Unit - IV - Memory Management

* Memory Management Requirements -

While surveying the various mechanisms and policies associated with memory management, it is helpful to keep in mind the requirements that memory management is intended to satisfy. [LIST 93] suggests five requirements.

- Relocation
- Protection
- Sharing
- Logical organisation
- Physical organisation.

* Memory Management Terms -

Frame :-

A fixed-length block of main memory.

A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.

A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into a available region of main memory (segmentation) or that segment may be divided into pages which can be individually copied into main memory (combined segmentation & paging).

* Addressing Requirements for a process –

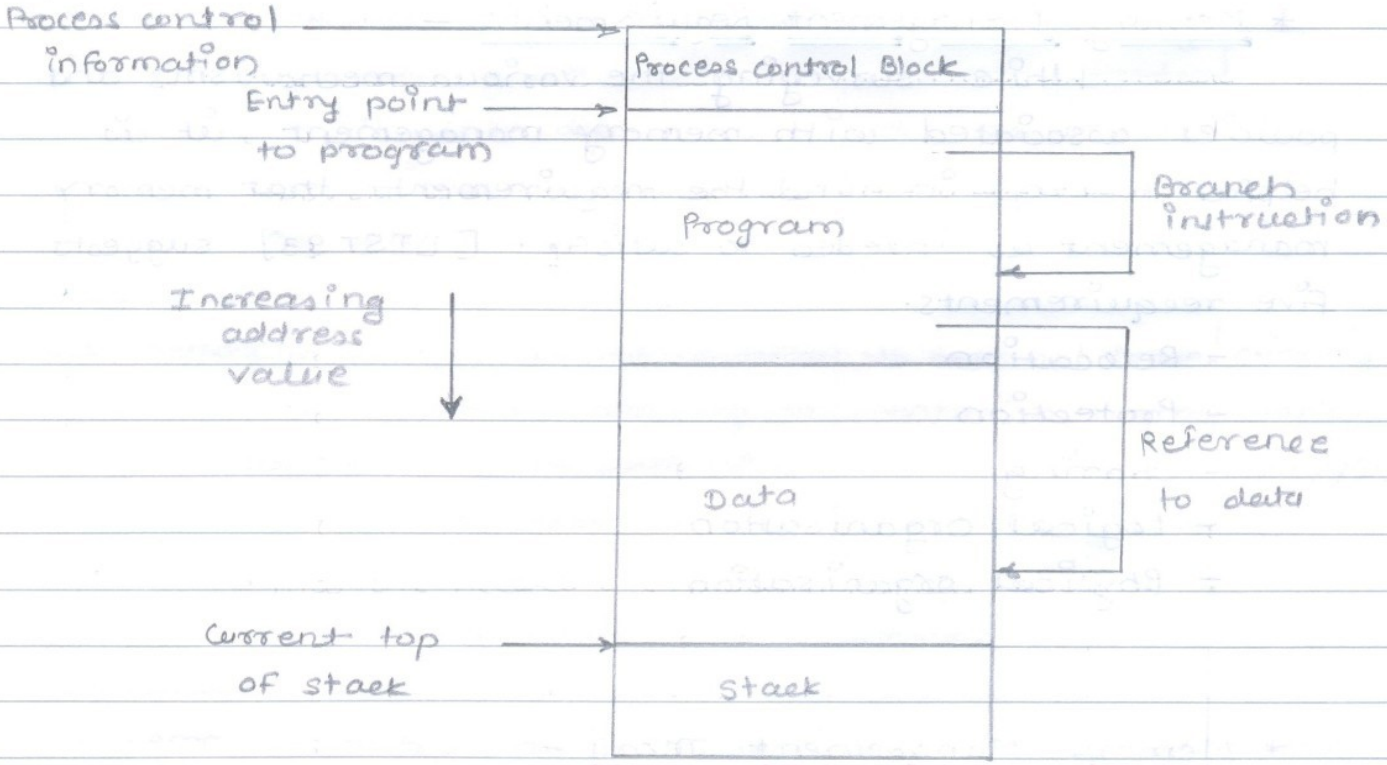


Fig – Addressing Requirements for a process

Note that the memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software).

* Physical organisation –

The main memory available for a program plus its data may be insufficient. In that case the programmer must engage in practice known as overlaying, in which the program and data are organised in such a way that various modules can be assigned the same region of memory, with a main program responsible for switching the modules in and out as needed. Even with the aid of compiler tools, overlay programming waste programmers time.

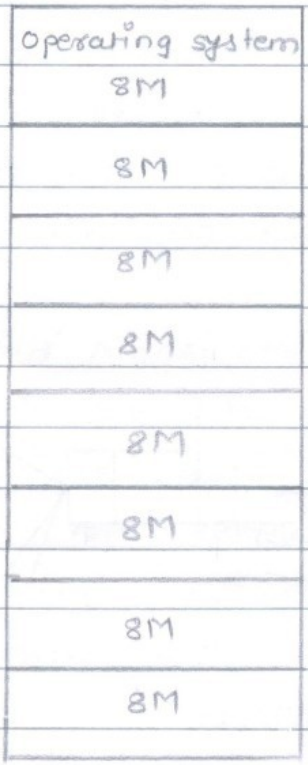
* Memory Partitioning *

Memory Management Techniques –

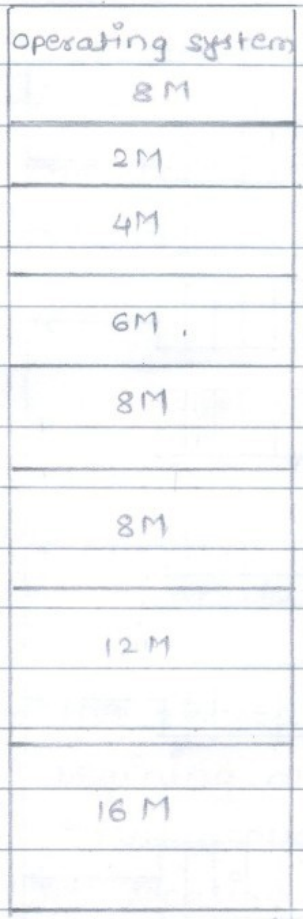
Technique	Description	Strengths	Weaknesses
Fixed partitioning	Main memory is divided into a no. of static partitions at system generation time. A process may be loaded into a partition of equal / greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; max. no. of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple paging	Main memory is divided into a no. of equal size frames. Each process is divided into no. of equal size pages of the same length as frames. a process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragementation.	A small amount of internal fragementation.
Simple segmentation	Each process is divided into a number of	No internal fragementation;	External fragementation.

Technique	Description	Strengths	Weaknesses
	segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	Improved memory utilisation & reduced overhead compared to dynamic partitioning.	
Virtual memory paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual memory segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation; higher degree of multiprogramming; large virtual address space; protection & sharing support	Overhead of complex memory management

* Fixed Partitioning *



a] Equal-size partitions



b) Unequal size portions

Main memory utilization is extremly inefficient. Any program, no matter how small, occupies an entire partition. In our example, there may be a program whose length is less than 2 Mbytes. yet it occupies an 8-Mbytes partition whenever it is swapped in. This phenomenon, in which there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition, is referred to as internal fragmentation.

* Placement Algorithm –

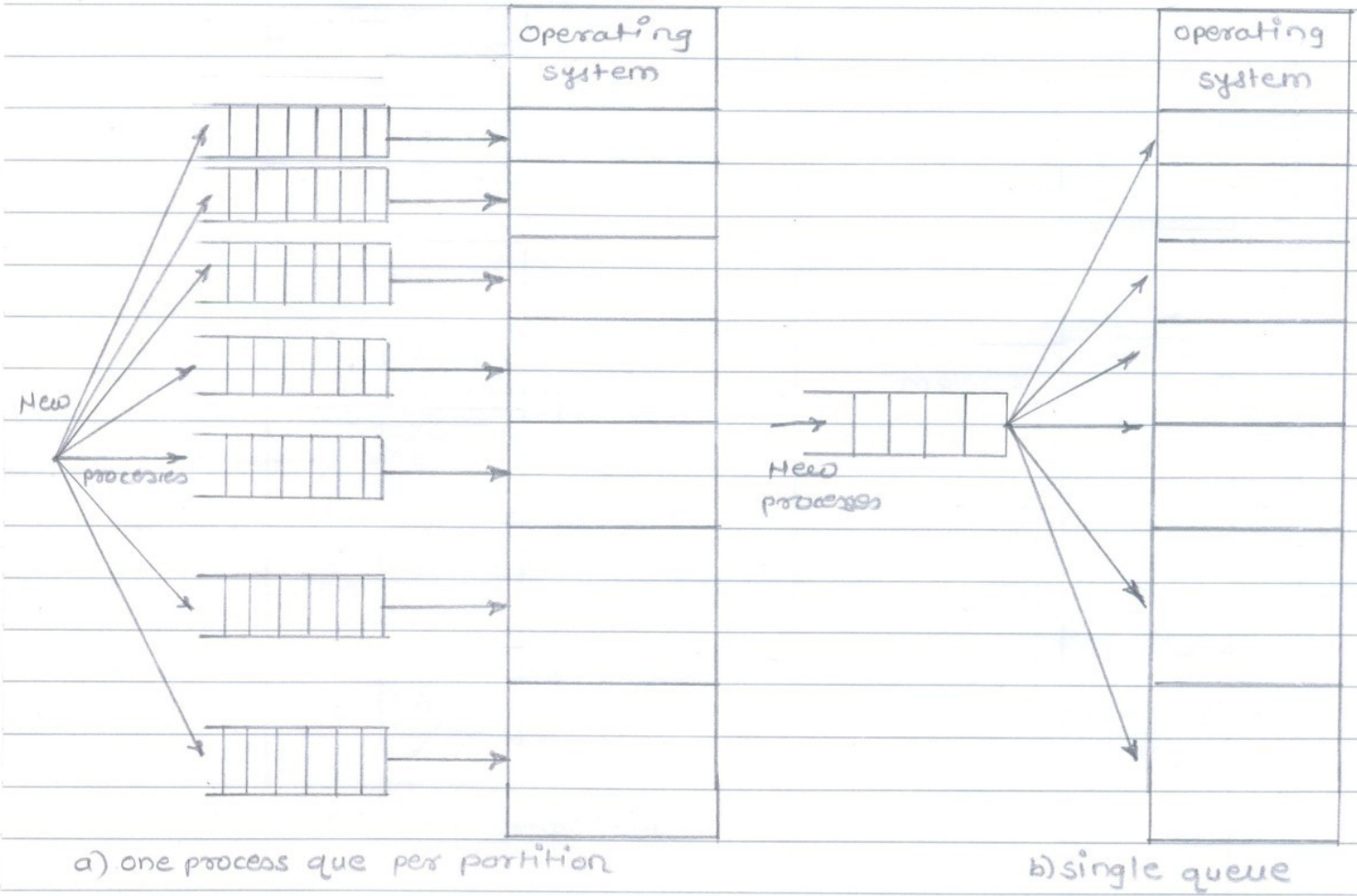


Fig – Memory Assignment for Fixed Partioning

The use of fixed partitioning is almost unknown today. One example of a sucessful operating system that did use this technique was an early IBM mainframe operating system, OS / MFT (Multiprogramming with a Fixed Number of Tasks.)

* Dynamic Partitioning *

As time goes on, memory becomes more and more fragmented, and memory utilization declines. This phenomenon is referred to as external fragmentation, indicating that the memory that is external to all partitions becomes increasingly fragmented. This is in

contrast to internal fragmentation.

One technique for overcoming external fragmentation is compaction. From time to time, the operating system shifts the processes so that they are contiguous & so that all of the free memory is together in one block.

* Placed Algorithm –

Three placement algorithms that might be considered are best-fit, first-fit and next-fit. All, of course, are limited to choosing among free blocks of main memory that are equal to or larger than the processes to be brought in. Best-fit chooses the block that is closest in size to the request. First-fit begins to scan memory from the beginning and chooses the first available block that is large enough. Next fit begins to scan memory from the location of the last placement, and chooses the next available block that is large enough.

Fig. shows an example memory configuration after a number of placement and swapping-out operations. The last block that was used was a 22-Mbyte block from which a 14-Mbyte partition was created. Fig. (b) shows the difference between the best-, first-, and next-fit placement algorithms in satisfying a 16-Mbyte allocation request. Best-fit will search the entire list of available blocks & make use of the 18-Mbyte block, leaving a 2-Mbyte fragment. First-fit results in a 6 M-byte fragment, and next-fit results in a 20-Mbyte fragment.

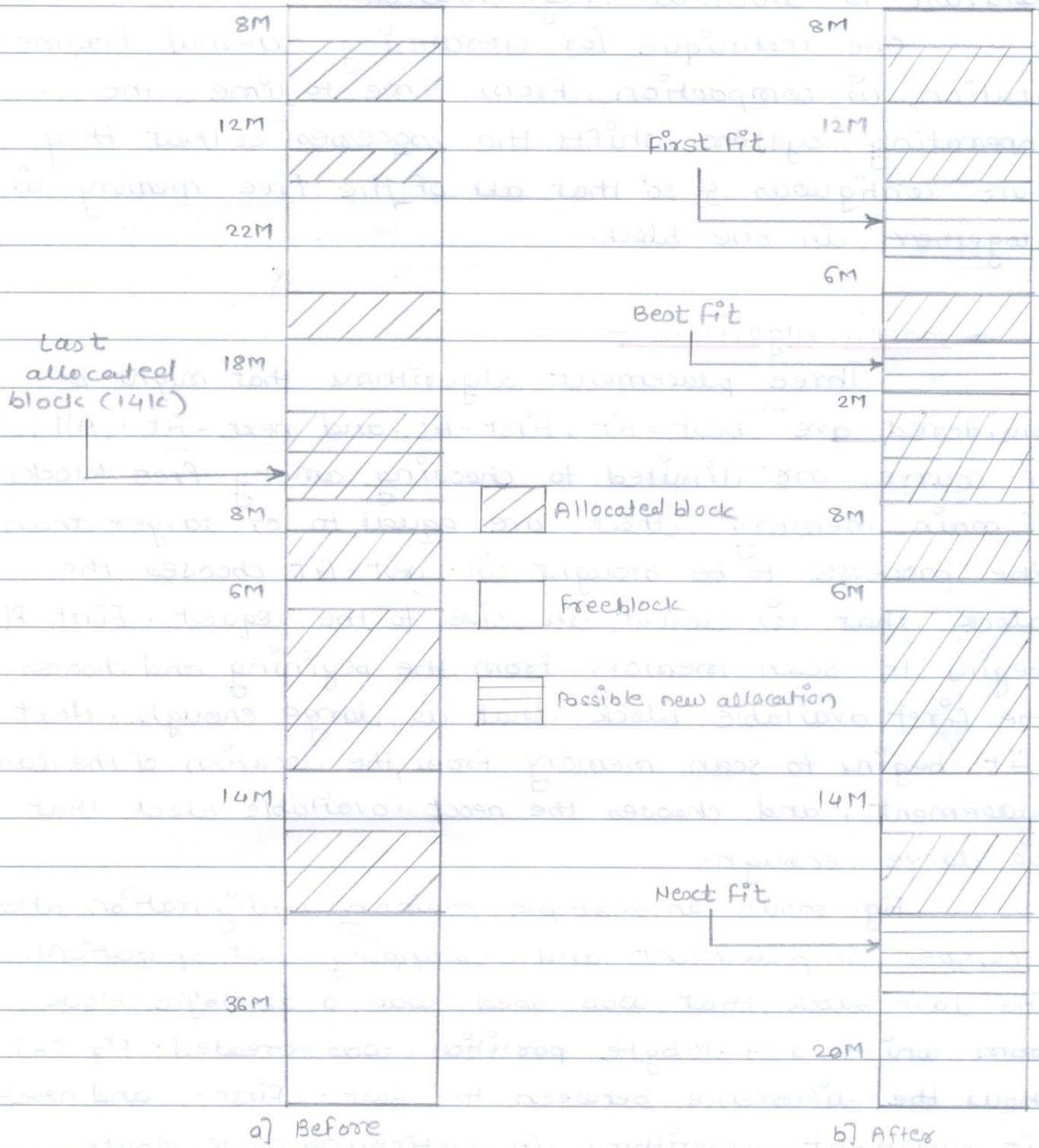


Fig - Example Memory Configuration before & after allocation of 16-Mbyte block

* Buddy System –

Both fixed and dynamic partitioning schemes have drawbacks. A fixed partitioning scheme limits the number of active processes & may use space inefficiently if there is a poor match between available partition sizes & process sizes. A dynamic partitioning scheme is more complex to maintain and includes the overhead of compaction. An interesting compromise is the buddy system ([KNUT 97], [PETE 77]).

In a buddy system, memory blocks are available of size 2^K words, $L \leq K \leq U$,
where

2^L = smallest size block that is allocated.
2^U = Largest size block that is allocated; generally 2^U is the size of the entire memory available for allocation.

Block Fig. gives an example using a 1-Mbyte initial block. The first request, A, is for 100 Kbytes, for which a 128 K block is needed. The initial block is divided into two 512K buddies. The first of these divided into two 256 K buddies, and the first of these is divided into two 128 K buddies, one of which is allocated to A. The next request B, requires a 256 K block. Such a block is already available & is allocated. The process continues with splitting and coalescing occuring as needed. Note that when E is released, two 128 K buddies are coalesced into a 256 K block, which is immediately coalesced with its buddy.

1-Mbyte block	1M					
Request 100K	A=128K	128K	256K	512K		
Request 240K	A=128K	128K	B=256K	512K		
Request 64K	A=128K	C=64K	64K	B=256K	512K	
Request 256K	A=128K	C=64K	64K	B=256K	D=256K	256K
Release B	A=128K	C=64K	64K	256K	D=256K	256K
Release A	128K	C=64K	64K	256K	D=256K	256K
Request 75K	E=128K	C=64K	64K	256K	D=256K	256K
Release C	E=128K	128K	256K	D=256K	256K	
Release E	512K			D=256K	256K	
Release D	1M					

Fig – Example of a Buddy System

* Relocation –

A logical address is a reference to a memory location independant of the current assignment of data to memory; a translation must be made to a physical address before the memory access can be achieved. A relative address is a particular example of logical address, in which the address is expressed as a location relative to some known point, usually a value in a processor register. A physical address, or absolute address, is an actual location in main memory.

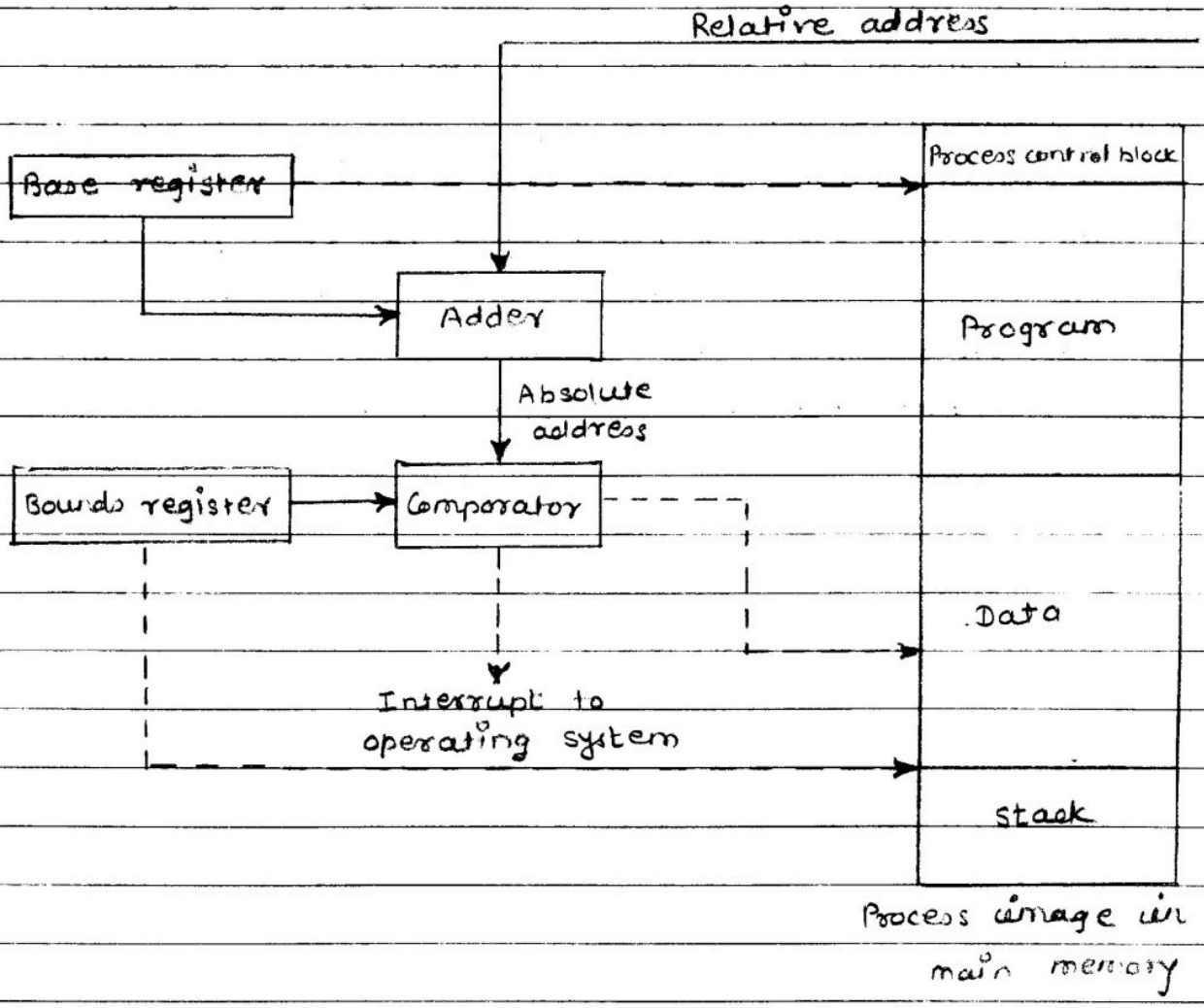


Fig - Hardware support for Relocation

* Paging –

Both unequal fixed-size and variable-size partitions are inefficient in the use of memory; the former results in internal fragmentation, the latter in external fragmentation. Suppose, however, that main memory is partitioned into equal-fixed size chunks that are relatively small, and that each process is also divided into small fixed size chunks of the same size. Then the chunks of a process, known as pages, could be assigned to available chunks of memory, known as frames, or page frames. We show in this section that the wasted space in memory for each process is due to internal fragmentation consisting of only a fraction of the last page of a process. There is no external fragmentation.

The OS maintains a page table for each process. The page table shows a frame location for each page of the process. Within the program, each logical address consists of a page number and an offset within the page.

Frame Number	Main Memory
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

a] Fifteen available frames

	Main Memory
0	A·0
1	A·1
2	A·2
3	A·3
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

b) Load Process A

	Main Memory
6	A·0
1	A·1
2	A·2
3	A·3
4	B·0
5	B·1
6	B·2
7	
8	
9	
10	
11	
12	
13	
14	

c) Load process B

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

Fig (A) – Assignment of Process to Free Frames

	Process A Page Table		Process B Page Table		Process C Page Table		Process D Page Table		Free frame list
0	0	0	—	0	7	0	4		13
1	1	1	—	1	8	1	5		14
2	2	2	—	2	9	2	6		
3	3			3	10	3	11		
						4	12		

Fig – Data structures for the Example of Fig (A) at time epoch (f)

* Segmentation —

A user program can be subdivided using segmentation, in which the program and its associated data are divided into a number of segments. It is not required that all segments of all programs be of the same length, although there is a maximum segment length. As with paging, a logical address using segmentation consists of two parts, in this case a segment number and an offset.

Whereas paging is invisible to the programmer, segmentation is usually visible & is provided as a convenience for organising programs and data. Typically the programmer or compiler will assign programs and data to different segments. For purpose of modular programming, the program or data may be further broken down into multiple segments. The principle inconvenience of this service is that the programmer must be aware of the maximum segment size limitation.

* Virtual Memory *

* Virtual Memory Technology —

Virtual memory	A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites and program generated addresses are translated automatically to the corresponding machine addresses. The size of virtual storage

	is limited by the addressing scheme of the computer system & by the amount of secondary memory available & not by the actual number of main storage locations.
Virtual address	The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.
Virtual address space	The virtual storage assigned to a process.
Address space	The range of memory addresses available to a process.
Real address	The address of a storage location in main memory.

* Hardware And Control Structures –

Comparing simple paging & simple segmentation, on the one hand, with fixed & dynamic partitioning, on the other, we see the foundation for a fundamental breakthrough in memory management. Two characteristics of paging and segmentation are the keys to this breakthrough:

1] All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process may be swapped in and out of main memory such that it occupies different regions of main memory at different times during the course of execution.

2] A process may be broken up into a number of pieces (pages or segments) and these pieces need not be contiguously located in main memory during execution. The combination of dynamic run-time address translation and the use of a page or segment table permits this.

* Paging –

The term virtual memory is usually associated with systems that employ paging, although virtual memory based on segmentation is also used. The use of paging to achieve virtual memory was first reported for the Atlas computer [KILB62] and soon came into widespread commercial use.

* Page Table Structure –

The basic mechanism for reading a word from memory involves the translation of a virtual, or logical, address, consisting of page number and offset, into a physical address, consisting of frame number and offset, using a page table.

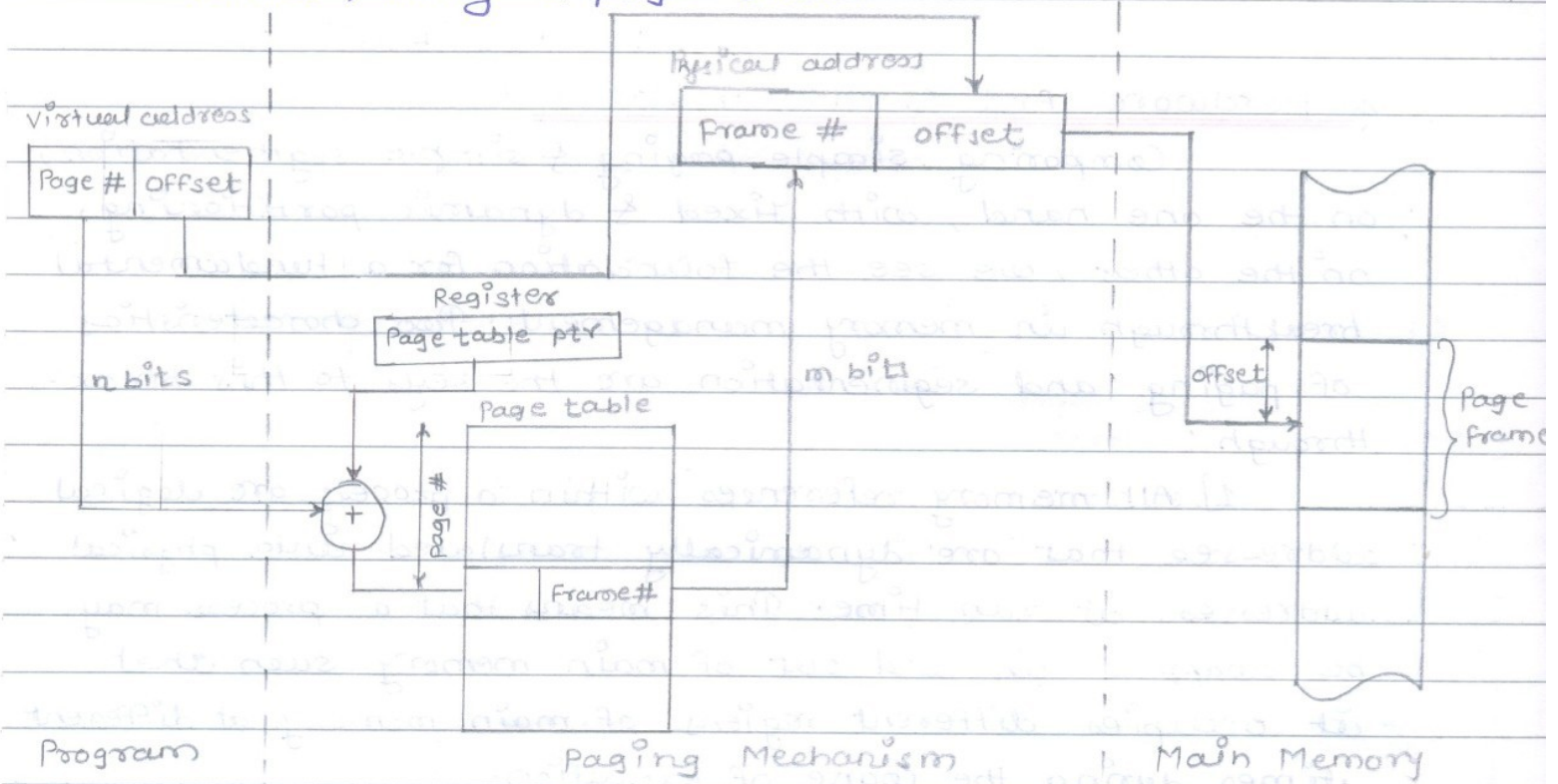


Fig – Address Translation in a Paging System

DATE :

* Translation Lookaside Buffer —

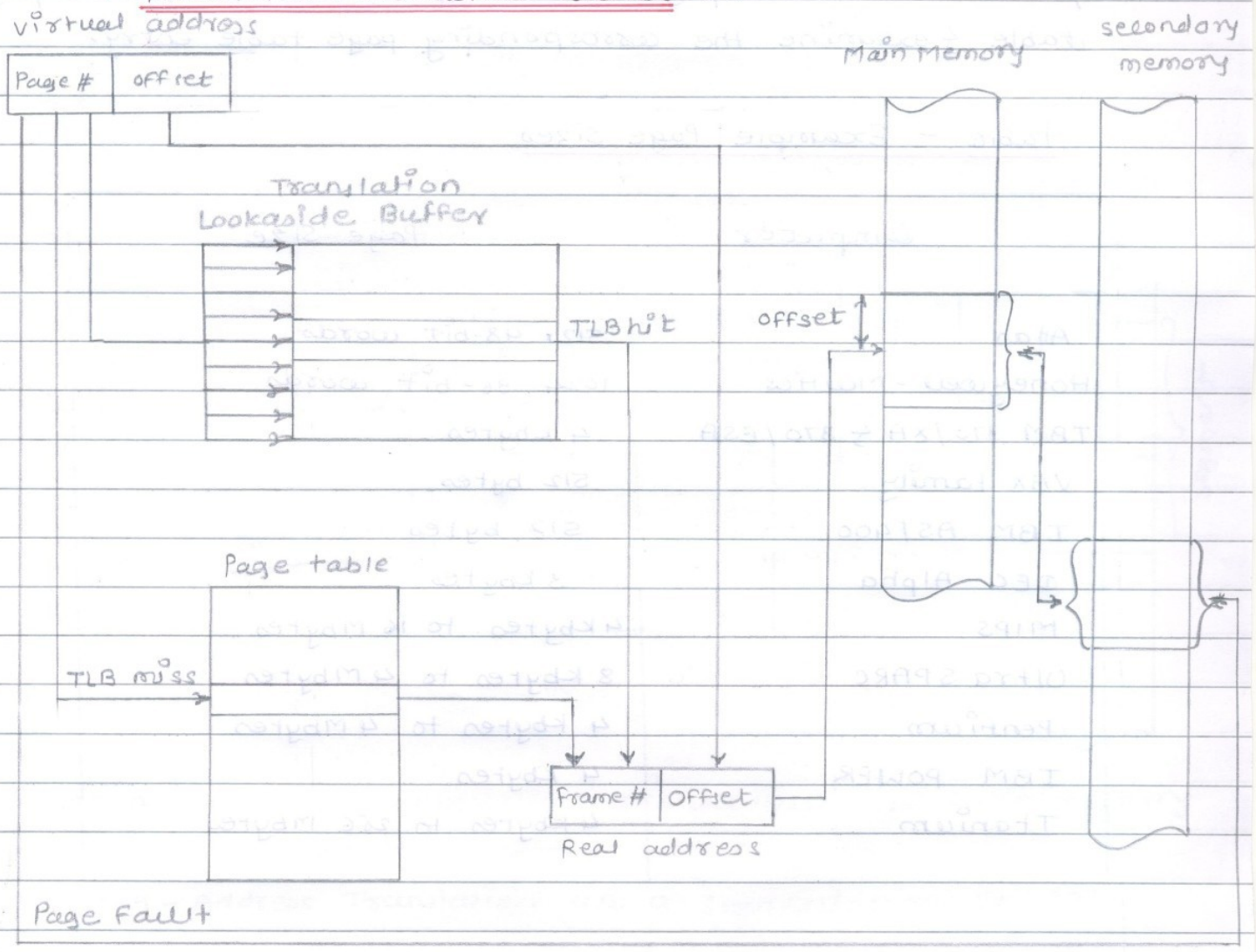


Fig – Use of a Translation Lookaside Buffer

Most virtual memory schemes make use of a special high-speed cache for page table entries, usually called a translation lookaside buffer (TLB). This cache functions in the same way as a memory cache & contains those page table entries that have been most recently used. The organisation of the resulting paging hardware is illustrated in above fig. Given a virtual address, the processor will first examine the TLB. If the desired page table entry is present (TLB hit), then the frame number is retrieved & real address is formed. If the desired page table entry is not found (TLB miss), then the

processor uses the page number to index the process page table & examine the corresponding page table entry.

Table - Example Page sizes

Computer	Page Size
Atlas	512 x 48-bit words
Honeywell - Multics	1024 36-bit words
IBM 370/XA & 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes to 16 Mbytes
Ultra SPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes to 4 Mbytes
IBM POWER	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

* Segmentation -

Virtual Memory Implications -

Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments. Segments may be of unequal, indeed dynamic, size. Memory references consists of a (segment number, offset) form of address.

* Address Translation in a segmentation system -

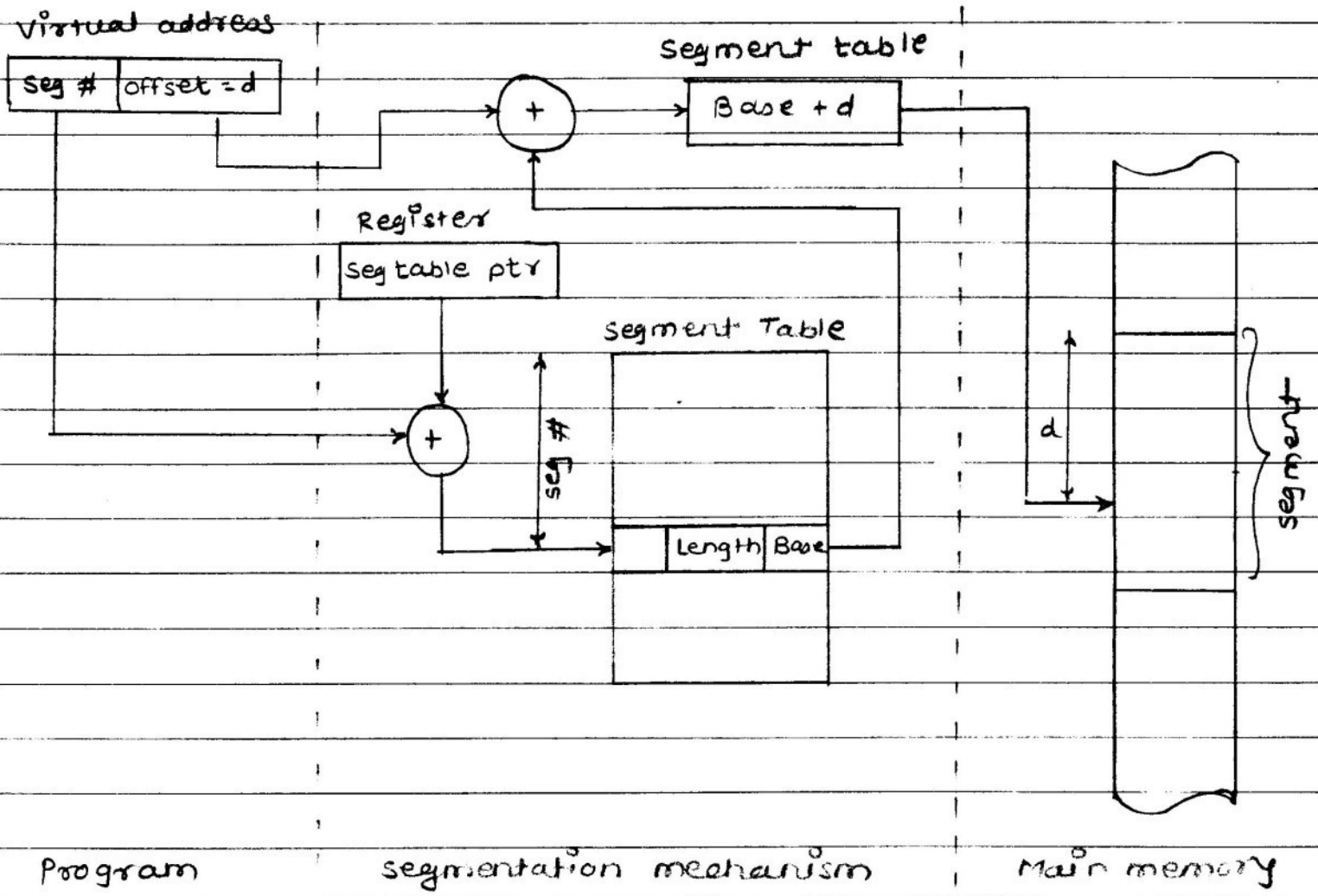


Fig - Address Translation in a segmentation system

* Protection and Sharing *

* Protection Relationships between segments -

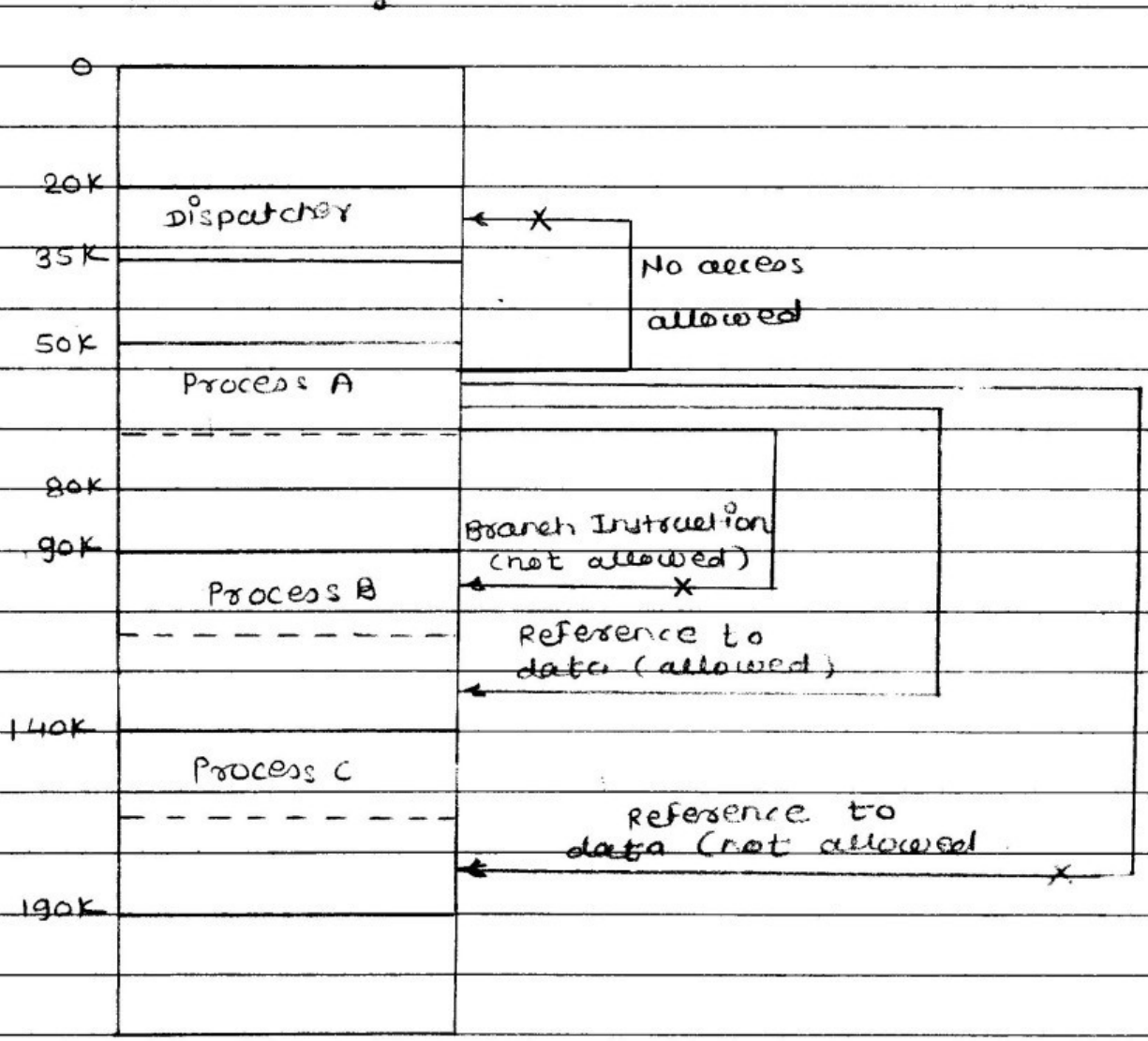


Fig - Protection Relationships between segments

* Operating System Software —

The design of the memory management portion of an operating system depends on three fundamental areas of choice:

- Whether or not to use virtual memory techniques.
- The use of paging or segmentation or both
- The algorithms employed for various aspects of memory management.

The choice made in first two areas depend on the hardware platform available. Thus, earlier UNIX implementations did not provide virtual memory because the processors on which the system ran did not support paging or segmentation. Neither of these techniques is practical without hardware support for address translation and other basic functions.

* Operating System Policies for Virtual Memory —

Fetch Policy	Resident set Management
Demand	Resident set size
Prepaging	Fixed
Placement Policy	Variable
Replacement Policy	Replacement scope
Basic Algorithms	Global
Optimal	Local
Least recently used (LRU)	Cleaning Policy
First-in-First-out (FIFO)	Demand
Clock	Precleaning
Page buffering	
	Load control
	Degree of multiprogramming

* Replacement Policy –

Basic Algorithms – Regardless of the resident set management strategy, there are certain basic algorithms that are used for the selection of a page to replace. Replacement algorithms that have been discussed in the literature include –

- Optimal
- Least recently used (LRU)
- First-in-first-out (FIFO)
- Clock.

* Behaviour of Four Page Replacement Algorithms –

Page address stream		2	3	2	1	5	2	4	5	3	2	5	2
OPT		2	3	3	3	3	3	3	3	3	3	3	3
					1	5	5	5	5	5	5	5	5
						F		F			F		
LRU		2	3	3	3	5	5	5	5	5	5	5	5
					1	1	1	4	4	4	2	2	2
						F		F		F	F		
FIFO		2	3	3	3	3	2	2	2	2	2	5	5
					1	1	1	4	4	4	4	4	2
						F	F	F		F		F	F
CLOCK →		2*	3*	3*	3*	3	2*	2*	2*	2	2*	2	2*
					1*	1	1	4*	4*	4	4	5*	5*
						F	F	F		F		F	

F = Page fault occurring after the frame allocation is initially filled.

Fig – Behaviour of four page replacement algorithms

* Page Buffering –

Although LRU and the clock policies are superior to FIFO, they both involve complexity & overhead not suffered with FIFO. In addition, there is the related issue that the cost of replacing a page that has been modified is greater than for one that has not, because the former must be written back out to secondary memory.

* Resident set Management –

	Local Replacement	Global Replacement
Fixed Allocation	– Number of frames allocated to process is fixed. – Page to be replaced is chosen from among the frame allocated to that process.	– Not possible
Variable Allocation	– The number of frams allocated to a process may be changed from time to time, to maintain the working set of a process. – Page to be replaced is chosen from among frames allocated to that process.	– Page to be replaced is chosen from all available frames in main memory; this causes the size of resident set of processes to vary.

* LINUX MEMORY MANAGEMENT *

Linux Virtual Memory –

Virtual Memory Addressing – Linux makes use of a three level page table structure, consisting of the following types of table (each individual table is the size of one page):

* Page directory –

An active process has a single page directory that is the size of one page. Each entry in the page directory points to one page of the page middle directory. The page directory must be in main memory for an active process.

* Page middle directory –

The page middle directory may span multiple pages. Each entry in the page middle directory points to one page in the page table.

* Page table –

The page table may also span multiple pages. Each page table entry refers to one virtual page of the process.

* Address Translation in Linux Virtual Memory Scheme.

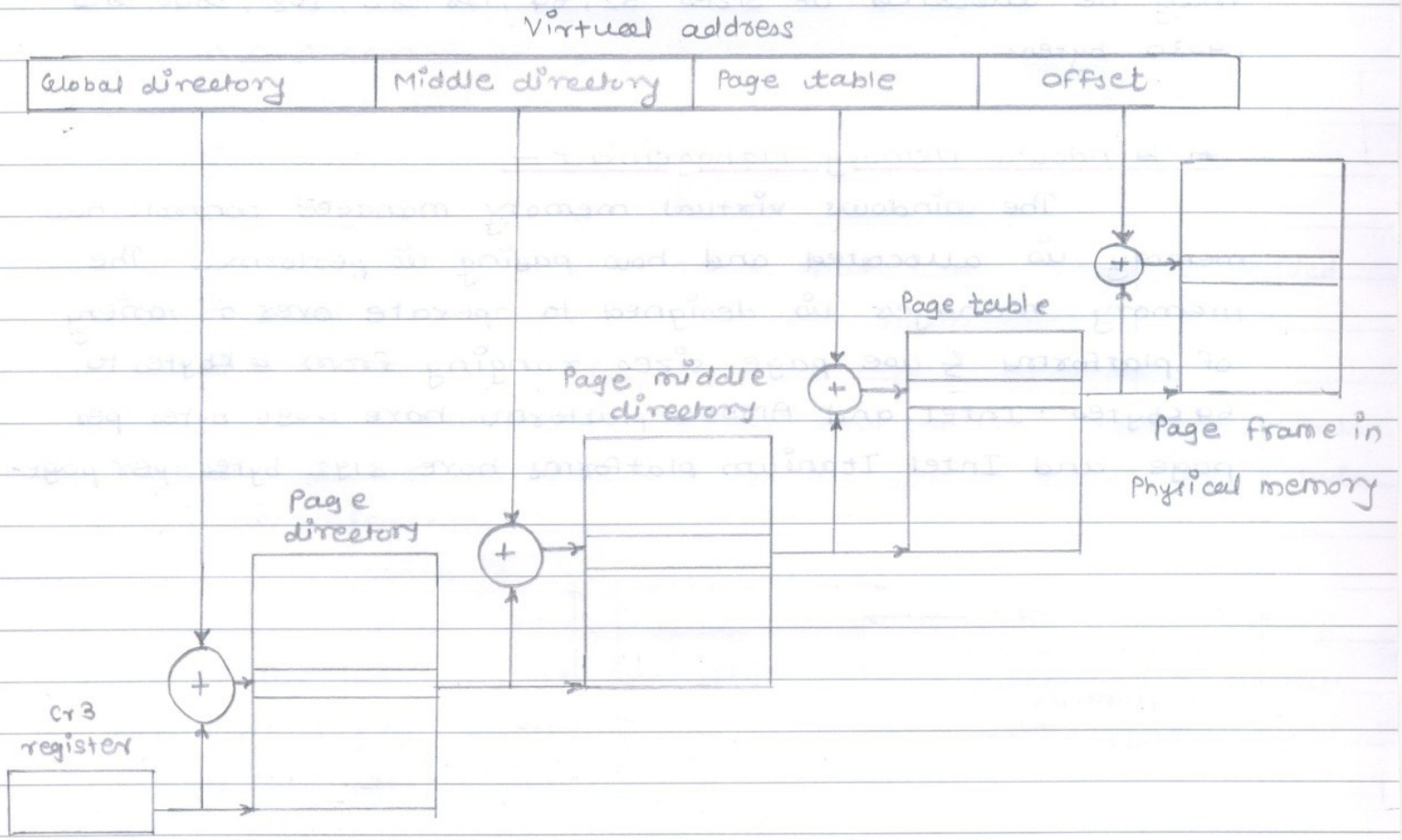


Fig - Address Translation in Linux Virtual Memory Scheme

* Kernel Memory Allocation -

The foundation of kernel memory ~~capability manage~~ allocation for Linux is the page allocation mechanism used for user virtual memory management. As in the virtual memory scheme, a buddy algorithm is used so that memory for the kernel can be allocated & deallocated in units of one or more pages. Because the minimum amount of memory that can be allocated in this fashion is one page, the page allocator alone would be inefficient because the kernel requires small short-term memory chunks in odd sizes. To accommodate these small chunks, Linux uses a scheme known as slab allocation [BONW94] within an allocated page. On a Pentium/x86 machine,

the page size is 4 Kbytes, and chunks within a page may be allocated of sizes 32, 64, 128, 252, 508, 2040 and 4080 bytes.

* Windows Memory Management –

The windows virtual memory manager controls how memory is allocated and how paging is performed. The memory manager is designed to operate over a variety of platforms & use page sizes ranging from 4 Kbytes to 64 Kbytes. Intel and AM064 platforms have 4096 bytes per page and Intel Itanium platforms have 8192 bytes per page.

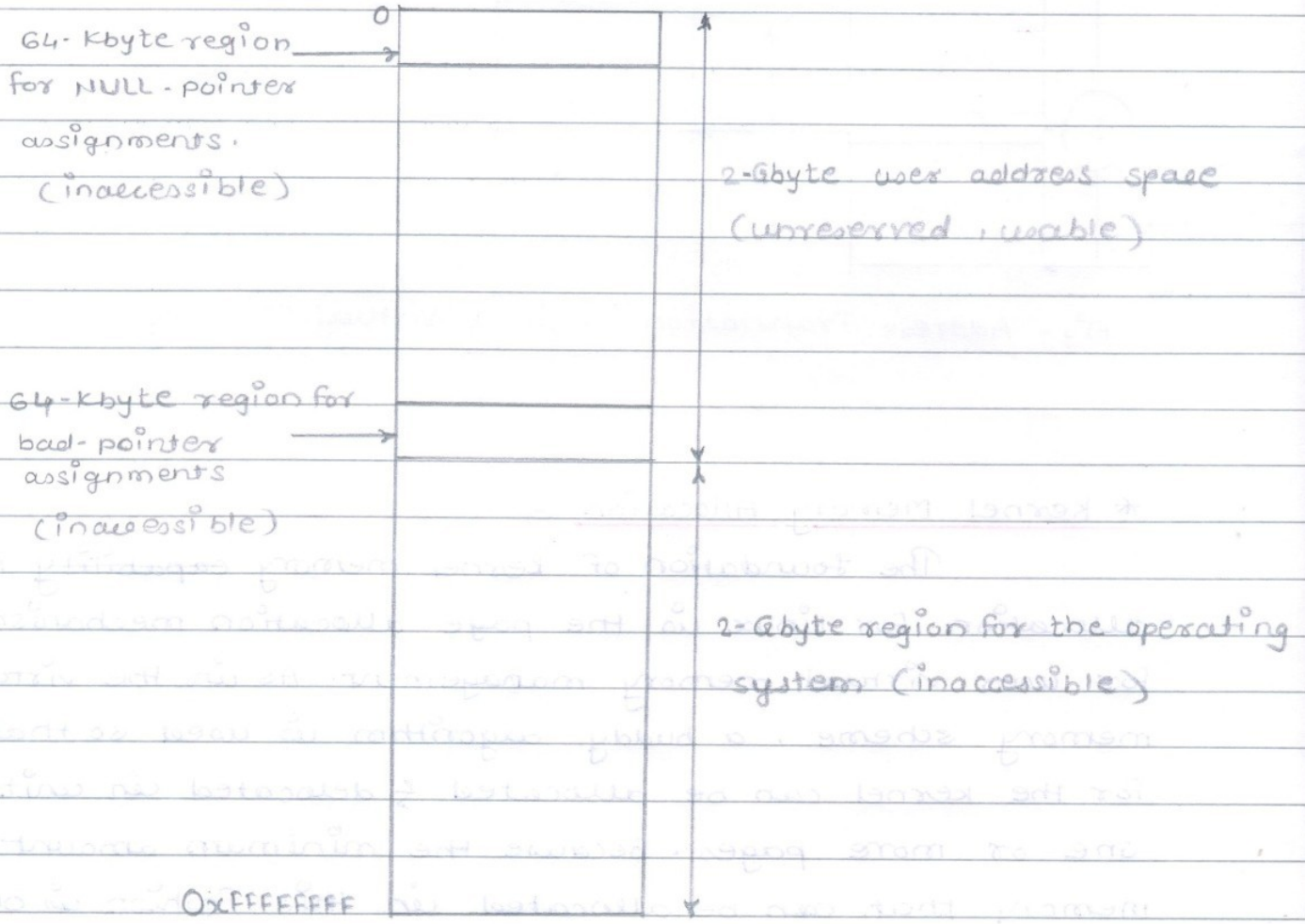


Fig - Windows Default 32-Bit Virtual Address Space

Fig. shows the default virtual address space seen by a normal 32-bit user process. It consists of four regions:

1) 0x00000000 to 0x0000FFFF —

Set aside to help programmers catch NULL-pointer assignments.

2) 0x00010000 to 0x7FFEFFFF —

Available user address space. This space is divided into pages that may be loaded into main memory.

3) 0x7FF0000 to 0x7FFFFFFF —

A guard page inaccessible to the user. This page makes it easier for the operating system to check on out-of bounds pointer references.

4) 0x80000000 to 0xFFFFFFFF —

System address space. This 2-Gbyte process is used for the Windows Executive, kernel, and device drivers.

On 64-bit platforms, 8TB of user address space is available in Windows Vista.

END