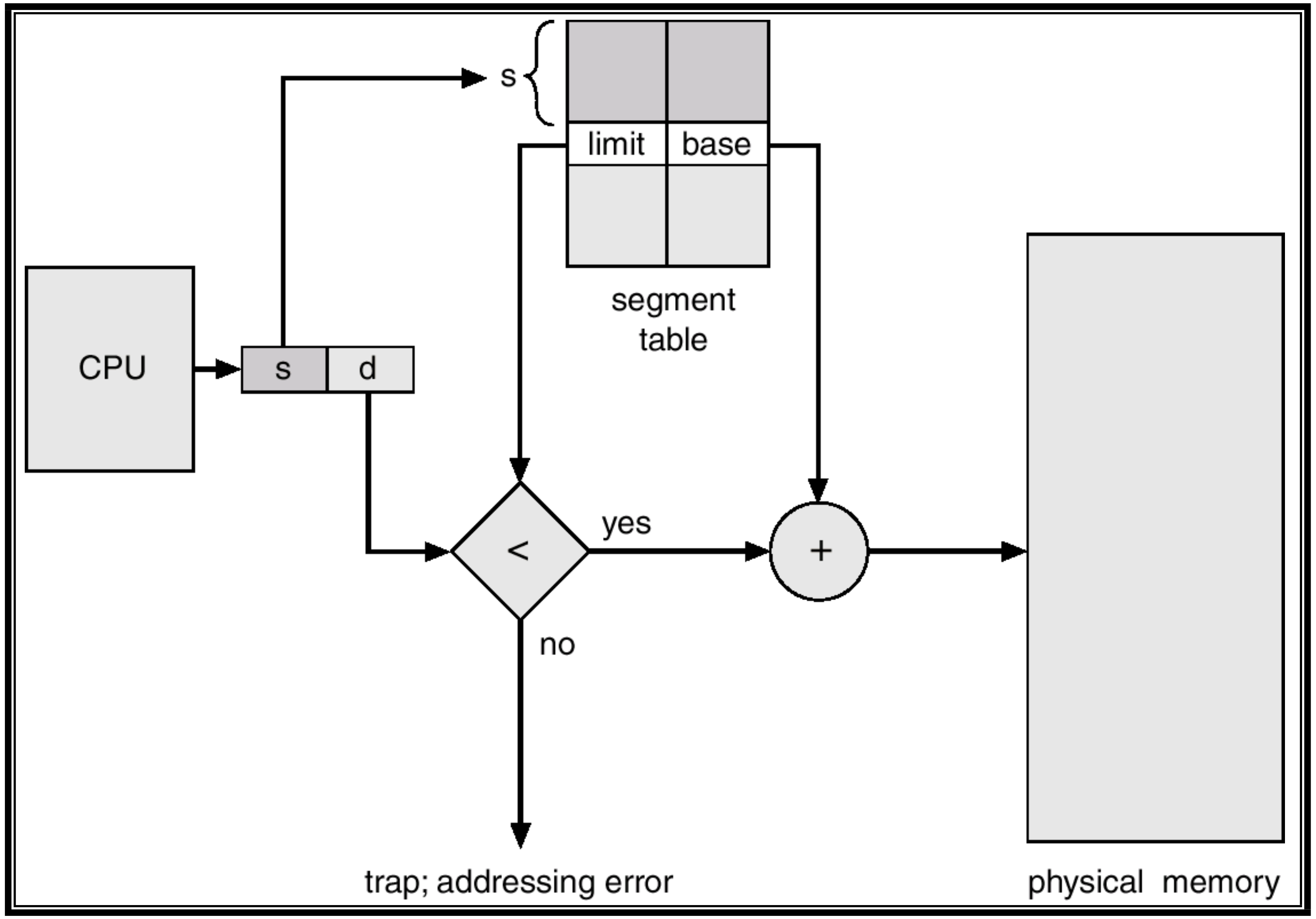
Example of Segmentation



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

Segmentation Hardware



Paging VS Segmentation

Sr No.	Paging	Segmentation
1	Non-Contiguous memory allocation	Non-contiguous memory allocation
2	Paging divides program into fixed size pages.	Segmentation divides program into variable size segments.
3	OS is responsible	Compiler is responsible.
4	Paging is faster than segmentation	Segmentation is slower than paging
6	It suffers from internal fragmentation	It suffers from external fragmentation
8	Logical address is divided into page number and page offset	Logical address is divided into segment number and segment offset
9	Page table is used to maintain the page information.	Segment Table maintains the segment information

Thank You