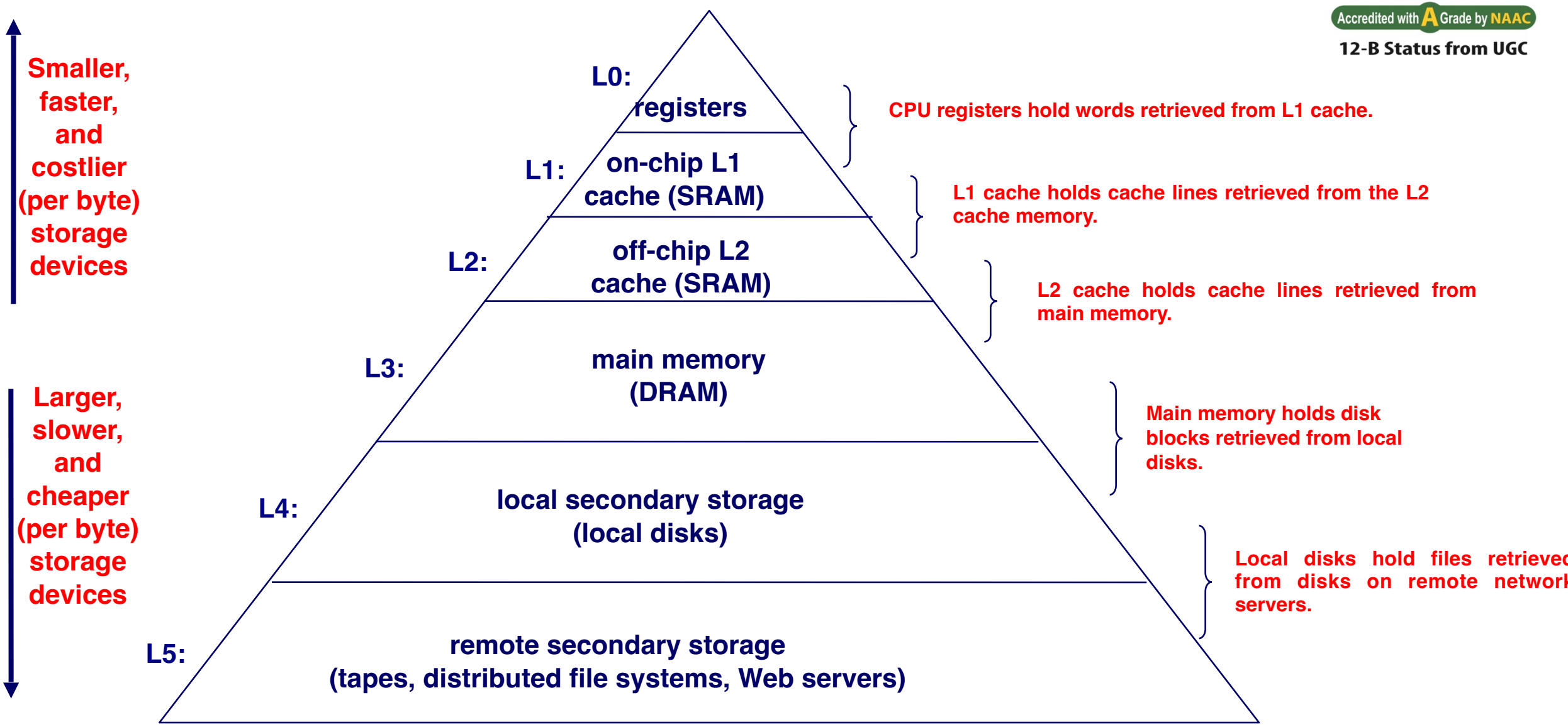
BCA 3rd Semester
Operating system
Subject Code: BCAC0022

Presented by:
Jayati Krishna
Assistant Professor, Dept. Of CEA
GLA University, Mathura

An Example Memory Hierarchy



From lecture-9.ppt

Which is best ?

Memory Hierarchy

- In computer architecture the **memory hierarchy** is a concept used to discuss performance issues in computer architectural design, algorithm predictions, and lower level programming constructs involving locality of reference.
- The memory hierarchy in computer storage separates each of its levels based on response time.
- Since response time, complexity, and capacity are related, the levels may also be distinguished by their performance and controlling technologies.

- **Primary Memory** The primary memory is also known as internal memory, and this is accessible by the processor . This memory includes main, cache, as well as CPU registers.
- **Secondary Memory** The secondary memory is also known as external memory, and this is accessible by the processor through an input/output module. This memory includes an optical disk, magnetic disk, and magnetic tape.

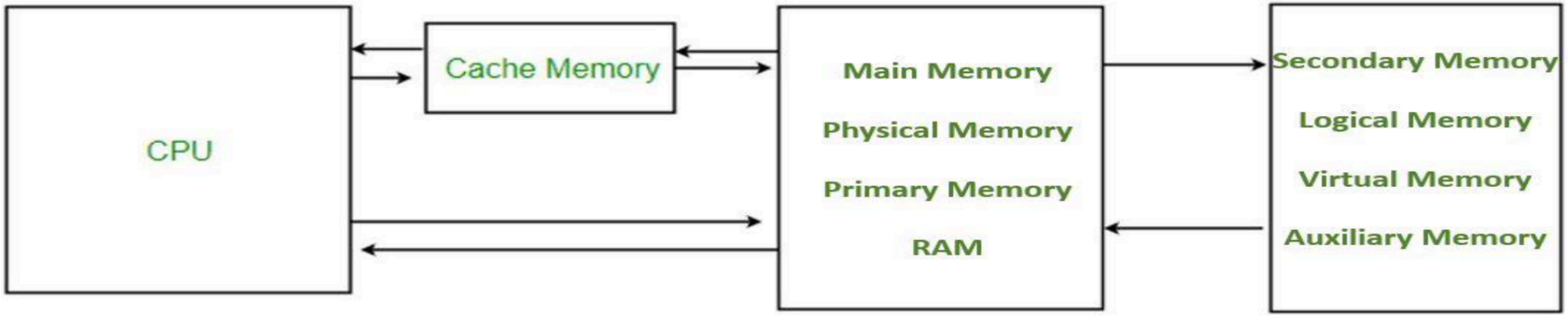
Characteristics of Memory Hierarchy

- The memory hierarchy characteristics mainly include the following.
- **Performance**
- The speed gap among the main memory as well as the CPU registers enhances because of the huge disparity in access time, which will cause the lower performance of the system.
- So, the enhancement was mandatory. The enhancement of this was designed in the memory hierarchy model due to the system's performance increase.
- **Ability**
- The ability of the memory hierarchy is the total amount of data the memory can store. Because whenever we shift from top to bottom inside the memory hierarchy, then the capacity will increase.

- **Access Time**
- The access time in the memory hierarchy is the interval of the time among the data availability as well as request to read or write. Because whenever we shift from top to bottom inside the memory hierarchy, then the access time will increase
- **Cost per bit**
- When we shift from bottom to top inside the memory hierarchy, then the cost for each bit will increase which means an internal Memory is expensive compared with external memory

Cache Memory

- The data or contents of the main memory that are used frequently by CPU are stored in the cache memory so that the processor can easily access that data in a shorter time.
- Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory, then the CPU moves into the main memory.
- Cache memory is placed between the CPU and the main memory. The block diagram for a cache memory can be represented as:

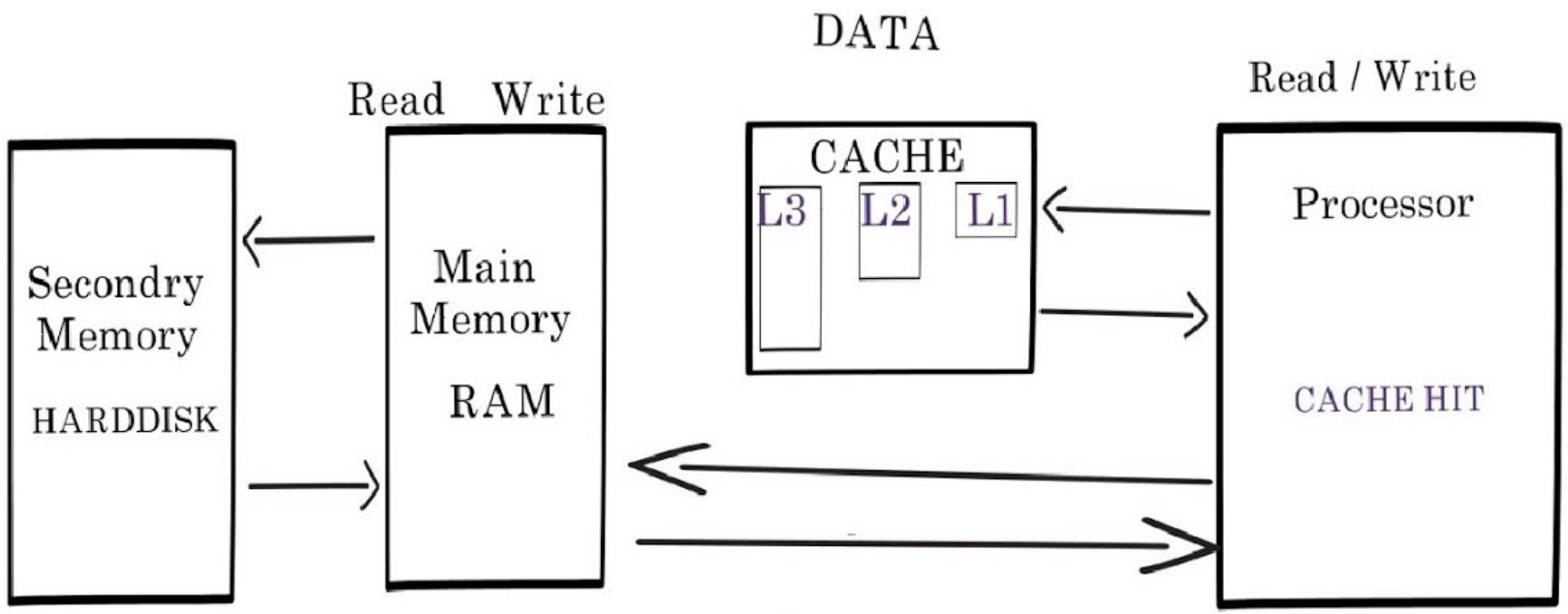


Showroom
4 MB

Go down
32 GB

Factory
8 TB

CACHE MEMORY BLOCK DIAGRAM



$$\text{HIT RATIO} = \frac{\text{Nos of Hits}}{\text{Total Access}} = \frac{\text{Nos of Hits}}{\text{Nos of Hits + Nos of Miss}}$$

Locality of Reference

- The references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of locality of reference. There are two types of locality of reference.
- **Spatial Locality**: Use of data elements in the nearby locations.
- **Temporal Locality**: Temporal locality refers to the reuse of specific data or resources, within a relatively small-time duration, i.e. Most Recently Used.

The basic operation of a cache memory is as follows:

- When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory.
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.
- A block of words one just accessed is then transferred from main memory to cache memory. The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed.

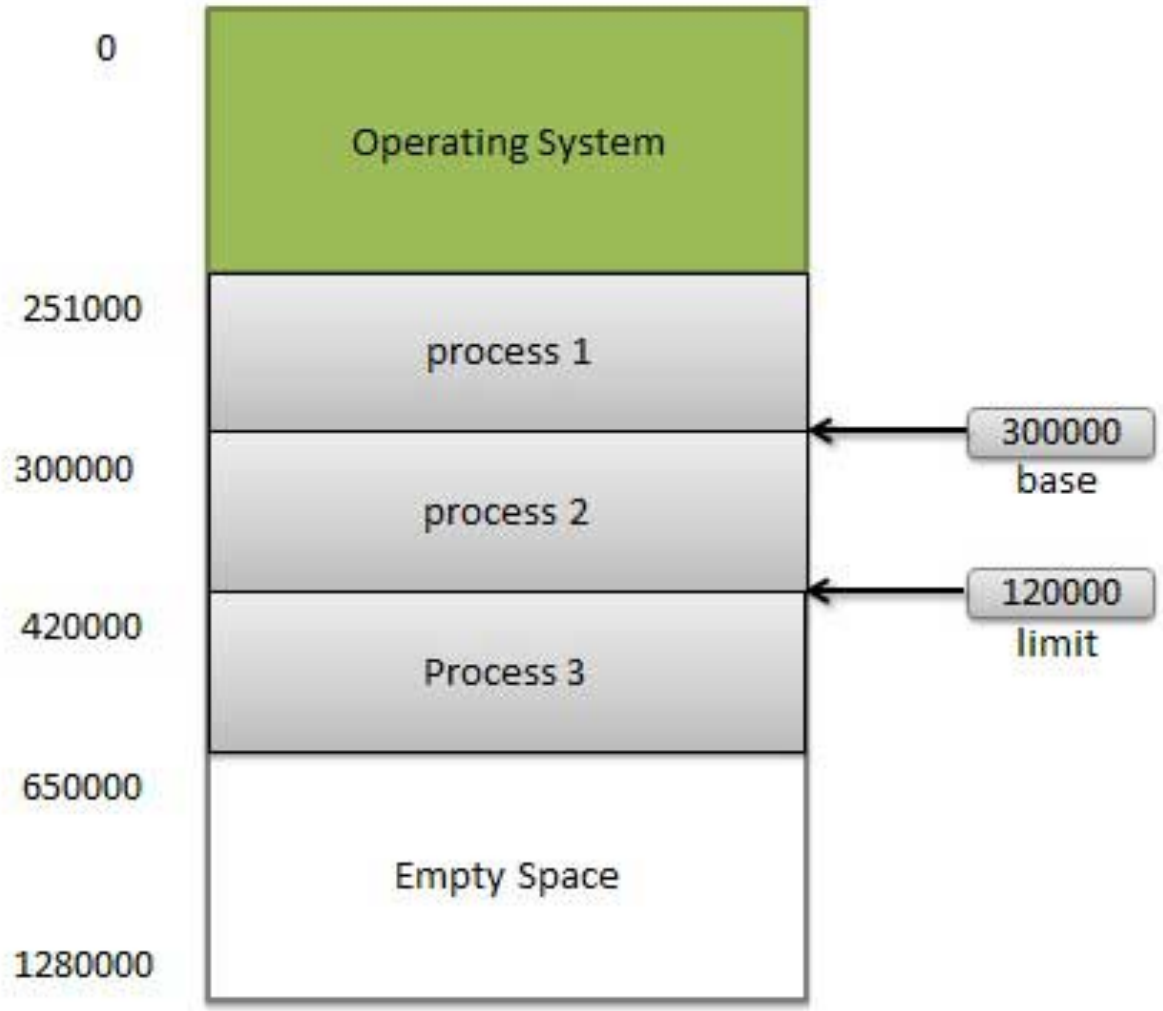
- The performance of the cache memory is frequently measured in terms of a quantity called **hit ratio**.
- When the CPU refers to memory and finds the word in cache, it is said to produce a **hit**.
- If the word is not found in the cache, it is in main memory and it counts as a **miss**.
- The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio.

Duties of OS

- Operating system is responsible for the following activities in connection with memory management:
- **Address Translation**: Convert logical addresses to physical addresses for data retrieval.
- **Memory Allocation and Deallocation**: Decide which processes or data segments to load or remove from memory as needed.
- **Memory Tracking**: Monitor which parts of memory are in use and by which processes.
- **Memory Protection**: Implement safeguards to restrict unauthorized access to memory, ensuring both process isolation and data integrity.

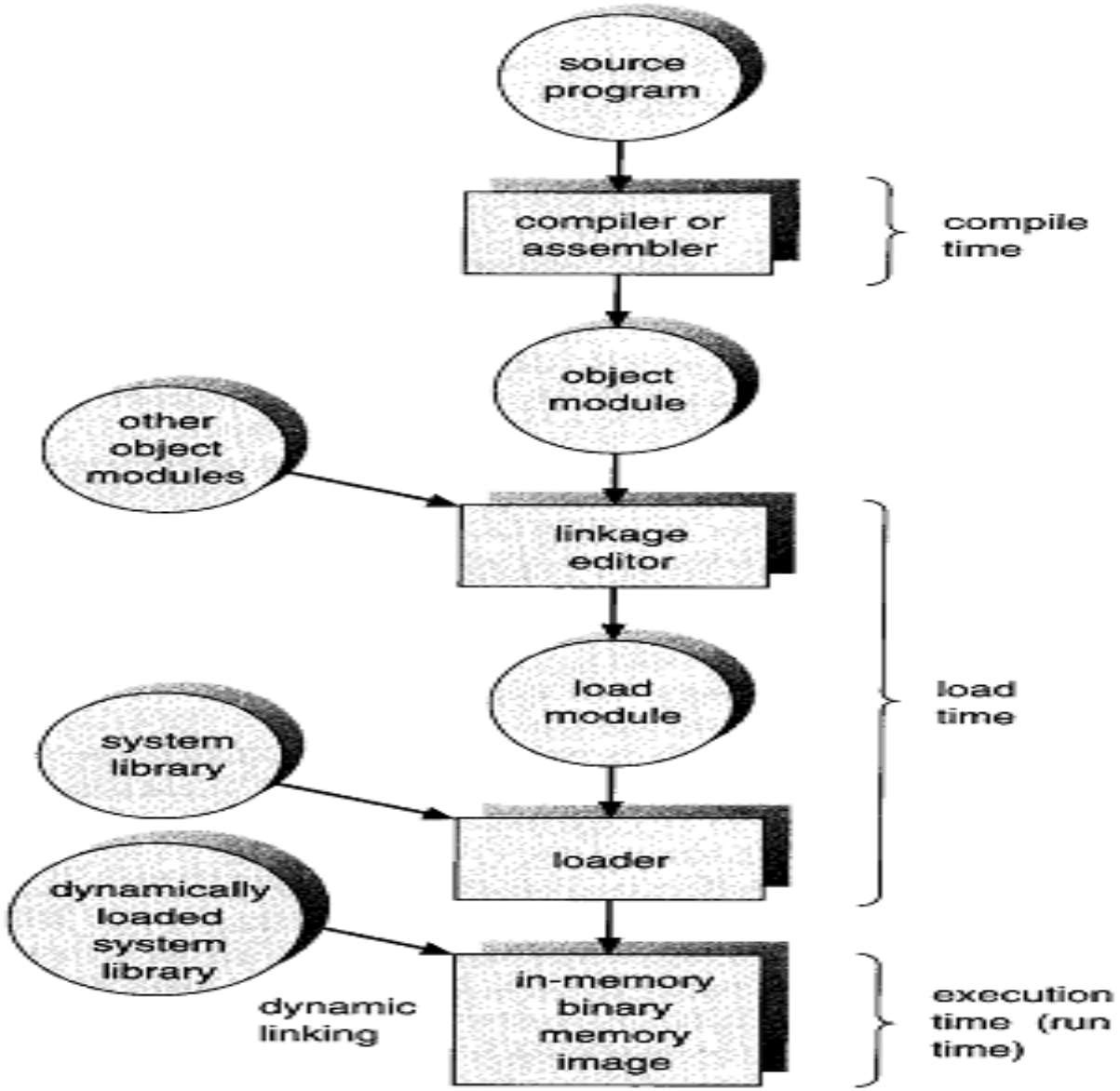
Memory Management

- The memory management services of an operating system are one of the basic services as it is needed:
- To ensure protection of different processes from each other (so that they do not interfere with each other's operation).
- To place the programs in memory (such that memory is optimum utilized and high degree of multiprogramming can be achieved).
- Memory management keeps track of each and every memory location either it is allocated to some process or it is free.
- It checks how much memory is to be allocated to processes. It decides which process will get memory at what time.
- It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.



- Memory management provides protection by using two registers, a base register and a limit register.
- The base register holds the smallest legal physical memory address and the limit register specifies the size of the range.
- For example, if the base register holds 300000 and the limit register is 120000, then the program can legally access all addresses from 300000 through 419999

Multistep processing of a user program



Logical Versus Physical Address Space

- An address generated by the CPU is commonly referred to as a **logical address (virtual address).**
- An address seen by the memory unit (that is, the one loaded into the **memory-address register** of the memory) is commonly referred to as a **physical address.**
- The set of all logical addresses generated by a program is a **logical** address **space.**
- the set of all physical addresses corresponding to these logical addresses is a **physical** address **space.**
-

Address Binding

- Address binding is the process of mapping the program's logical or virtual addresses to corresponding physical or main memory addresses.
- In other words, a given logical address is mapped by the MMU (Memory Management Unit) to a physical address.
- **CPU generates the logical or virtual address** for an instruction/data to be fetched from RAM.
- The logical address undergoes translation by the MMU or address translation unit in particular.
- The output of this process is the appropriate physical address or the location of code/data in RAM.

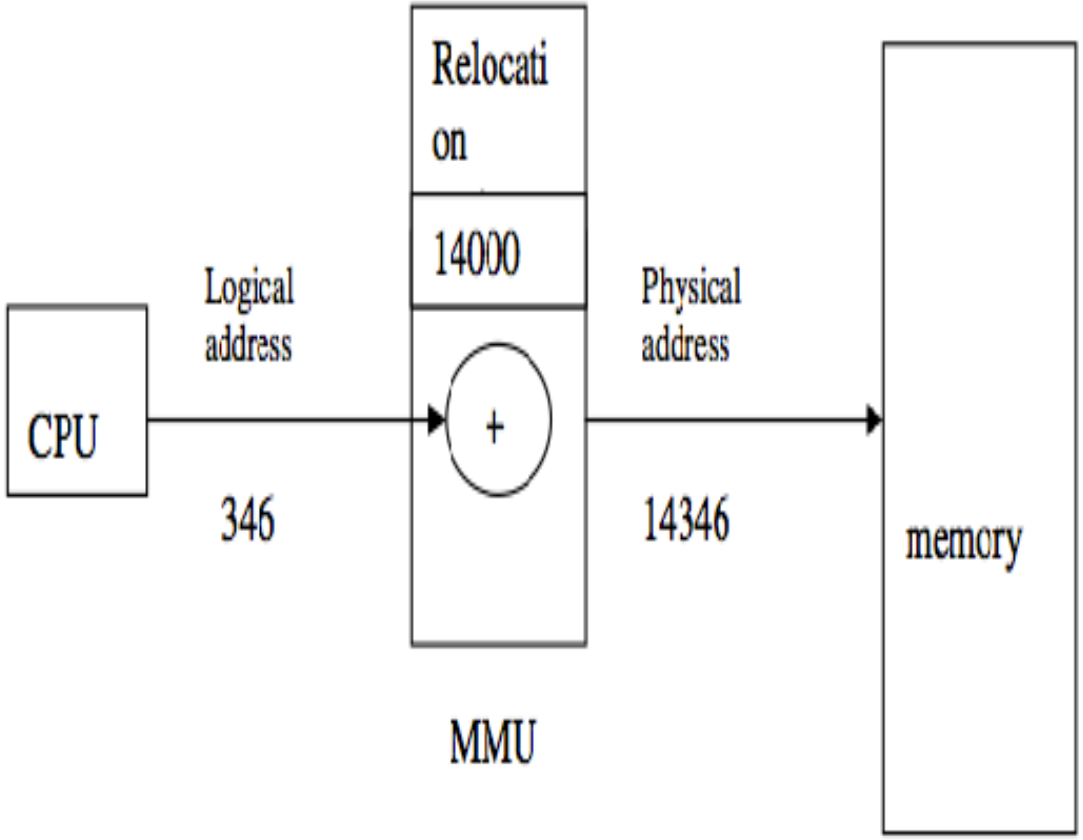


Figure 1

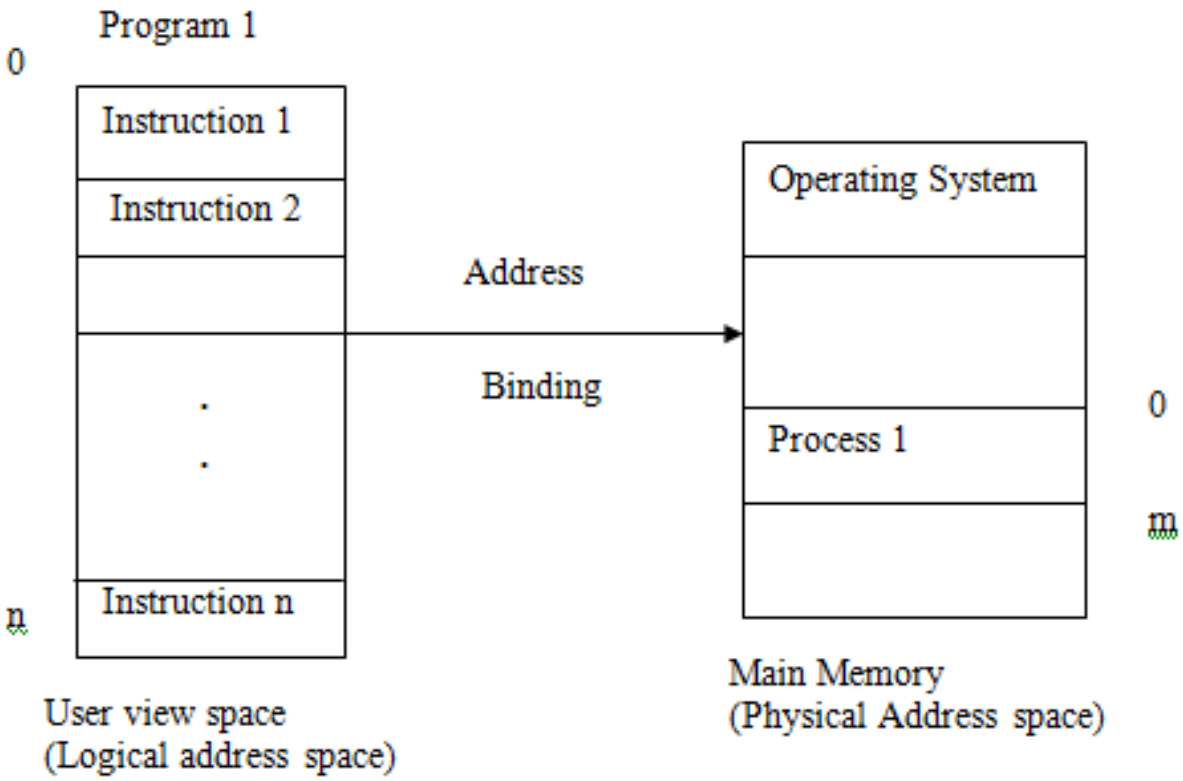


Figure 2

Dynamic Loading

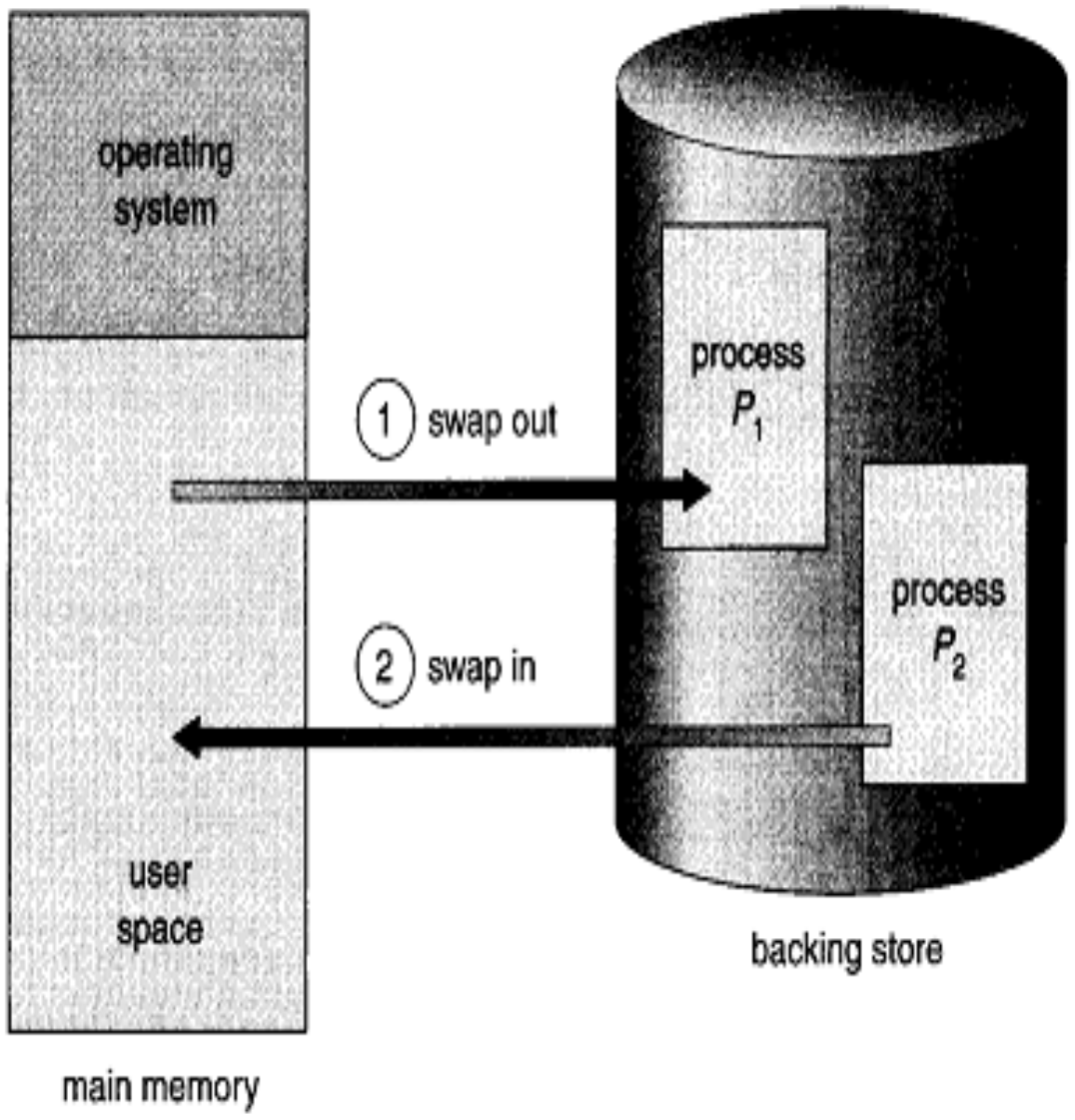
- With dynamic loading, a routine is not loaded until it is called.
- All routines are kept on disk in a relocatable load format. The main program is loaded into memory and is executed.
- When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded.
- If not, the relocatable loader is called to load the desired routine into memory and to update the program's address tables to reflect this change. Then control is passed to the newly loaded routine.
- **Advantages**
- Unused routine is never loaded
- It does not need special support from O/S
- O/S provides library routine to implement dynamic loading

Dynamic Linking:

- The concept of dynamic linking is similar to the dynamic binding. Without this facility, all program on a system need to have a copy of their language library.
- With the dynamic linking, a **stub** is included in the image for each library routine reference.
- The stub is a small piece of code that indicates how to load the library, if the routine is not already present.
- When this stub is executed, it checks to see whether the needed routine is already in memory.
- If it is not in memory, the program loads it into memory. Either way stub replaces itself with the address of the routine and executes the routine.

Swapping

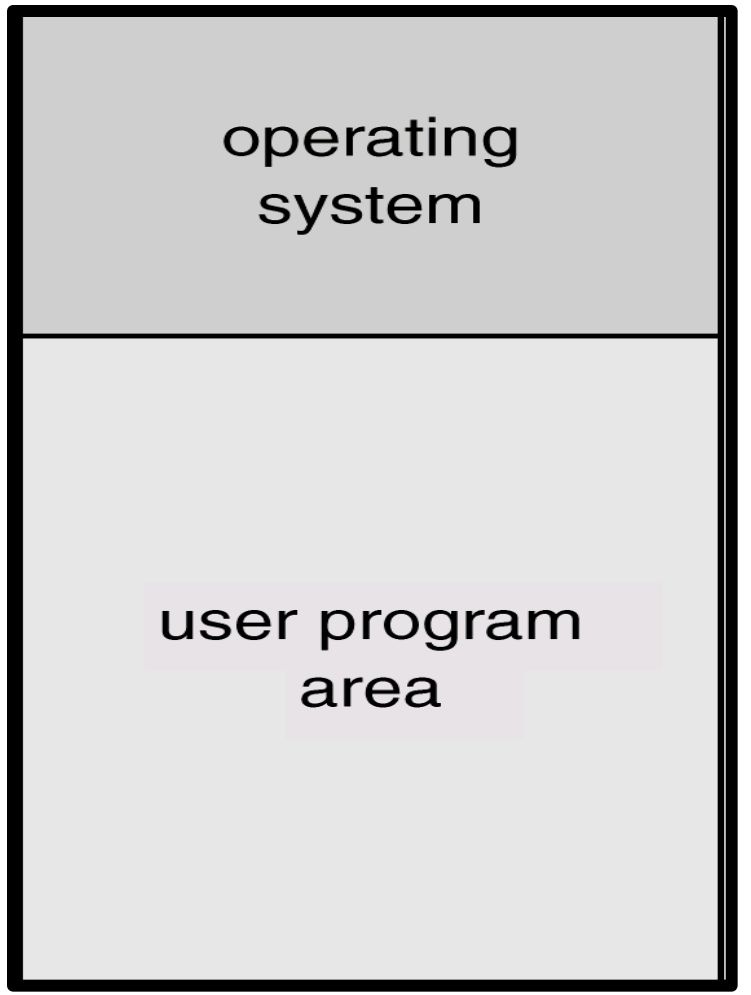
- A process must be in memory to be executed. A process, however, can be swapped temporarily out of memory to a **backing store** and then again brought back into memory for continued execution.
- For example, in a multiprogramming environment with a **round-robin CPU-scheduling algorithm**,
- when a quantum expires, the memory manager will start to swap out the process that just finished and to swap another process into the memory space that has been freed.
- In the meantime, the CPU scheduler will allocate a time slice to some other process in memory. When each process finishes its quantum, it will be swapped out with another process.



- This swapping policy is also used in priority-based scheduling algorithms.
- If a **higher-priority process** arrives and wants service, the memory manager can swap out the lower-priority process and then load and execute the higher-priority process.
- When the higher-priority process finishes, the lower-priority process can be swapped back in and continued.
- This variant of swapping is sometimes called **roll out, roll in.**

Contiguous Memory Allocation

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.

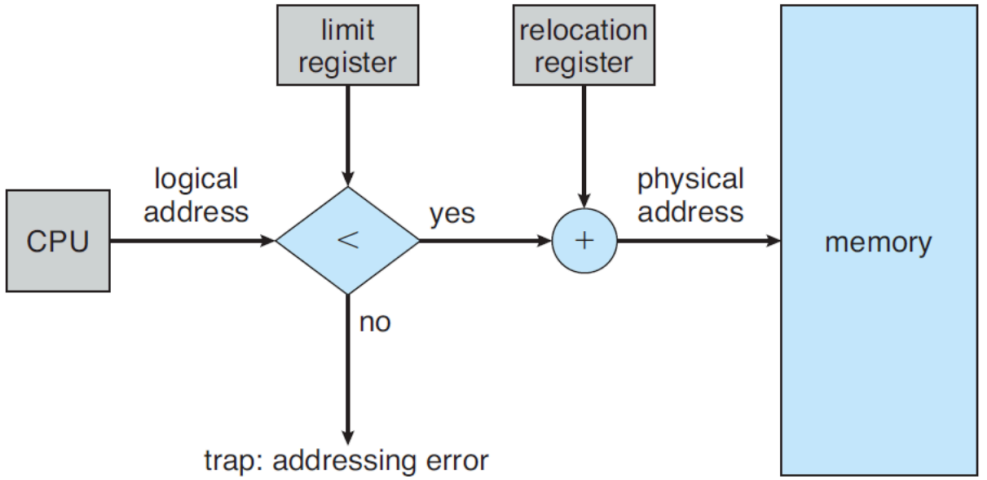


Contiguous allocation policy

- We know that when a process is required to be executed it must be loaded to main memory, by policy has two implications.
    - It must be loaded to main memory completely for execution.
    - Must be stored in main memory in contiguous fashion.

Address Translation in Contiguous Allocation

- Here we use a Memory Management Unit(OS) which contains a relocation register, which contains the base address of the process in the main memory and it is added in the logical address every time.
- In order to check whether address generated to CPU is valid(with in range) or invalid, we compare it with the value of limit register, which contains the max no of instructions in the process.
- So, if the value of logical address is less than limit, then it means it's a valid request and we can continue with translation otherwise, it is a illegal request which is immediately trapped by OS.

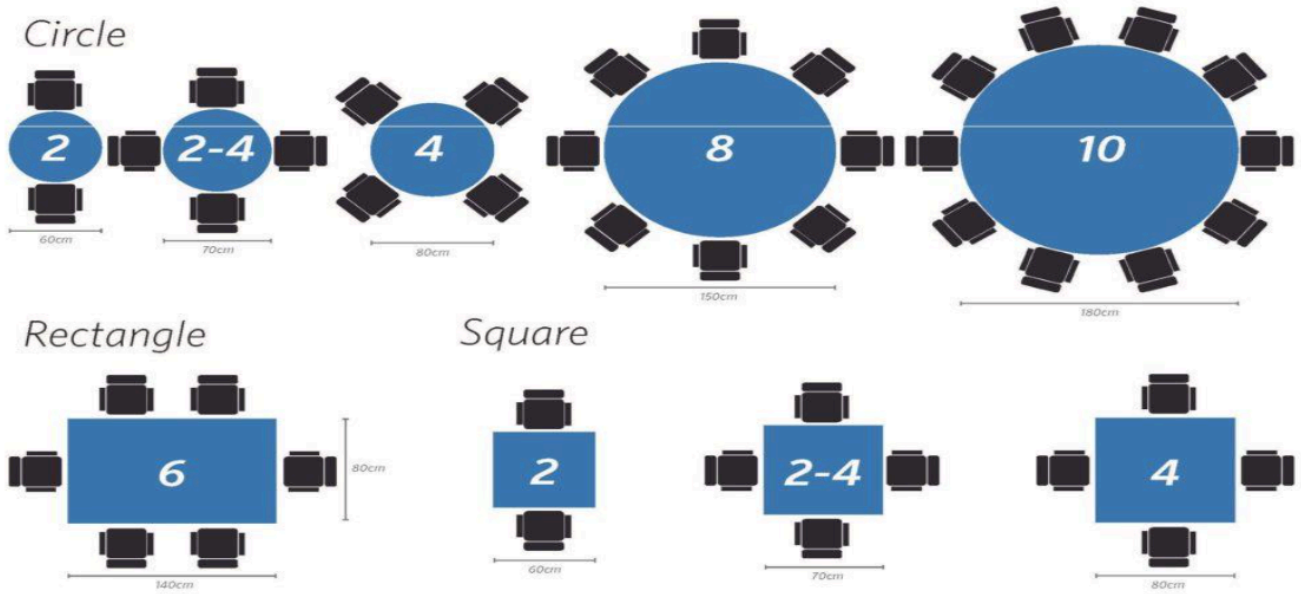


Space Allocation Method in Contiguous Allocation

- **<u>Variable size partitioning</u>**: -In this policy, in starting, we treat the memory as a whole or a single chunk & whenever a process request for some space, exactly same space is allocated if possible and the remaining space can be reused again.

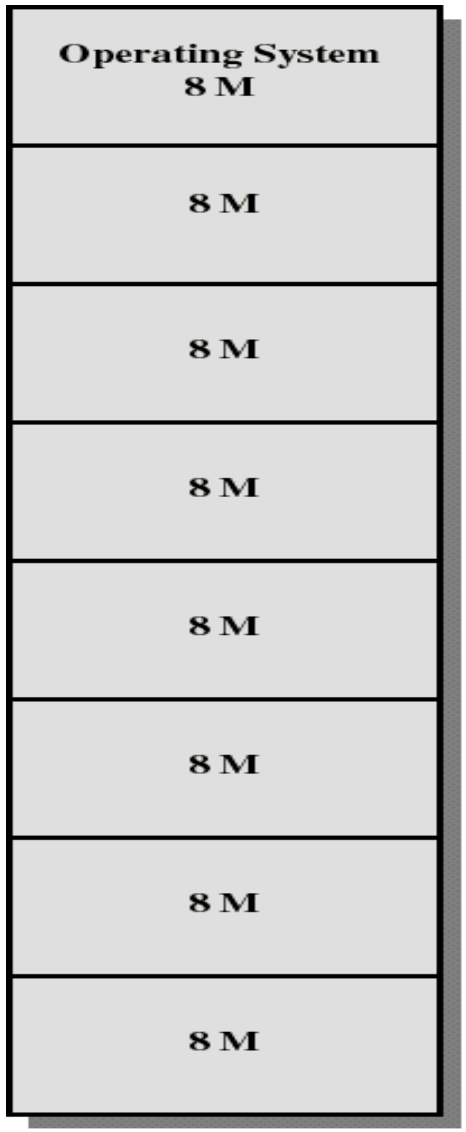
Space Allocation Method in Contiguous Allocation

- **<u>Fixed size partitioning: -</u>** here, we divide memory into fixed size partitions, which may be of different sizes, but here if a process request for some space, then a partition is allocated entirely if possible, and the remaining space will be wasted internally.

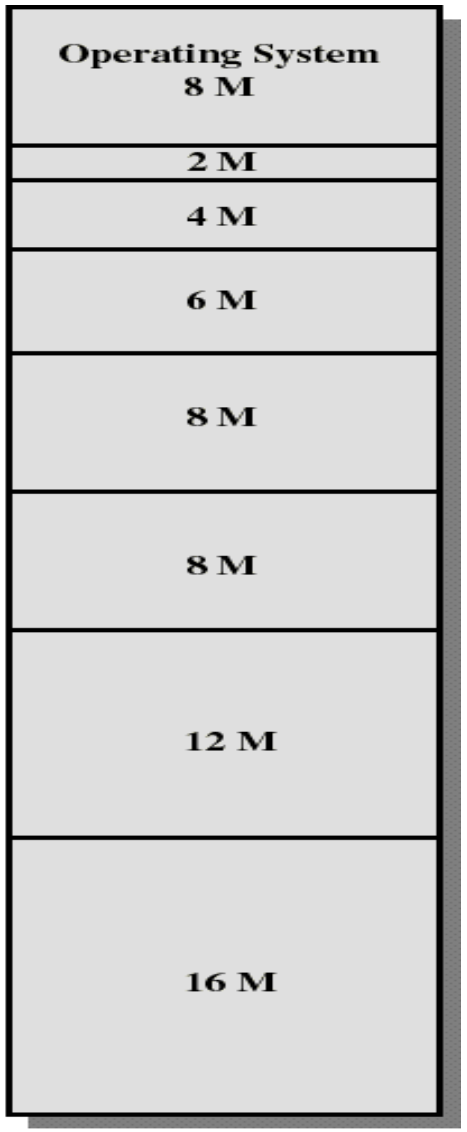


Memory Allocation: Fixed Partitioning

- Divide main memory into a set of non overlapping regions called partitions
- Partitions can be of equal or unequal sizes
- any process whose size is less than or equal to a partition size can be loaded into the partition
- if all partitions are occupied, the operating system can swap a process out of a partition



Equal-size partitions



Unequal-size partitions

- **First fit policy:** - It states searching the memory from the base and will allocate first partition which is capable enough. It is fast, but fragments.
    - **Advantage**: - simple, easy to use, easy to understand
    - **Disadvantage**: -poor performance, both in terms of time and space.
- **Best fit policy:** - We search the entire memory and will allocate the smallest partition which is capable enough. It is slow, but small fragments
    - **Advantage**: - perform best in fix size partitioning scheme.
    - **Disadvantage**: - difficult to implement, perform worst in variable size partitioning as the remaining spaces which are of very small size.

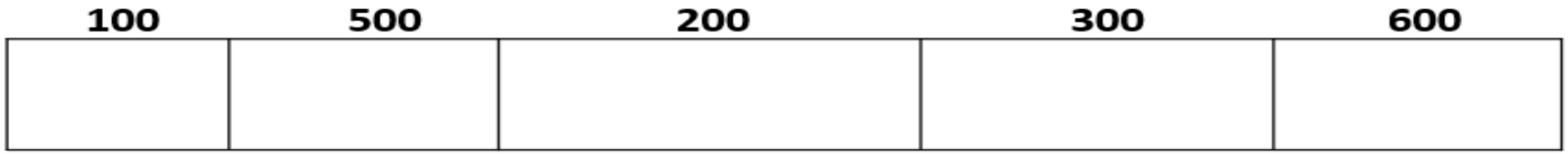
- **Worst fit policy**: - It also searches the entire memory and allocate the largest partition possible.It is slow, but leaves large holes
    - **Advantage**: - perform best in variable size partitioning
    - **Disadvantage**: - perform worst in fix size partitioning, resulting into large internal fragmentation.

Fixed Partitioning – Problems

- Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called internal fragmentation.
- Unequal-size partitions lessens these problems but they still remain...
- Equal-size partitions was used in early IBM's OS/MFT (Multiprogramming with a Fixed number of Tasks)

Practice 1

- Consider five memory partitions of size 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB, where KB refers to kilobyte. These partitions need to be allotted to four processes of sizes 212 KB, 417 KB, 112 KB and 426 KB in that order. Perform the allocation of processes using-First Fit Algorithm, Best Fit Algorithm and Worst Fit Algorithm

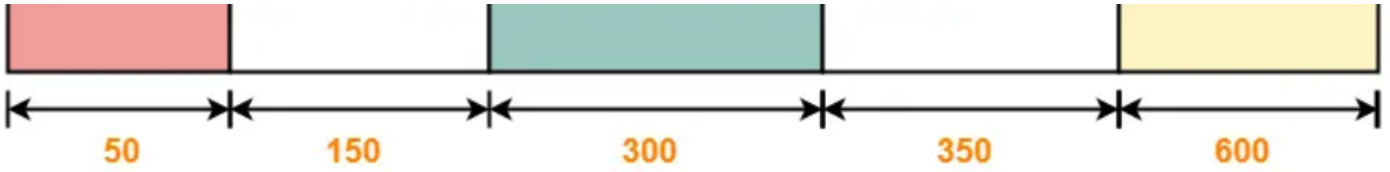


Practice 2

- Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB in order. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order. Perform the allocation of processes using-
    - First Fit Algorithm,
    - Best Fit Algorithm
    - Worst Fit Algorithm

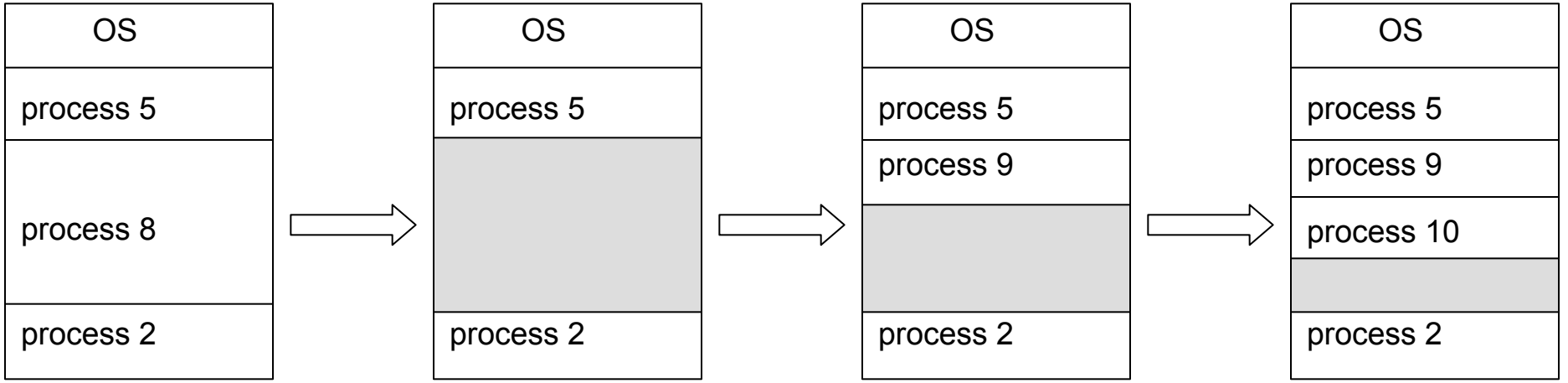
Practice 3

- Consider the following heap (figure) in which blank regions are not in use and hatched regions are in use- The sequence of requests for blocks of size 300, 25, 125, 50 can be satisfied if we use(which option is correct)-
    - Either first fit or best fit policy (any one)
    - First fit but not best fit policy
    - Best fit but not first fit policy
    - None of the above



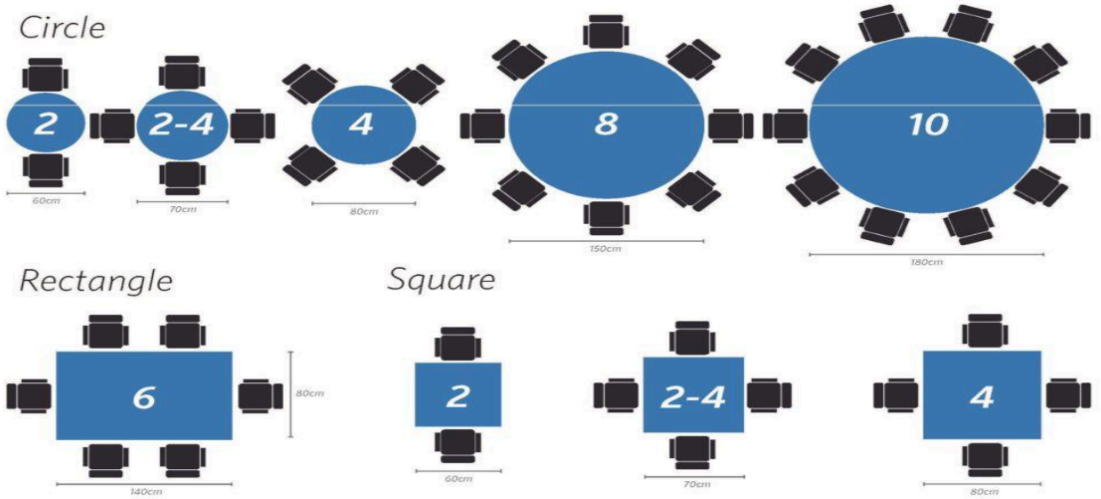
Multiple Partition Allocation: Dynamic

- *Hole* – block of available memory; holes of various size are scattered throughout memory.
- When a process arrives, it is allocated contiguous memory from a hole large enough to accommodate it.
- Operating system maintains information about:
  a) allocated partitions b) free partitions (hole)



External Fragmentation

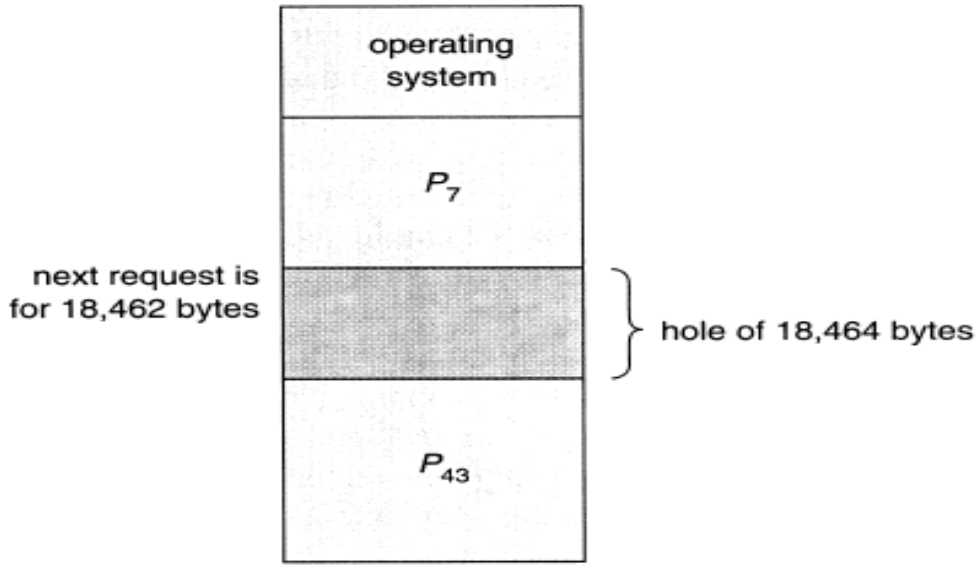
- External fragmentation is a function of contiguous allocation policy. The space requested by the process is available in memory but, as it is not being contiguous, cannot be allocated this wastage is called external fragmentation.



Internal Fragmentation

- Internal fragmentation is a function of fixed size partition which means, when a partition is allocated to a process. Which is either the same size or larger than the request then, the unused space by the process in the partition Is called as internal fragmentation.

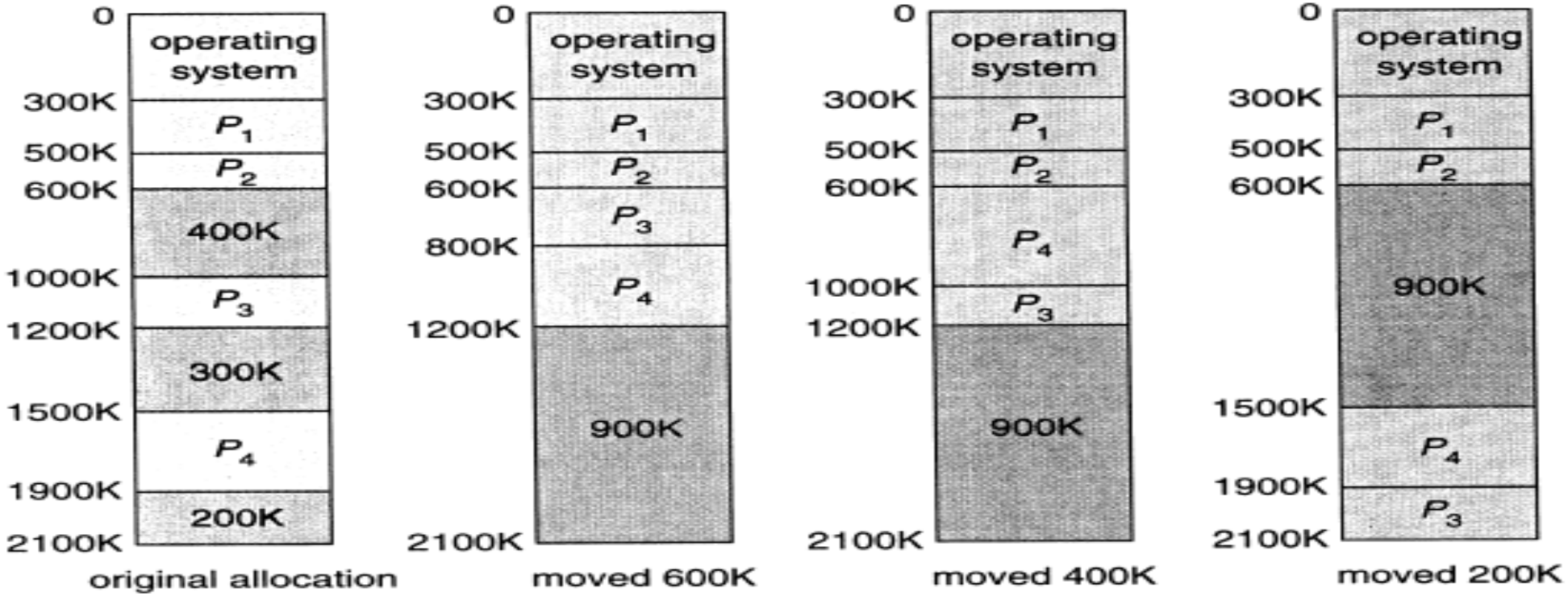
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
  - Occurs when memory is allocated in fixed size pieces



Compaction

- One solution to the problem of external fragmentation is compaction. The goal is to move the memory contents so as to place all free memory together in one large block. However, compaction is not always possible.
- If addresses are relocated dynamically, relocation requires only moving the program and data and then changing the base register to reflect the new base address.
- The compaction moves all processes toward one end of memory and all holes in the other direction, producing one large hole of available memory. This scheme can be expensive.
- Eg: disk defragmenter in Windows.
- Internal fragmentation can be aimed by having partitions of several sizes and assigning a program based on the best fit. However, still internal fragmentation is not fully eliminated.

Compaction Options



Non-contiguous Allocation

Two principal approaches:

- Segmentation
- Paging

Full Form	Units	Bytes
1 Bit	Binary Digit (0/1)	
1 Nibble	4 bits	
1 Byte	8 bits	
1 kilobyte(KB)	1024 byte	2^{10} bytes
1 Megabyte(MB)	1024 KB	2^{20} bytes
1 Gigabyte (GB)	1024 MB	2^{30} bytes
1 Terabyte(TB)	1024 GB	2^{40} bytes
1 Petabyte(PB)	1024 TB	2^{50} bytes
1 Exabyte(EB)	1024 PB	2^{60} bytes
1 Zettabyte(ZB)	1024 EB	2^{70} bytes
1 Yottabyte(YB)	1024 ZB	2^{80} bytes
1 Brontobyte	1024 YB	2^{90} bytes
1 Geopbyte	1024 Brontobyte	2^{100} bytes

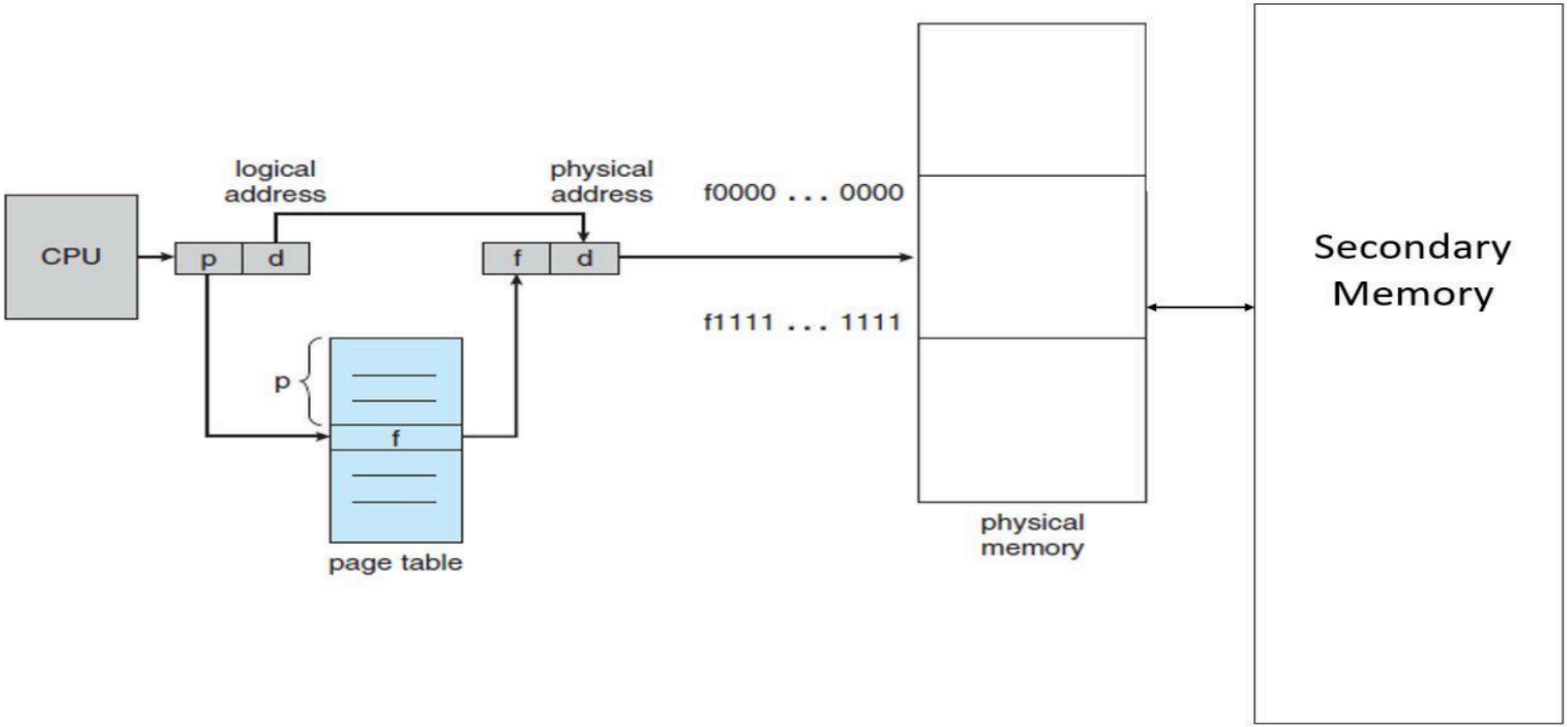
Paging

- It is a memory management scheme that eliminates the need for contiguous allocation of physical memory. It avoids external fragmentation.
- This scheme permits the physical address space of a process to be non – contiguous.
- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Every address generated by the CPU is divided into two parts:
- **page number (p)**
- **page offset (d)**

- Secondary memory is divides into fixed size partition(because management is easy) all of them of same size called pages(easy swapping and no external fragmentation).
- Main memory is divided into fix size partitions (because management is easy), each of them having same size called frames(easy swapping and no external fragmentation).
- Size of frame = size of page
- In general number of pages are much more than number of frames (approx. 128 time)

Translation Process

- CPU generate a logical address is divided into two parts - p and d where p stands for page no and d stands for instruction offset.
- The page number(p) is used as an index into a Page table
- Page table base register(PTBR) provides the base of the page table and then the corresponding page no is accessed using p.
- Here we will finds the corresponding frame no (the base address of that frame in main memory in which the page is stored)
- Combine corresponding frame no with the instruction offset and get the physical address. Which is used to access main memory.



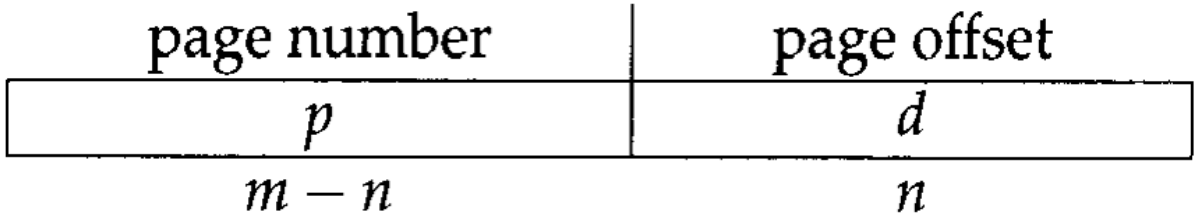
Page Table

- Page table is a data structure not hardware.
- Every process have a separate page table.
- Number of entries a process have in the page table is the number of pages a process have in the secondary memory.
- Size of each entry in the page table is same it is corresponding frame number.
- Page table is a data structure which is it self stored in main memory.

- **Advantage**
  - Removal of External Fragmentation
- **Disadvantage**
  - Translation process is slow as Main Memory is accessed two times(one for page table and other for actual access).
  - A considerable amount of space a waisted in storing page table(meta data).
  - System suffers from internal fragmentation(as paging is an example of fixed size partition).
  - Translation process is difficult and complex to understand and implement.

Page Addressing

- If the size of logical address space is 2^m and a page size is 2^n addressing units (bytes or words), then the high-order **m – n** bits of a logical address designate ***the page number,*** and the **n** low-order bits designate the ***page offset***. Thus, the logical address is as follows:
- where p is an index into the page table and d is the displacement within the page.



- Address Length in bits= **n**
- No. of Locations = $\mathbf{2^n}$
- Memory Size= No. of Location * Size of each Location
- Q. Suppose you have address of 32 Bit, then how many locations will be there?
- Ans- $\mathbf{2^{32}}$
- Q- Each Location is of 1 Byte, Then total memory size will be?
- Ans- $\mathbf{2^{32}}$ *** 1 Byte= 4 GB**

- Address Length in bits= U**pper Bound($\log_2 n$)**
- Memory Size. / Size of each Location = No. of Location
- Q. Suppose we have memory 512 MB, what will be address length when each location is of 1 B.
- Ans- **2^{29} (29 bits)**

- **For Main Memory-**

- Physical Address Space = Size of main memory
- Size of main memory = Total number of frames x Page size
- Frame size = Page size
- If number of frames in main memory = 2^X, then number of bits in frame number = X bits
- If Page size = 2^X Bytes, then number of bits in page offset = X bits
- If size of main memory = 2^X Bytes, then number of bits in physical address = X bits

- **For Process-**
- Virtual Address Space = Size of process
- Number of pages the process is divided = Process size / Page size
- If process size = 2^X bytes, then number of bits in virtual address space = X bits

For Page Table-

- Size of page table = Number of entries in page table x Page table entry size
- Number of entries in pages table = Number of pages the process is divided
- Page table entry size = Number of bits in frame number + Number of bits used for optional fields if any

1) How to calculate number of bits in logical address and physical address when logical address space of 8 pages of 1024 word each, mapped to physical memory of 32 frames?

Size of logical address space is

No. of pages * Page size = 8 * 1024 = 2^3 * 2 ^10 = 2^13

No. of bits for logical address is 13

Size of Physical address space is

2^5 * 2^10 = 2^15

No. of bits for physical address is 15

2) Assume a page size of 1K and a 15-bit logical address space. How many pages are in the system?

2^5 = 32.

3) Assuming a 15-bit address space with 8 logical pages. How large are the pages?

2^12 = 4K(4096).

1. Calculate the size of memory if its address consists of 22 bits and the memory is 2-byte addressable.
2. Calculate the number of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable.
3. Consider a system with byte-addressable memory, 32 bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is _____.
4. Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?

1. Calculate the size of memory if its address consists of 22 bits and the memory is 2-byte addressable. **8MB**
2. Calculate the number of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable. **32 Bits**
3. Consider a system with byte-addressable memory, 32 bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is **4 MB**
4. Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?

2 MB

Practice

Consider a paged memory system with logical address of 26-bits and physical address of 32-bits. The page size is 2KB) Further consider that one page table entry size is 4byes. Find the following-

1.Bits in page offset

2.Number of pages in process

3.Bits for page number

4.Number of frames in physical memory

5.Bits for frame number

6.Page table size

Here, in the logical address, *n= 2 and m = 4. Using a page size* of 4 bytes and a physical memory of 32 bytes (8 pages), we show how user's view of memory can be mapped into physical memory.

•Logical address 0 is page 0, offset 0.

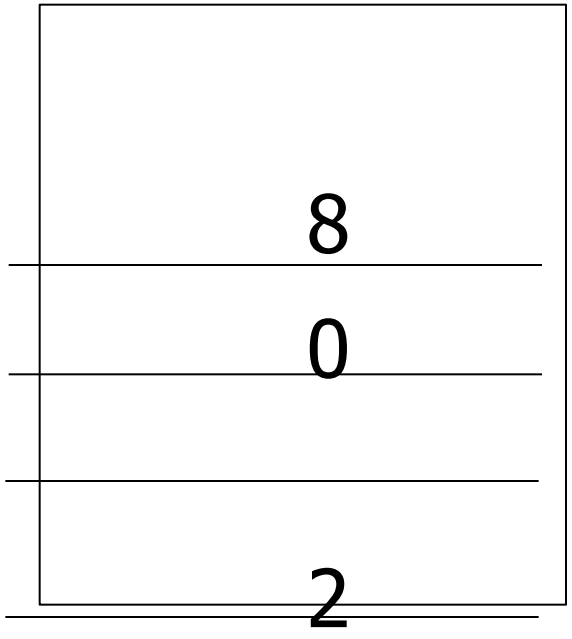
•Indexing into the page table, we find that page 0 is in frame 5. Thus, ***logical address 0 maps to physical address 20*** [= (5 x 4) + 0]

•Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 x 4) + 3]

•Logical address 4 is page 1, offset 0; Thus, logical address 4 maps to physical address 24 [= (6 x 4) + O]

•Logical address 13 maps to physical address 9

Consider logical address 1025 and the following page table for some process P0. Assume a 15-bit address space with a page size of 1K. What is the physical address to which logical address 1025 will be mapped?



Consider logical address 1025 and the following page table for some process P0. Assume a 15-bit address space with a page size of 1K. What is the physical address to which logical address 1025 maps?

8
0
2

Step 1. Convert to binary:

000010000000001

Consider logical address 1025 and the following page table for some process P0. Assume a 15-bit address space with a page size of 1K. What is the physical address to which logical address 1025 maps?

8
0
2

Step2. Determine the logical page number:

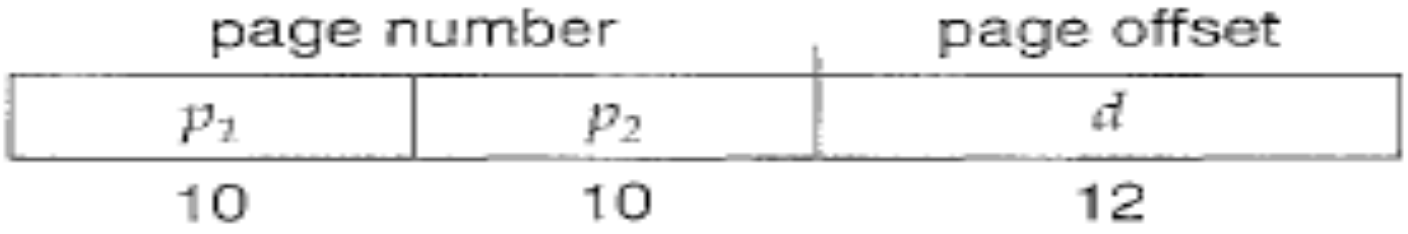
Since there are 5-bits allocated to the logical page, the address is broken up as follows:

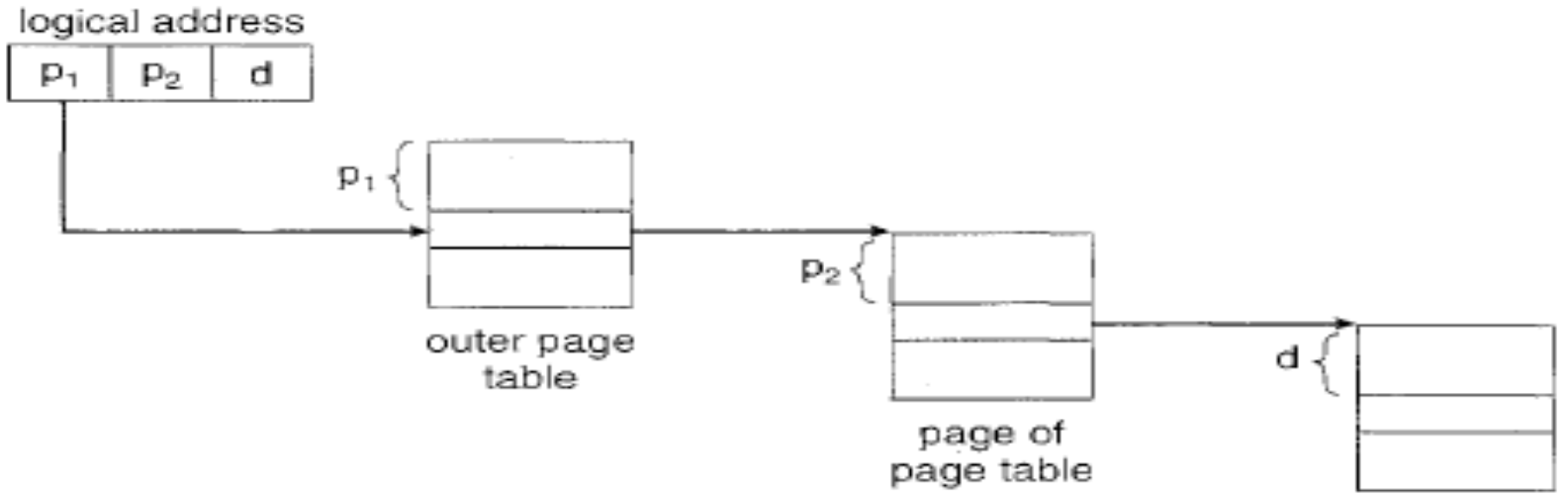
00001

Hierarchical Paging

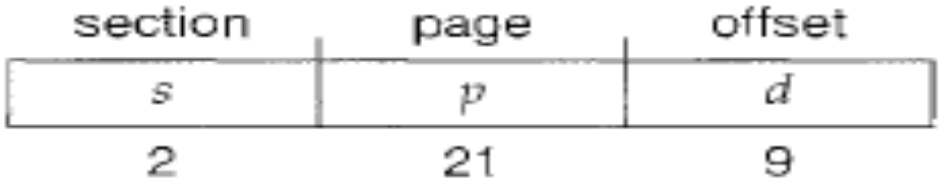
- Most modern computer systems support a large logical address space (2^{32} to 2^{64}). In such an environment, the page table itself becomes excessively large.
- One simple solution *to this problem is to divide the page table* into smaller pieces. We can accomplish this division in several ways.
- **Two-level paging algorithm:** In this the page table itself is also paged. For example, consider again the system with a 32-bit logical address space and a page size of 4 KB.
- A logical address is divided into a page number consisting of 20 bits and a page offset consisting of 12 bits.
- Because we page the page table, the page number is further divided into a 10-bit page number and a 10-bit page offset. Thus, a logical address is as follows:

- where p1 is an index into the outer page table and *P2 is the displacement* within the page of the outer page table. Because address translation works from
- the outer page table inward, this scheme is also known as a **forward-mapped page table.**

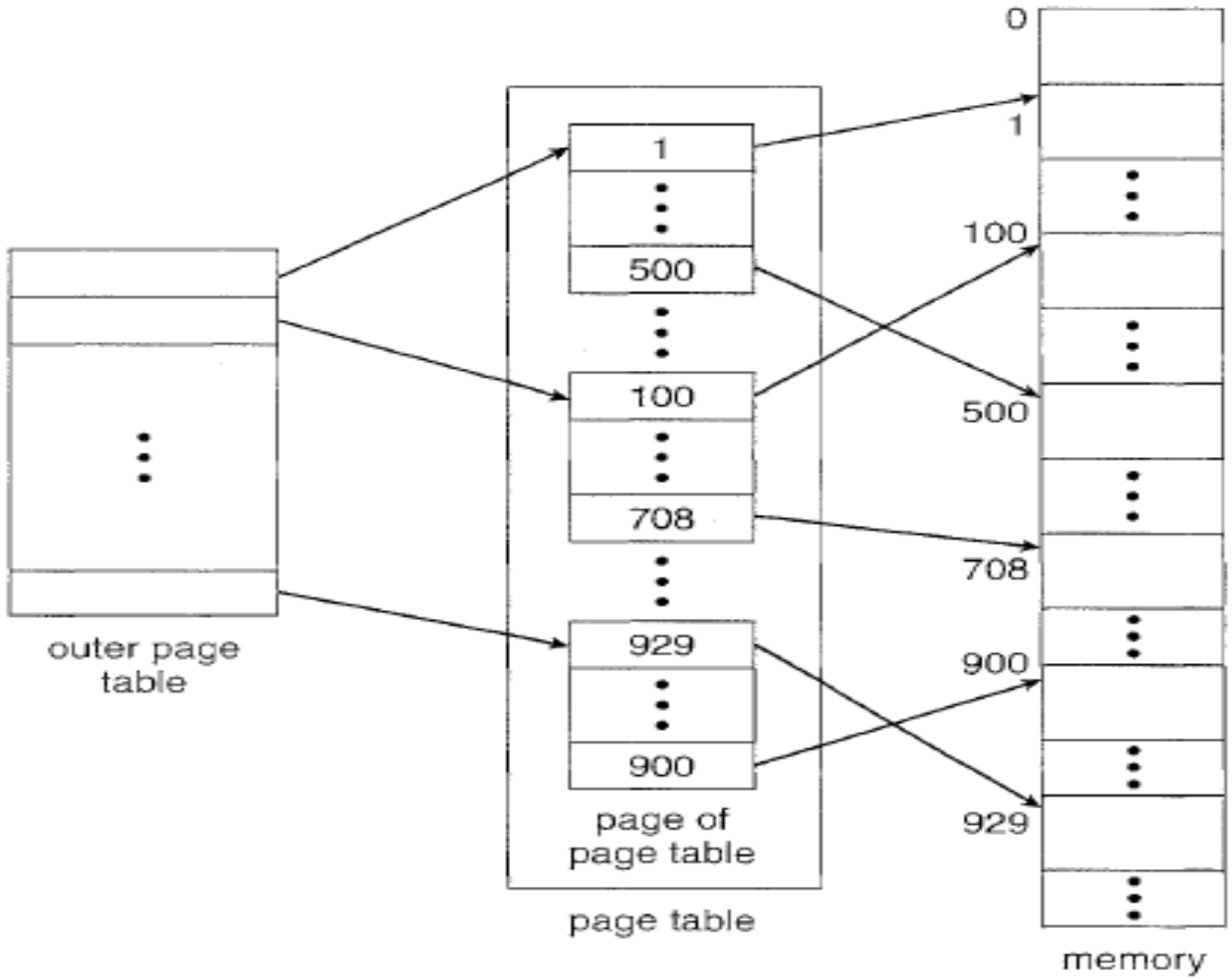




For Example : The VAX architecture supports a variation of two-level paging



The next step would be a four-level paging scheme, where the second-level
outer page table itself is also paged, and so forth.



Hashed Page Tables

- A common approach for handling address spaces larger than 32 bits is to use a with the hash value being the virtual page number. Each entry in the hash table contains a linked list of elements that hash to the same location.

- Each element consists of three fields: (1) the virtual page number, (2) the value of the mapped page frame, and (3) a pointer to the next element in the linked list.

- The algorithm works as follows: The virtual page number in the virtual address is hashed into the hash table. The virtual page number is compared with field 1 in the first element in the linked list.
- If there is a match, the corresponding page frame (field 2) is used to form the desired physical address.
- If there is no match, subsequent entries in the linked list are searched for a matching virtual page number

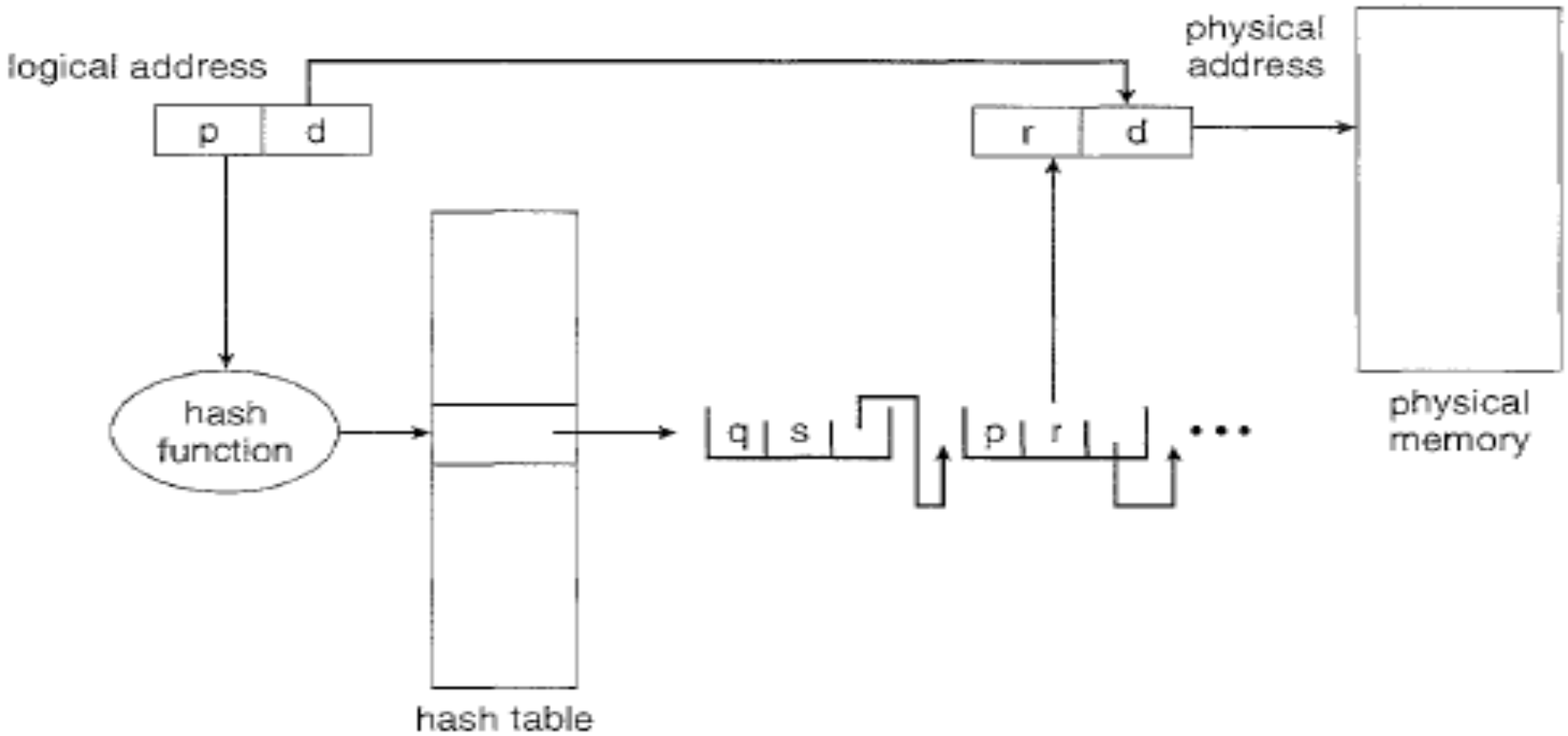


Figure 8.16 Hashed page table.

Mechanism

- To map two dimensional user-defined addresses into one-dimensional physical addresses, a segment table is used.

- Each entry in the segment table has a *segment base* and a *segment limit.*

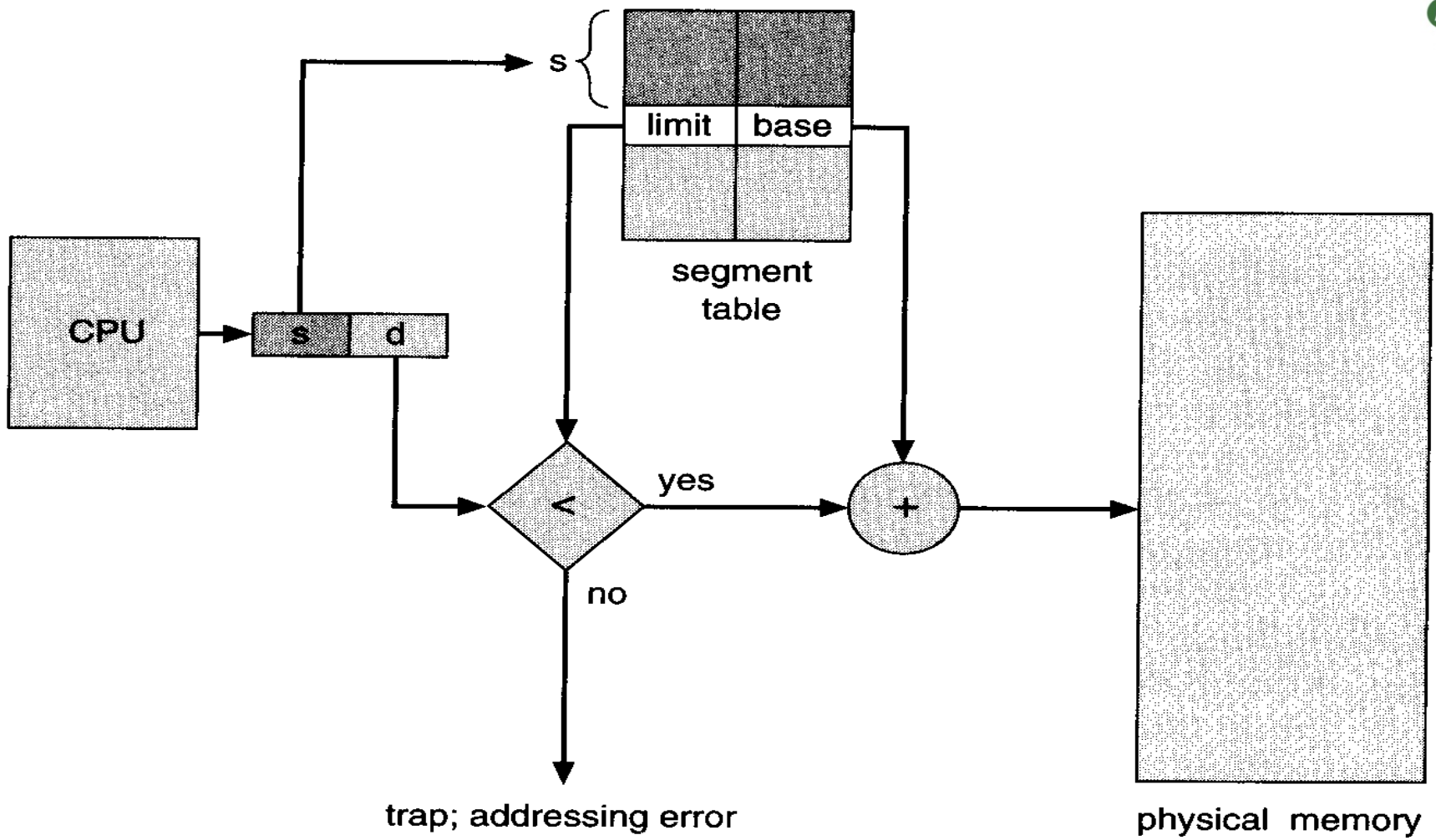
- The segment base contains the starting physical address where the segment resides in memory, whereas the segment limit specifies the length of the segment.

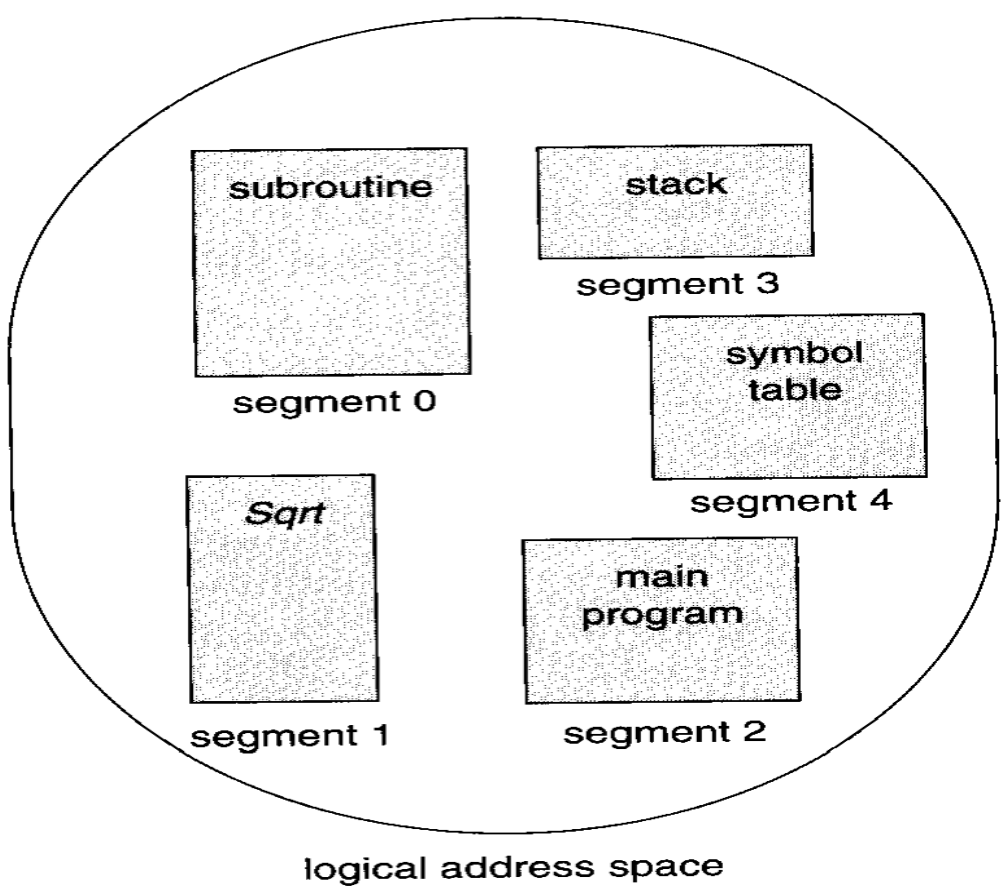
Segmentation

- Segmentation divides logical address space into a collection of segments. Each segment has a name and a length.

- The addresses specify both the segment name and the offset within the segment.
- The user specifies each address by two quantities: a segment name and an offset.
- **< segment-number, offset >.**
- *For simplicity of implementation, segments are numbered and are referred by a segment number, rather than by a segment name.*

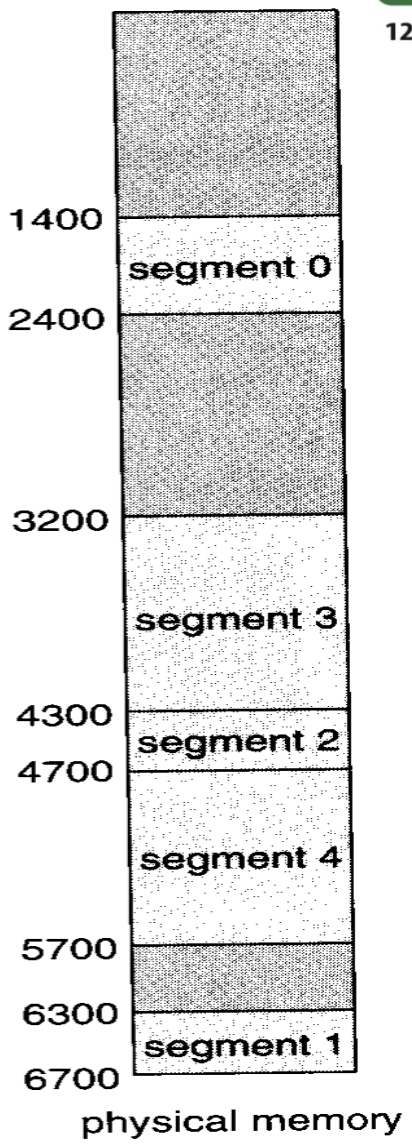
- A logical address consists of two parts: a segment number, *s,* and an offset into that segment, *d.* The segment number is used as an index to the segment table.
- The offset *d* of the logical address must be between 0 and the segment limit. If it is not, we trap an invalid address (logical addressing attempt beyond, end of segment).
- When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.
- The segment table is thus essentially an array of base-limit register pairs.





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

segment table



physical memory

Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for the following logical addresses?

a. 0,430
b. 1,10
c. 2,500
d. 3,400
e. 4,112

- Answer:
- a. 219 + 430 = 649
- b. 2300 + 10 = 2310
- c. illegal reference, trap to operating system
- d. 1327 + 400 = 1727
- e. illegal reference, trap to operating system

Assuming a 1 KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):

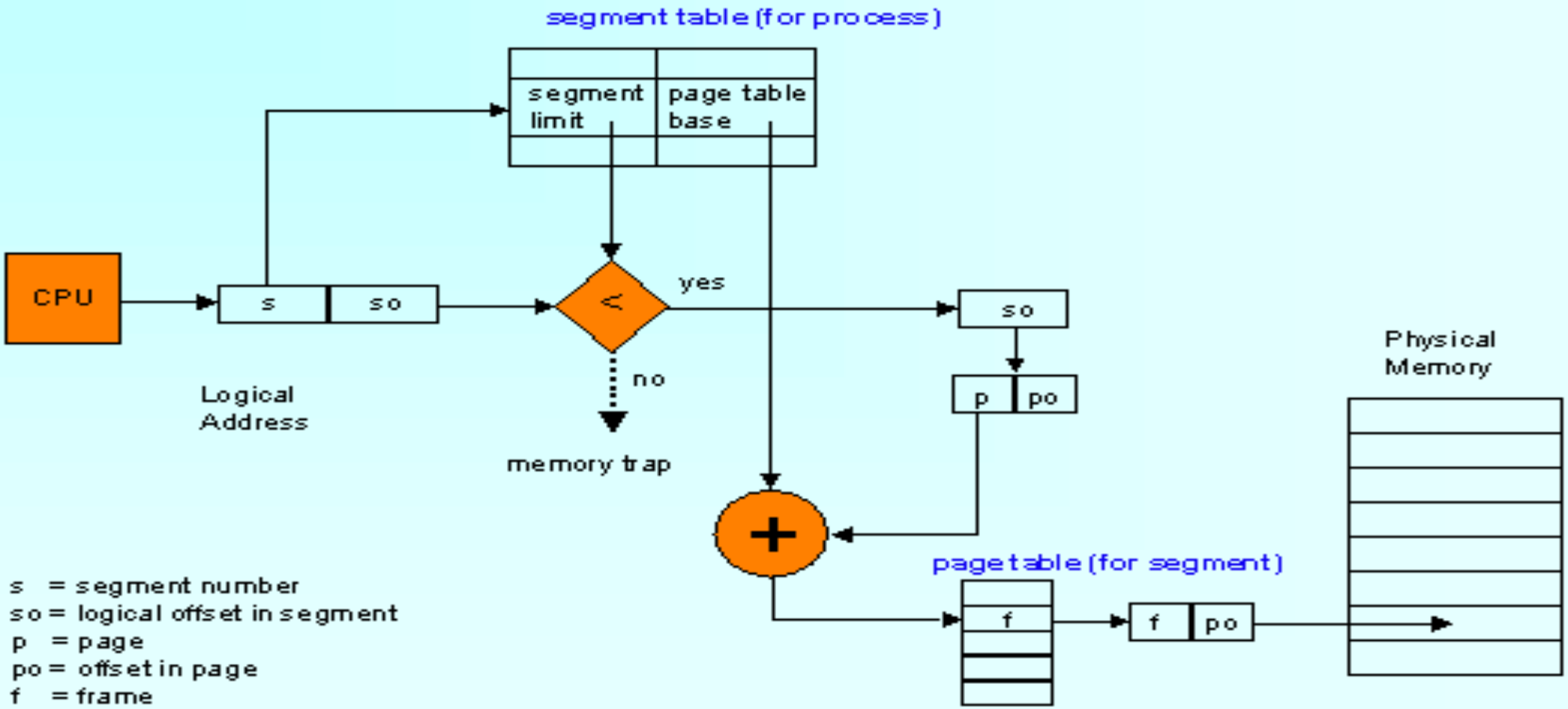
a. 2375
b. 19366
c. 30000
d. 256
e. 16385

Answer:

a. page = 2; offset = 327
b. page = 18; offset = 934
c. page = 29; offset = 304
d. page = 0; offset = 256
e. page = 16; offset = 1

- Paging eliminates external fragmentation and thus provides efficient use of main memory.
- Segmentation is visible to the programmer and has the ability to handle modularity, and support for sharing and protection.
- To combine the advantages of both, some systems are equipped with processor hardware and operating system software to provide both.

Architecture for Segmentation with Paging



- In a combined paging/segmentation system a user address space is broken up into a number of segments.

- Each segment is in turn broken up into a number of fixed-size pages which are equal in length to a main memory frame.

- From the programmer`s point of view, a logical address still consists of a segment number and a segment offset. From the system`s point of view, the segment offset is viewed as a page number and page offset for a page within the specified segment.

- When a particular process is running, a register holds the starting address of the segment table for that process.
- Presented with a virtual address, the processor uses the segment number portion to index into the process segment table to find the page table for that segment.
- Then, the page number portion of the virtual address is used to index the page table and look up the corresponding frame number.
- This combined with the offset portion of the virtual address to produce the desired real address.

Virtual Memory

- A problem how to fit large programs into small amount of physical memory.

- The OS keeps only that part in the memory, which is currently required to execute the process at that time and rest on the hard disk.

- The virtual memory management technique thus allows the execution of the processes which are not completely in the memory.

- The main advantage of this memory management technique is to process the programs, which are larger than the main memory.

- With the use of this technique the programmers don't have the problem of the memory space while developing and implementing a program.

In virtual memory storage, the programmers get illusion of much larger memory than the size of actual memory available in the system.

Advantages:

•The size of user's program would no longer be restricted by the available size of physical memory. Programmers would be able to write their programs without worry.

•Since each user utilizes less physical memory, more programs can stay in the memory, thus increasing CPU utilization

Virtual memory is commonly implemented by

Demand Paging

Demand Paging

•With demand-paged virtual memory, pages are only loaded when they are demanded during program execution;

•. A demand-paging system is similar to a paging system with swapping.

•When a process is to be swapped in, only those necessary pages which are necessary are bring into the memory. Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.

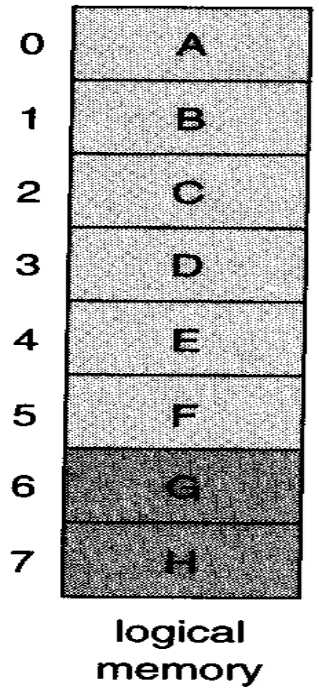
•With this scheme, we need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk

•. The **valid-invalid bit** scheme can be used for this purpose.

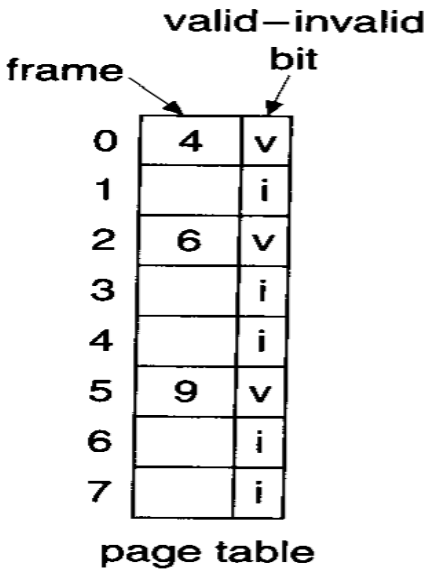
•When this bit is set to "valid" the associated page is both legal and in memory. If the bit is set to "invalid," the page is not legal or is valid but is currently on the disk.

•If the program tries to execute a page that was not swapped in memory, in that case **page fault trap** occurs. Page fault trap is a result of OS failure to bring a valid part of the program into memory.

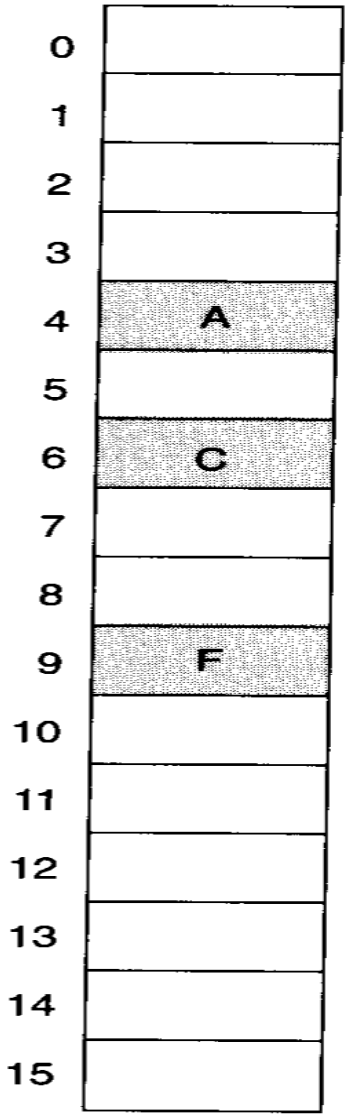
•Whenever a running program experiences a page fault, it just get suspended until the missing page is swapped inside the memory.



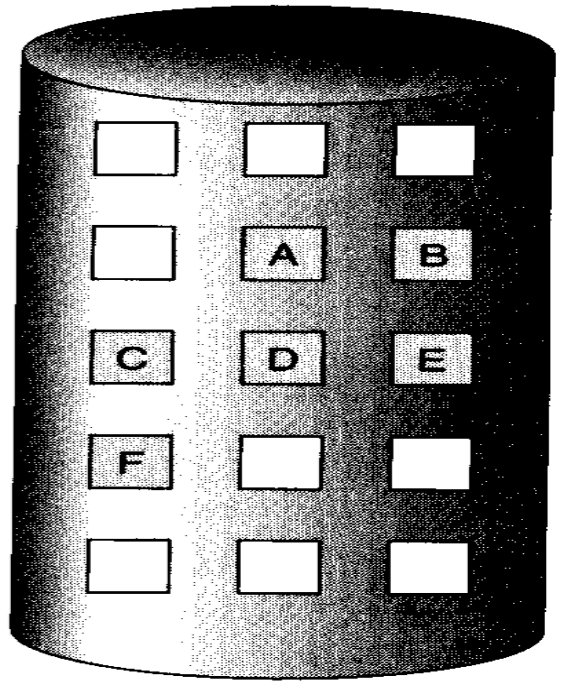
logical memory

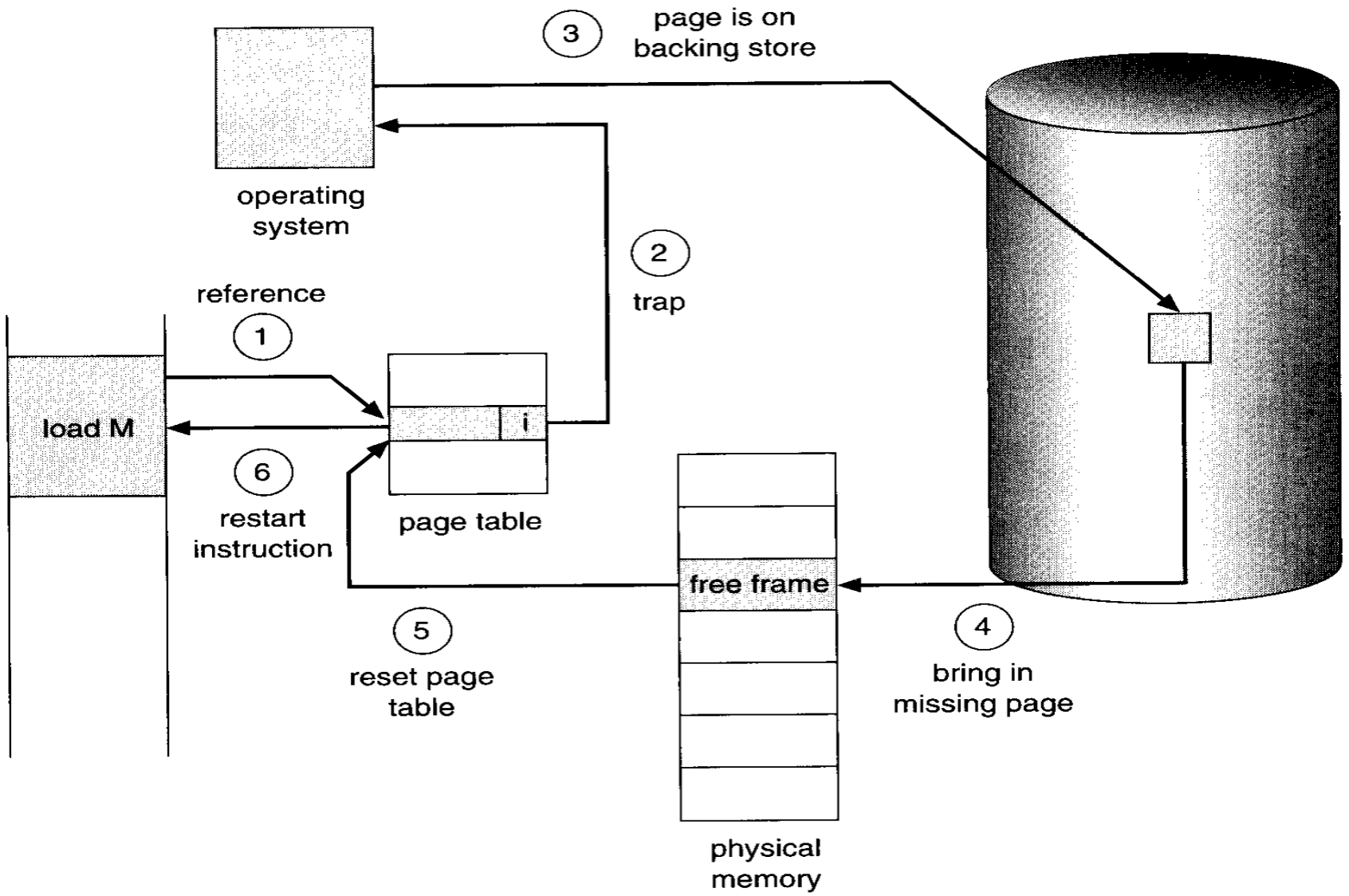


page table



physical memory





Pure demand paging is where no pages are initially brought into memory - a process begins execution by demand paging the first instructions.

The hardware to support demand paging is the same as the hardware for paging and swapping:

• **Page table.** This table has the ability to mark an entry invalid through a valid-invalid bit.

• **Secondary memory.** This memory holds those pages that are not present in main memory. It is known as the swap device, and the section of disk used for this purpose is known as **swap space.**

The procedure for handling this page fault is straight forward:

1)We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

2)If the reference was invalid, we terminate the process. If it was valid, but page is missing, a process to bring that page in should be initiated.

3) We find a free frame (by taking one from the free-frame list).

4) We schedule a disk operation to read the desired page into the

Page Replacement

➢While a user process is executing, a page fault occurs.

➢ The operating system determines where the desired page is residing on the disk but then finds that there are *no* free frames on the free-frame list as all memory is in use.

➢If no frame is free, we find one that is not currently being used and free it. We can free a frame by writing its contents to swap space and changing the page table (and all other tables) to indicate that the page is no longer in memory.

➢ We can now use the freed frame to hold the page for which the process faulted. So the basic steps included in page replacement are given below:

1. Find the location of the desired page on the disk.

2. Find a free frame:

   a. If there is a free frame, use it.
   b. If there is no free frame, use a page-replacement algorithm to select a victim frame.
   c. Write the victim frame to the disk; change the page and frame tables accordingly.

3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Restart the user process.

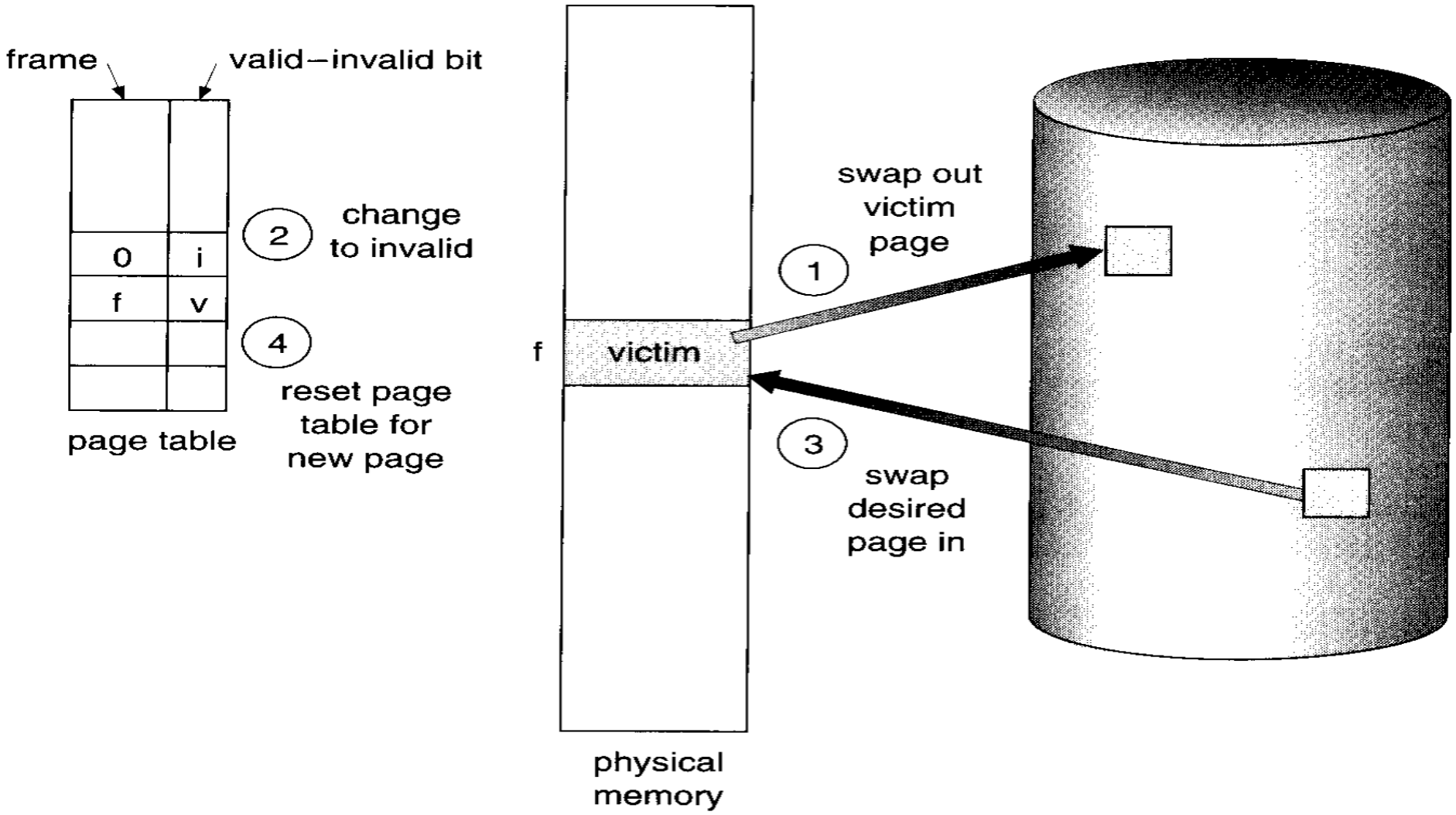
So this process involves two page transfers. This situation doubles the page fault service time and increases the effective access time.

➢To overcome this problem, a bit known as **modify/dirty bit** can be associated with a page.

➢This bit is set whenever any word is written into the page and thus indicates that the page has been modified. Now, when a page is selected to be replaced, its modify bit is checked.

➢If it is set, it implies that this page was modified(when it was in main memory) and therefore must be written back to the disk, otherwise this page should not be written back to the disk, because no changes have been made to it was read from the disk into the memory.

➢The new demanded page(the one, which is swapped-in) can now replace it as it.



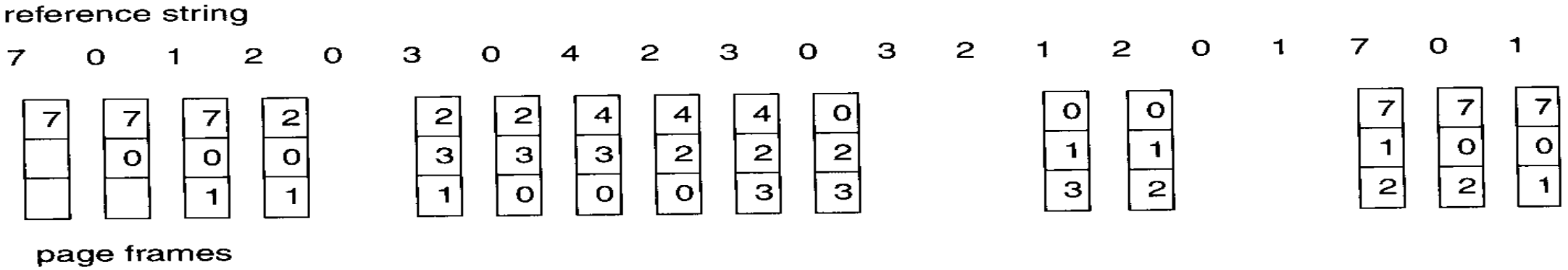
Page Replacement Algorithms

1) FIFO Page Replacement

The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm. A FIFO replacement algorithm notes the time, when that page was brought into memory. When a page must be replaced, the oldest page is chosen. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue For example, we use the reference string

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2,1, 2, 0, 1, 7, 0,1

for a memory with three frames.



Least recent used (LRU) page replacement algorithm →

this algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.

What is the total number of page faults that will occur while processing the page reference string given below-

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 ,3 with 4 page frames. Find number of page faults.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4

Total Page Fault = 6

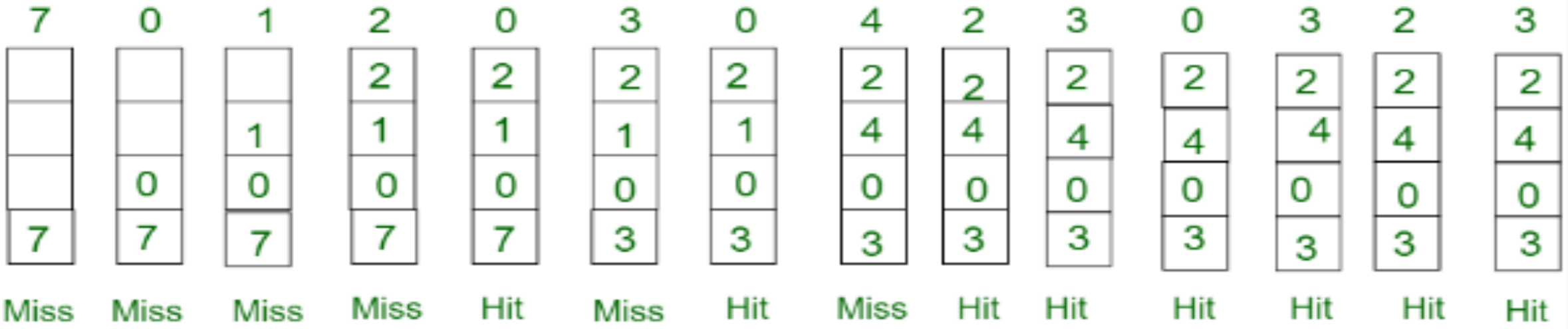
Optimal Page Replacement algorithm →

- This algorithm replaces the page that will not be referred by the CPU in future for the longest time.
- It is practically impossible to implement this algorithm.
- This is because the pages that will not be used in future for the longest time can not be predicted.
- However, it is the best known algorithm and gives the least number of page faults.
- Hence, it is used as a performance measure criterion for other algorithms

EX:- Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0 page frame. Find number of page fault.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4



Total Page Fault = 6

Practice

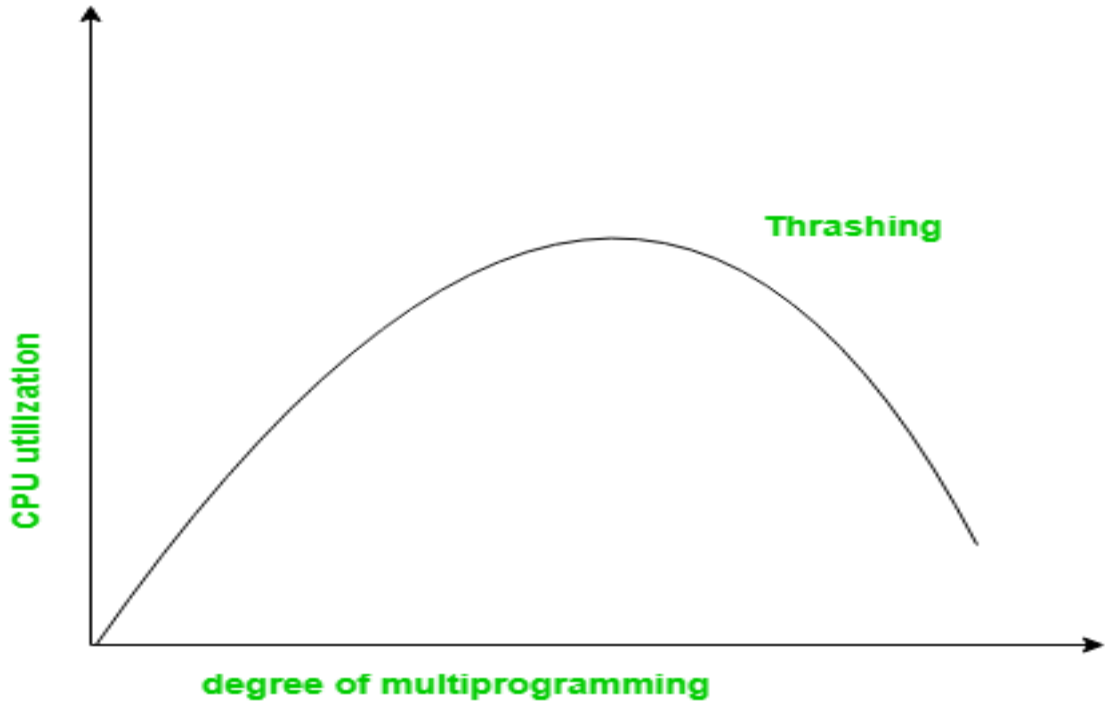
Solve using FIFO,LRU and Optimal

- Reference String: `1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5`
- Number of Frames: 3

- Reference String: `2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2`
- Number of Frames: 3

Thrashing

Thrashing is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.



- If this page fault and then swapping happening very frequently at higher rate, then operating system has to spend more time to swap these pages. This state is called thrashing. Because of this, CPU utilization is going to be reduced.

- The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory thereby increasing the degree of multiprogramming.

- This would result in a further decrease in the CPU utilization triggering a chained **reaction of higher page faults followed by an increase in the degree of multiprogramming, called Thrashing**

I/O communication Technique

- Data transfer between the CPU and I/O devices may be handled in a variety of techniques.
- Some techniques use the CPU as an intermediate path, others transfer the data directly to and from the memory unit.
- Data transferred to and from peripherals is said to be **I/O communication**. Generally, I/O communication is done in 3 ways. These are
  1. Programmed I/O
  2. Interrupt-initiated I/O
  3. Direct Memory Access (DMA)

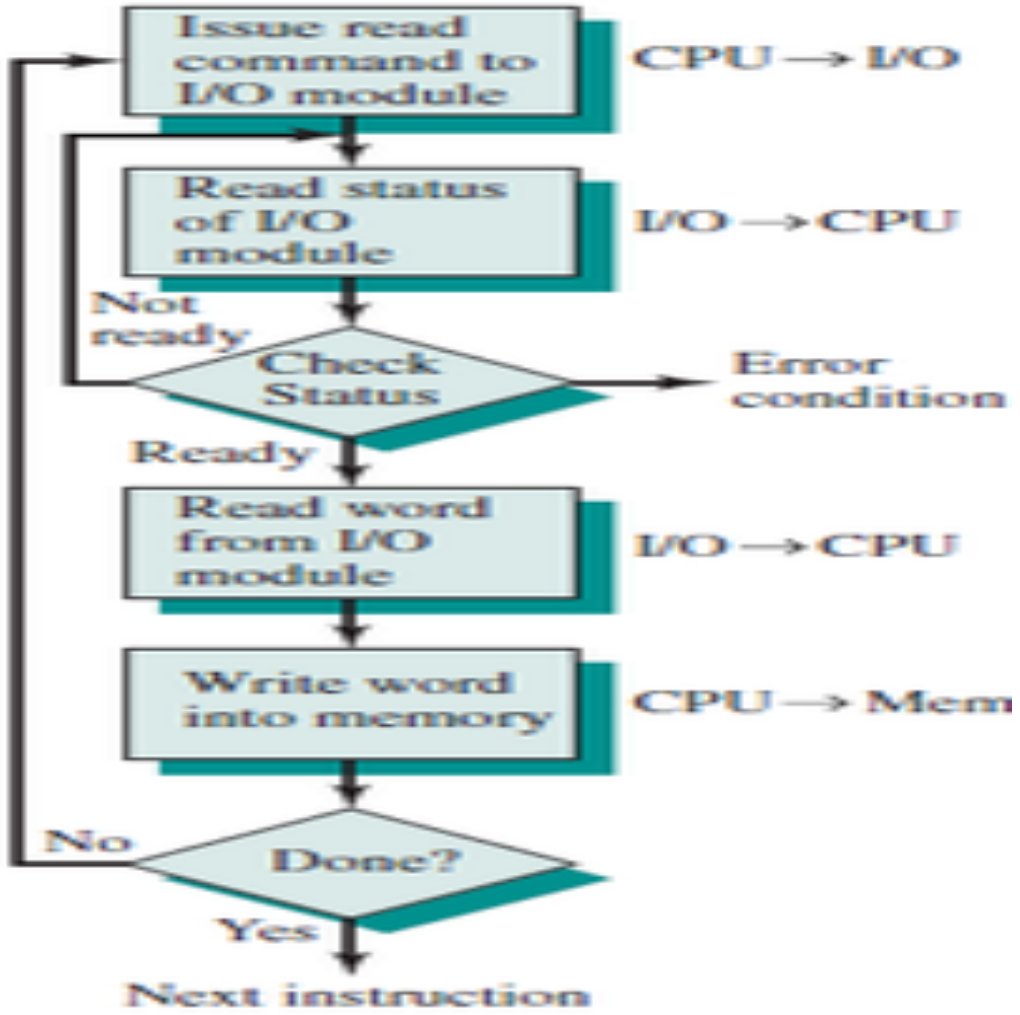
1. Programmed I/O

With programmed I/O, data are exchanged between the processor and the I/O module.

The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data.

When the processor issues a command to the I/O module, it must wait until the I/O operation is complete.

If the processor is faster than the I/O module, this is wasteful of processor time.



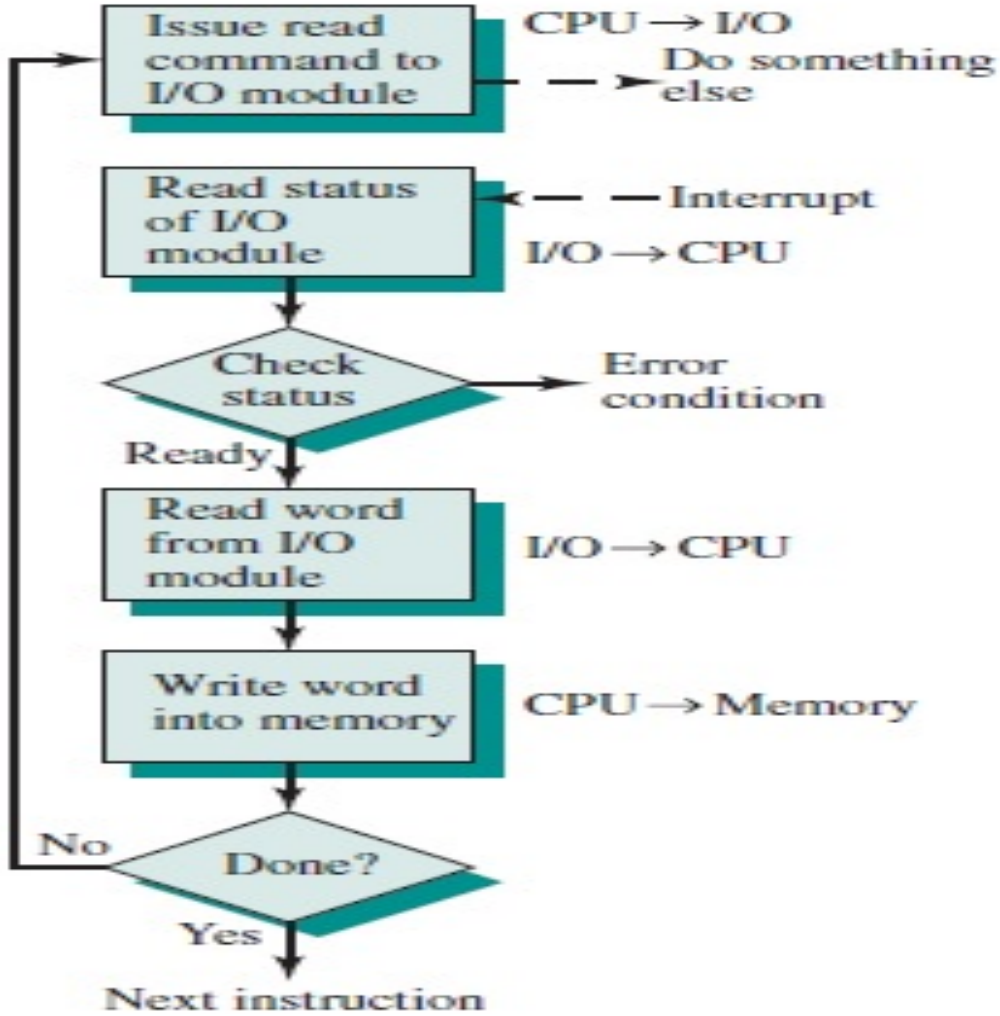
- The processor is executing a program and encounters an instruction relating to I/O operation.
- The processor then executes that instruction by issuing a command to the appropriate I/O module.
- The I/O module will perform the requested action based on the I/O command issued by the processor (READ/WRITE) and set the appropriate bits in the I/O status register.
- The processor will periodically check the status of the I/O module until it find that the operation is complete.

2. Interrupt-initiated I/O

Interrupt I/O is a way of controlling input/ output activity whereby a peripheral or terminal that needs to make or receive a data transfer sends a signal.

This will cause a program interrupt to be set. At a time appropriate to the priority level of the I/O interrupt. Relative to the total interrupt system, the processors enter an interrupt service routine.

The function of the routine will depend upon the system of interrupt levels and priorities that is implemented in the processor.

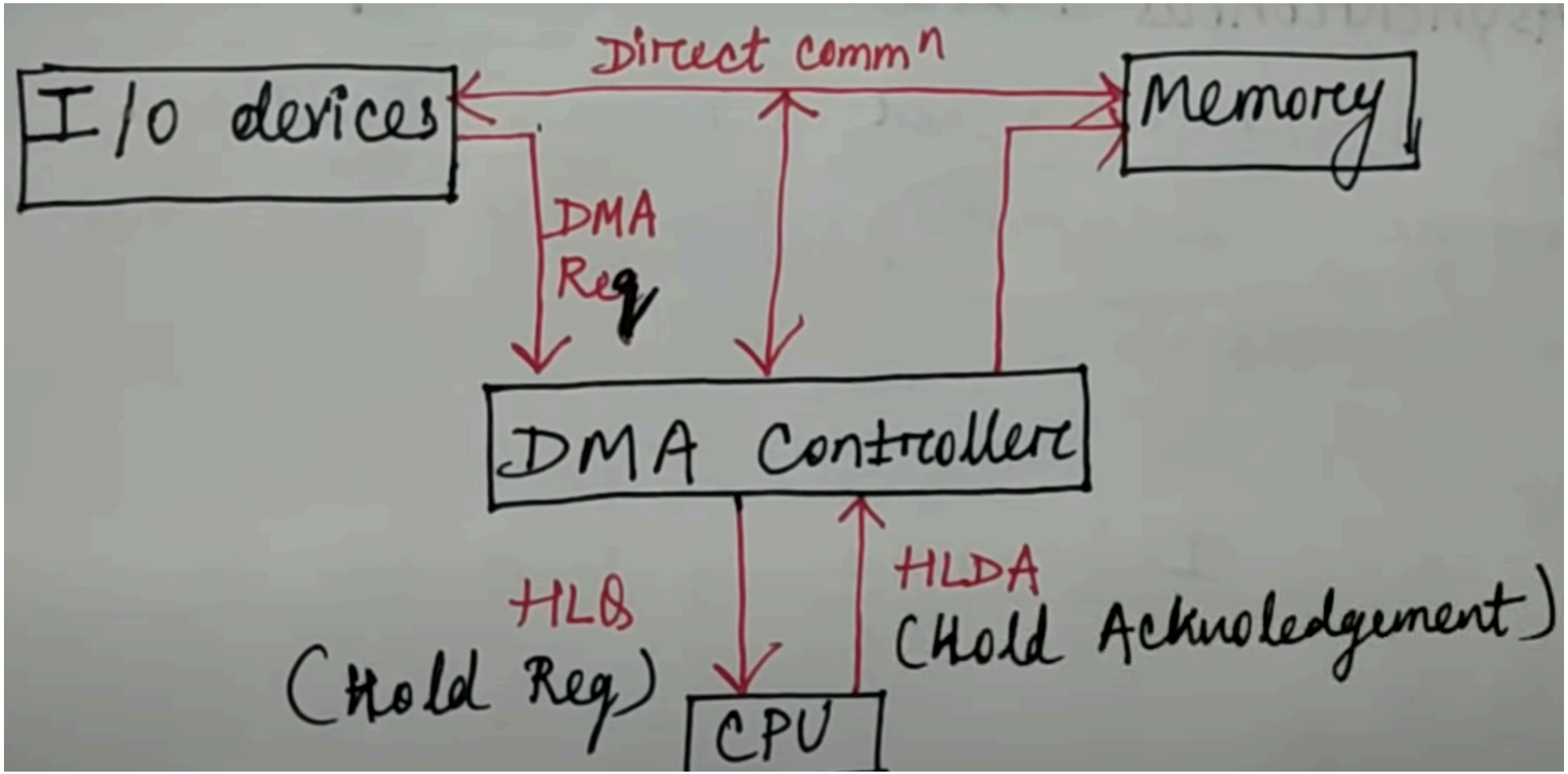


- A device driver initiates an I/O request on behalf of a process.
- The device driver signals the I/O controller for the proper device, which initiates the requested I/O.
- The device signals the I/O controller that is ready to retrieve input, the output is complete or that an error has been generated.
- The CPU receives the interrupt signal on the interrupt-request line and transfer control over the interrupt handler routine.
- The interrupt handler determines the cause of the interrupt, performs the necessary processing and executes a "*return from*" interrupt instruction.
- The CPU returns to the execution state prior to the interrupt being signaled.
- The CPU continues processing until the cycle begins again.

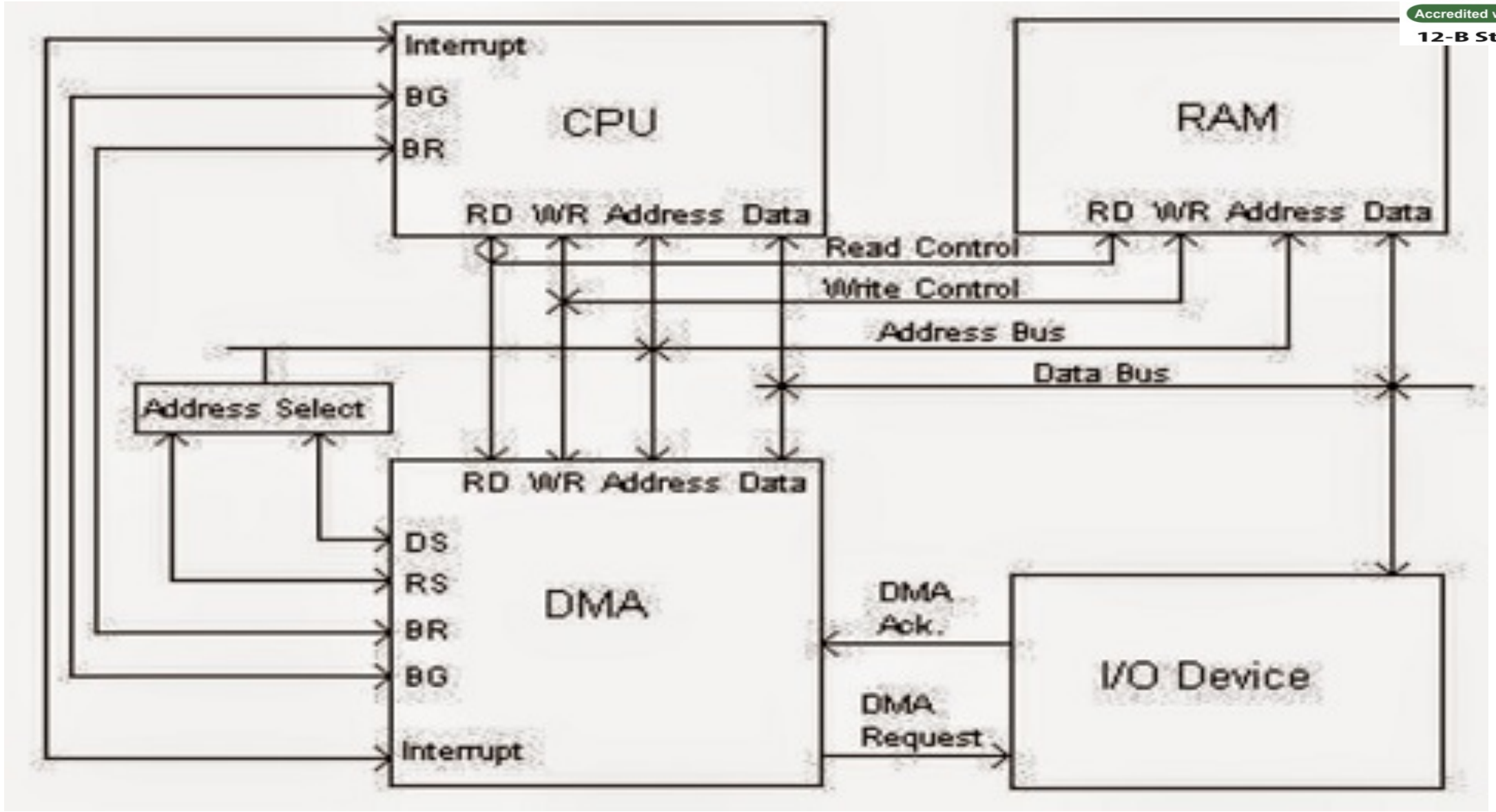
Direct Memory Access (DMA)

- Direct Memory Access is a technique for transferring data within main memory and external device without passing it through the CPU.
- DMA is a way to improve processor activity and I/O transfer rate by taking-over the job of transferring data from processor, and letting the processor to do other tasks.
- This technique overcomes the drawbacks of other two I/O techniques which are the time consuming process when issuing command for data transfer and tie-up the processor in data transfer.
- It is more efficient to use DMA method when large volume of data has to be transferred. For DMA to be implemented, processor has to share its' system bus with the DMA module.

Direct Memory Access (DMA) Working



1. I/O device wants to send a data to memory.
2. I/O device has to send DMA req to DMA controller.
3. DMA sends HLQ to CPU.
4. DMA waits till CPU sends HLDA to DMA
5. CPU- Slave and DMA- Master
6. CPU leaves control over system Bus.
7. CPU is in hold state and DMA has to manage operations over system bus b/w CPU, memory and I/O devices.

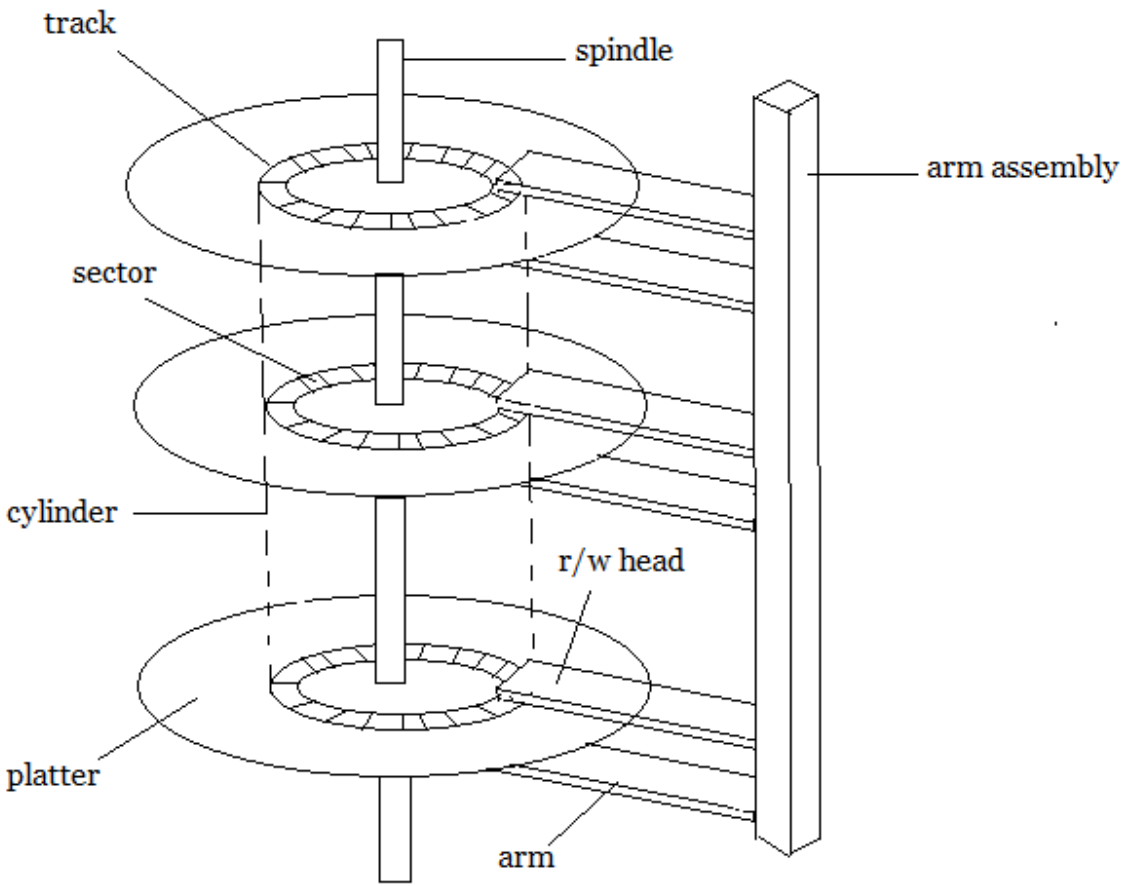


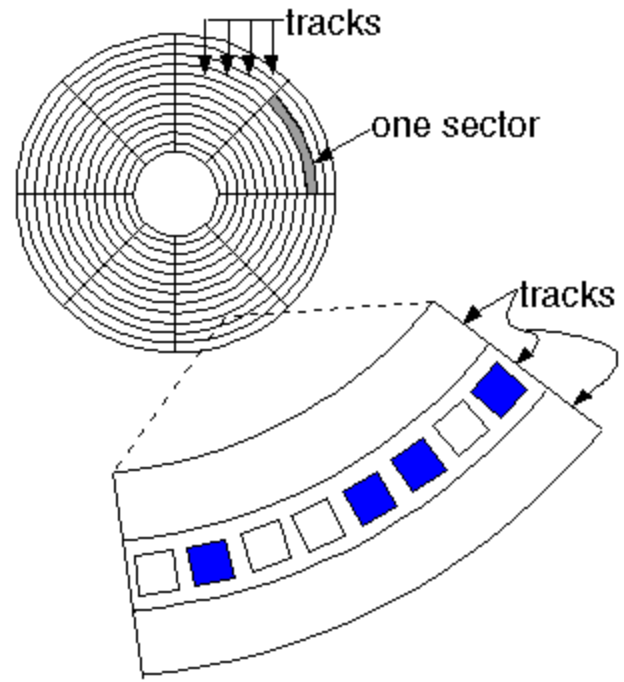
DMA transfer in a computer system

- The DMA request line is used to request a DMA transfer.
- The bus request (BR) signal is used by the DMA controller to request the CPU to relinquish control of the buses.
- The CPU activates the bus grant (BG) output to inform the external DMA that its buses are in a high-impedance state (so that they can be used in the DMA transfer.)
- The address bus is used to address the DMA controller and memory at given location
- The Device select (DS) and register select (RS) lines are activated by addressing the DMA controller.
- The RD and WR lines are used to specify either a read (RD) or write (WR) operation on the given memory location.
- The DMA acknowledge line is set when the system is ready to initiate data transfer.
- The data bus is used to transfer data between the I/O device and memory.
- When the last word of data in the DMA transfer is transferred, the DMA controller informs the termination of the transfer to the CPU by means of the interrupt line.

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of *logical blocks, where the logical block is the smallest unit of* transfer.
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
  - Sector 0 is the first sector of the first track on the outermost cylinder.
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.


track
spindle
arm assembly
sector
cylinder
r/w head
platter
arm


tracks
one sector
tracks

Access Time

- **Seek Time**
  – Time it takes the head assembly to travel to the desired track
  – Seek time may vary depending on the current head location

  Rotational Latency
  – Delay waiting for the rotation of the disk to bring the required disk sector under the head
  – Depends on the speed of the spindle motor

  CLV vs CAV
- **Data Rate**
  – Time to get data off the disk

- Alternatively, the disk rotation speed can stay constant, and the density of bits decreases from inner tracks to outer tracks to keep the data rate constant. This method is used in hard disks and is known as **constant** angular **velocity (CAV).**
- CLV (Constant linear Velocity) – spindle speed (rpm) varies depending on position of the head. So as to maintain constant read (or write) speeds.
- – Used in audio CDs to ensure constant read rate at which data is read from disk
  - CAV (Constant angular velocity) -- spindle velocity is always a constant. Used in hard disks. Easy to engineer.
- – Allows higher read rates because there are no momentum issues

Disk Management

- Low-level formatting, or physical formatting— Dividing a disk into sectors that the disk controller can read and write.
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk.
  - –Partition the disk into one or more groups of cylinders
  - –Logical formatting or “making a file system”.
- Boot block initializes system.– The bootstrap is stored in ROM.–Bootstrap loader program.
- Methods such as sector sparing used to handle bad blocks.

Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of main memory.
- Swap-space can be carved out of the normal file system ,or , more commonly, it can be in a separate disk partition
- .Swap-space management

  – 4.3BSD allocates swap space when process starts; holds text segment(the program) and data segment.

  – Kernel uses swap mapsto track swap-space use.

  – Solaris 2 allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created

Disk Reliability

- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively.
- Disk striping uses a group of disks as one storage unit.
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data.
  - *Mirroring or shadowing keeps duplicate of each disk.*
  - *Block interleaved parity uses much less redundancy.*

Disk Scheduling

Overview

- Introduction
- Various Scheduling algorithms
  - FCFS(First come first serve)
  - SSTF(shortest seek time first)
  - SCAN Scheduling
  - C-SCAN Scheduling
  - LOOK Scheduling

Disc Scheduling

- I/O request issues a system call to the OS.

  - If desired disk drive or controller is available, request is served immediately.

  - If busy, new request for service will be placed in the queue of pending requests.

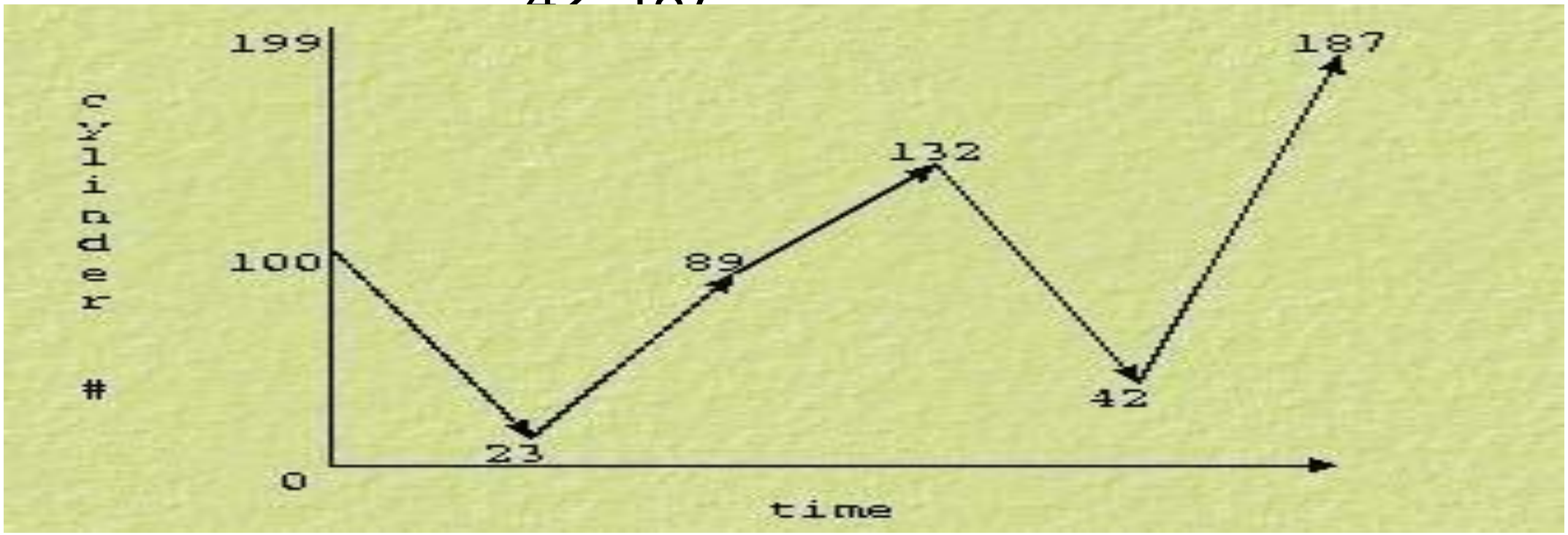
  - When one request is completed, the OS has to choose which pending request to service next.

FCFS Scheduling

- Simplest, perform operations in order requested
- no reordering of work queue
- no **starvation**: every request is serviced
- Doesn't provide fastest service
- Ex: a disk queue with requests for I/O to blocks on cylinders
  23, 89, 132, 42, 187
  With disk head initially at 100

FCFS

23, 89, 132, 42, 187



(100-23)+(89-23)+(132-89)+(132-42)+(187-42)

77+66+43+90+145=421

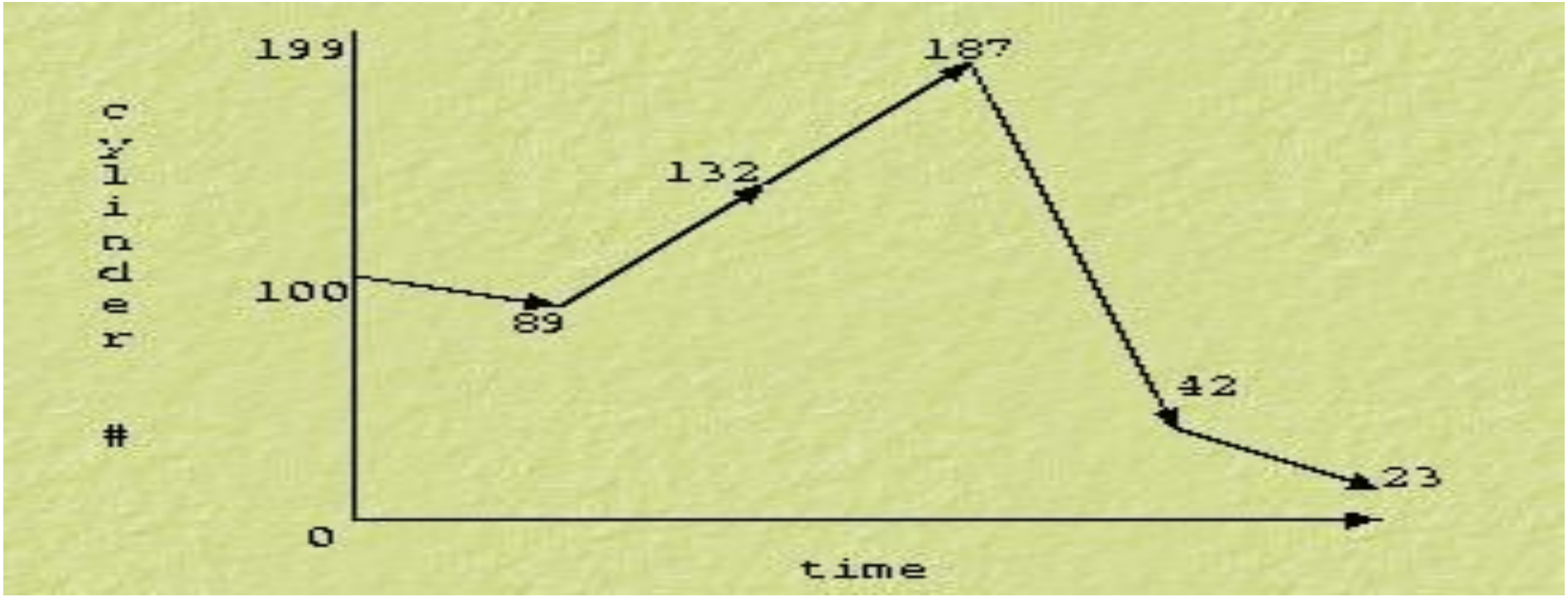
If the requests for cylinders 23 and 42 could be serviced together, total head movement could be decreased substantially.

SSTF Scheduling

- Like SJF, select the disk I/O request that requires the least movement of the disk arm from its current position, regardless of direction
- reduces total seek time compared to FCFS.
- Disadvantages
  - **starvation** is possible; stay in one area of the disk if very busy
  - switching directions slows things down
  - Not the most optimal

SSTF

23, 89, 132, 42, 187



(100-89)+(132-89)+(187-132)+(187-42)+(42-23)

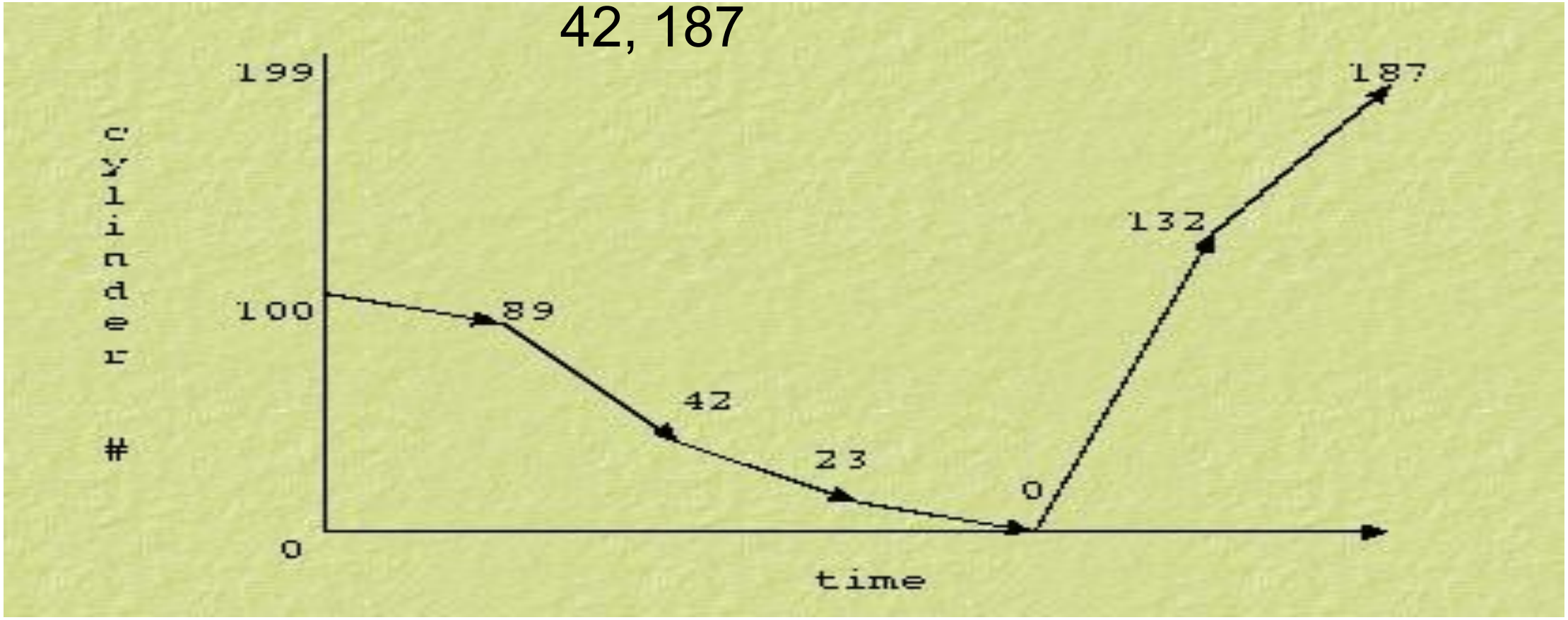
11+43+55+145+19=273

SCAN

- go from the outside to the inside servicing requests and then back from the outside to the inside servicing requests.
- Sometimes called the elevator algorithm.
- Reduces variance compared to SSTF.
- If a request arrives in the queue
  - just in front of the head
  - Just behind

SCAN

23, 89, 132, 42, 187



(100-89)+(89-42)+(42-23)+(23-0)+(132-0)+(187-132)

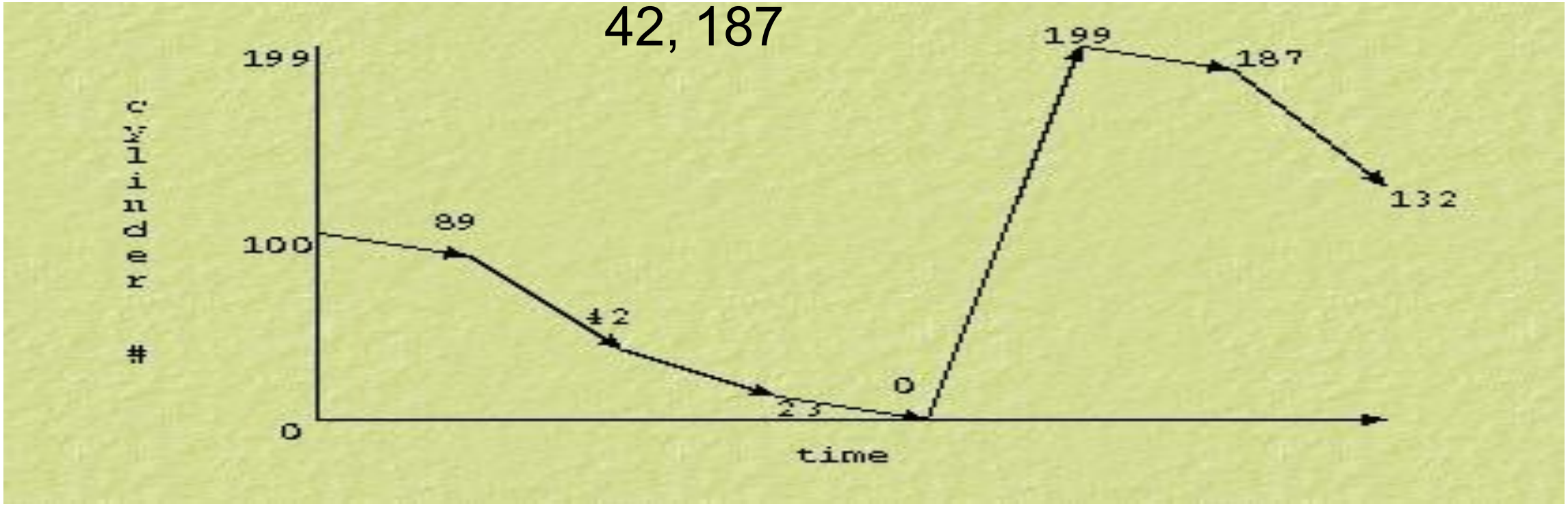
11+47+19+23+132+55=287

C-SCAN

- Circular SCAN
- moves inwards servicing requests until it reaches the innermost cylinder; then jumps to the outside cylinder of the disk without servicing any requests.
- Why C-SCAN?
  - Few requests are in front of the head, since these cylinders have recently been serviced. Hence provides a more uniform wait time.

C-SCAN

23, 89, 132, 42, 187



(100-89)+(89-42)+(42-23)+(23-0)+(199-0)+(199-187)+(187-132)

11+47+19+23+199+12+55=366

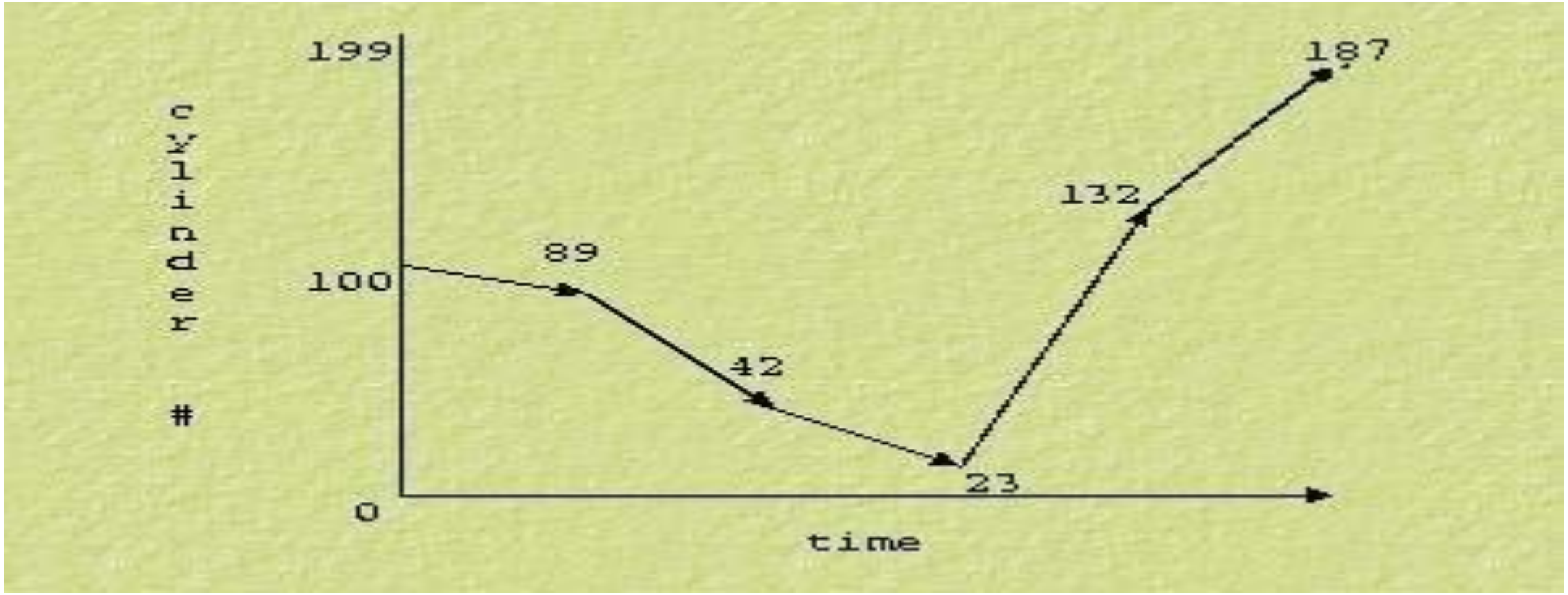
Head movement can be reduced if the request for cylinder 187 is serviced directly after request at 23 without going to the disk 0

LOOK

- like SCAN but stops moving inwards (or outwards) when no more requests in that direction exist

LOOK

23, 89, 132, 42, 187



(100-89)+(89-42)+(42-23)+(132-23)+(187-132)
11+47+19+109+55=241

Compared to SCAN, LOOK saves going from 23 to 0 and then back.

Most efficient for this sequence of requests

C-LOOK

23, 89, 132, 42, 187

Sequence will be 100,89,42,23,187,132,

(100-89)+(89-42)+(42-23)+(187-23)+(187-132)
=296

Which one to choose?

- Performance depends on number and type of requests.
- SSTF over FCFS.
- SCAN, C-SCAN for systems that place a heavy load on the disk, as they are less likely to cause starvation.
- Default algorithms, SSTF or LOOK

Practice Problems

1.A disk with 1000 cylinders, numbers 0 to 999, compute the number of tracks the disk arm must move to satisfy all the request in the disk queue. Assume the last request serviced was at track 345 and the head is moving towards track 0.The queue in FIFO order contains request for the following tracks: 123, 874, 692, 475, 105, 376.
Perform the computation for the following
scheduling algorithms:
(i) FIFO, (ii) SSTF, (iii) SCAN (iv)LOOK.

Practice Problems

2.A disk with 200 cylinders, numbers 0 to 199, compute the number of tracks the disk arm must move to satisfy all the request in the disk queue. Assume the last request serviced was at track 53 and the head is moving towards track 0.The queue in FIFO order contains request for the following tracks: 98 183 37 122 14 124 65 67
Perform the computation for the following
scheduling algorithms:
(i) FIFO, (ii) SSTF, (iii) SCAN (iv)LOOK (v)CSCAN.

Practice Problems

3.A disk with 200 cylinders, numbers 0 to 199, compute the number of tracks the disk arm must move to satisfy all the request in the disk queue. Assume the last request serviced was at track 50 and the head is moving towards track 0.The queue in FIFO order contains request for the following tracks: 82 170 43 140 24 16 190

Perform the computation for the following scheduling algorithms:

(i) FIFO, (ii) SSTF, (iii) SCAN (iv)LOOK (v)CSCAN (vi)CLOOK

Practice Problems

4. Consider a disk with 4 platters, 2 surfaces per platter, 1000 tracks per surface, 50 sectors per track, and 512 bytes per sector. What is the disk capacity?

5. A storage system consists of 10 disk drives, each with 6 platters, 4 surfaces per platter, 2000 tracks per surface, 100 sectors per track, and 512 bytes per sector. What is the total capacity of the storage system? If 7 disk drives are currently filled with data, what is the overall disk utilization percentage?

6. Consider a disk system having 60 cylinders. Disk requests are received by a disk drive for cylinders 10, 22, 20, 2, 40, 6, and 38, in that order. Assuming the disk head is currently at cylinder 20, what is the time taken to satisfy all the requests if it takes 2 milliseconds to move from one cylinder to an adjacent one and Shortest Seek Time First (SSTF) algorithm is used ?

Practice Problems

4. Consider a disk with 4 platters, 2 surfaces per platter, 1000 tracks per surface, 50 sectors per track, and 512 bytes per sector. What is the disk capacity?(**204.8 MB**)

5. A storage system consists of 10 disk drives, each with 6 platters, 4 surfaces per platter, 2000 tracks per surface, 100 sectors per track, and 512 bytes per sector. What is the total capacity of the storage system? If 7 disk drives are currently filled with data, what is the overall disk utilization percentage? (**70%)**

6. Consider a disk system having 60 cylinders. Disk requests are received by a disk drive for cylinders 10, 22, 20, 2, 40, 6, and 38, in that order. Assuming the disk head is currently at cylinder 20, what is the time taken to satisfy all the requests if it takes 2 milliseconds to move from one cylinder to an adjacent one and Shortest Seek Time First (SSTF) algorithm is used ?(**120 milliseconds**)

Practice Problems

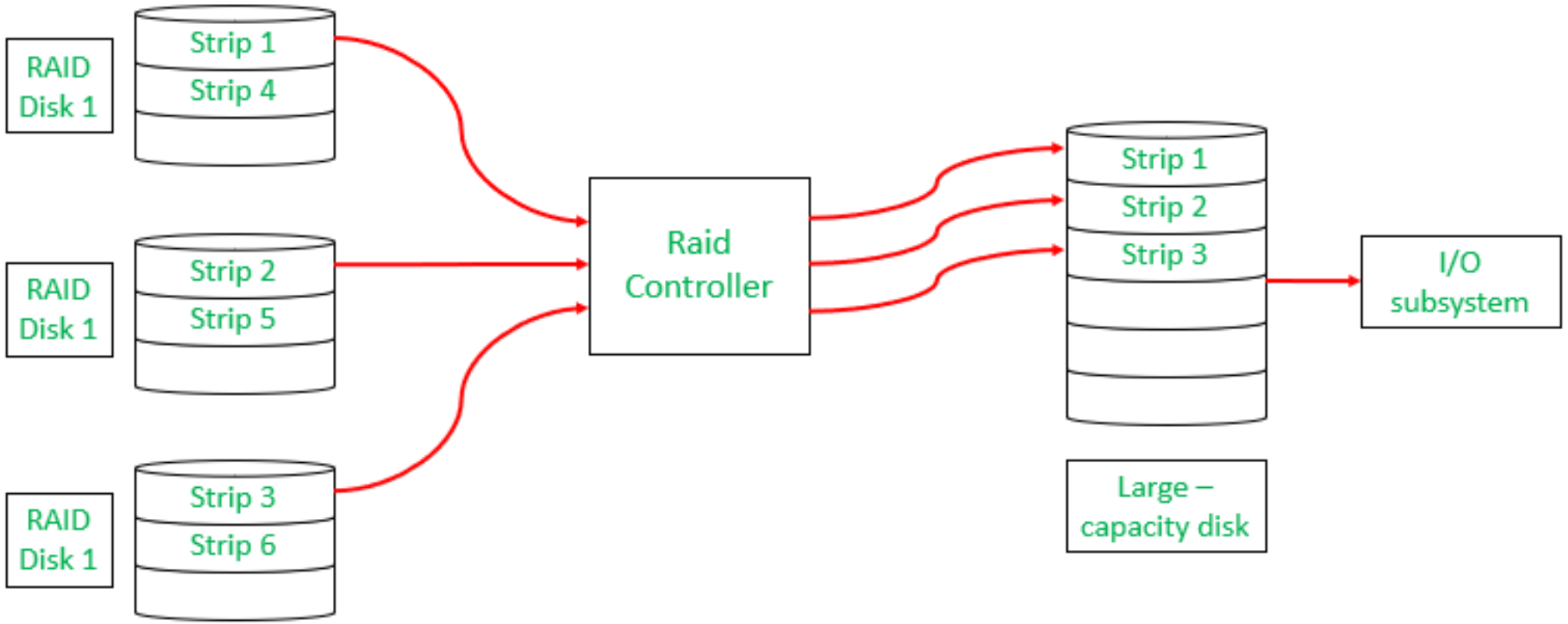
7. Consider a disk where each sector contains 512 bytes and there are 400 sectors per track and 1000 tracks on the disk. If disk is rotating at speed of 1500 RPM, find the total time required to transfer file of size 1 MB. Suppose seek time is 4ms?

8. Consider a system with 8 sector per track and 512 bytes per sector. Assume that disk rotates at 3000 rpm and average seek time is 15ms standard. Find total time required to transfer a file which requires 8 sectors to be stored. a) Assume contiguous allocation b) Assume Non- contiguous allocation

Practice Problems

7. Consider a disk where each sector contains 512 bytes and there are 400 sectors per track and 1000 tracks on the disk. If disk is rotating at speed of 1500 RPM, find the total time required to transfer file of size 1 MB. Suppose seek time is 4ms?(**229 MB**)

RAID



RAID

- RAID stands for Redundant Array of Independent Disks.
- Combines multiple physical drives into a single logical unit to enhance performance and fault tolerance
- Data is distributed across multiple disks.
- Redundancy is added to ensure data recovery in case of disk failure.
- Components are RAID Controller (hardware/software) and Array of physical drives.

RAID

Common RAID Levels:

RAID 0: Striping (Performance).

RAID 1: Mirroring (Redundancy).

RAID 5: Striping with Parity (Balanced).

RAID 6: Double Parity (High Fault Tolerance).

RAID 10: Combination of RAID 1 + RAID 0 (Performance + Redundancy)

RAID

Advantages of RAID

- Improved read/write speeds.
- Fault tolerance and redundancy.
- Scalability for larger storage needs.

Limitations of RAID

- Not a Backup: RAID protects against hardware failure but not data loss due to user error or malware.
- Cost: Higher implementation costs for certain levels.
- Complexity: Can be complex to set up and manage.

File System

File System

- A File is a collection of related information that is recorded on secondary storage.

- files represent programs (both source and object forms) and data.

- A file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user.

- NTFS(Window),FAT(DOS),UNIX(Unix File system),Linux (Extended File System,)Big Data(ZFS File system)

File Attributes

- **Name** – only information kept in human-readable form
- **Extension** (Type)
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size

- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure

File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider **text file, source file, executable file**

File Operations

- File is an **abstract data type**
- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file – seek(Unix Command)**

- **Delete**
- **Truncate**
- ***Open(F_i)*** – search the directory structure on disk for entry F_i, and move the content of entry to memory
- ***Close (F_i)*** – move the content of entry F_i in memory to directory

- structure on disk

File Types – Name, Extension

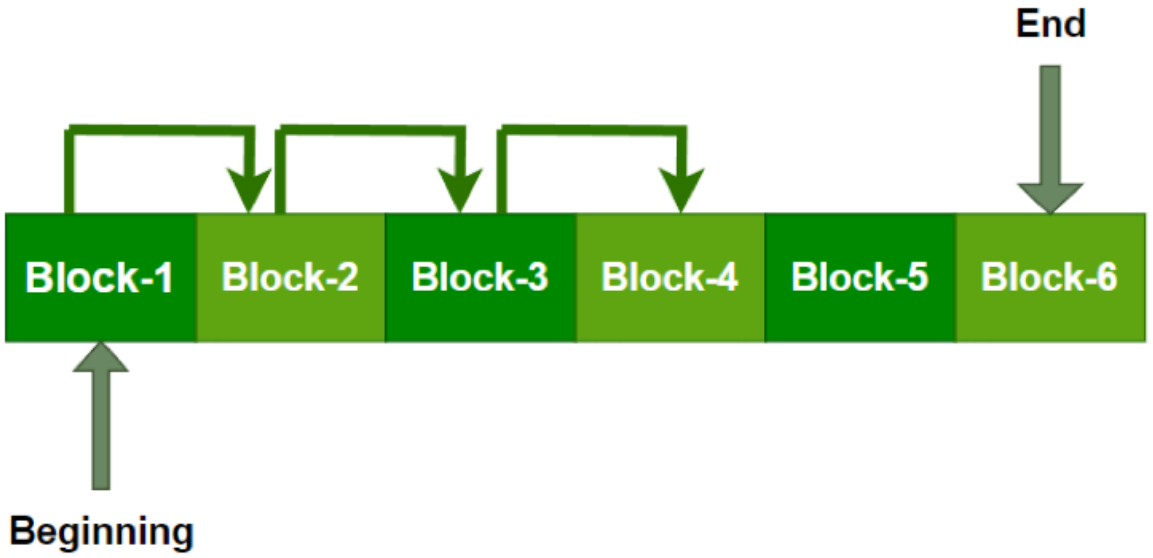
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access Methods

1. Sequential Access

- Information in the file is processed in order, one record after the other.
- for example, editors and compilers usually access files in this fashion.
- Reads and writes make up the bulk of the operations on a file.

Sequential Access Method



- A read operation—*read next*—reads the next portion of the file.

- Similarly, the write operation—*write next*—appends to the end of the file and gives the new end of file.

- Such a file is not able to skip forward or backward *n* records for some integer *n*.

2. Direct Access

- **Direct access** (or **relative** access). based on a file is made up of fixed length **logical records**

- Allow programs to read and write records rapidly in no particular order.

- A disk model of a file, since disks allow random access to any file block.
-
- There are no restrictions on the order of reading or writing for a direct-access file.

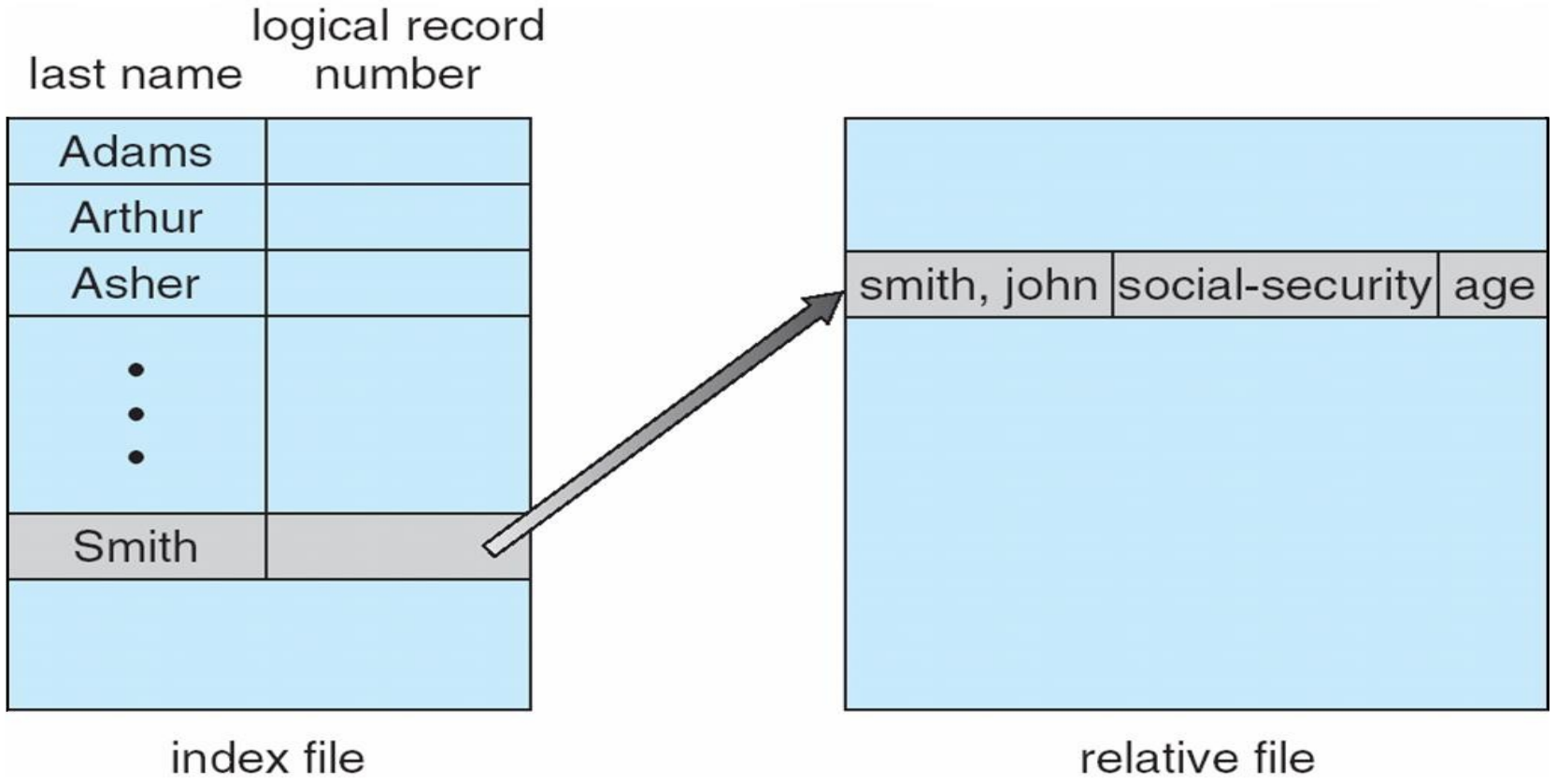
- Direct-access files are of great use for immediate access to large amounts of information.
- we can compute block contains the answer and then read that block directly to provide the desired information.
- For the direct-access method, the file operations must be modified to include the block number as a parameter.
- we have *read n,* where *n* is the block number, rather than *read next,* and *write n* rather than *write next.*

Other Access Methods

- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)

- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS
- VMS operating system provides index and relative files as another example (see next slide)

Example of Index and Relative Files



Allocation Methods

It is a method which tells how to allocate space to the files so that disk space is utilized effectively and files can be accessed quickly.

There are three major methods of allocating disk space are:

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

1) Contiguous Allocation

Contiguous allocation of a file is defined by the disk address and length of the first block. If the file is *n* blocks long and starts at location *b,* then it occupies blocks *b, b* + 1, *b* + *2,* ..., *b* + *n* – *1* . no head movement

Advantages

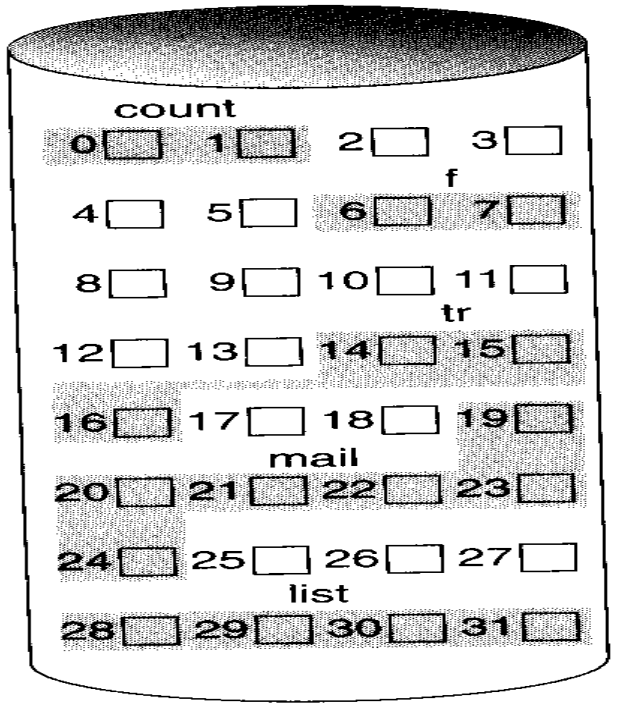
i) A file that has been allocated contiguously is easy to access.

ii) Both, sequential and direct access can be supported by contiguous allocation.

Disadvantage

i) Finding space for a new file.

ii) The general dynamic storage-allocation problem which involves how to satisfy a request of size *n* from a list of free holes.



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Fig: Contiguous allocation of Disk space

2) Linked Allocation

each file is a linked list of disk blocks The directory contains a pointer to the first and last blocks of the file.

For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25. Each block contains a pointer to the next block.

Disadvantage

i) The major problem is that it can be used effectively only for sequential-access files.
ii) space required for the pointers
iii) Reliability.

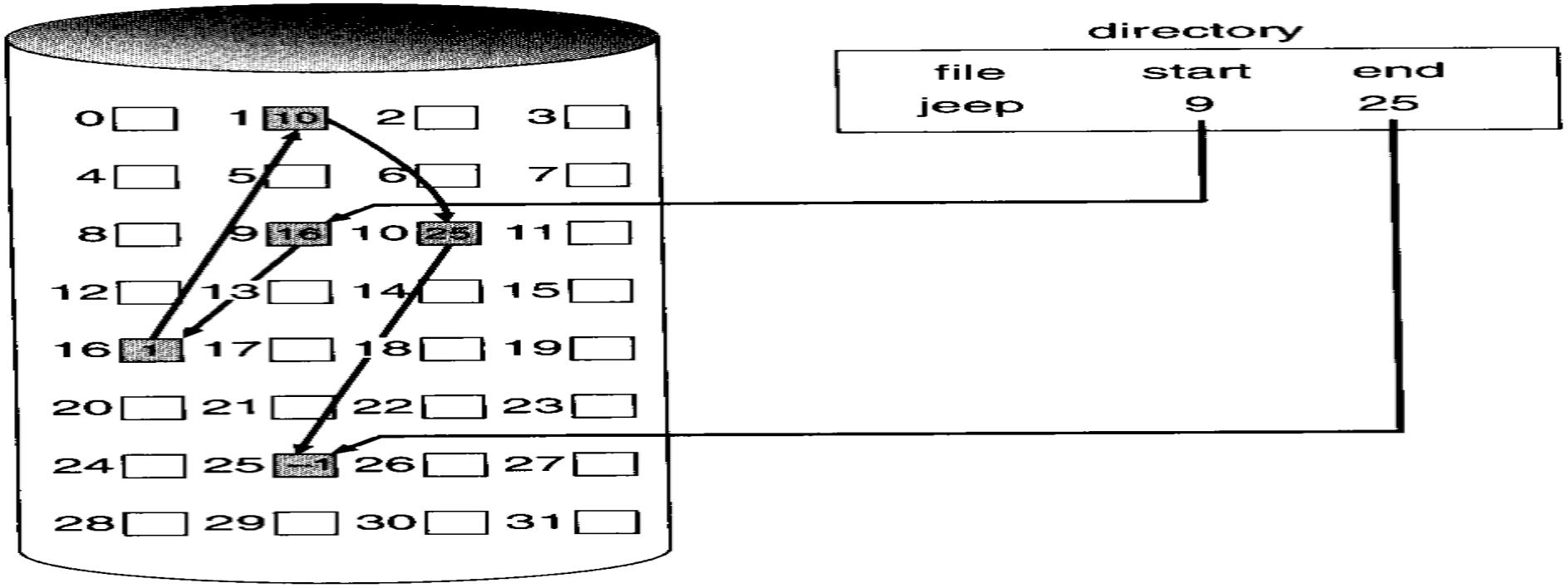


Fig: Linked allocation of Disk space

Indexed Allocation

- Each file has its own index block, which is an array of disk-block addresses.

- The *i*th entry in the index block points to the *i*th block of the file. The directory contains the address of the index block.

- To find and read the ith block, we use the pointer in the *i*th index-block entry.

- When the file is created, all pointers in the index block are set to *nil*. When the *i*th block is first written, a block is obtained from the free-space manager, and its address is put in the *i*th index-block entry.

Advantage

i) Indexed allocation supports direct access, without suffering from external fragmentation because any free block on the disk can satisfy a request for more space.

Disadvantage

i) Indexed allocation does suffer from wasted space as the pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

File-allocation table (FAT).

- This simple but efficient method of disk-space allocation is used by the MS-DOS and OS/2 operating systems.
- A section of disk at the beginning of each volume contains this table. The table has one entry for each disk block and is indexed by block number(like linked list).
- The directory entry contains the block number of the first block of the file. The table entry indexed by that block number contains the block number of the next block in the file.
- This chain continues until the last block, which has a special end-of-file value as the table entry.

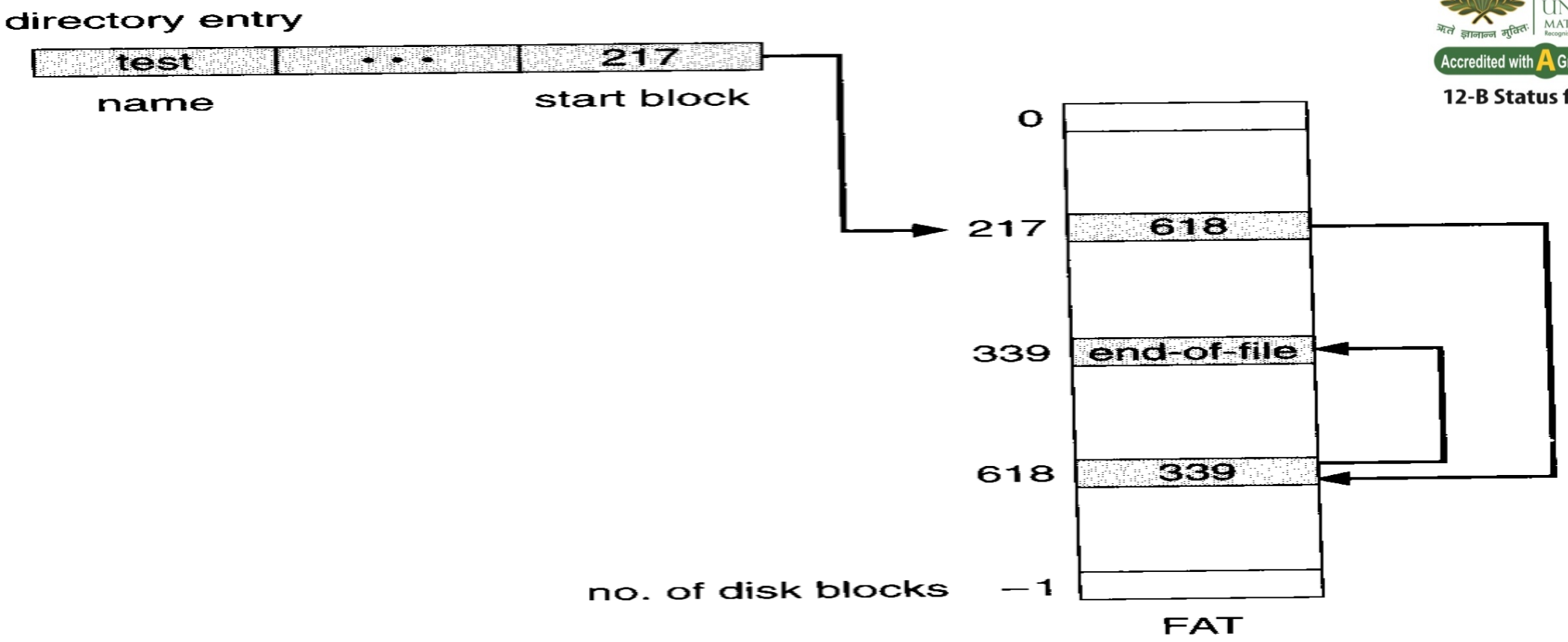


Fig: File Allocation Table

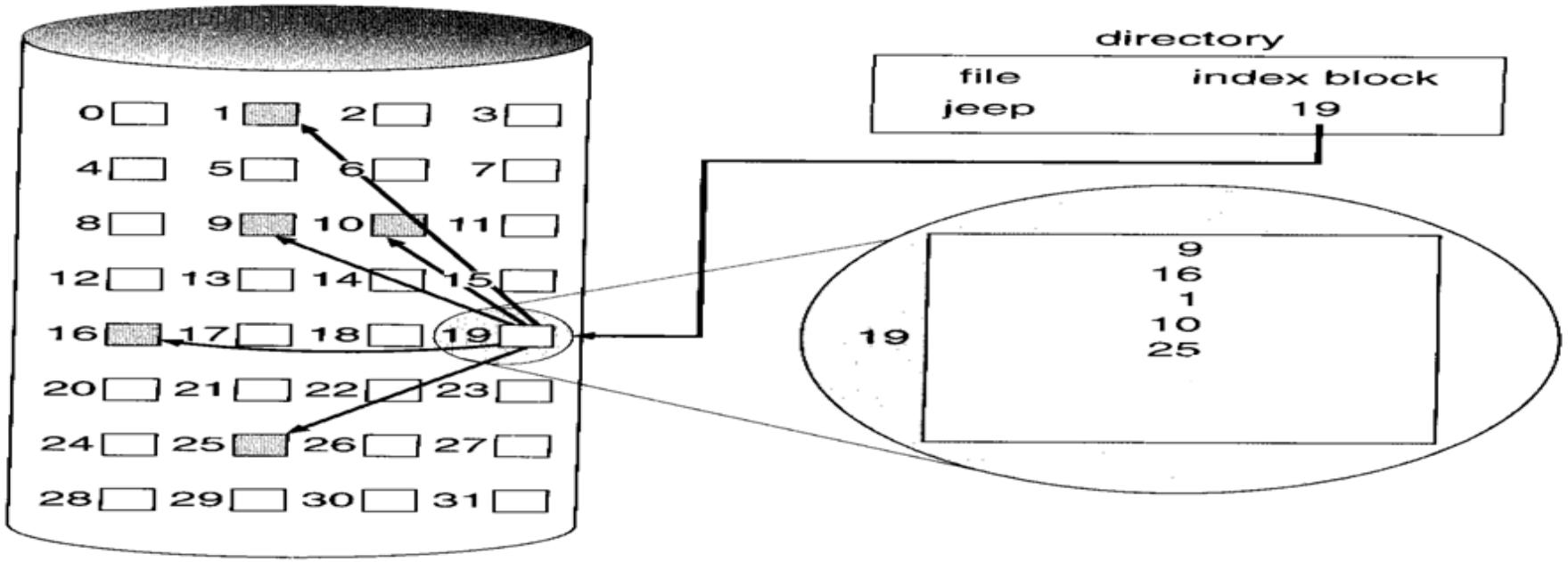
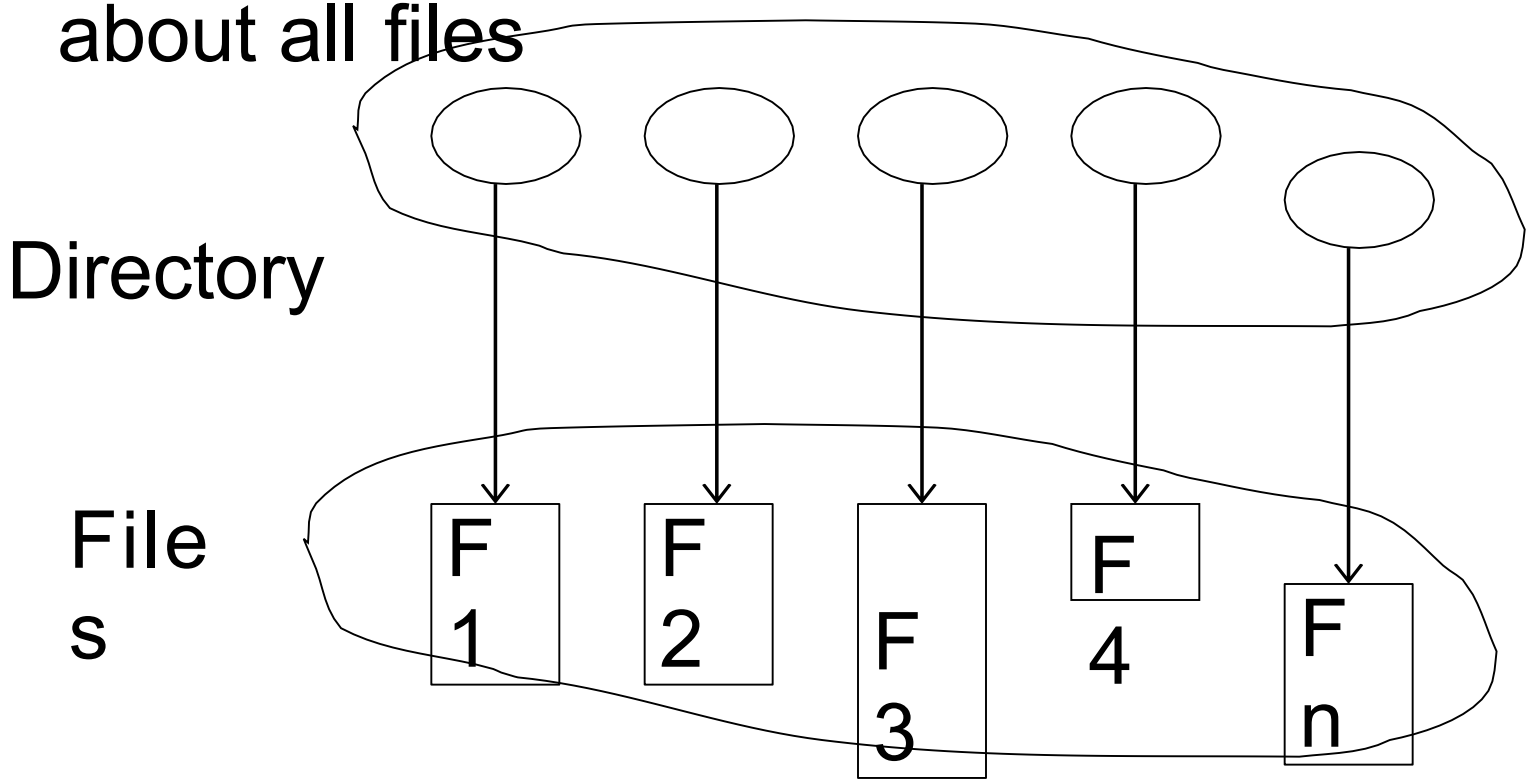


Fig: Indexed allocation of disk space

Directory Structure

- A collection of nodes containing information about all files

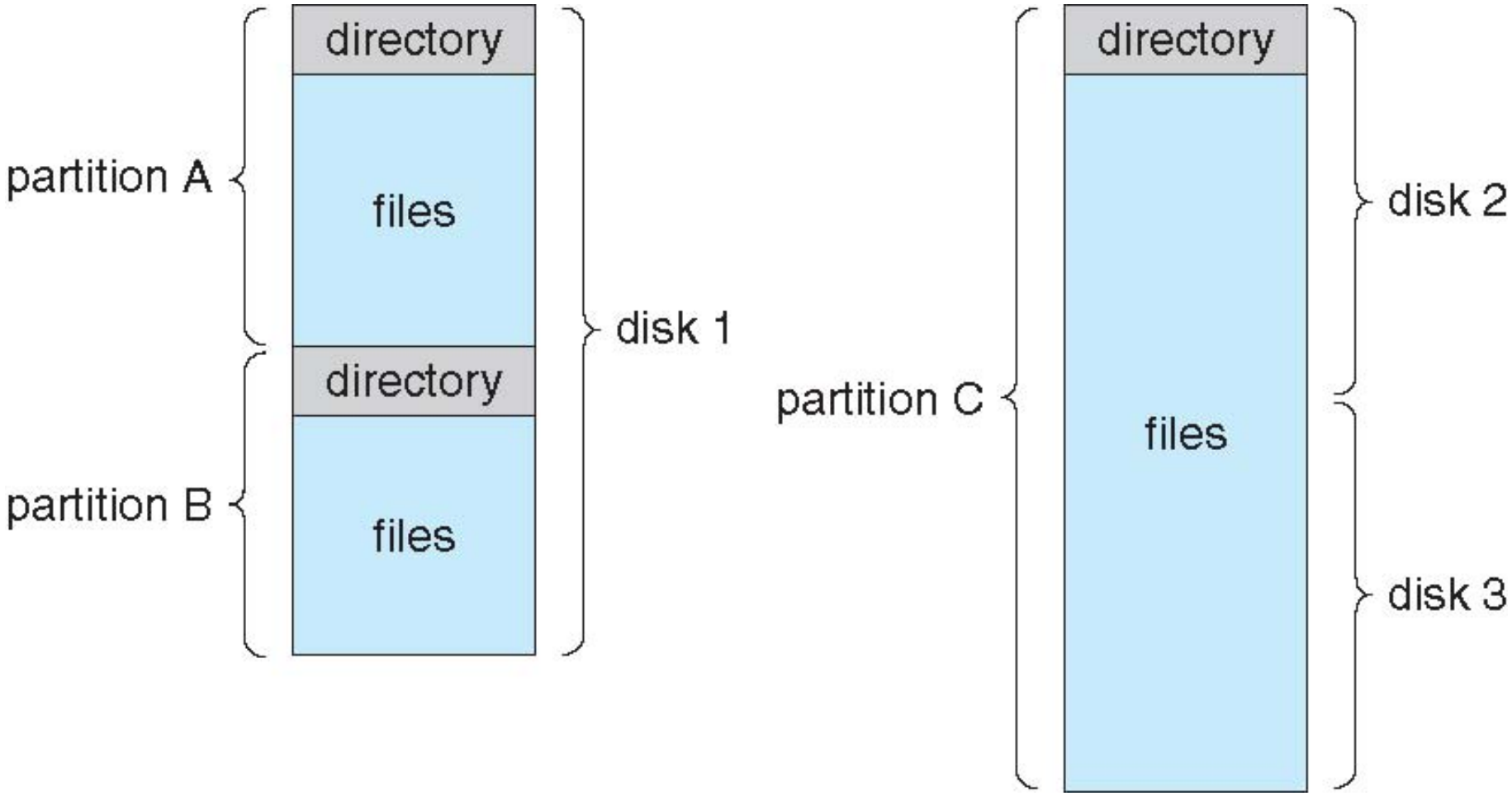


Both the directory structure and the files reside on disk

Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

A Typical File-system Organization



Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

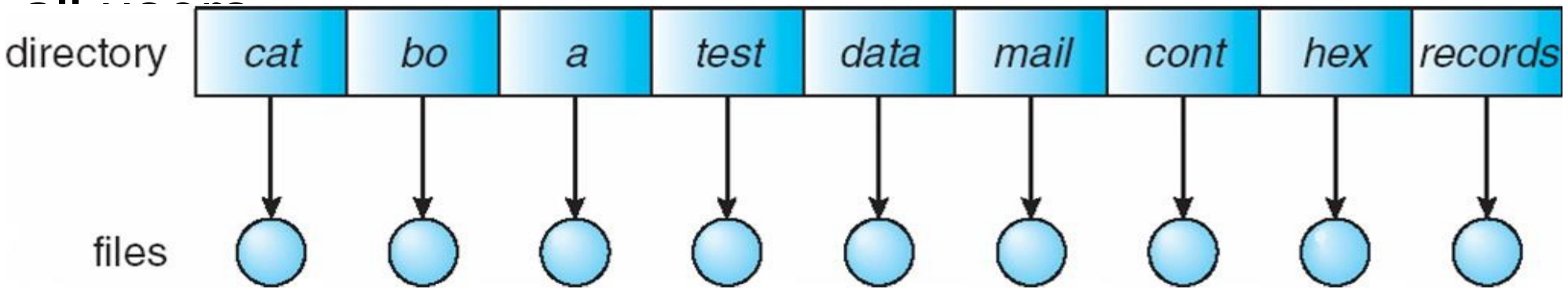
Directory Organization

Organize the Directory (Logically) to Obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

- A single directory for



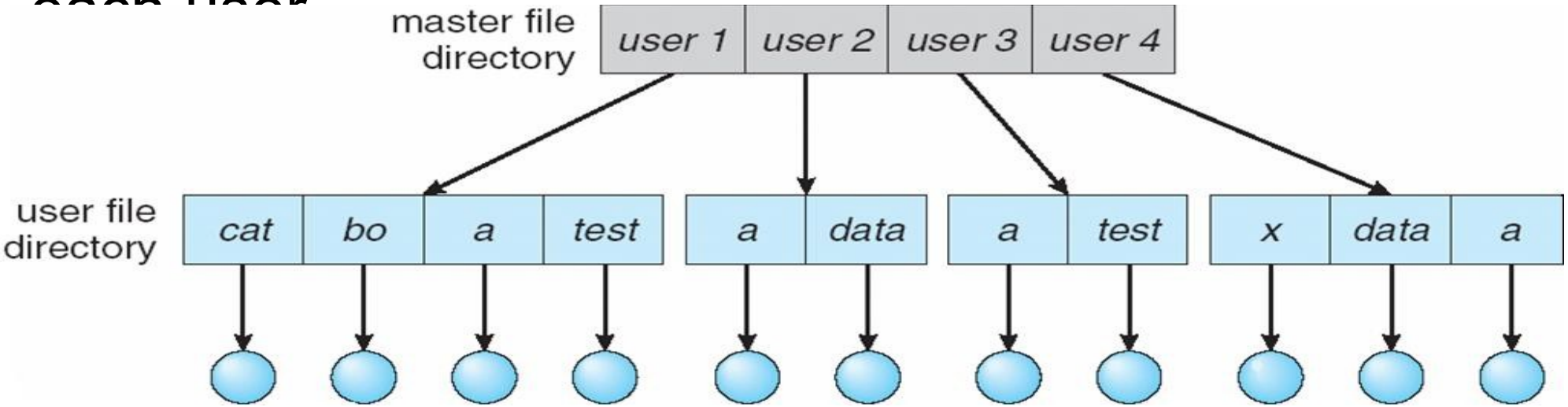
Naming problem

Grouping problem

- All files are contained in the same directory, which is easy to support and understand.
- when the number of files increases or when the system has more than one user.
- Since all files are in the same directory, they must have unique names.
- If two users call their data file *test,* then the unique-name rule is violated.

Two-Level Directory

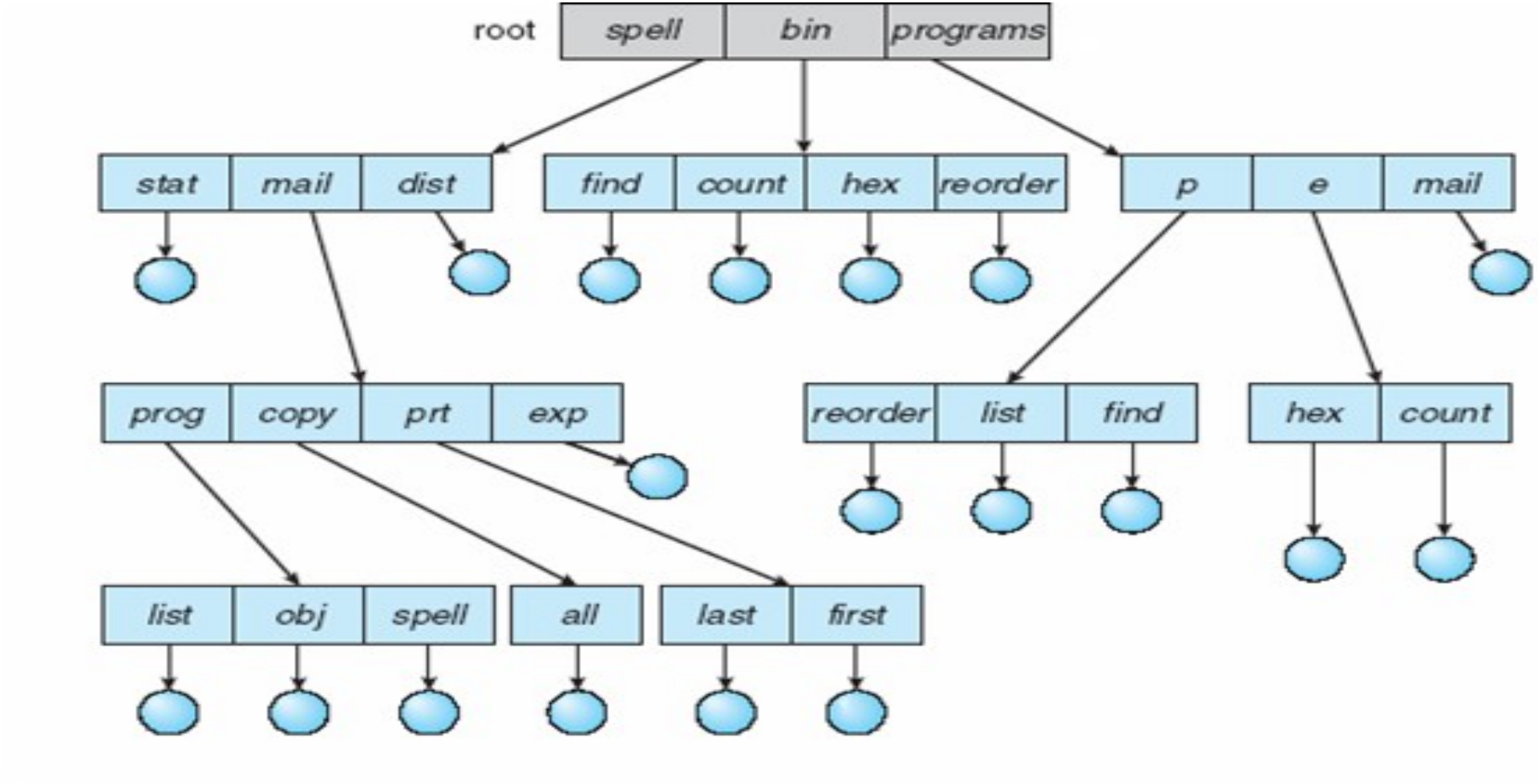
- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

- Each user has his own User File directory (UFD). When a user job starts or a user logs in, the system's Master File directory (MFD) is searched.
- The MFD is indexed by user name or account number, and each entry points to the UFD for that user.
- When a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.
- To delete a file, the operating system confines searches to the local UFD thus, it cannot accidentally delete another user's file that has the same name. Thus this structure isolates one user from another.

Tree-Structured Directories



- The natural generalization is to extend the directory structure to a tree of arbitrary height.

- This generalization allows users to create their own subdirectories and to organize their files accordingly.

- The tree has a root directory, and every file in the system has a unique path name.A directory contains a set of files or subdirectories. All directories have the same internal format.

- Special system calls are used to create and delete directories. normally, each process has a current directory.

- The **current directory** contains files that are related to the process.

- When reference is made to a file, the current directory is searched.

- If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory to be the directory holding that file.

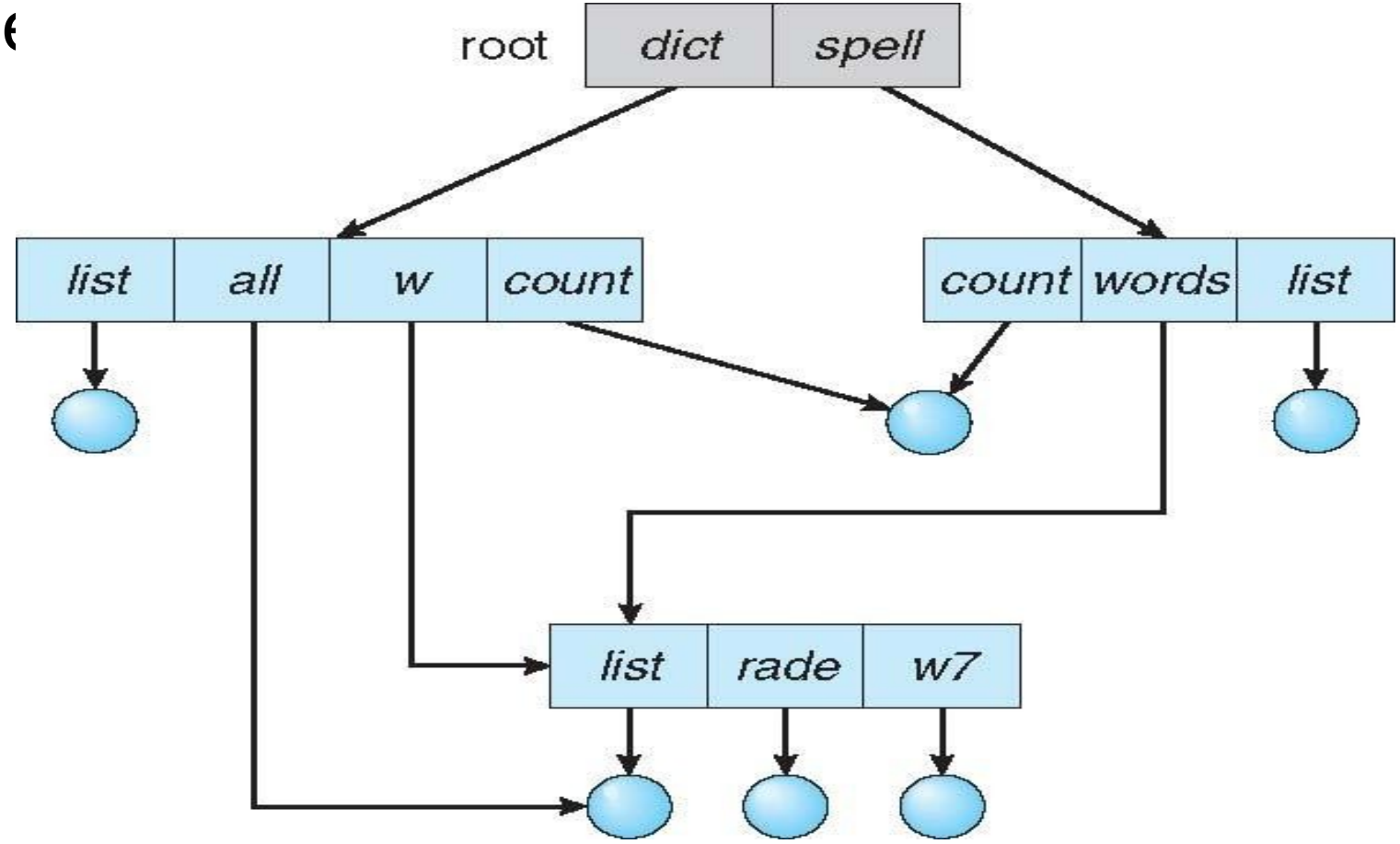
Path names can be of two types.

- An **absolute path name** begins at the root and follows a path down to the specified file, giving the directory names on the path.

- A **relative path name** defines a path from the current directory. For example, in the tree-structured file system (from above fig), if the current directory is *root/spell/mail,* then the relative path name is: *prt / first*, if refers to the same file as does the absolute path name *root/ spcll /mail / prt / first.*

Acyclic-Graph Directories

- Have shared subdirectories and file



- The common subdirectory should be *shared*. *A* shared directory *or* file will exist in the file system in two (or more) places at once.

- An **acyclic graph(** graph with no cycles) allows directories to share subdirectories and files. The *same* file or subdirectory may be in two different directories.

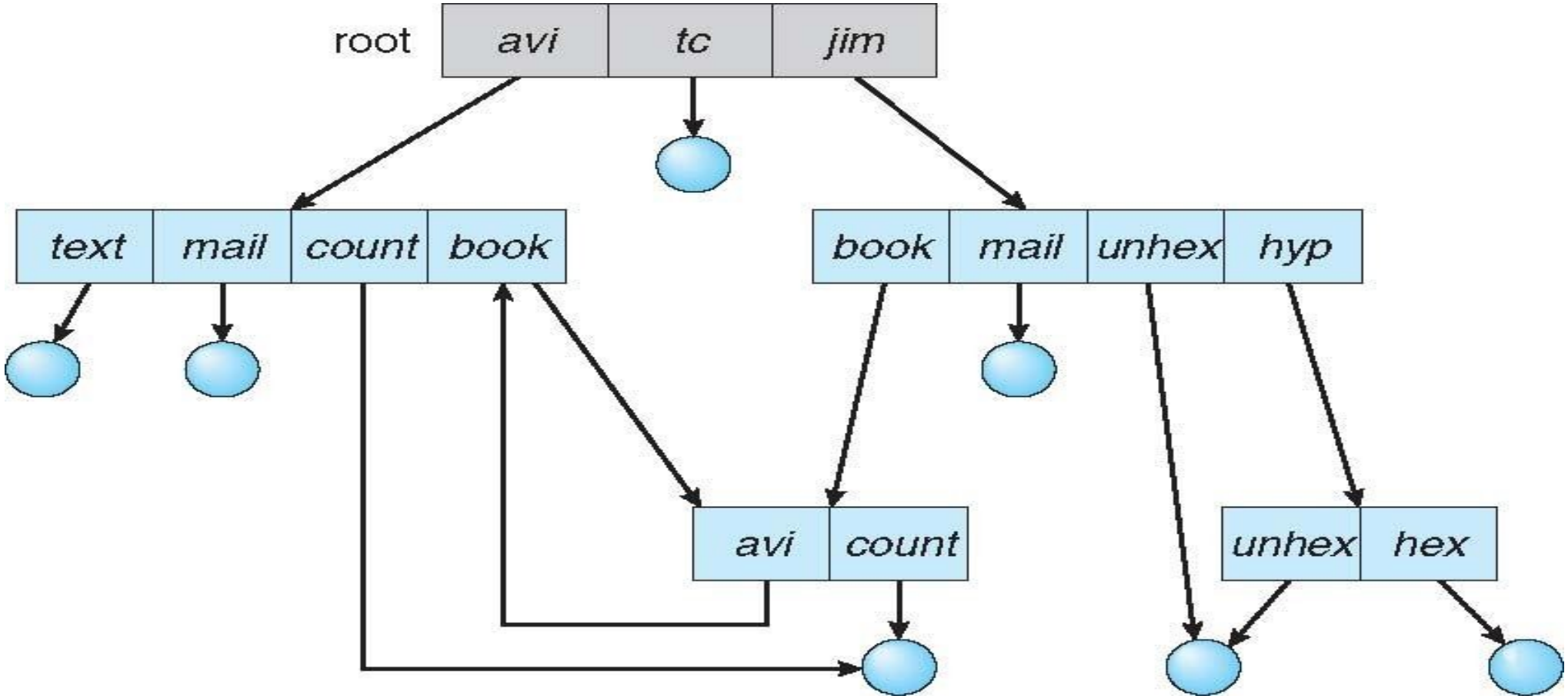
- The acyclic graph is a natural generalization of the tree-structu directory scheme.

- A shared file (or directory) is not the same as two copies of the file. With a shared file, only *one* actual file exists, so any changes made by one person are immediately visible to the other.

- When people are working as a team, all the files they want to share can be put into one directory.

- An acyclic-graph directory structure is more flexible than is a simple tree structure, but it is also more complex.

General Graph Directory



- A serious problem with using an acyclic-graph structure is ensuring that there are no cycles.

- If we start with a two-level directory and allow users to create subdirectories, a tree-structured directory results.

- It should be fairly easy to see that simply adding new files and subdirectories to an existing tree-structured directory preserves the tree-structured nature.

- However, when we add links to an existing tree-structured directory, the tree structure is destroyed, resulting in a simple graph structure

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom

- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	1 1 0
c) **public access**	1	⇒	0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

Attach a group to a file

```
chgrp      G      game
```