

Chapter 5

Memory Management

--By R.G.B

Memory ???

- Devices used to store information for use in a system
- Information stored in form of bits

Types of memory

1. Primary

- Holds data and programs used by a process that is executing
- Ex: RAM

2. Secondary

- Non-volatile memory used to store data when a process is not executing
- Ex: Hard Disk

Memory Organization and Hierarchy

Processor memory

Main memory

Virtual memory

Secondary memory

Memory Management:

- Allocation and Free of memory accordingly
- The management of main memory is critical to the computer system.
- In Uni-programming system, Main memory is divided into two parts:
 1. one part for the OS
 2. one part for the program currently being executed
- In multiprogramming system, the user part of the memory must be further subdivided to accommodate multiple processes
- The task of subdivision is carried out dynamically by the Operating System and is known as Memory Management

Schemes For Memory Management

- **Contiguous Memory Allocation**
- **Non-Contiguous Memory Allocation**

Contiguous Memory Allocation

- Each logical object is placed in a set of memory locations with strictly consecutive addresses
- Contiguous memory allocation is one of the oldest memory allocation schemes.
- The size of the process is compared with the amount of contiguous main memory available to execute the process.
- If sufficient contiguous memory is found, the process is allocated memory to start its execution
- Otherwise, it is added to a queue of waiting processes until sufficient free contiguous memory is available

Non-Contiguous Memory Allocation

- It implies that a single logical object may be placed in non-consecutive sets of memory locations
- A program is divided into blocks or **segments** that the system may place in nonadjacent slots in main memory.
- This allows making use of holes (unused gaps) in memory that would be too small to hold whole programs.
- More overhead to operating system

Partitioning

- Fixed partition
- Variable partition

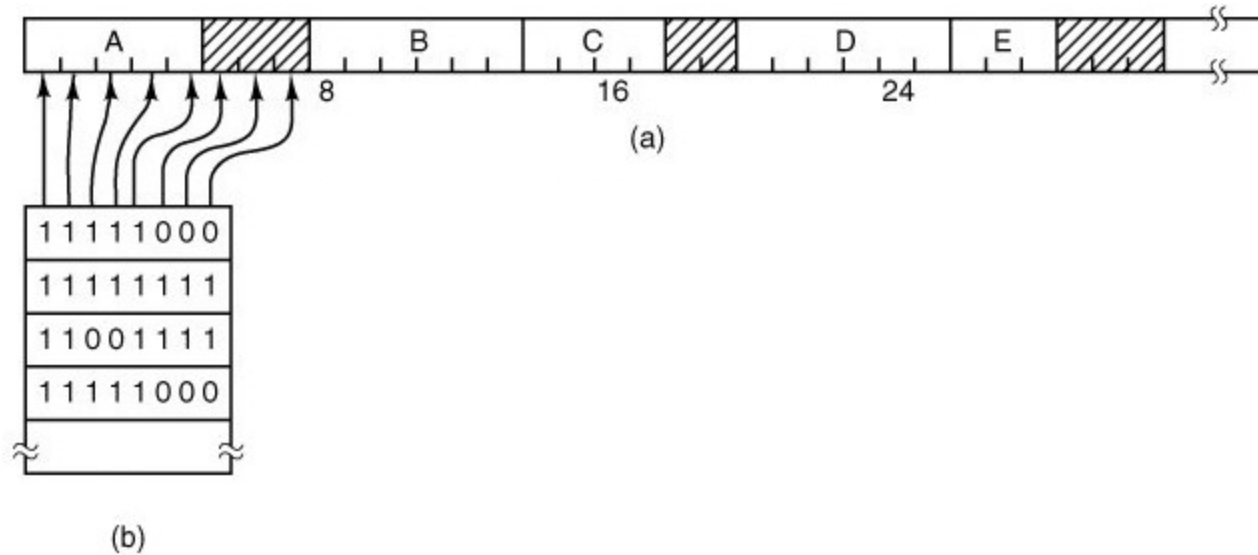
Managing Variable partition

- Management of Dynamic memory
- Two ways to keep track of memory usage
 1. Bitmaps
 2. Linked list

Bitmap

- Memory is divided up into allocation units, perhaps as small as a few words and perhaps as large as several kilobytes.
- Corresponding to each allocation unit is a bit in the bitmap, which is 0 if the unit is free and 1 if it is occupied (or vice versa)
- Design issues
 1. The smaller the allocation unit, the larger the bitmap.
 2. Allocation unit is chosen large, the bitmap will be smaller, but appreciable memory may be wasted

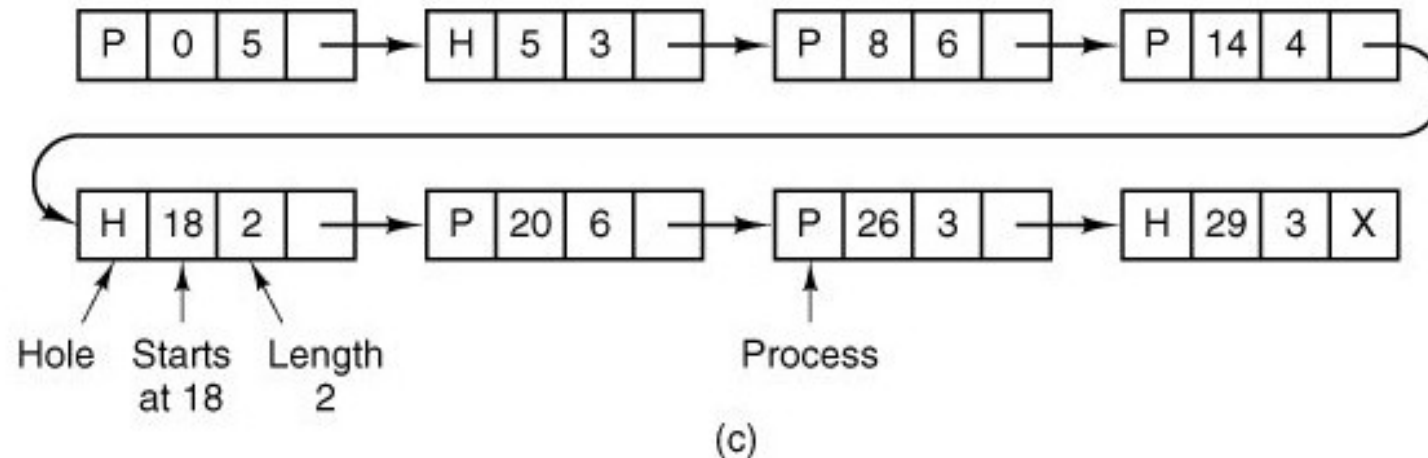
Bitmaps



- (a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions (0 in the bitmap) are free.
- (b) The corresponding bitmap

Linked list

- Maintain a linked list of allocated and free memory segments, where a segment is either a process or a hole between two processes
- Segment list is kept sorted by address
- Sorting this way has the advantage that when a process terminates or is swapped out, updating the list is straightforward



Placement Schemes

- First Fit
- Next Fit
- Best Fit
- Worst Fit
- Quick Fit

First Fit

- Process manager scans along the list of segments until it finds a hole that is big enough.
- The hole is then broken up into two pieces, one for the process and one for the unused memory, except in the statistically unlikely case of an exact fit.
- First fit is a fast

Next Fit

- It works the same way as first fit, except that it keeps track of where it is whenever it finds a suitable hole
- The next time it is called to find a hole
- It starts searching the list from the place where it left off last time, instead of always at the beginning, as first fit does

Best Fit

- Best fit searches the entire list and takes the smallest hole that is adequate
- Rather than breaking up a big hole that might be needed later, best fit tries to find a hole that is close to the actual size needed
- Efficient utilization of memory
- Overhead to PM

Worst Fit

- Always take the largest available hole, so that the hole broken off will be big enough to be useful.
- Simulation has shown that worst fit is not a very good idea either
- Memory wastage (inefficient)

Quick Fit

- Maintains separate lists for some of the more common sizes requested.
- **For example**, it might have a table with n entries, in which the first entry is a pointer to the head of a list of 4-KB holes, the second entry is a pointer to a list of 8-KB holes, the third entry a pointer to 12-KB holes, and so on.
- Finding a hole of the required size is extremely fast
- Sort by hole size, namely, when a process terminates or is swapped out, finding its neighbours to see if a merge is possible is expensive.

Buddy Memory Allocation

- Memory allocation algorithm that divides memory into partitions to try to satisfy a memory request as suitably as possible.
- System makes use of splitting memory into halves to try to give a best-fit

	0	128k	256k	512k	1024k
start	1024k				
A=70K	A	128	256	512	
B=35K	A	B 64	256	512	
C=80K	A	B 64	C 128	512	
A ends	128	B 64	C 128	512	
D=60K	128	B D	C 128	512	
B ends	128	64 D	C 128	512	
D ends	256		C 128	512	
C ends	512			512	
end	1024k				

Fragmentation

- Free memory space is broken into little pieces.
- It happens after sometimes that processes can not be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Types of Fragmentation

- **External fragmentation: Total** memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous so it can not be used
- **Internal fragmentation: Memory** block assigned to process is bigger. Some portion of memory is left unused as it can not be used by another process
- External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.
- **Compaction → Solution**

Relocation

- When a program is run it does not know in advance what location it will be loaded at Therefore, the program cannot simply generate static addresses (e.g. from jump instructions).
- Instead, they must be made relative to where the program has been loaded.

Protection

- Once you can have two programs in memory at the same time there is a danger that one program can write to the address space of another program
- Dangerous and should be avoided
- **Memory protection** is a way to control memory access rights on a computer, and is a part of most modern operating systems
- The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it

Coalescing & Compacting

- **Coalescing** is act of merging adjacent free blocks of memory
- Memory **compaction** is the process of moving allocated objects together, and leaving empty space together

Logical Vs Physical Memory(Address)

- An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address.
- Logical address is also known as a Virtual address.
- Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.
- The set of all logical addresses generated by a program is referred to as a logical address space. The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.

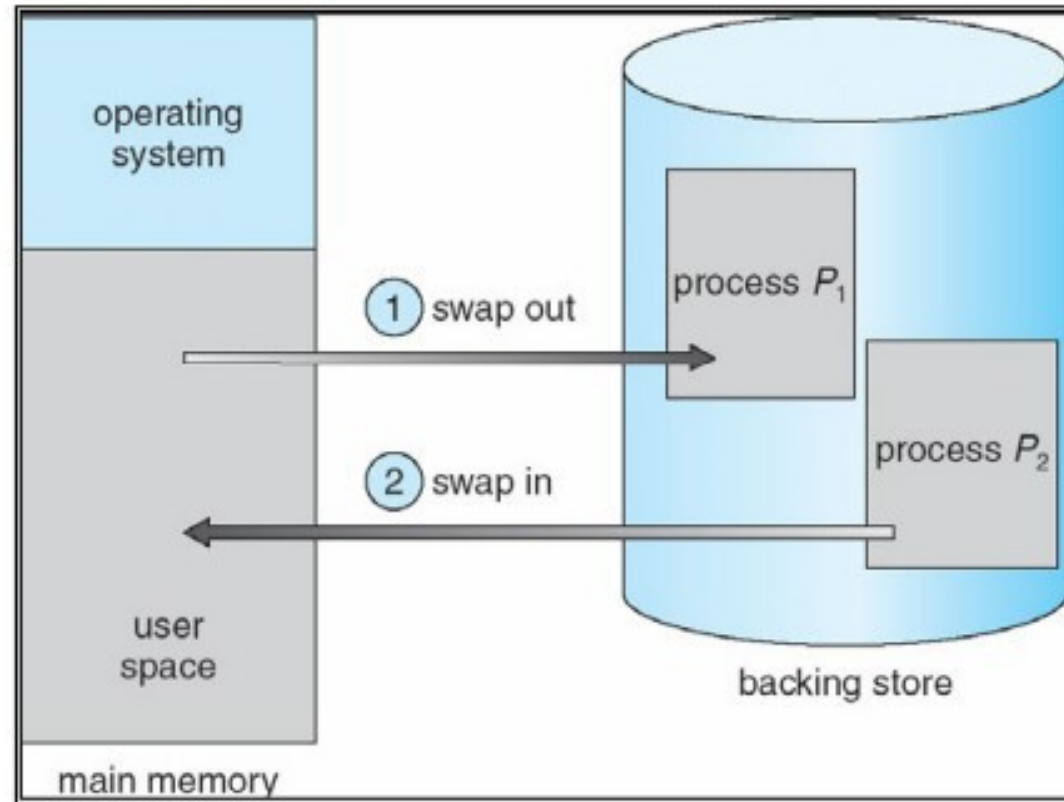
Logical Vs Physical Memory(Address)

- The run-time mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device.
- MMU uses following mechanism to convert virtual address to physical address.
- The value in the base register is added to every address generated by a user process which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

Swapping

- A process must be in memory to be executed.
- In case of memory is less
- A process, however can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.
- Algorithms used to swapping(FIFO, Priority, SJF, RR,)

Swapping

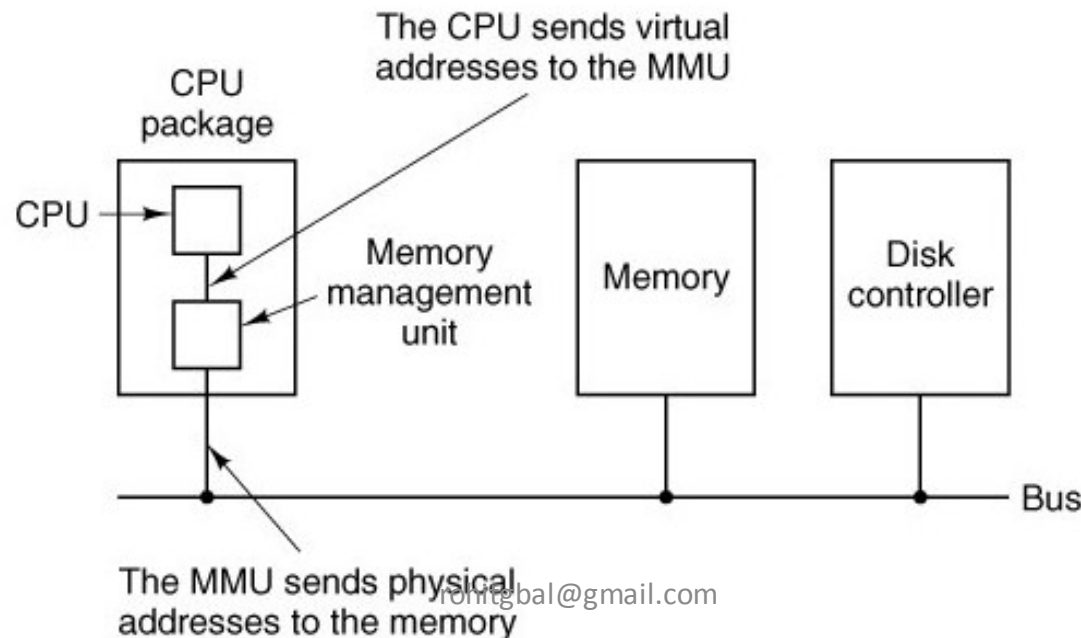


Virtual Memory

- The basic idea behind virtual memory is that the combined size of the program, data, and stack **may exceed the amount of physical memory** available for it.
- Parts of the program currently in use in main memory, and the rest on the disk(HDD)
- **For example**, A 512-MB program can run on a 256-MB machine by carefully choosing which 256 MB to keep in memory at each instant, with pieces of the program being swapped between disk and memory as needed
- Virtual memory systems separate the memory addresses used by a process from actual physical addresses

Paging

- Virtual system uses a techniques called paging that permits the physical address space of a process to be non-contiguous
- Program-generated addresses are called **virtual addresses** and form the **virtual address space**



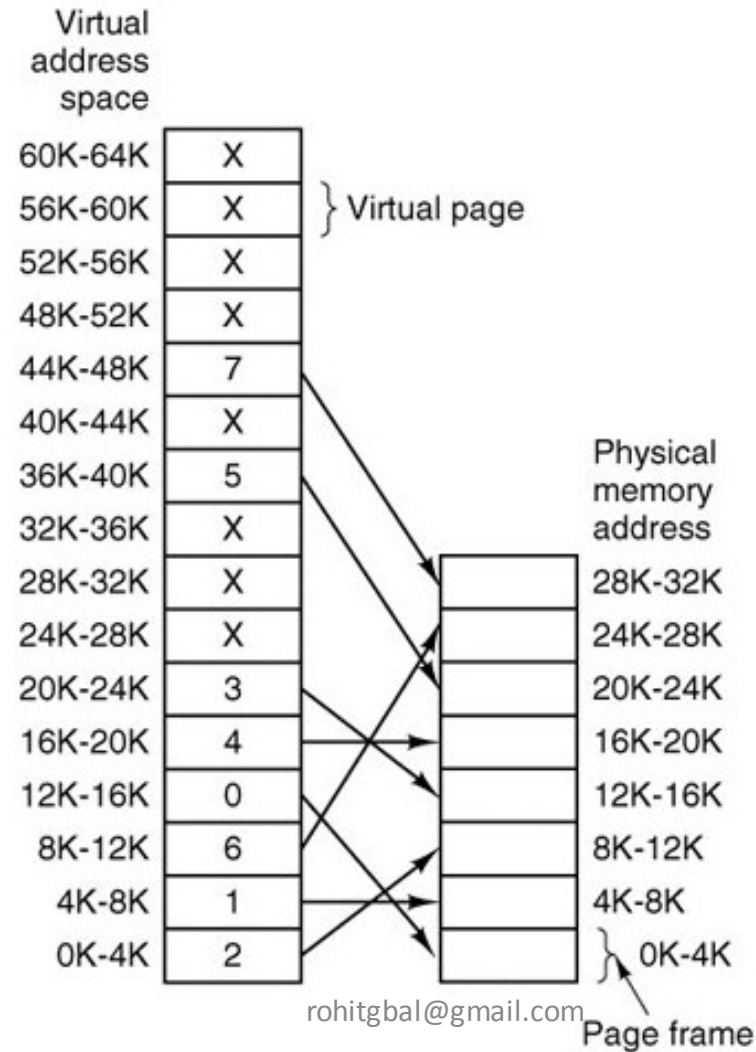
Paging

- Implementing paging involves breaking physical memory into fixed size block called **frames**
- Breaking logical memory into blocks of the same size called **pages**. size is power of 2, between 512 bytes and 8,192 bytes
- When a process is to be executed, its pages are loaded into any available memory frames from the backing store.
- The backing store is divided into fixed-size block that are of the same size as memory frames.
- ***Backing store is slower and cheaper than main memory***

Page Table

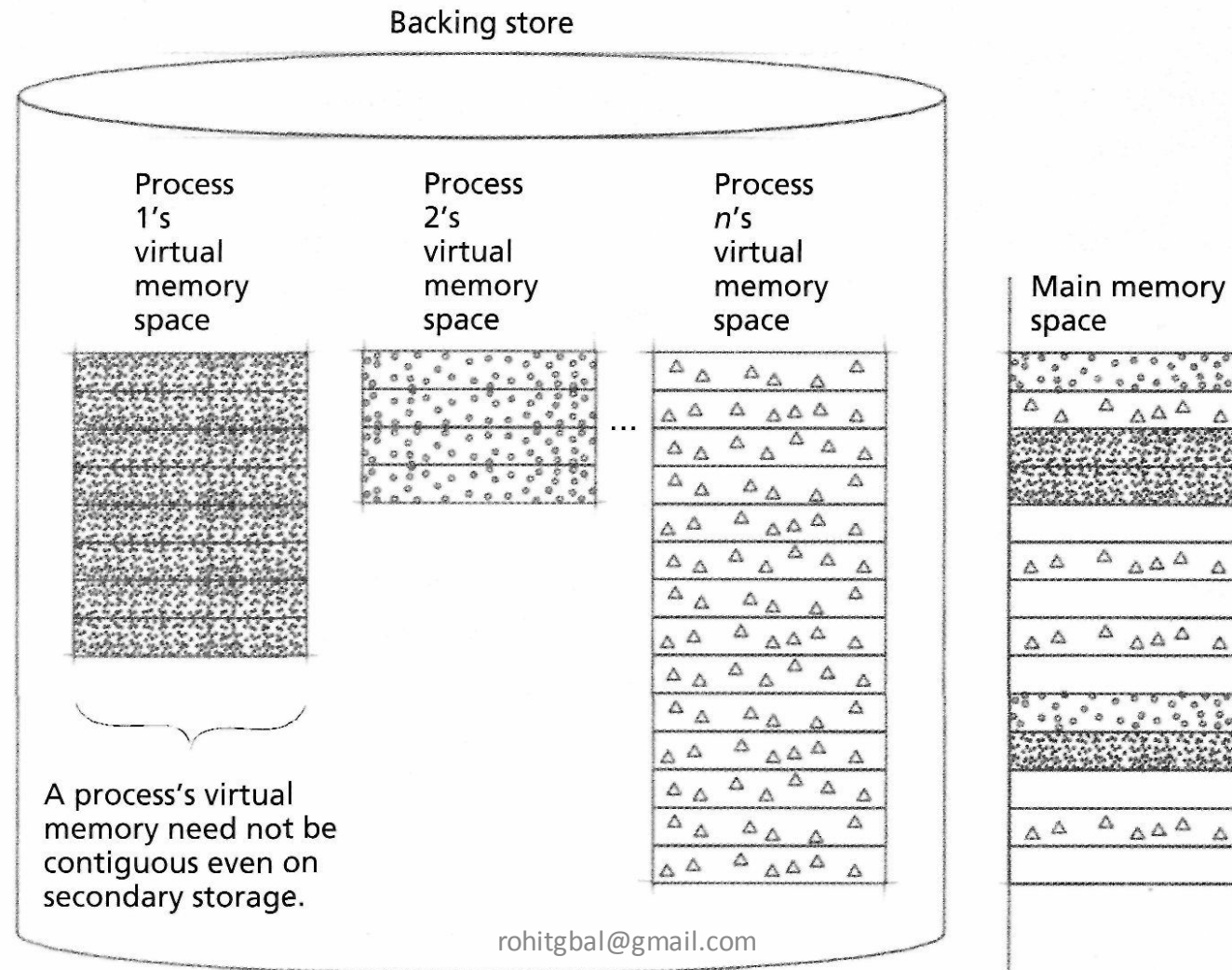
- Page table store relation between memory and backing store
- The virtual address space is divided up into units called **pages**.
- The corresponding units in the physical memory are called **page frames**
- The pages and page frames are always the same size
- A **present/absent bit** keeps track of which pages are physically present in memory

Page Table



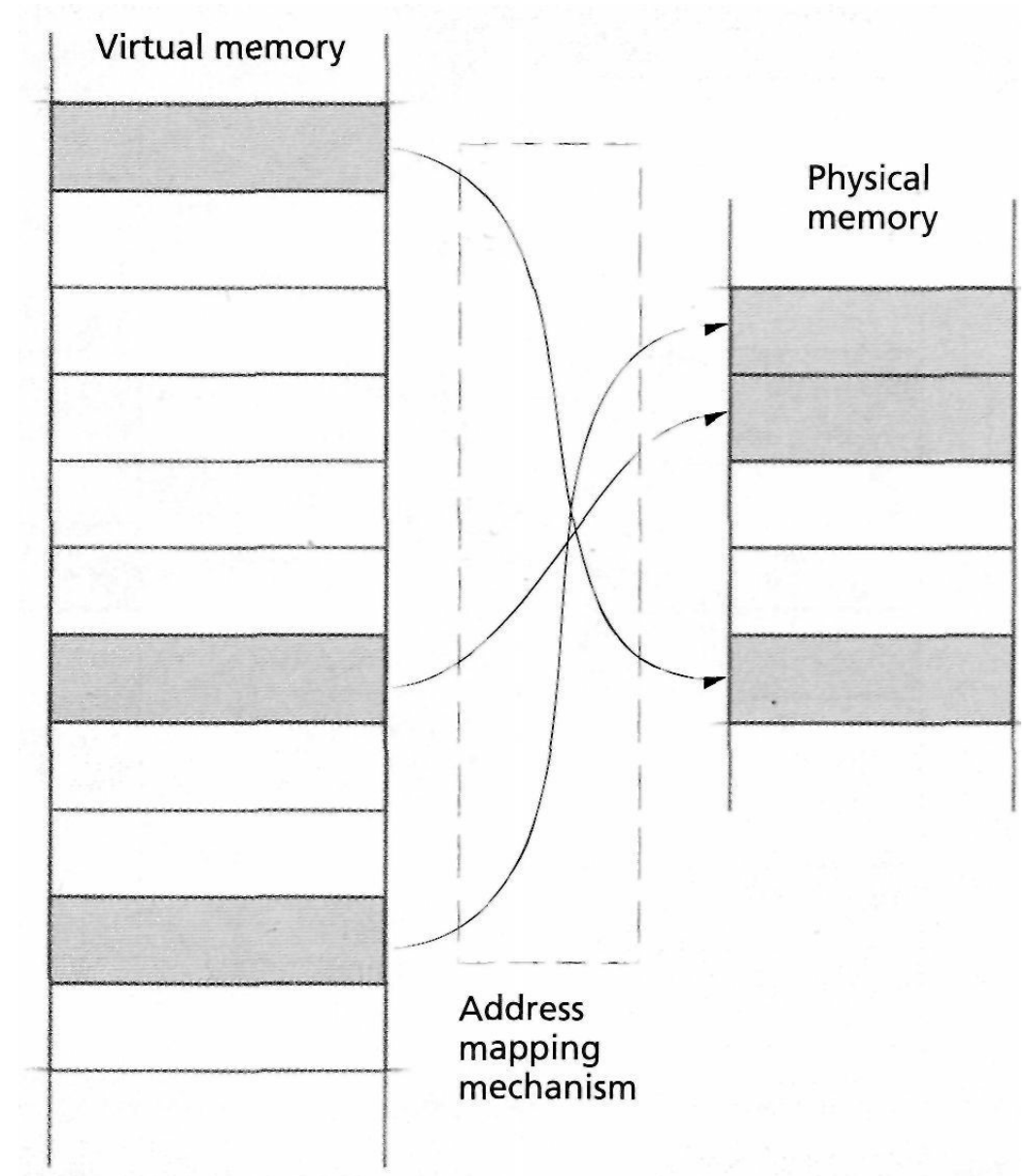
Mapping

- Read Address To Virtual Address
- Process references only physical address- executed in **Main Memory**
- **Address Translation**- should be quick
- **Dynamic Address Translation**
- No need for contiguous memory



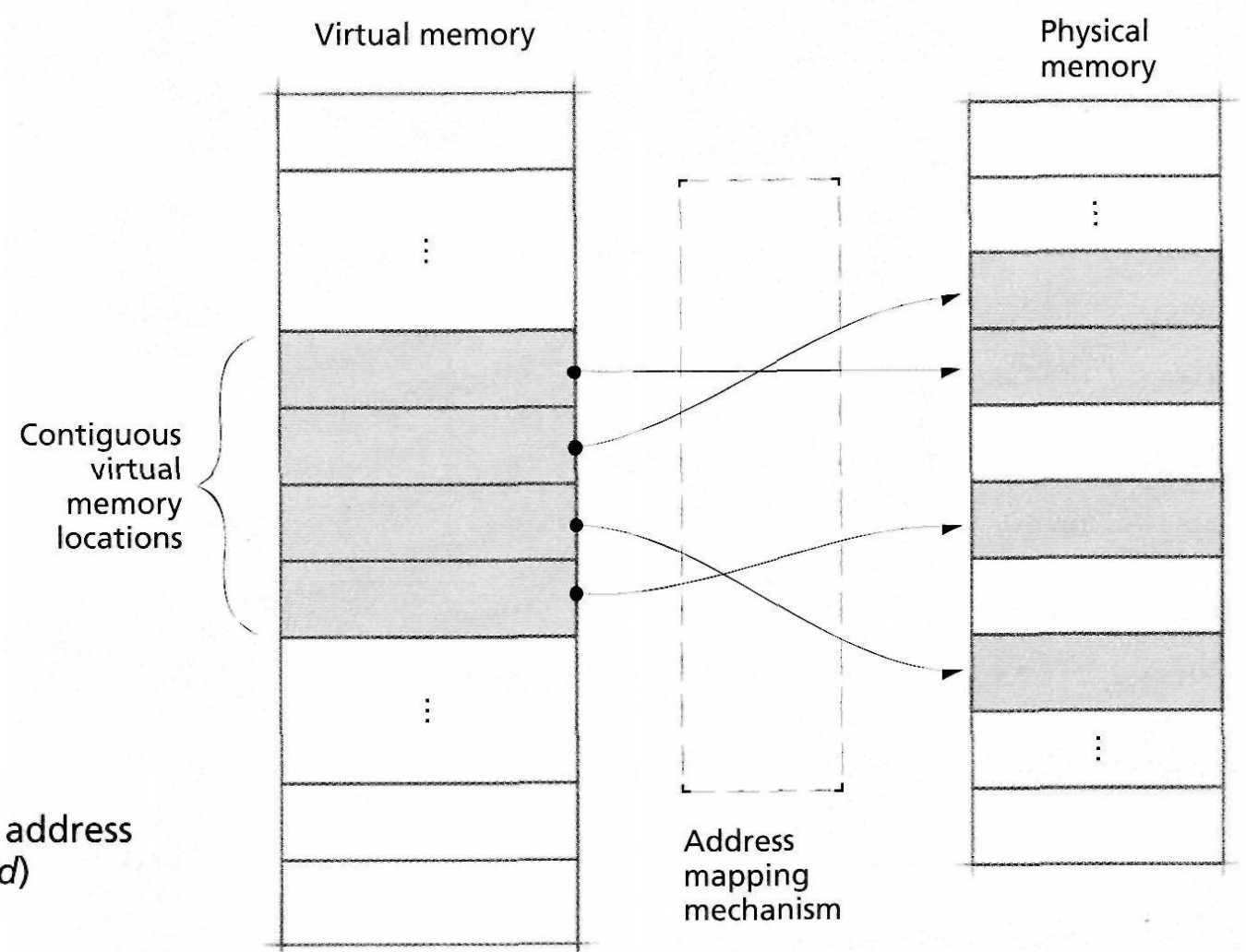
Block Mapping

- Address translation maps – Indicate which regions of a process's virtual address space, V , are currently in main memory and where they are located

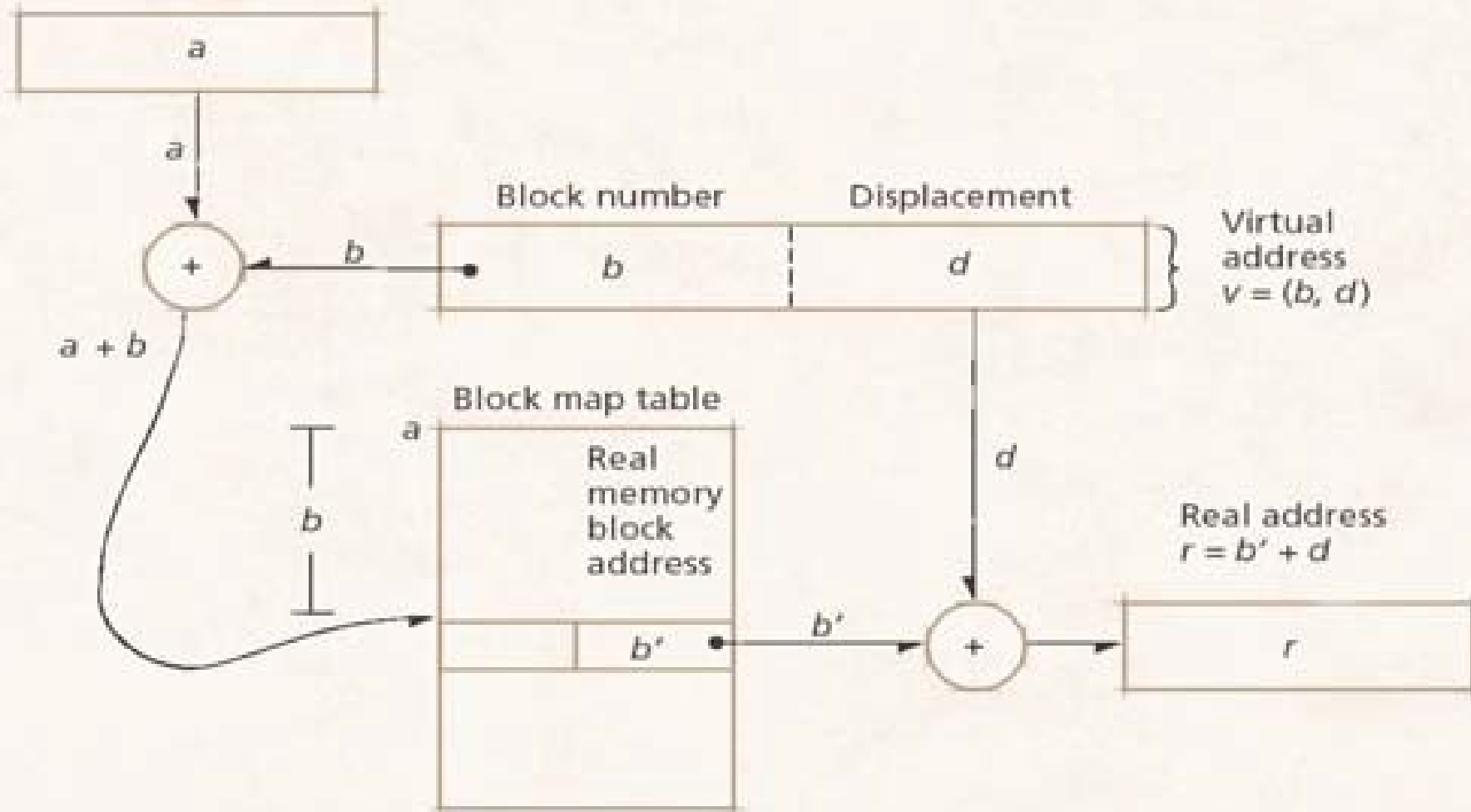


Block Mapping

- **Artificial Contiguity**
- Contiguous virtual addresses may not correspond to contiguous real memory addresses
- System represents addresses ordered pairs



Block map origin
register containing
base address of
process's block
map table



Pages Vs Segments

- **Pages**

- Blocks are fixed size
- Technique is called paging

- **Segments**

- Blocks maybe of different size
- Technique is called segmentation

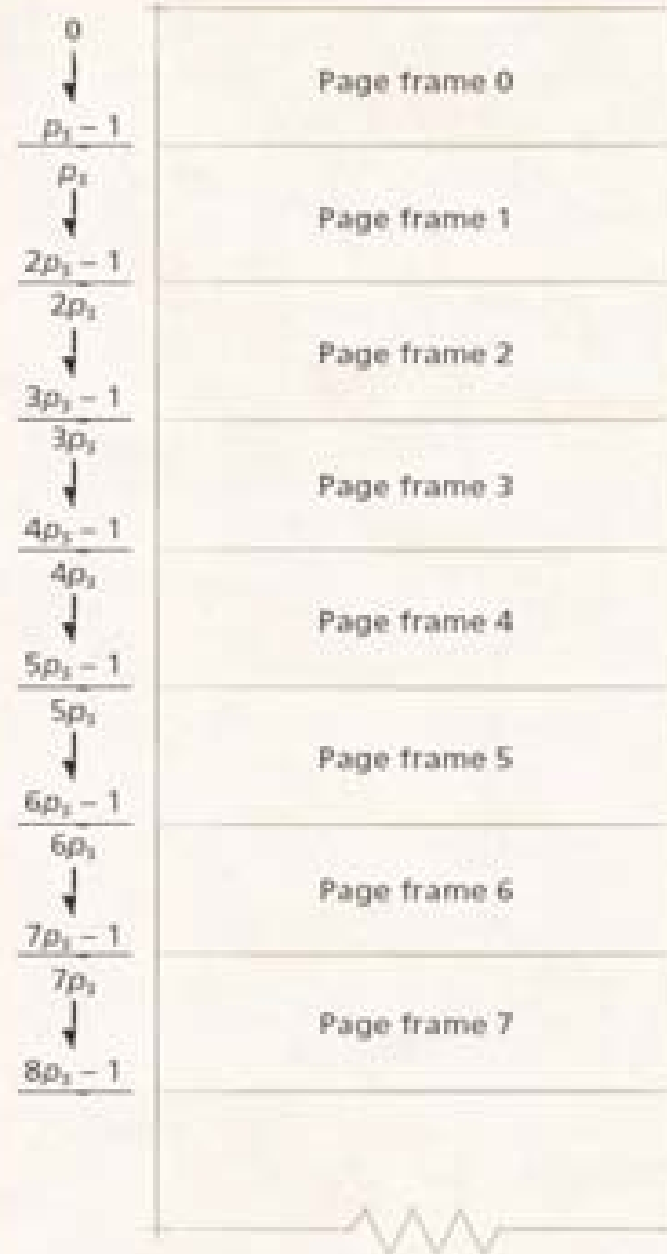
Paging

- Paging uses fixed-size block mapping
- Virtual address in paging system is an ordered pair $v = (p, d)$
- p is the number of the page in virtual memory on which the referenced item resides
- d is the displacement from the start of page p at which the referenced item is located



Page Frame

- Page frame
- Fixed-size block of main memory
- Begins at a main memory address that is an integral multiple of
- fixed page size (ps)



Page frame number	Page frame size	Range of physical memory addresses
0	p_1	$0 \rightarrow p_1 - 1$
1	p_1	$p_1 \rightarrow 2p_1 - 1$
2	p_1	$2p_1 \rightarrow 3p_1 - 1$
3	p_1	$3p_1 \rightarrow 4p_1 - 1$
4	p_1	$4p_1 \rightarrow 5p_1 - 1$
5	p_1	$5p_1 \rightarrow 6p_1 - 1$
6	p_1	$6p_1 \rightarrow 7p_1 - 1$
7	p_1	$7p_1 \rightarrow 8p_1 - 1$
...		

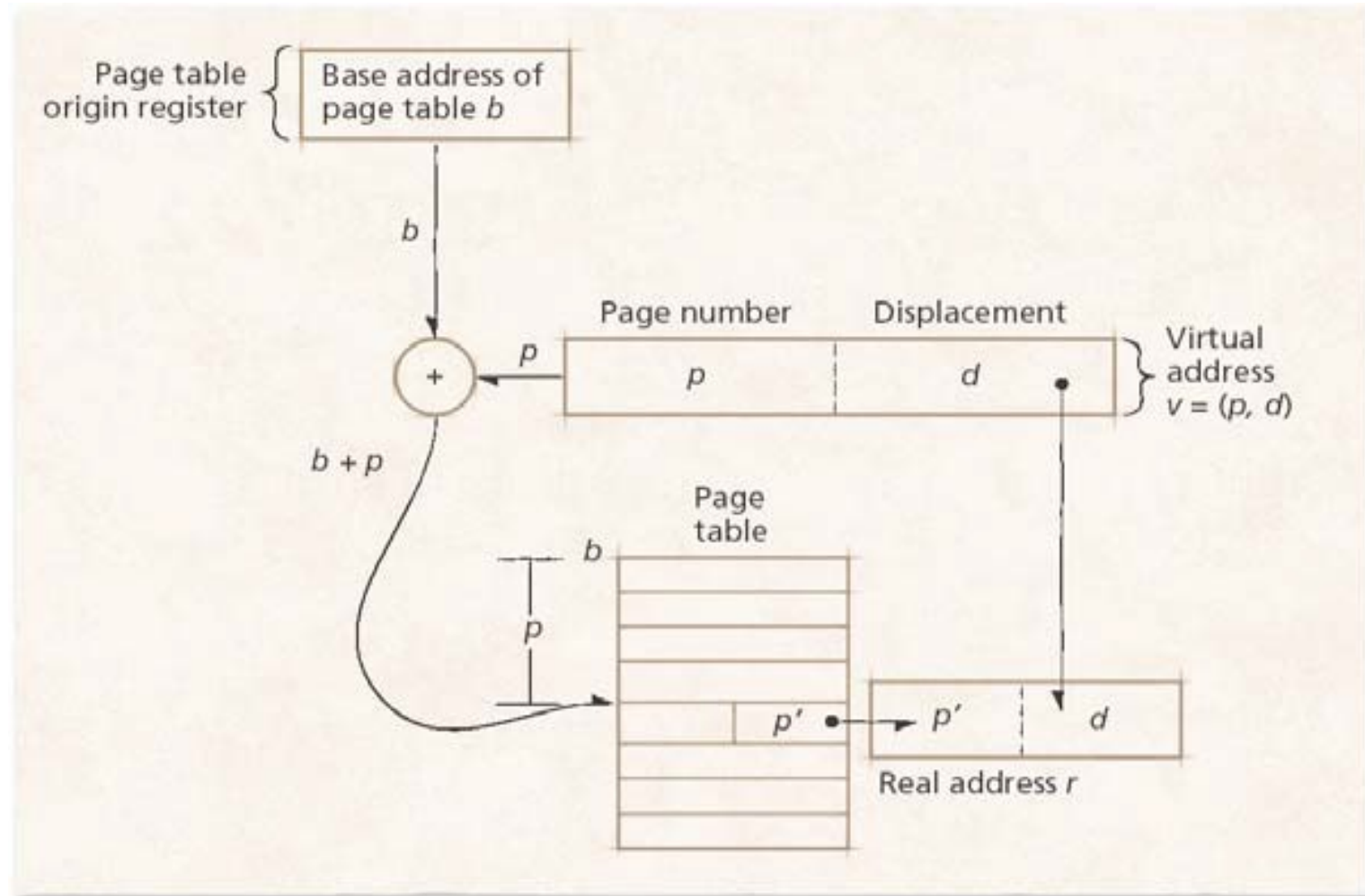
Page Table Entry (PTE)

- Indicates that virtual page p corresponds to page frame p'
- Contains a resident bit to indicate if page is in memory
- If so, PTE stores the page's frame number
- Otherwise, PTE stores the location of the page on secondary
- storage

Paging Address Translation by Direct Mapping

- Dynamic address translation under paging is similar to block address translation
- Process references virtual address $v = (p, d)$
- DAT adds the process's page table base address, b , to referenced page number, p
- $b + p$ forms the main memory address of the PTE for page p
- System concatenates p' with displacement, d , to form real address, r

Paging Address Translation by Direct Mapping



Paging Address Translation by Associative Mapping

- **Maintaining entire page table in cache memory is often not viable**
 - Due to cost of high-speed, location-addressed cache memory and relatively large size of programs
- **Increase performance of dynamic address translation**
 - Place entire page table into content-addressed associative memory
 - Every entry in associative memory is searched simultaneously
 - Content-addressed cache memory is also prohibitively expensive

Paging Address Translation with Direct/Associative Mapping

- **Compromise between cost and performance**
- Most PTEs are stored in direct-mapped tables in main memory
- Most-recently-used PTEs are stored in high-speed set-associative cache memory called a Translation Lookaside Buffer (TLB)
- If PTE is not found in TLB, the DAT mechanism searches the table
- in main memory
- Can yield high performance with relatively small TLB due to locality

Translation Lookaside Buffer (TLB)

- It is a Memory Cache(Hardware)
- Stores recent translation from Physical to virtual address
- Inside the MMU and consists of a small number of entries
- Implemented as Content Addressable Memory(CAM)
- Search key is V Address
- Result is P Address
- If Present TLB hit
- Not Present TLB miss
- Check the page table – **page walk**(Time consuming)

A TLB to speed up paging.

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

TLB Software

- Many modern RISC machines, including the SPARC, MIPS, HP PA, and PowerPC, do nearly all of this page management in software
- Instead of MMU– OS manages page table entries and page fault

Inverse Page Table

- Address space consists of 2^{32} bytes, with 4096 bytes per page, then over 1 million page table entries are needed.
- $4294967296/4096=1048576$
- At least consume 4MB for page table entry
- To reduce memory space
- Only save physical memory address

Page Fault

- Trap(exception) to the software raised by the hardware when a program accesses a page that is mapped in the virtual address space, but not loaded in physical memory
- The typical case the operating system tries to handle the page fault by making the required page accessible at a location in physical memory or kills the program in the case of an illegal access

Page Replacement Algorithms

1. Optimal page replacement algorithm
2. Not recently used page replacement
3. First-In, First-Out page replacement
4. Second chance page replacement
5. Clock page replacement
6. Least recently used page replacement
7. Working set Page Replacement
8. WS Clock Page

Optimal Page Replacement Algorithm

- The algorithm has lowest page fault rate of all algorithm
- Replace the page which will not be used for longest period of time i.e future knowledge of reference string is required
- Replace the page that will not be referenced for the longest time.
- Impossible to implement

The Not Recently Used Page Replacement Algorithm

- Two status bit associated with each page.
- R is set whenever the page is referenced (read or written)
- M is set when the page is written to (i.e., modified).
- When a page fault occurs, the operating system inspects all the pages and divides them into four
- categories based on the current values of their R and M bits:
- Class 0: not referenced, not modified.
- Class 1: not referenced, modified.
- Class 2: referenced, not modified.
- Class 3: referenced, modified.
- The NRU (Not Recently Used) algorithm removes a page at random from the lowest numbered nonempty class.

First In First Out

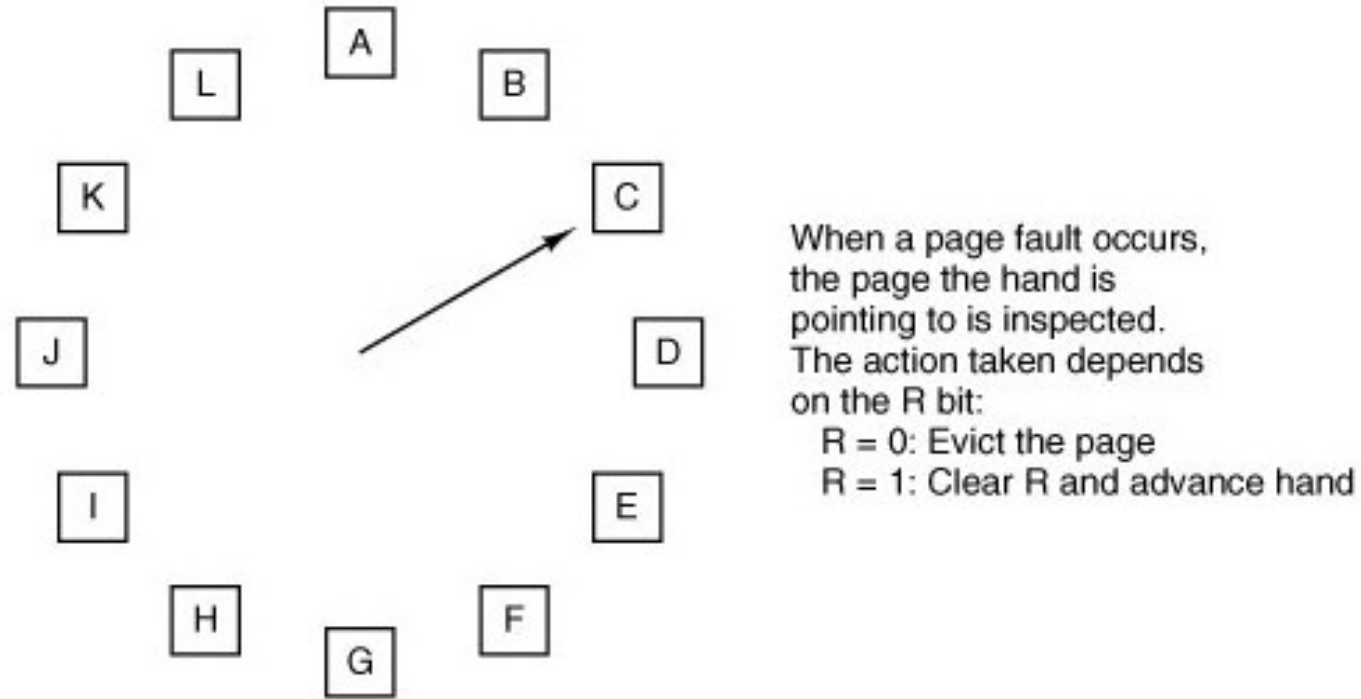
- Oldest page in the physical memory is the one selected for replacement.
- Very simple to implement.
- Keep a list
- On a page fault, the page at the head is removed and the new page added to the tail of the list
- **Issues**
- Poor replacement policy
- FIFO doesn't consider the page usage.

Second Chance Page Replacement Algorithm

- A simple modification to FIFO that avoids the problem of heavily used page.
- It inspects the R bit If it is 0, the page is both old and unused, so it is replaced immediately.
- If the R bit is 1, the bit is cleared, the page is put onto the end of the list of pages, and its load time is updated as though it had just arrived in memory. Then the search continues

Clock Replacement Algorithm

- Keep all the page frames on a circular list in the form of a clock



Clock Replacement Algorithm

- When a page fault occurs, the page being pointed to by the hand is inspected.
- If its R bit is 0, the page is evicted, the new page is inserted into the clock in its place, and the hand is advanced one position
- If R is 1, it is cleared and the hand is advanced to the next page. This process is repeated until a page is found with $R = 0$

Least Recently Used

- Page that has not been used for longest period of time is selected for replacement
- LRU is theoretically realizable, it is not cheap.
- To fully implement LRU, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear.
- The difficulty is that the list must be updated on every memory reference.

Working Set Page Replacement

- Working set pages— set of pages currently using
- Entire working set in memory— no page faults
- Before process is running algorithm make sure page is in primary memory
- Page fault: Remove page which is not in working set
- R – Reference bit
- T – no of clock ticks

Working Set Page Replacement


1. If $R=1$ current virtual time is set to “Time of last used” in page table
2. If $R=0$ and age is calculated(Current VT-Time of last used)
 - a) If $\text{age} > t$ – Page not working and removed from working set
 - b) If $\text{age} \leq t$ Page is working not removed

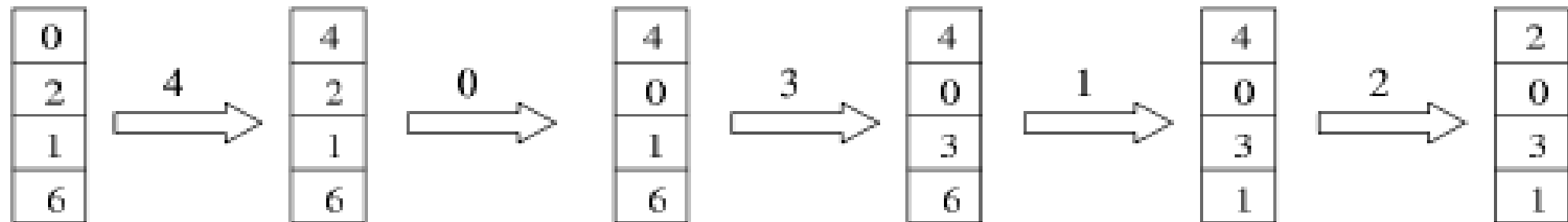
WS Clock Page Replacement

- Similar to Ws
- Arrange pages in circular manner like a clock
 1. If $R=1$ current virtual time is set to “Time of last used” in page table
 2. If $R=0$ and age is calculated(Current VT-Time of last used)
 - a) If $\text{age} > t$ – Page not working and removed from working set
 - b) If $\text{age} \leq t$ Page is working not removed

FIFO

- Consider the following reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Compulsory Misses 

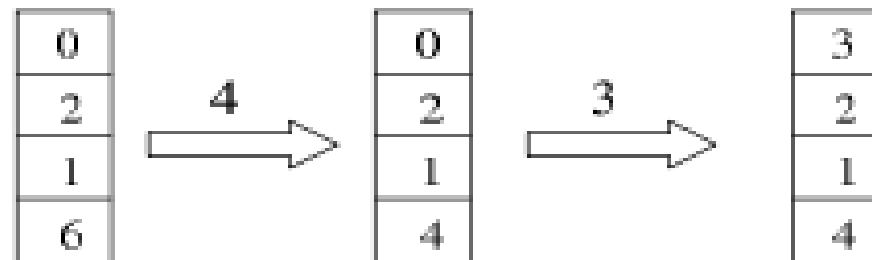


- Fault Rate = $9 / 12 = 0.75$

Optimal Page Replacement

- Consider the following reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

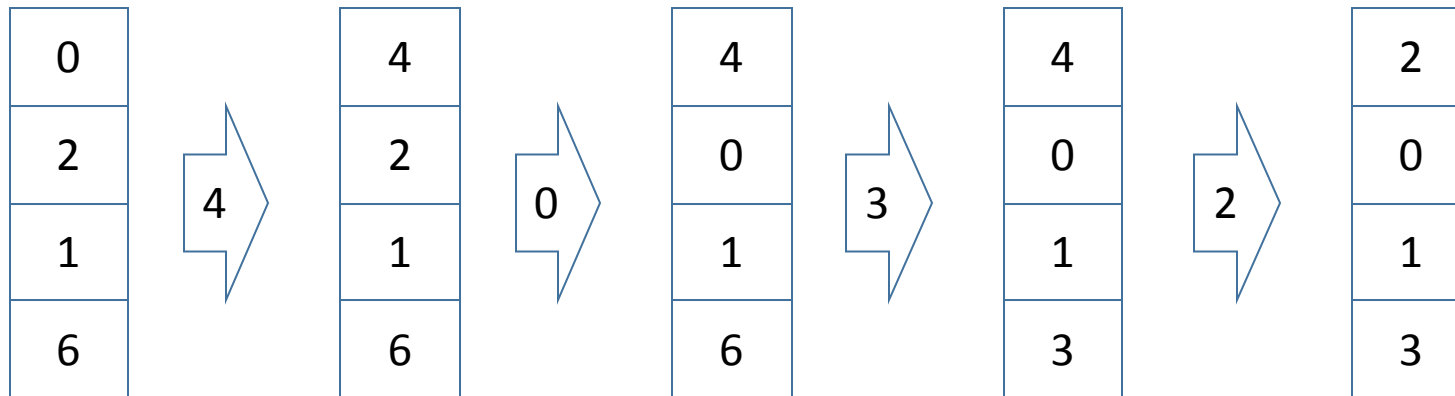
Compulsory Misses 



- Fault Rate = $6 / 12 = 0.50$
- With the above reference string, this is the best we can hope to do

LRU

0	2	1	6	4	0	1	0	3	1	2	1
X	X	X	X	X	X			X		X	

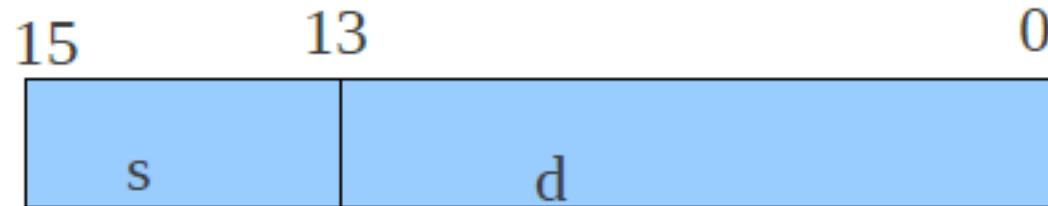


Fault rate=8/12

Segmentation

- segmentation is another techniques of non-contiguous memory allocation method
- variable size
- User view
- The general division can be: main program, set of subroutines, procedures, functions and set of data structures(stack, array etc)
- Segment number
- Segmentation maintains multiple separate virtual address spaces per process.
- Allows each table to grow or shrink, independently

Segment Address



Segment
number

Displacement

Fig: Virtual address space or logical address
space (16 bit)

Segment

