Limitations of Paging:

All the memory management techniques using paging learnt so far required that entire process be in memory for it to be executed.

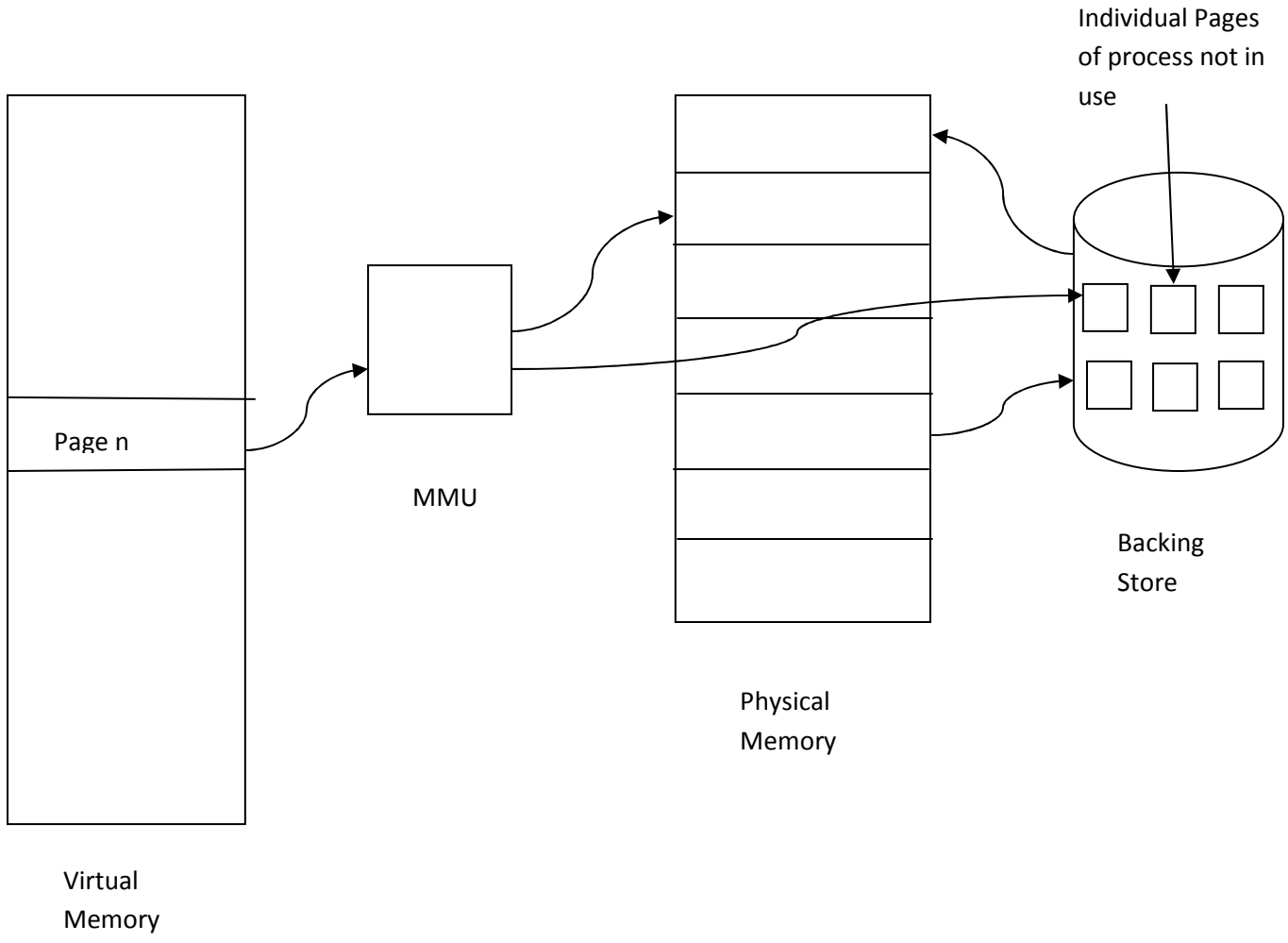
Every logical page address should have a corresponding physical frame in memory. Due to this program cannot have its logical address space larger than physical address space.

In many cases it is not necessary to have the entire program inside memory.

Solution:

Allow the program to have a larger virtual address space than the physical address space.

Only load those pages to the program into physical memory that are required. Use a backing store to hold all the individual pages of process not in use.



Advantage:

Program no longer constrained to physical memory size.

More programs could run since each would now occupy less memory.

How it is implemented?

Process Swapping:

Remove a process from memory and write it to disk and bring a new process from disk.

Moving entire process in and out of memory onto the disk is costly. So instead of doing it at process level it is better to do it at page level.

Demand paging:

Load the pages of a process in memory only when they are needed.

How to do Demand Paging

Some mechanism is needed to know which page of process is stored in memory and which is stored on disk. Need some hardware support to distinguish the origin of pages

Solution:

Use of valid/in-valid bit of MMU.

When the bit is VALID associated page is legal and in memory.

When bit is INVALID or 0 page is either invalid or currently on disk.

The page table entry for a page in memory is set as usual. The pages not in memory are either marked invalid or the actual address of page on disk is stored.

So a bit 0 or INVALID in MMU will cause a page fault

Steps involved in case of page fault:

1. Page fault will cause a trap to Operating system
2. Check if the reference was a valid or invalid memory access
3. If invalid terminate the process
4. If valid free frame (use one from free frame list or kick some frame out.)
5. Perform disk read to get the page

6. Re update the page table and set the valid bit
7. Restart instruction that was interrupted by trap

Performance of Demand Paging

Typical Memory access time: ma: 200 nanoseconds

A typical hard disk has average latency of 3 milliseconds a seek of 5 milliseconds and transfer time of 0.05 milliseconds.

Typical disk access time: 8 milliseconds

Let p be the page fault rate. % of memory access that cause page fault.

Effective memory access time = (1-p)*200 + p(8milliseconds + 200nanoseconds)

= 200 nanoseconds + p*8,000,000 nanoseconds

Thus effective access time is directly proportional to page fault rate. If we want a performance degradation of less than 10% $p < 0.0000025$