Tishk International University
Science Faculty
IT Department

Operating Systems

Lecture 4: Memory Management

3rd Grade - Fall Semester

Instructor: Alaa Ghazi

Lecture 4: Memory Management - Agenda

- Background
  - ❑ Main Memory
  - ❑ Cache Memory
  - ❑ Hardware Address Protection
  - ❑ Address Binding
- Memory Management Approaches
  1. Single Contiguous Model
  2. Partition with Contiguous Allocation
  3. Swapping
  4. Segmentation
  5. Paging
- Virtual Memory Basics
- Optimizing Applications Performance

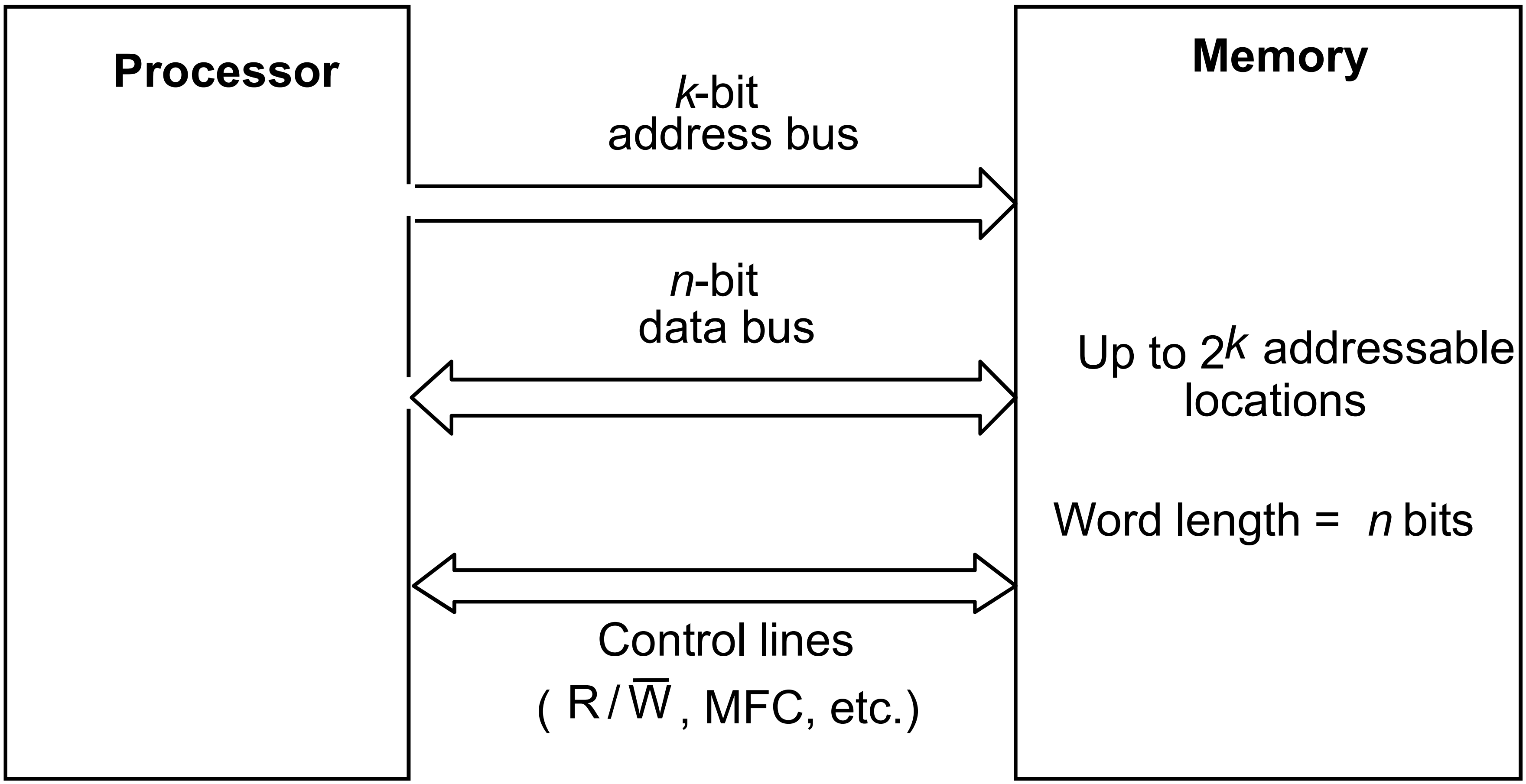
Background

- The second most important hardware resource in the computer system after the CPU is the Memory.
- Memory accesses and memory management are a very important part of modern computer operation. Every instruction has to be fetched from memory before it can be executed, and most instructions involve retrieving data from memory or storing data in memory or both.
- The introduction of multi-tasking OSes increases the need for complex memory management, ***because*** as processes are swapped in and out of the CPU, so must their code and data be swapped in and out of memory, all at high speeds and without interfering with any other processes.

What is Main Memory

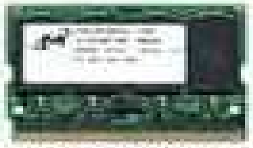
- **Main Memory or (Random Access Memory - RAM):** is the area in a computer in which data is stored for quick access by the computer's processor.
- RAM speed is measured in nanoseconds (billionths of a second), while magnetic and SSD storage is measured in milliseconds (thousandths of a second).
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly

CPU-Main Memory Connection (not required in the exam)

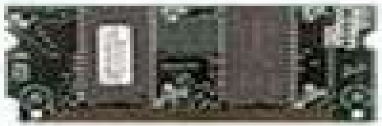


Memory Modules Samples (not required in the exam)

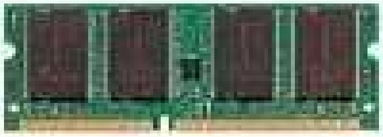
Note, as well as the different number of pins, the different spacing of the slots in the connector-edge



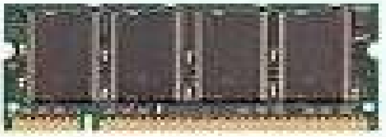
MicroDIMM (rare)



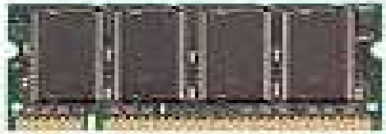
72 pin SODIMM (rare)



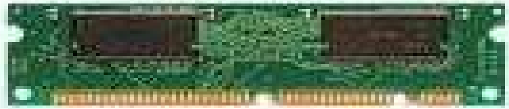
144 pin SDRAM SODIMM



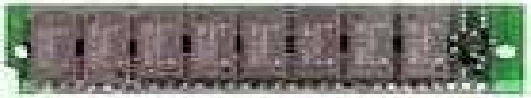
200 pin DDR SODIMM



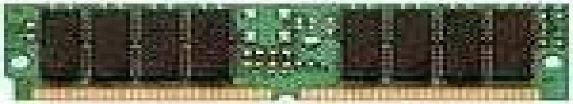
200 pin DDR-2 SODIMM



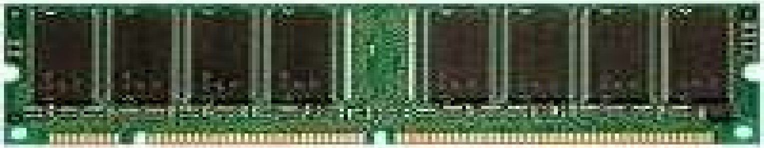
100 pin DIMM printer RAM



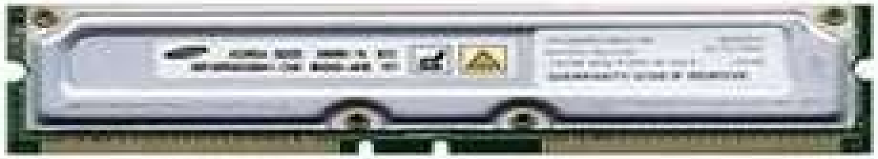
30 pin SIMM



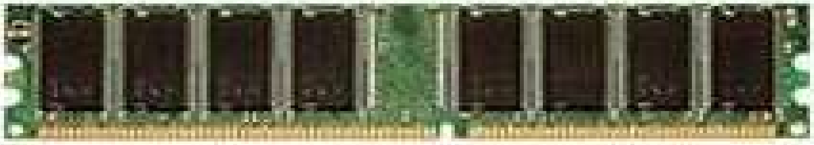
72 pin SIMM



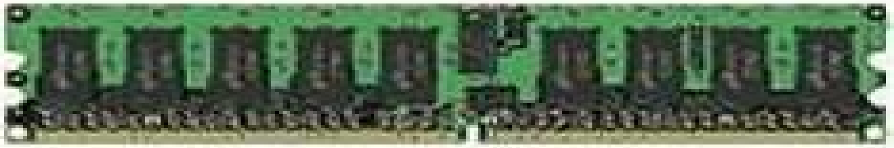
168 pin SDRAM DIMM



184 pin RAMBus RDRAM RIMM



184 pin DDR DIMM

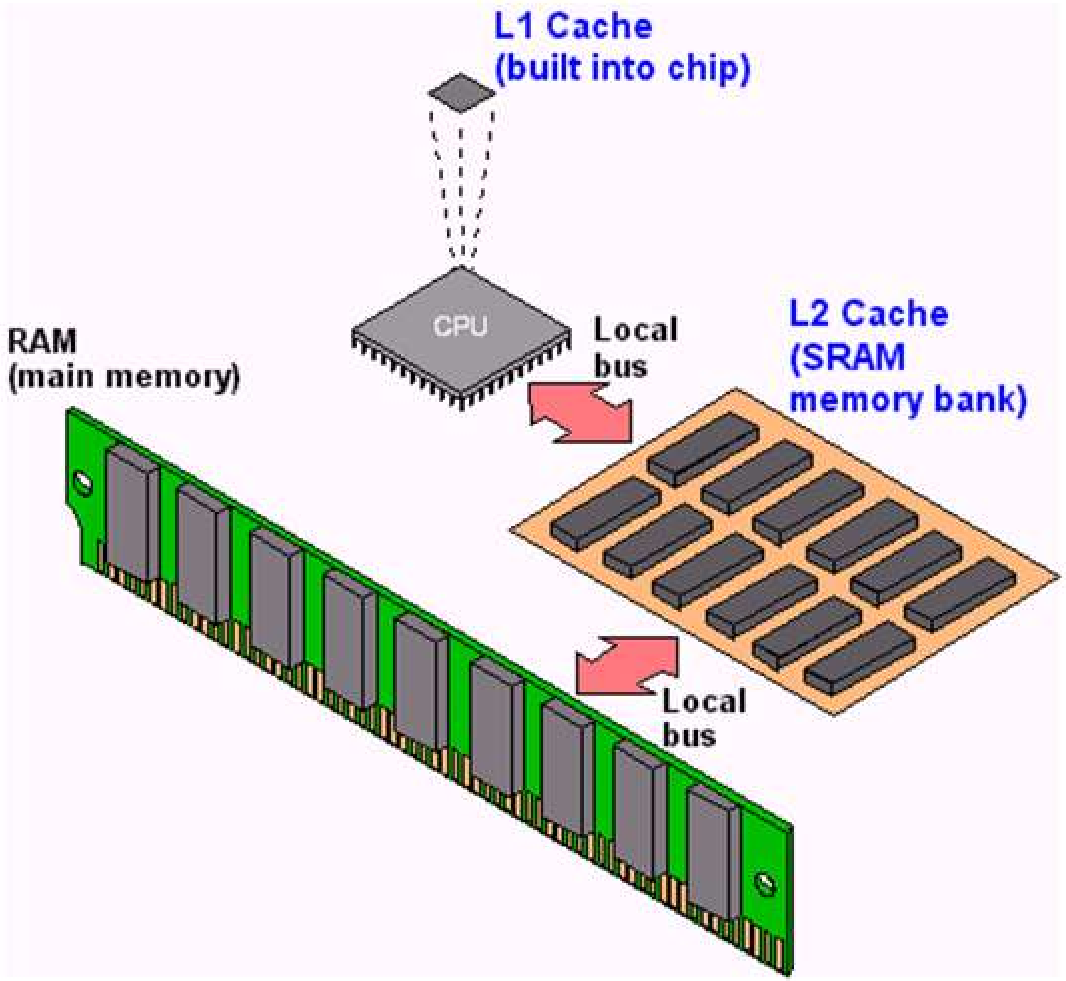


240 pin DDR-2 DIMM

Cache Memory

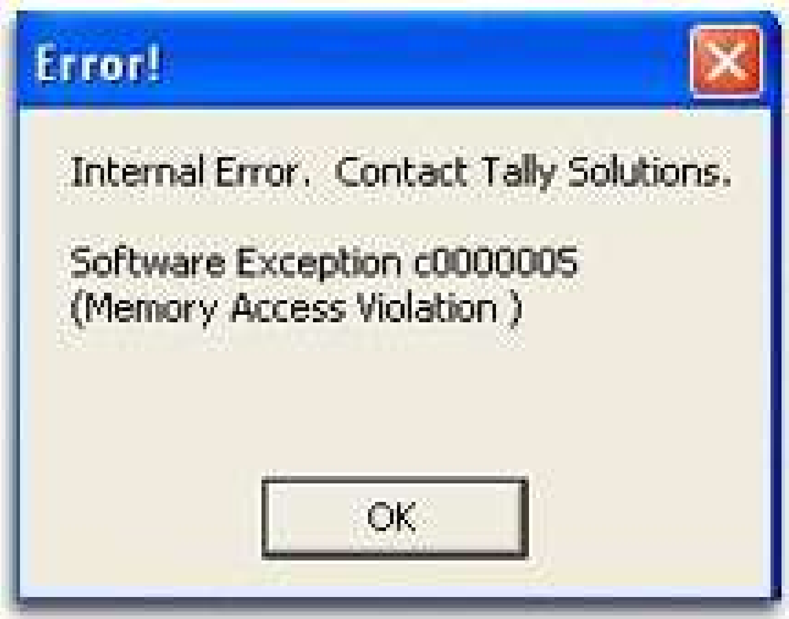
- **Cache Memory**: is a small-sized type of volatile computer memory that provides high-speed data access to a processor and stores frequently used computer code and data.
- It is the fastest memory in a computer, and is typically integrated onto the motherboard and directly embedded in the processor or main random access memory (RAM).

Cache Memory Levels – Physical (not required in the exam)



Hardware Address Protection

- **Hardware Address Protection** is the restriction of User processes so that they only access memory locations that "belong" to that particular process.
- A pair of **base** and **limit registers** define the logical address space of a process.
- CPU must check every memory access generated in user mode to be sure it is between base and limit registers for that process.



Error!

Internal Error. Contact Tally Solutions.

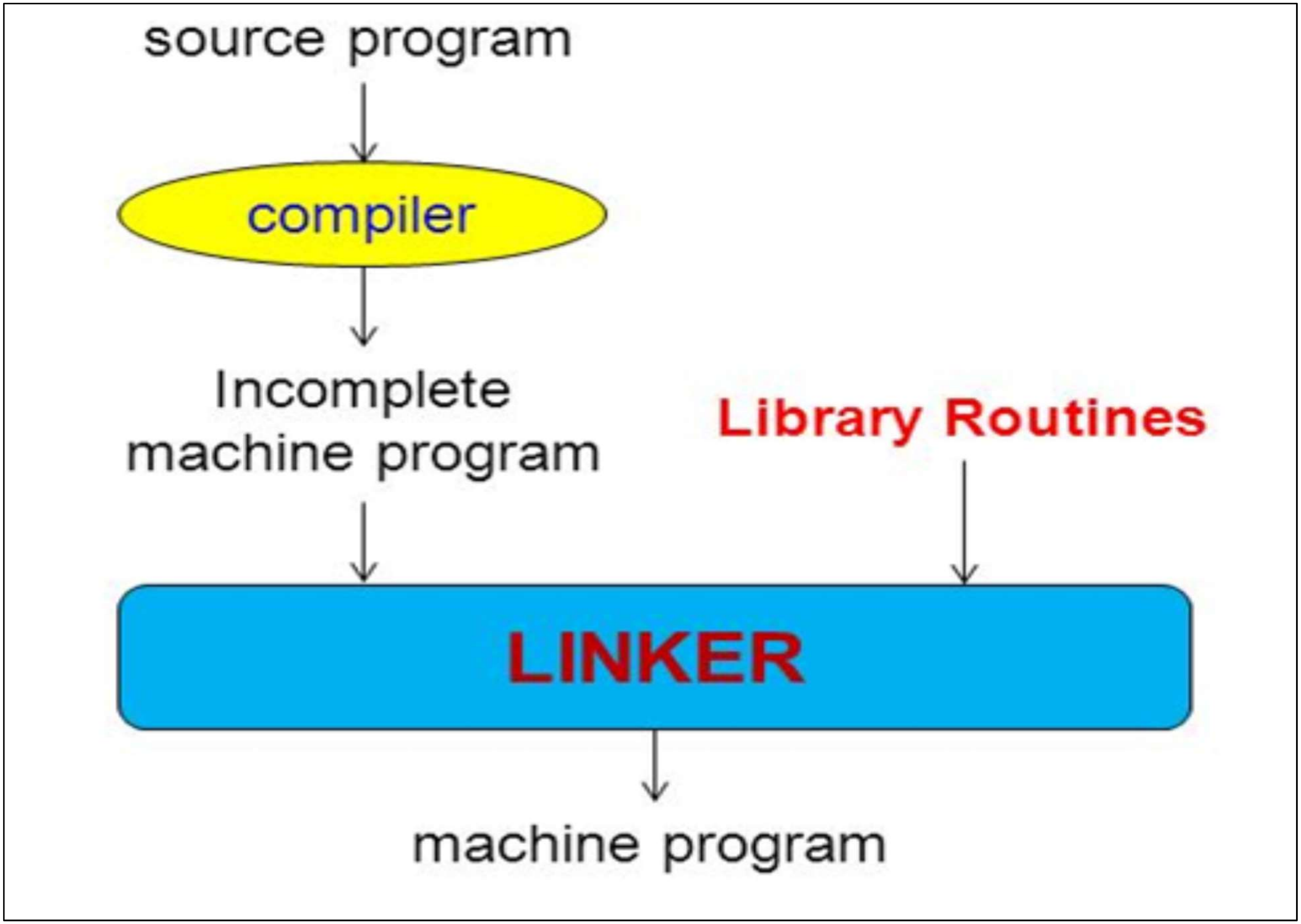
Software Exception c0000005
(Memory Access Violation)

Libraries Linking

- **Static linking** – It is the case when system libraries and program code are combined by the loader into the binary program image.
- **Dynamic linking** – It is the case when linking of the routines to the main program is postponed until execution time.
- **Stub** is a small piece of code, used to locate the appropriate memory-resident library routine and replaces itself with the address of the routine, and executes the routine
- **Advantages of Dynamic Linking** and Shared Libraries:

1. Less program loading time
2. Less memory space
3. Less disk space to store binaries

Static Linking Diagram

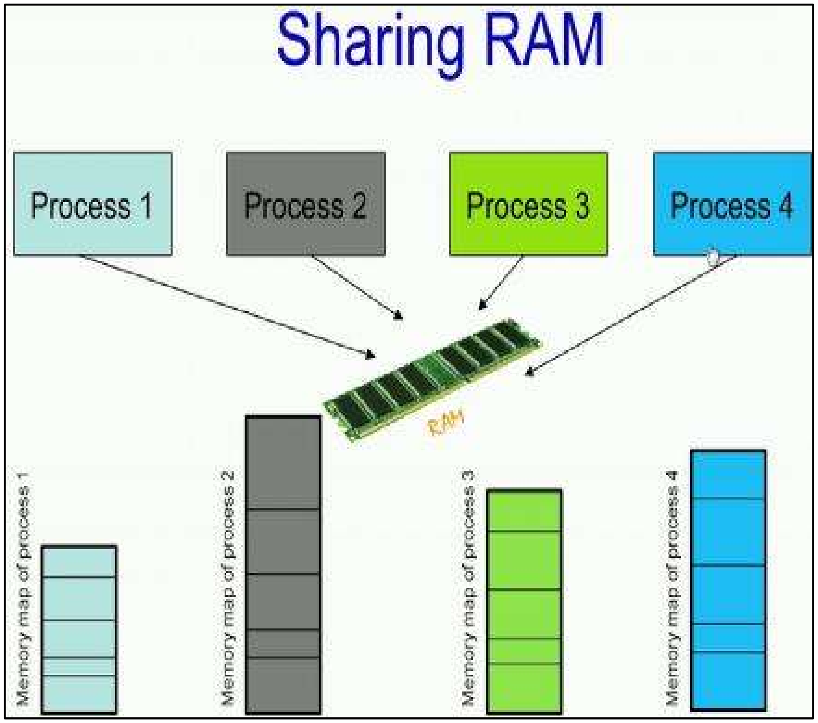


Address Binding Schemes

- Address binding of instructions and data to memory addresses can happen at three different schemes:
  - **Compile time**: when the ***absolute code*** will be generated by the compiler, containing actual physical addresses
  - **Load time**: the compiler must generate ***relocatable code***, which references addresses relative to the start of the program.
  - **Execution time**: If a program can be moved around in memory during execution, then binding must be delayed until execution time. This is the method which is implemented by most **modern OSes.**

Memory Management Approaches

1. **Single Contiguous Model**
2. **Partition with Contiguous Allocation**
3. **Swapping**
4. **Segmentation**
5. **Paging**



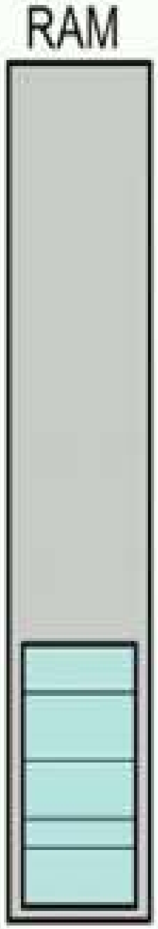
1. Single Contiguous Model

- Is one of the most primitive ways of managing memory especially done for the older on a operating systems.
- So RAM is occupied by one process at a time. Essentially at any particular instant there is only one process and its memory size is restricted by the RAM size.
- After this process completes executing, the next process will be loaded into the RAM.

Single Contiguous Model

- No sharing
  - One process occupies RAM at a time
  - When one process completes, another process is allocated RAM

Process memory size is restricted by RAM size



2. Partition with Contiguous Allocation

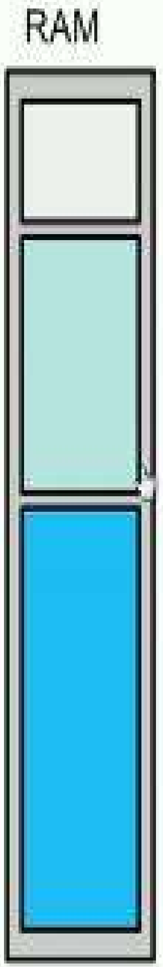
- In this approach at any instant of time, we could have multiple processes that occupy the RAM simultaneously but each process should be contained in a single contiguous partition of memory.
- **This approach is a slight improvement over the single contiguous model.**
- Main memory is usually partitioned into two **parts**:
  - Resident operating system area.
  - User processes area.
- **The Partition Table** would have the base address of a process, the size of the process and a process identifier.
- When a process completes execution, the area in RAM that it holds will be de-allocated. Consequently the entry corresponding to the partition table will be free.

Partition Model

- As long sufficient contiguous space is available, new processes are allocated memory.

Partition Table

Memory Address	Size	Process	Usage	
0x0	120k	4	In use	🟦
120k	60k	1	In use	▨
180k	30k		Free	□

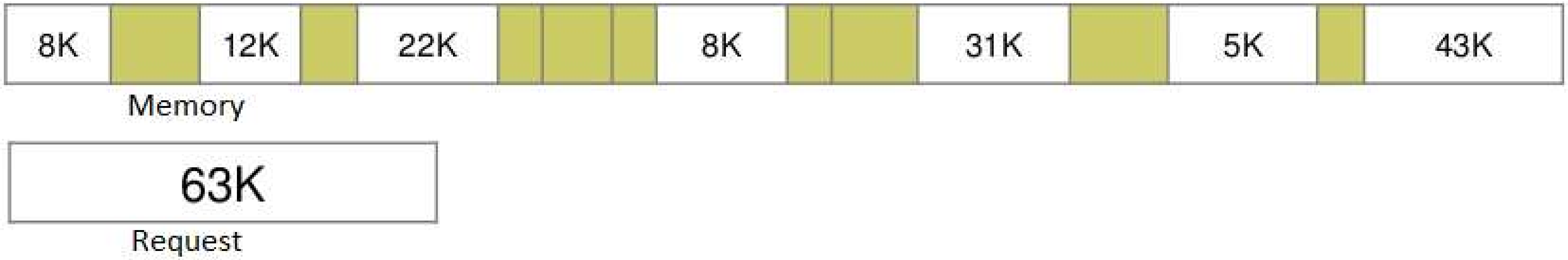


Limitations of
Partition with Contiguous Allocation

- Each Process needs to be entirely in RAM.
- Allocation needs to be in contiguous memory
- External Fragmentation
- Limit the size of the process by RAM size
- Performance Degradation

External Fragmentation

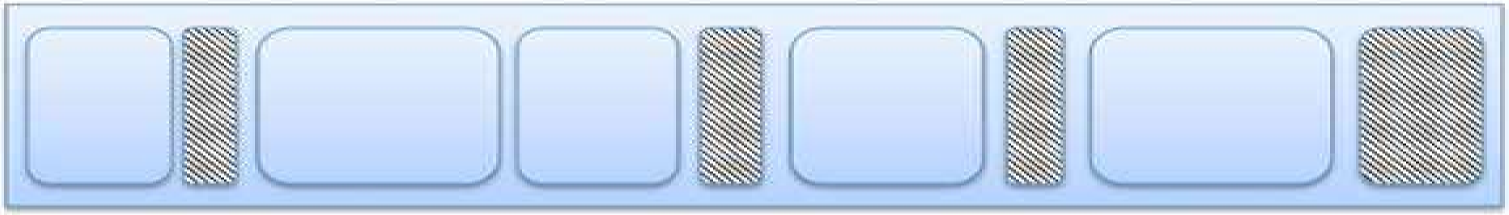
- **External Fragmentation** – It is the case when total memory space exists to satisfy a request, but it is not contiguous. This is a problem with Partition with Contiguous Allocation memory management approach.



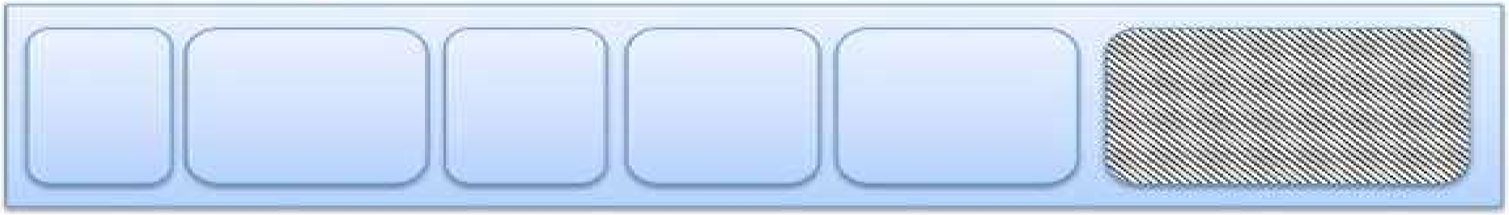
Compaction

- **Compaction:** is shuffling memory contents to place all free memory together in one large block
- Compaction is a solution to Reduce external fragmentation.
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time

Fragmented memory before compaction



Memory after compaction



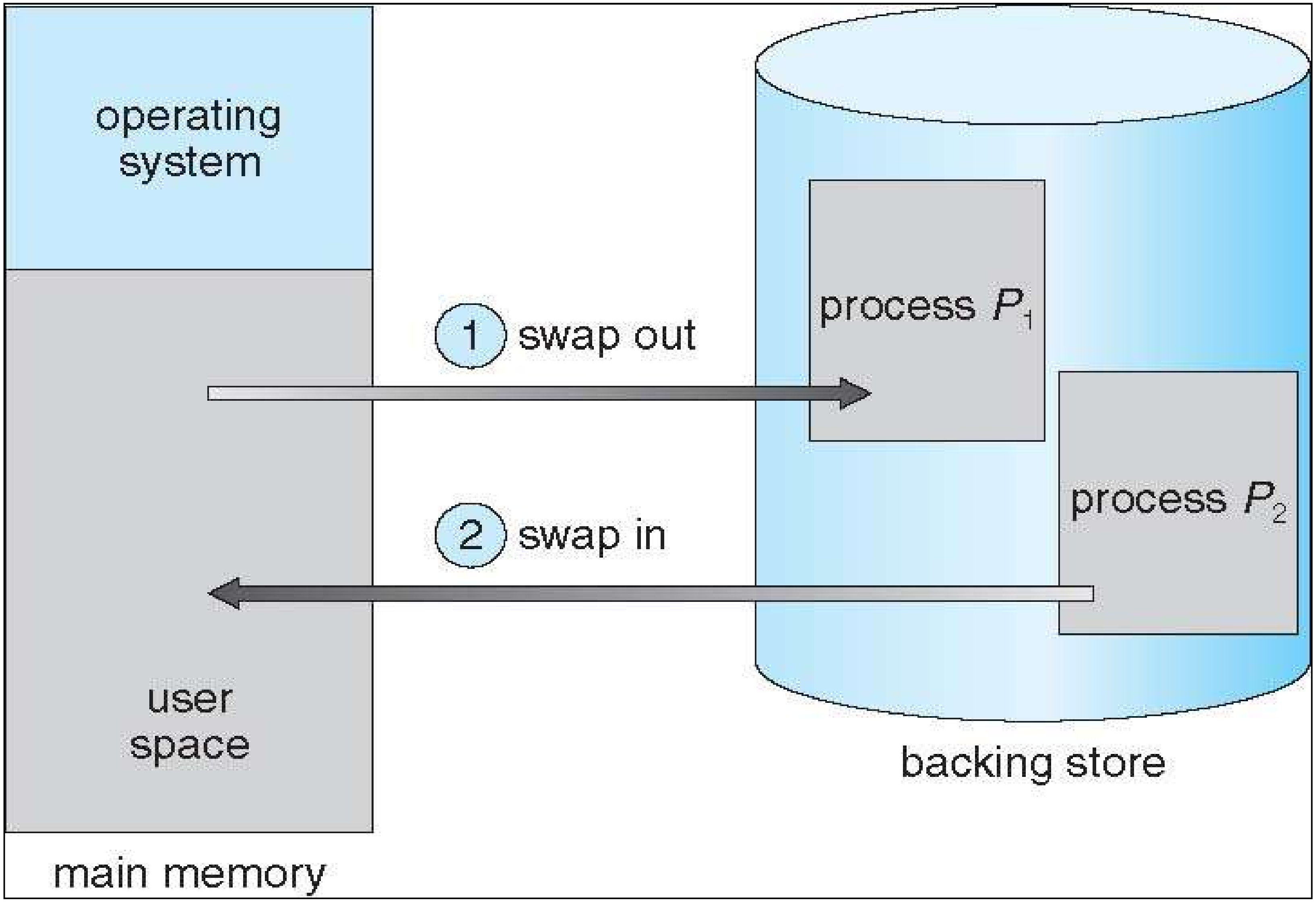
3. Swapping

- **Swapping** is a technique in which a process can be swapped temporarily out of memory to Backing Store (HD or SSD), and then brought back into memory for continued execution
- **Backing store** – is a fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- Standard swapping is not used in modern operating systems but a modified version is commonly used that is Enable Swap only when free memory is extremely low.

Swapping (Cont.)

- **Q\** Does the swapped out process need to swap back in to same physical addresses?
- A\\ If compile-time or load-time address binding schemes are used, then processes must be swapped back into the same memory location. If execution time binding is used, then the processes can be swapped back into any available location.
- **Q\** What is the swapping procedures that are found on current operating systems (i.e., UNIX, Linux, and Windows)?
  - Swapping normally disabled
  - Started if more than threshold amount of memory is allocated
  - Disabled again once memory demand is reduced below threshold

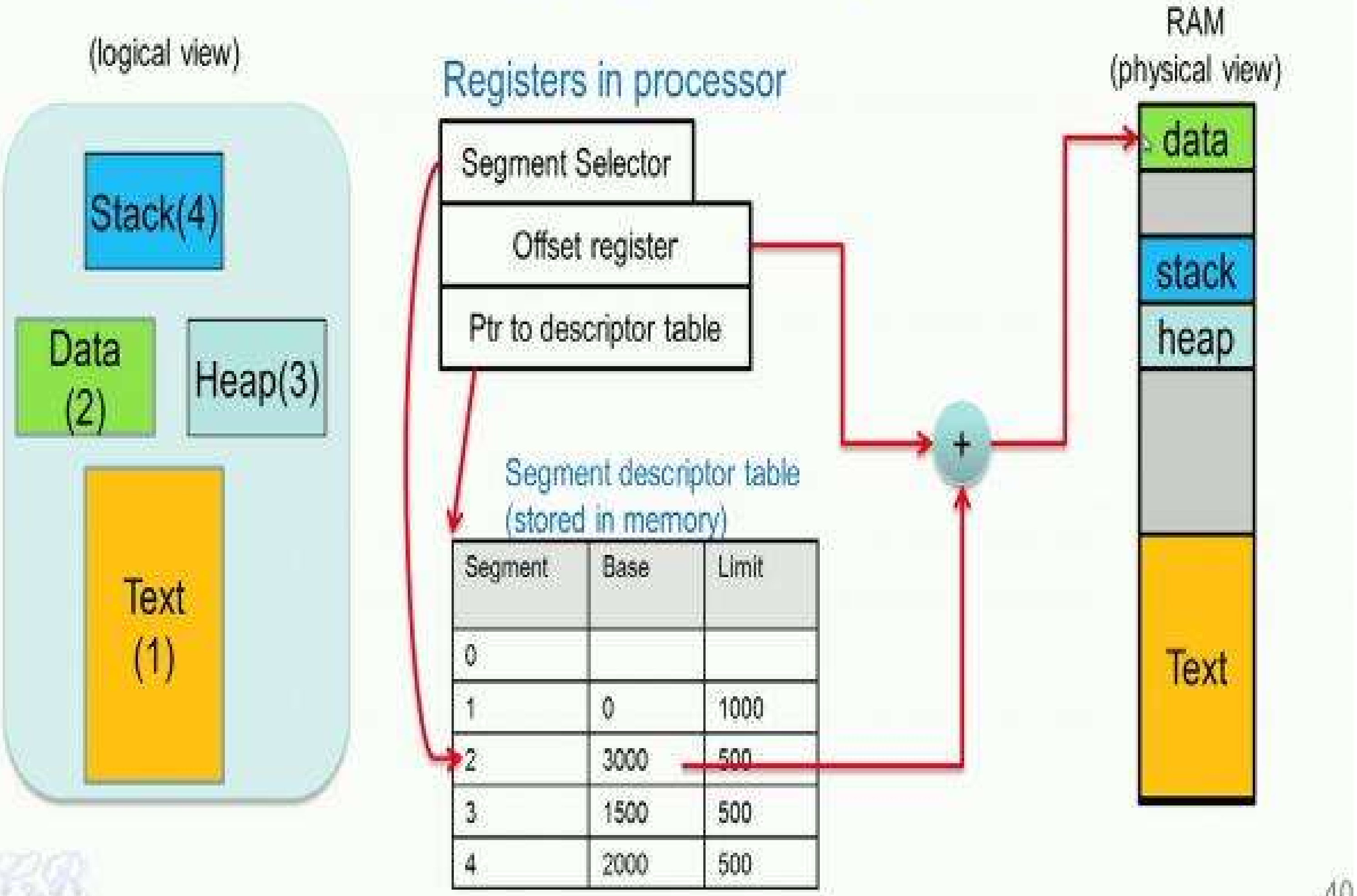
Schematic View of Swapping



4. Segmentation

- **Segmentation**: is a Memory-management approach that supports the view of the program as a collection of segments by segmenting processes and loading them into different non-contiguous addressed spaces in memory with a segment number and an offset.
- In Segmentation we have a **Segment Descriptor Table** which is stored in memory. Each row in the segment descriptor table refers to one particular segment. For instance, the Data segment 2 is at an offset 2 in the Segment descriptor table, and the offset would specify the Base address in RAM and the Limit of the segment.

Address Mapping with Segmentation



Segment	Base	Limit
0		
1	0	1000
2	3000	500
3	1500	500
4	2000	500

5. Paging

- **Paging** is to divide physical memory into fixed-sized blocks called **frames** and divide logical memory into blocks of same size called **pages**
- Physical address space of a process can be noncontiguous.
- Paging eliminates external fragmentation , but it still suffers from Internal fragmentation
- Page Size is a power of 2, usually between 512 bytes and 16 Mbytes
- To load a process of size S where

(N-1) pages < S < (N) pages,

it is required to find ***N*** free frames to load the process.

Page Table

The **page table** is the table used to look up what frame a particular page is stored in at the moment. It translates logical to physical addresses.

Associative Memory is a special fast-lookup hardware cache that can solve the two memory access problem if page table is stored in main memory.

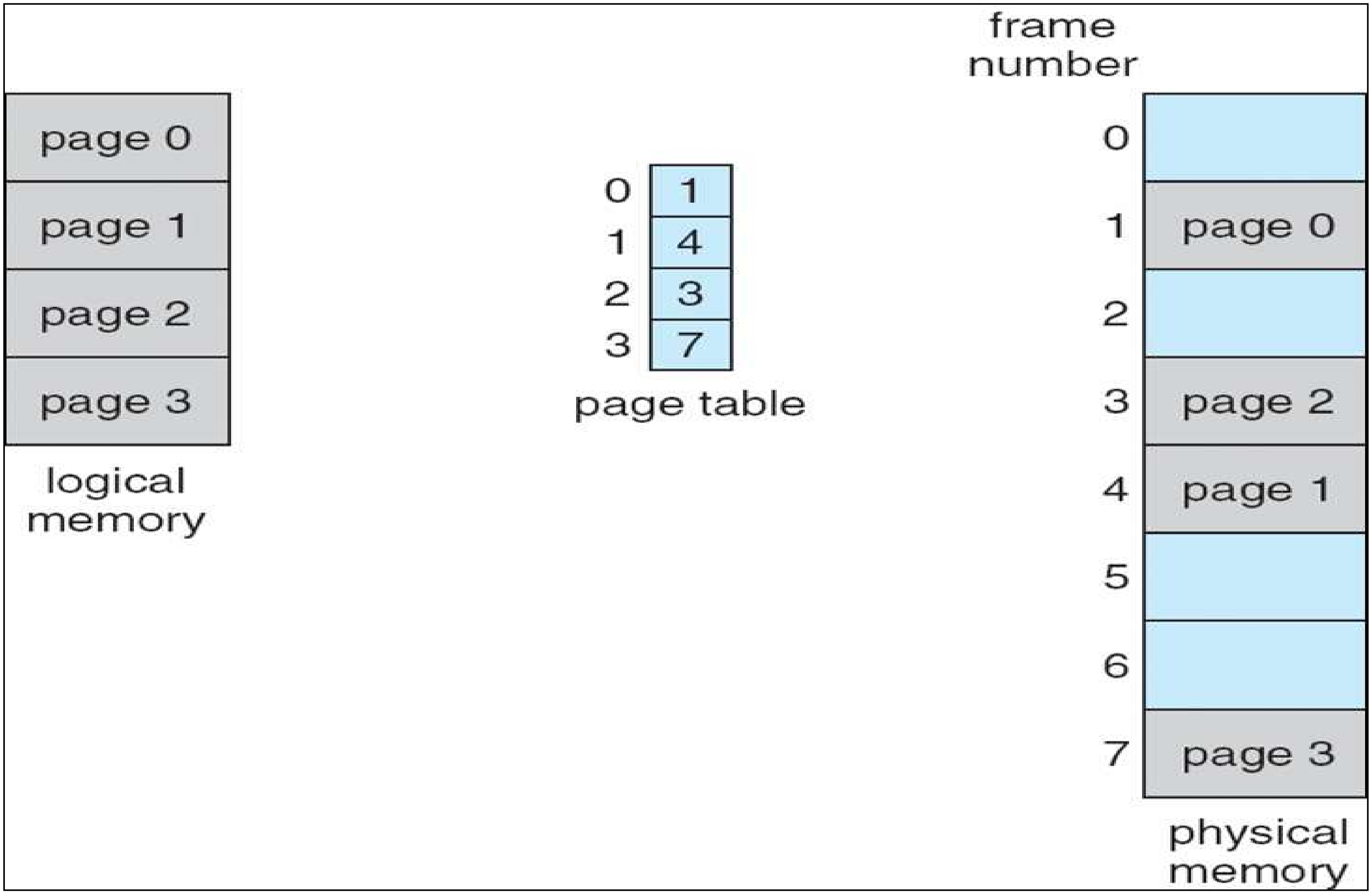
Q\ Why Page size selection is critical?

- A large page will result in increase of internal fragmentation
- A small page size will increase the size of the page table

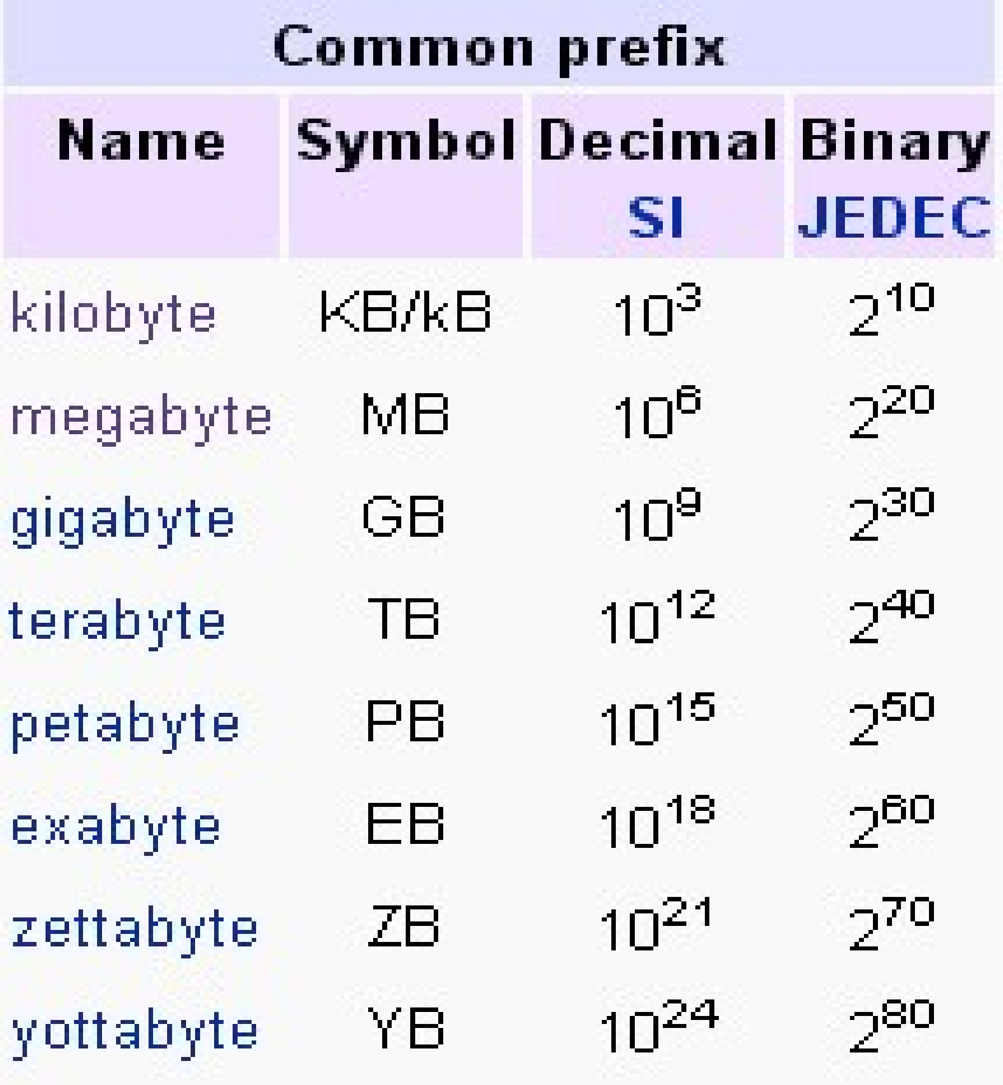
The **Page Table Structure** can be:

- **Basic Paging**: A single page table which stores the page number and the offset
- **Hierarchical Paging**: A multi-level table which breaks up the virtual address into multiple page tables.

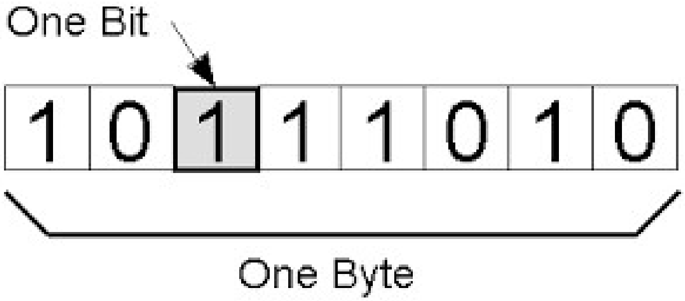
Basic Paging Diagram



What is Kilobyte?



Common prefix			
Name	Symbol	Decimal SI	Binary JEDEC
kilobyte	KB/kB	10^3	2^{10}
megabyte	MB	10^6	2^{20}
gigabyte	GB	10^9	2^{30}
terabyte	TB	10^{12}	2^{40}
petabyte	PB	10^{15}	2^{50}
exabyte	EB	10^{18}	2^{60}
zettabyte	ZB	10^{21}	2^{70}
yottabyte	YB	10^{24}	2^{80}



**1 kilobyte =
1024 bytes**

The Page Table size Calculation

- **Example**: Calculate the **page table size** considering

  (32-bit) logical address space and Entry size = 4 Bytes

  - For the two cases below

  A) Page size of 2 K Bytes

  B) Page size of 4 K Bytes

- **Solution**:

  A) Page size of 2 KB = $2 * 2^{10}$ = 2^{11} Bytes

  No. of table entries = address space / page size

  = $(2^{32} / 2^{11}) = 2^{21} = 2 * 2^{20}$ = 2 M entries

  Page table size = table entries * entry size = 2 M * 4 = 8 M Bytes

  B) Page size of 4 KB = $4 * 2^{10}$ = 2^{12} Bytes

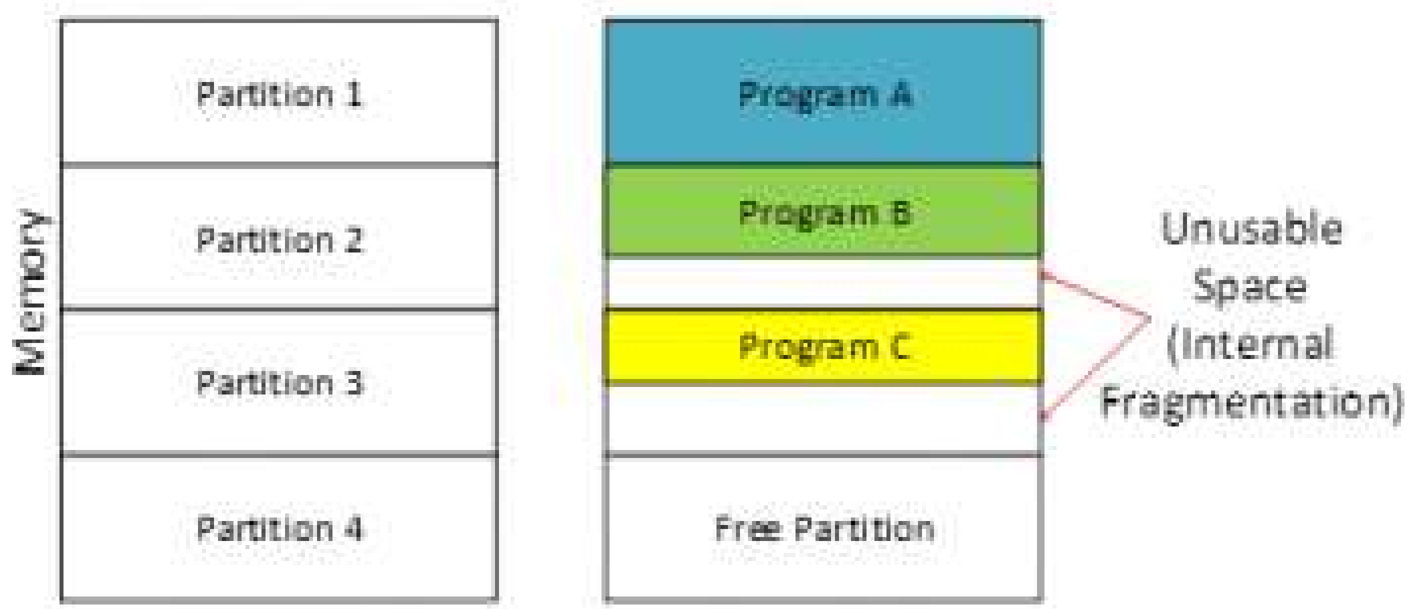
  No. of table entries = address space / page size

  = $(2^{32} / 2^{12}) = 2^{20}$ = 1 M entries

  Page table size = table entries * entry size = 1 M * 4 = 4 M Bytes

Internal Fragmentation

- **Internal Fragmentation** – is this size difference in memory that happens when we divide memory to fixed partitions and then allocated memory may be slightly larger than requested memory.



Calculating Internal Fragmentation

Example: Calculate the number of pages and internal fragmentation considering:

A) Page size = 2048 bytes

B) Page size = 4096 bytes

Assuming that Process size = 70450 bytes for both cases.

Solution:

A) No. of pages= **ceiling (**process size / page size**)**

= **ceiling (**70450 / 2048**)** = 35 pages

Internal fragmentation = (No. of pages * page size) – process size

= 35 * 2048 – 70450 = 1230 bytes

B) No. of pages= **ceiling (**process size / page size**)**

= **ceiling (**70450 / 4096**)** = 18 pages

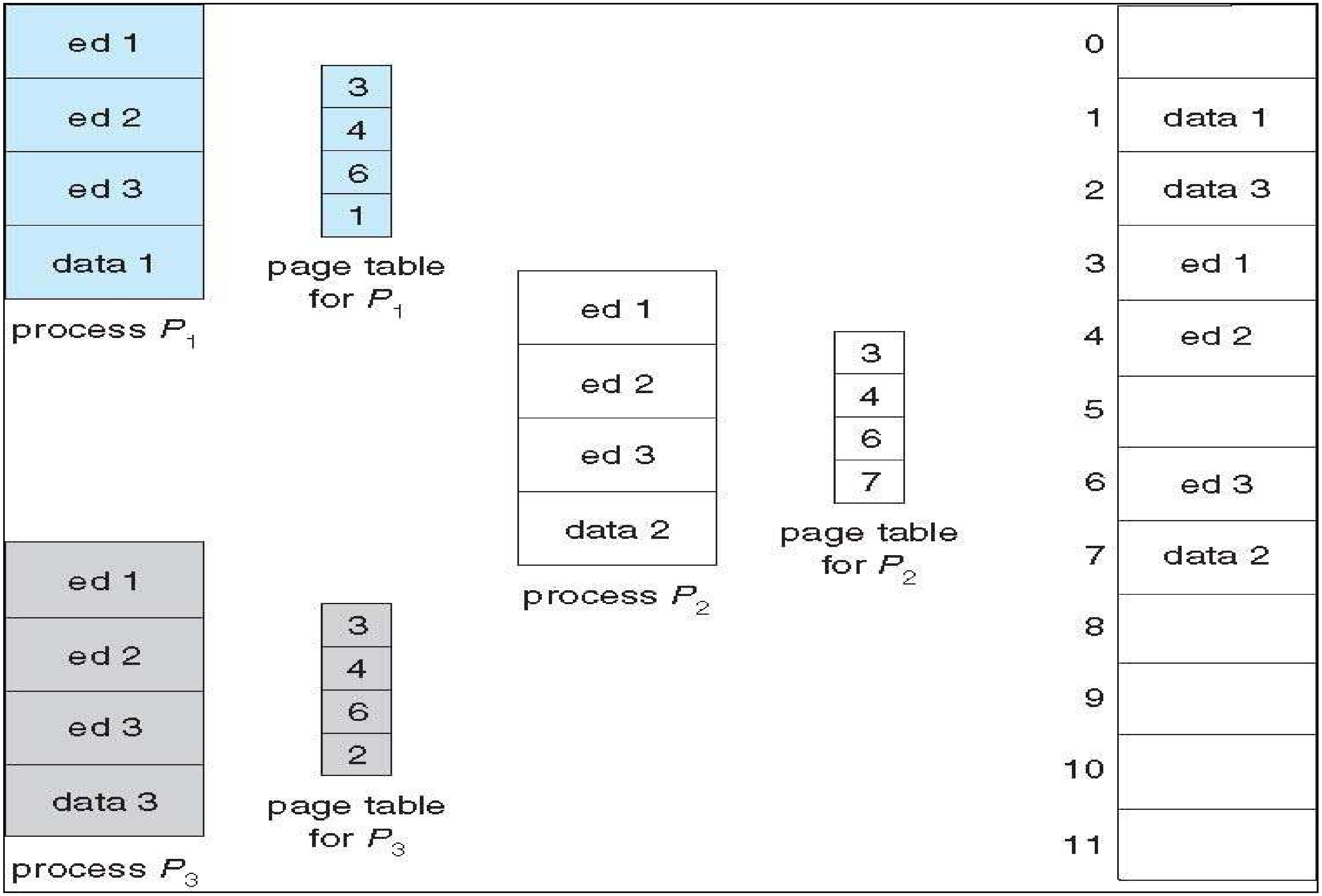
Internal fragmentation = (No. of pages * pages size) – process size

= 18 * 4096 – 70450 = 3278 bytes

Shared Pages

- **Shared code**
  - One copy of read-only code can be shared among processes (i.e., text editors, compilers, window systems)
  - It is also useful for inter-process communication if sharing of read-write pages is allowed
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Diagram



Virtual Memory Basics

- **Virtual Memory** increases the available memory of the computer by enlarging the "address space," or places in memory where data can be stored. It does this by using hard disk space for additional memory allocation.
- However, since the hard drive is much slower than the RAM, data stored in virtual memory must be mapped back to real memory in order to be used.
- Q\ Why most real processes do not need all their pages?
  - Error handling code is not needed unless that error occur.
  - Only a small fraction of the arrays are actually used.
  - Certain routines of programs are rarely used.

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to virtual memory
  - **Logical address (virtual address)** – it is the address generated by the process executing currently on the CPU.
  - **Physical address** – it is the address seen by the memory management unit
- **Memory-Management Unit**: is the Hardware device that at run time maps virtual to physical address
- The user process deals with logical addresses; it never sees the real physical addresses

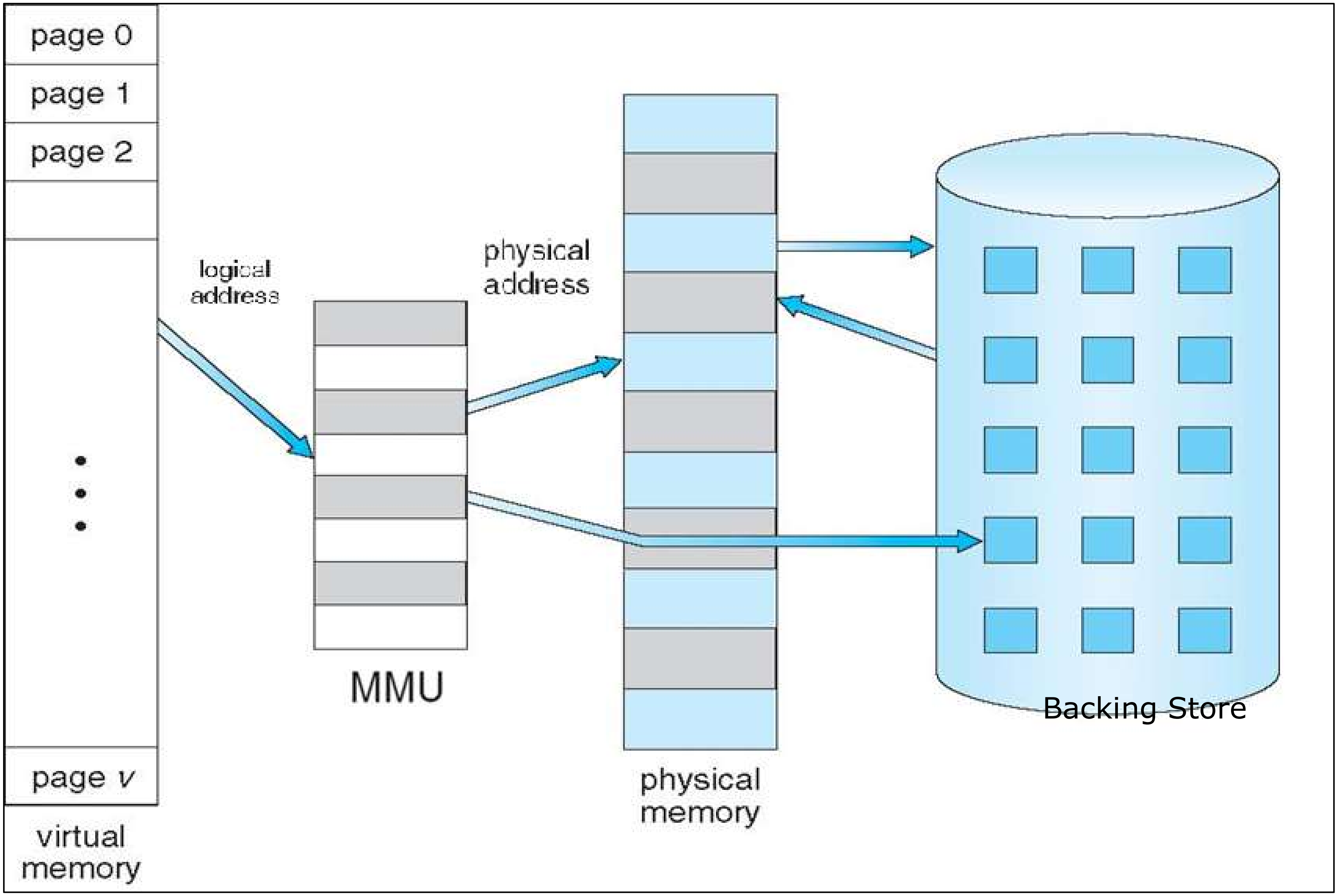
Benefits of Virtual Memory

- In virtual memory, Logical address space can therefore be much larger than physical address space
- Every process executing in the system would have its own process page table
- **Benefits** of Virtual Memory are:
  - Only part of the program needs to be in memory for execution
  - Allows address spaces to be shared by several processes
  - More programs can run concurrently

Page Fault: is a type of trap raised when a running process accesses a memory **page** that is not currently in the physical RAM

Page replacement is finding some page in memory, which is not really in use, in order to page it out.

The General Layout of Virtual Memory



Thrashing and Memory Leaks

- **Thrashing**: it is the case when a process does not have “enough” pages, then page-fault rate will become very high and the process will be busy swapping pages in and out
- **Memory Leak:** occurs when an application is using more RAM than it normally does which in turn slows down the system, causing it to struggle with performing even the basic tasks.

Optimizing Applications Performance

- Optimizing Applications depends heavily on memory organization in the computer system. The next few points offer some pointers for **improving the performance of applications under Windows :**

1. Adding More Physical Memory
2. Defragment the Hard Drive containing the paging file
3. Installing Applications to the Fastest Hard Drive
4. Getting the Latest Device Drivers
5. Move Extra workload to Another Server

1. Adding More Physical Memory

- All applications run in RAM, of course, so the more RAM you have, the less likely it is that Windows will have to store excess program or document data in the page file on the hard disk, which is a real performance killer.

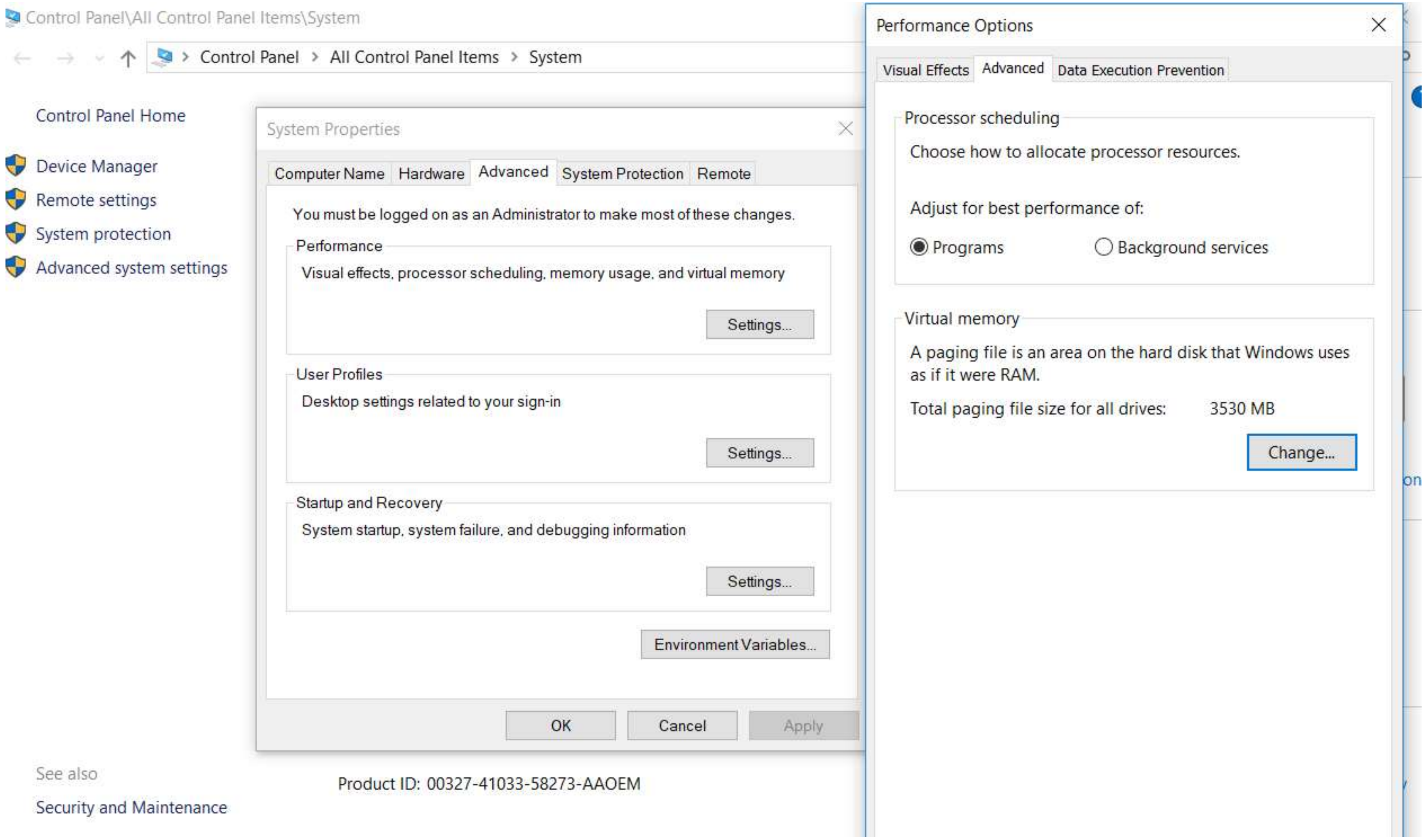
Use one of the following **Windows monitoring tools** to watch the available memory:

- **Task Manager** —Display the Performance tab and watch the Physical Memory: Available value. .
- **Resource Monitor** —Display the Memory tab and watch the Available to Programs value. .
- **Performance Monitor** —Start a new counter, open the Memory category, and then select the Available Mbytes counter.

2. Defragment the Hard Drive containing the Paging File

- This is needed when the page file is located on a disk that is heavily used by other applications.
- **Hard Disk Defragmentation** is to organize the files parts in contiguous sectors on the disk, thereby improving computer performance and maximizing disk space.
- The **solution steps** are:
  1. move the page file to another drive temporarily,
  2. set the paging file on the original drive to be fragmented to 0 MB.
  3. reboot the system to enable the other paging file to be used.
  4. perform the disk defragmentation on the original drive.
  5. set the paging file on the original drive to the necessary values, and reboot.

Paging file in Windows 10 (not required in the exam)


Control Panel\All Control Panel Items\System

Control Panel > All Control Panel Items > System

System Properties

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

Product ID: 00327-41033-58273-AAOEM

Performance Options

Processor scheduling

Choose how to allocate processor resources.

Adjust for best performance of:

● Programs ○ Background services

Virtual memory

A paging file is an area on the hard disk that Windows uses as if it were RAM.

Total paging file size for all drives: 3530 MB

3. Installing Applications to the Fastest Hard Drive

- If your system has multiple hard drives that have different performance ratings, install your applications on the fastest drive. This enables Windows to access the application's data and documents faster.

4. Getting the Latest Disk Drivers

- If your application works with a device, check with the manufacturer or Windows Update to see whether a newer version of the device driver is available. In general, the newer the driver, the faster its performance.

5. Move Extra workload to Another Server

- For Server Machine, you may also elect to offload some of the workload to another system.