Operating System : DAY 4

Mrs.Akshita.S.Chanchlani

SunBeam Infotech

akshita.chanchlani@sunbeaminfo.com

Memory Allocation Types

- **Contiguous** Memory Allocation **- One process – One piece of memory.**
  - **Single** Partition Allocation.
  - **Multiple** Partition Allocation
    - **Fixed** size partitioning (equal size, unequal size)
    - **Dynamic** Partitioning (First-Fit, Best-Fit , Worst-fit)
      - Problem (Internal and External Fragmentation)

- **Non Contiguous** Memory Allocation - **One process – Many pieces of memory.**
  - **Paging**
  - **Segmentation**

Relocation

- Because of swapping and compaction, a process may occupy different main memory locations during its lifetime
- Hence physical memory references by a process cannot be fixed
- This problem is solved by distinguishing between **logical address** and **physical address**

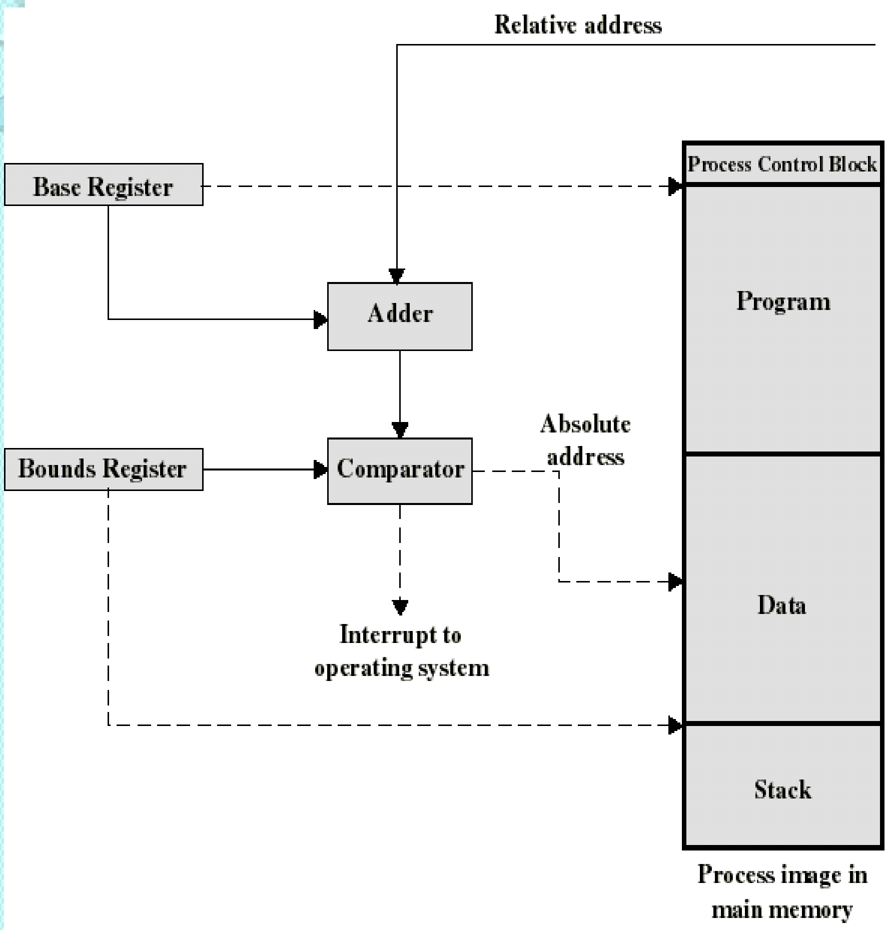
Address Types

- A **physical address** (absolute address) is a physical location in main memory
- A **logical address** is a reference to a memory location independent of the physical structure/organization of memory
- Compilers produce code in which all memory references are logical addresses
- A **relative address** is an example of logical address in which the address is expressed as a location relative to some known point in the program (ex: the beginning)

Address Translation

- Relative address is the most frequent type of logical address used in pgm modules (ie: executable files)
- Such modules are loaded in main memory with all memory references in relative form
- Physical addresses are calculated "on the fly" as the instructions are executed
- For adequate performance, the translation from relative to physical address must by done by hardware

Simple example of hardware translation of addresses



- When a process is assigned to the running state, a base register (in CPU) gets loaded with the starting physical address of the process
- A bound register gets loaded with the process's ending physical address
- When a relative addresses is encountered, it is added with the content of the base register to obtain the physical address which is compared with the content of the bound register
- This provides hardware protection: each process can only access memory within its process image

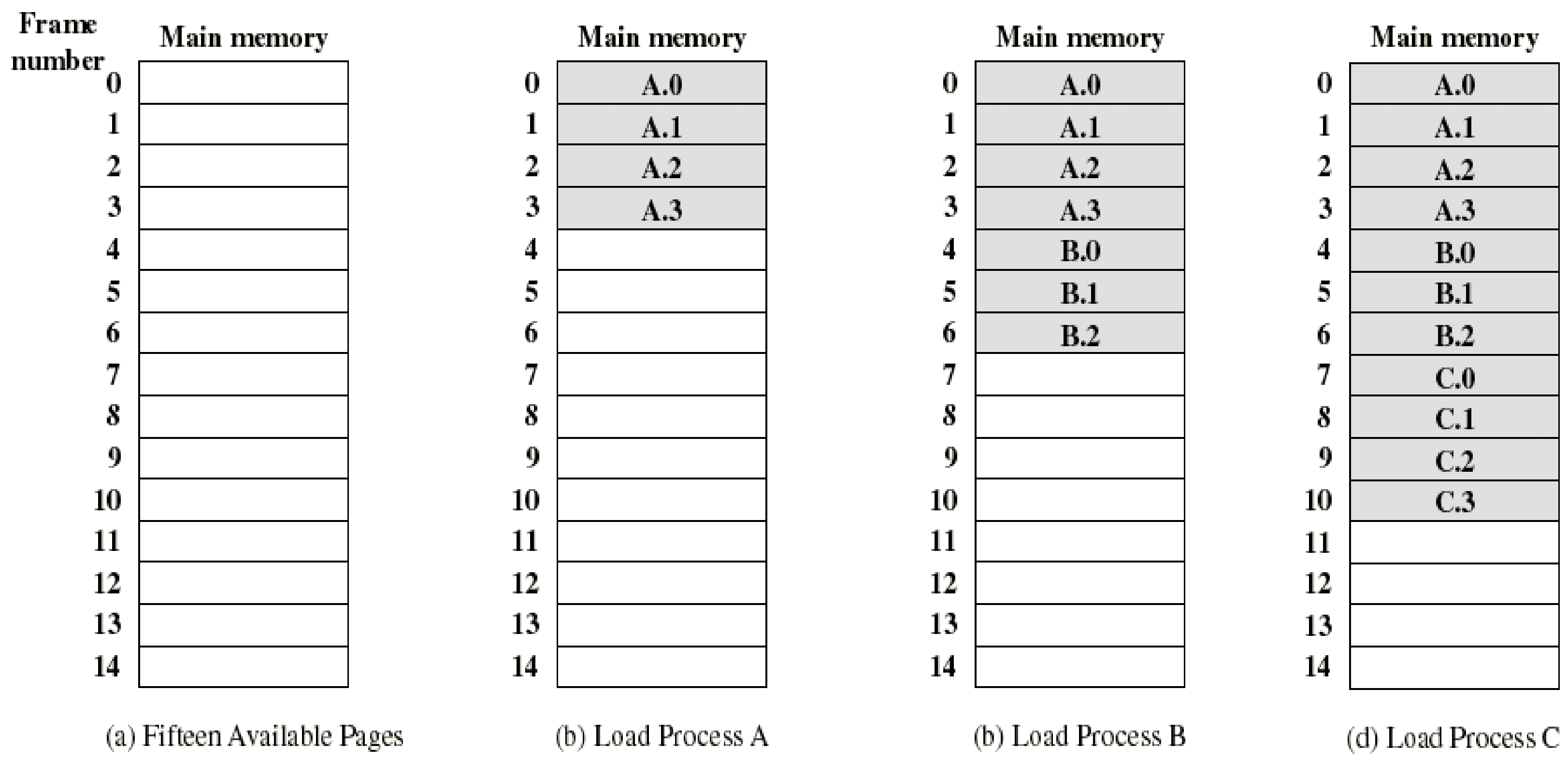
Simple Paging

- Main memory is partition into equal fixed-sized chunks (of relatively small size)
- Trick: each process is also divided into chunks of the same size called **pages**
- The process pages can thus be assigned to the available chunks in main memory called **frames** (or page frames)
- Consequence: a process does not need to occupy a contiguous portion of memory

Paging

- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***n*** pages, need to find *n* free frames and load program
- Set up a page table to translate logical to physical addresses

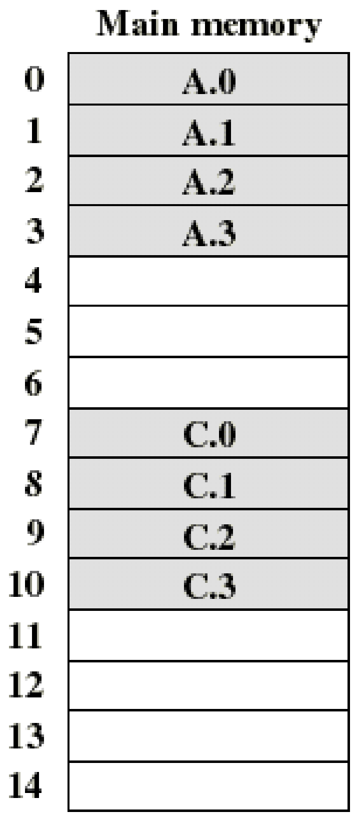
Example of process loading



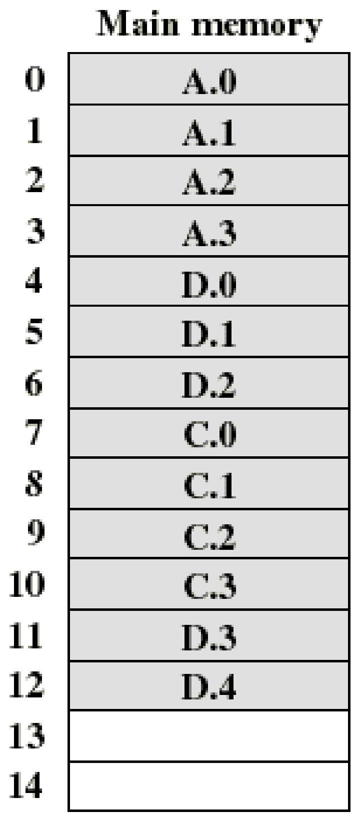
- Now suppose that process B is swapped out

Example of process loading (cont.)

- When process A and C are blocked, the pager loads a new process D consisting of 5 pages
- Process D does not occupied a contiguous portion of memory
- There is no external fragmentation



(e) Swap out B



(f) Load Process D

Page Tables

0	0
1	1
2	2
3	3

Process A page table

0	—
1	—
2	—

Process B page table

0	7
1	8
2	9
3	10

Process C page table

0	4
1	5
2	6
3	11
4	12

Process D page table

13
14

Free frame list

- The OS now needs to maintain (in main memory) a **page table** for each process
- Each entry of a page table consist of the frame number where the corresponding page is physically located
- The page table is indexed by the page number to obtain the frame number
- A free frame list, available for pages, is maintained

Logical address used in paging

•Within each program, each logical address must consist of a page number and an offset within the page.

•A CPU register always holds the starting physical address of the page table of the currently running process.

•Presented with the logical address (page number, offset) the processor accesses the page table to obtain the physical address (frame number, offset)

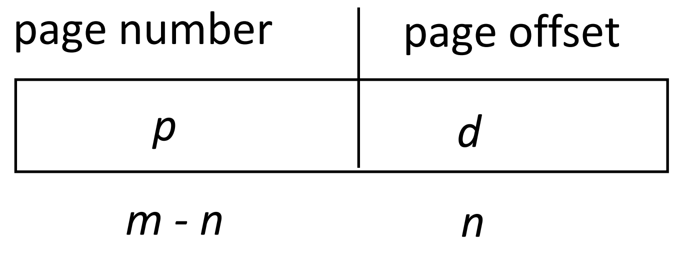
Address Translation Scheme

•Address generated by CPU is divided into:

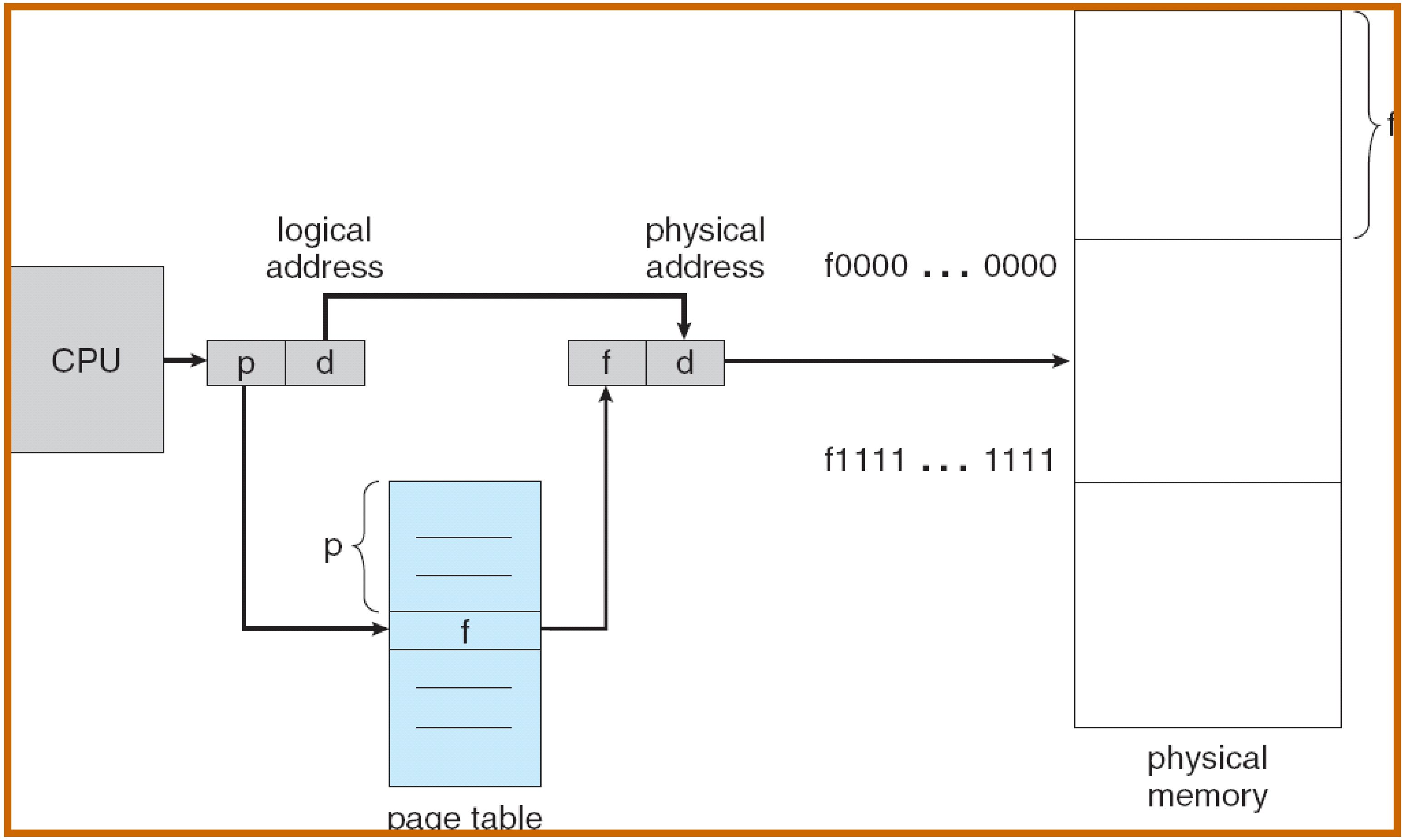
–**Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory

–**Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit

–For given logical address space 2^m *and page size* 2^n



Paging Hardware



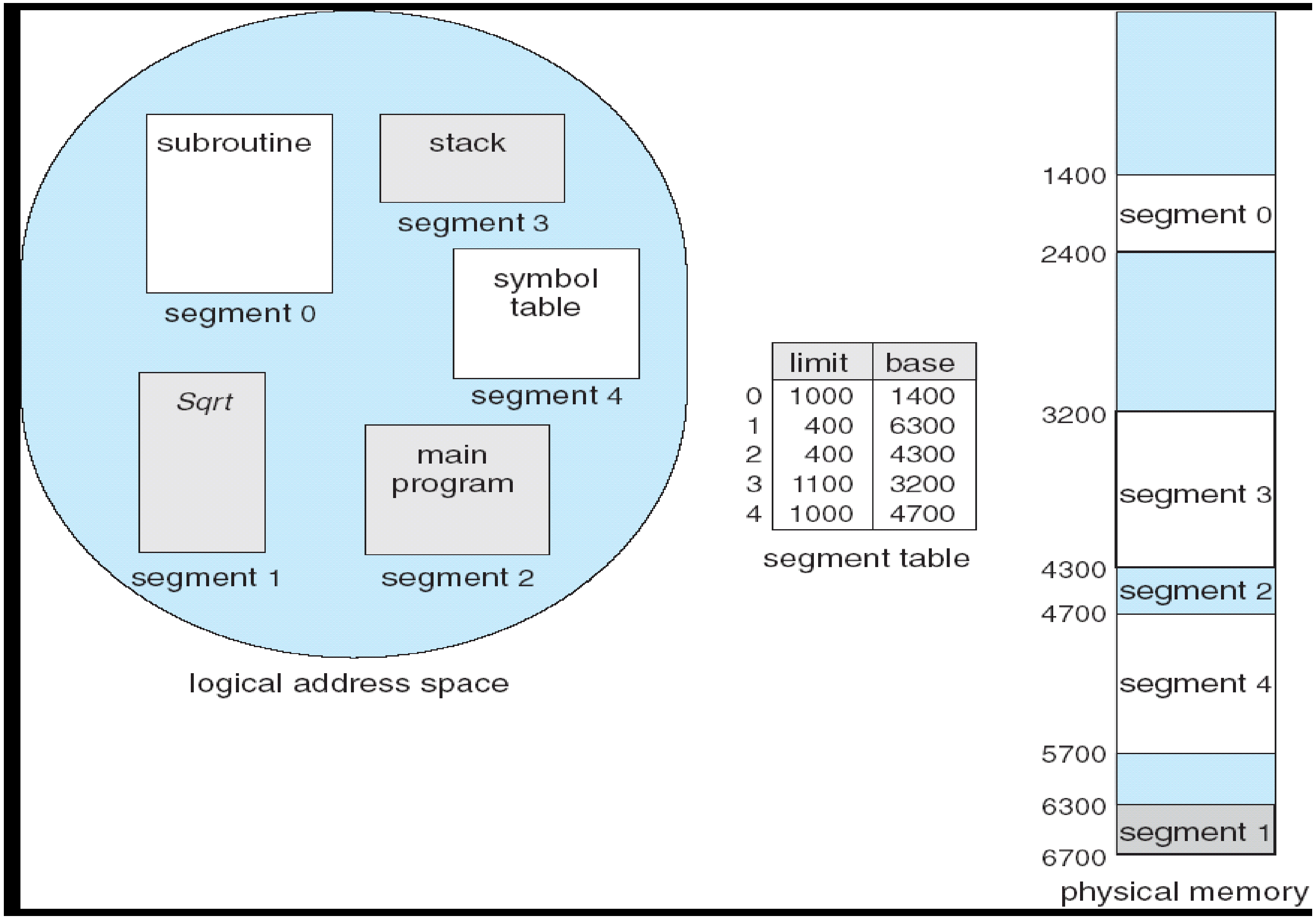
Segmentation

- Each program is subdivided into blocks of non-equal size called segments
- ***Segment***: a region of logically contiguous memory
- ***Segmentation-based transition***: use a table of base-and-bound pairs
- When a process gets loaded into main memory, its different segments can be located anywhere.
- Each segment is fully packed with instructs/data: no internal fragmentation
- There is external fragmentation; it is reduced when using small segments

Segmentation

- In contrast with paging, segmentation is visible to the programmer
  - provided as a convenience to organize logically programs (ex: data in one segment, code in another segment)
  - must be aware of segment size limit
- The OS maintains a **segment table** for each process. Each entry contains:
  - the starting physical addresses of that segment.
  - the length of that segment (for protection)

Segmentation Example

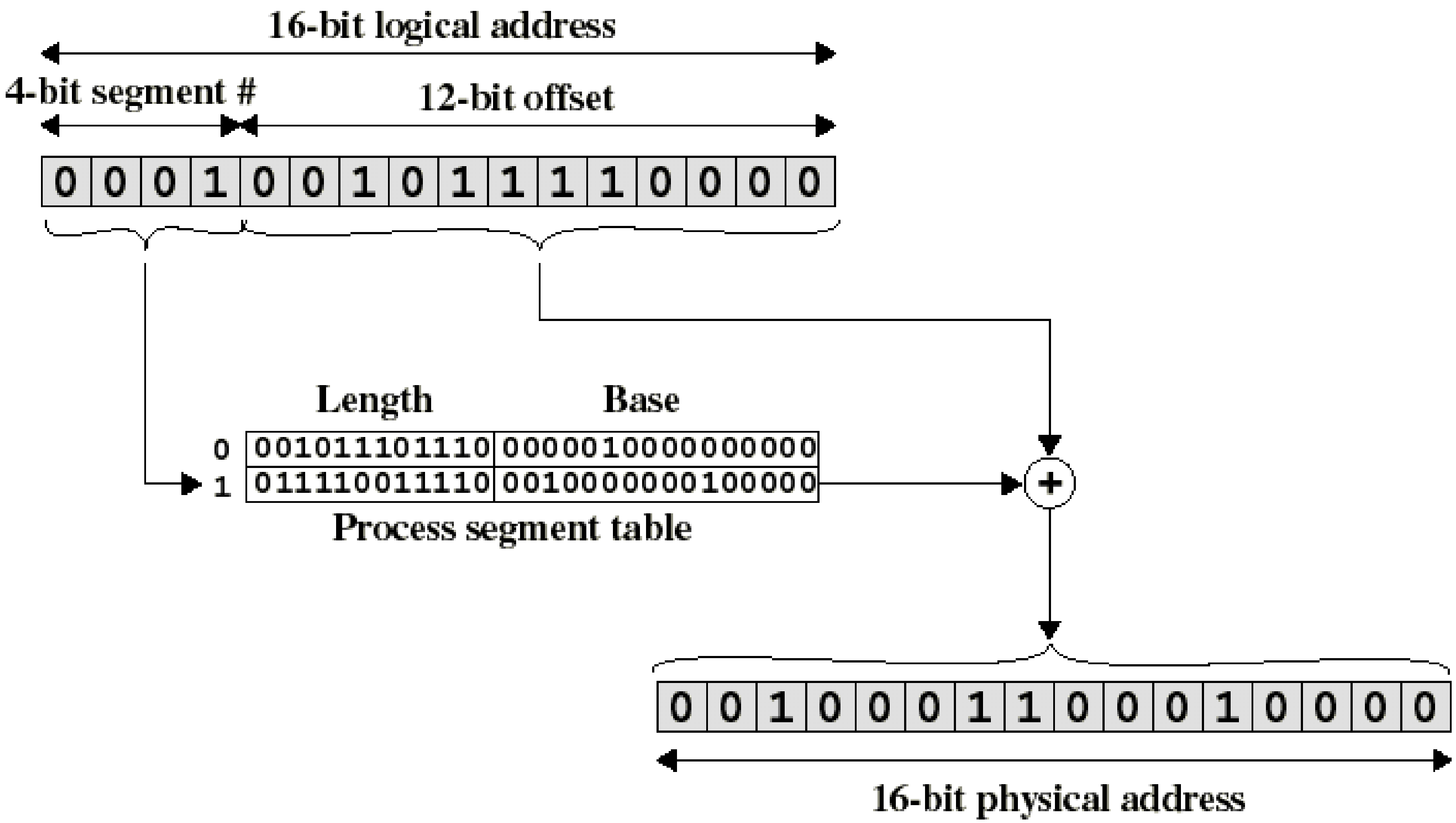


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Logical address used in segmentation

- When a process enters the Running state, a CPU register gets loaded with the starting address of the process's segment table.
- Presented with a **logical address (segment number, offset) = (n,m),** the CPU indexes (with n) the segment table to obtain the starting physical address k and the length l of that segment
- The physical address is obtained by **adding** m to k (in contrast with paging)
  - the hardware also compares the offset m with the length l of that segment to determine if the address is valid

Logical-to-Physical Address Translation in segmentation



Simple segmentation and paging comparison

- Segmentation requires more complicated hardware for address translation
- Segmentation suffers from external fragmentation
- Paging only yield a small internal fragmentation
- Segmentation is visible to the programmer whereas paging is transparent
- Segmentation can be viewed as commodity offered to the programmer to organize logically a program into segments and using different kinds of protection (ex: execute-only for code but read-write for data)
  - for this we need to use protection bits in segment table entries

Virtual memory

•**Virtual memory** – separation of user logical memory from physical memory:

–Only part of the program needs to be in memory for execution.

–Logical address space can therefore be much larger than physical address space.

–Allows address spaces to be shared by several processes.

–Allows for more efficient process creation.

–More programs running concurrently.

–Less I/O needed to load or swap processes.

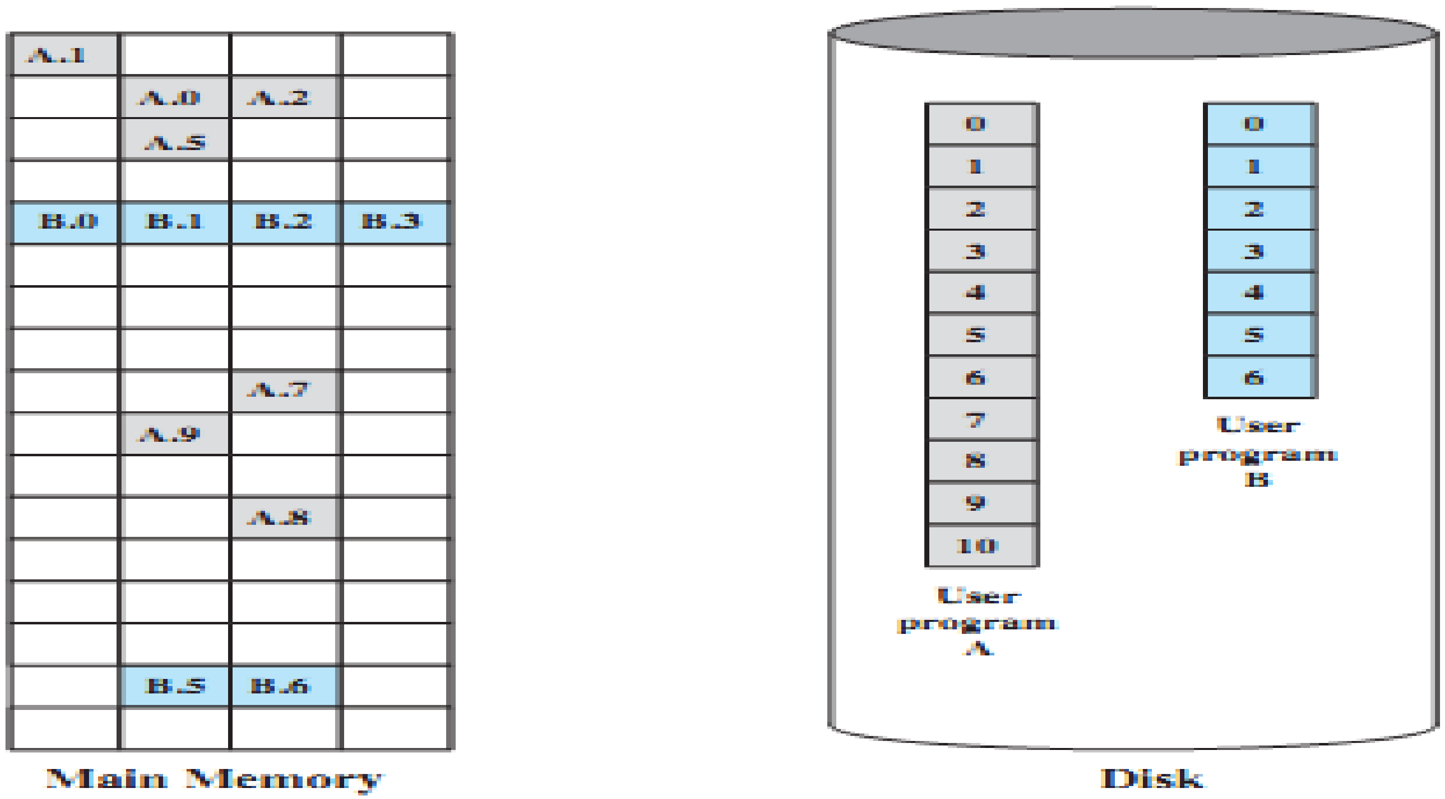
•Virtual memory gives the programmer the impression that he/she is dealing with a huge main memory (relying on available disk space). The OS loads automatically and on-demand pages of the running process.

•A process image may be larger than the entire main memory.

•The required pages need to be loaded into memory whenever required.

Virtual memory is implemented using Demand Paging or Demand Segmentation.

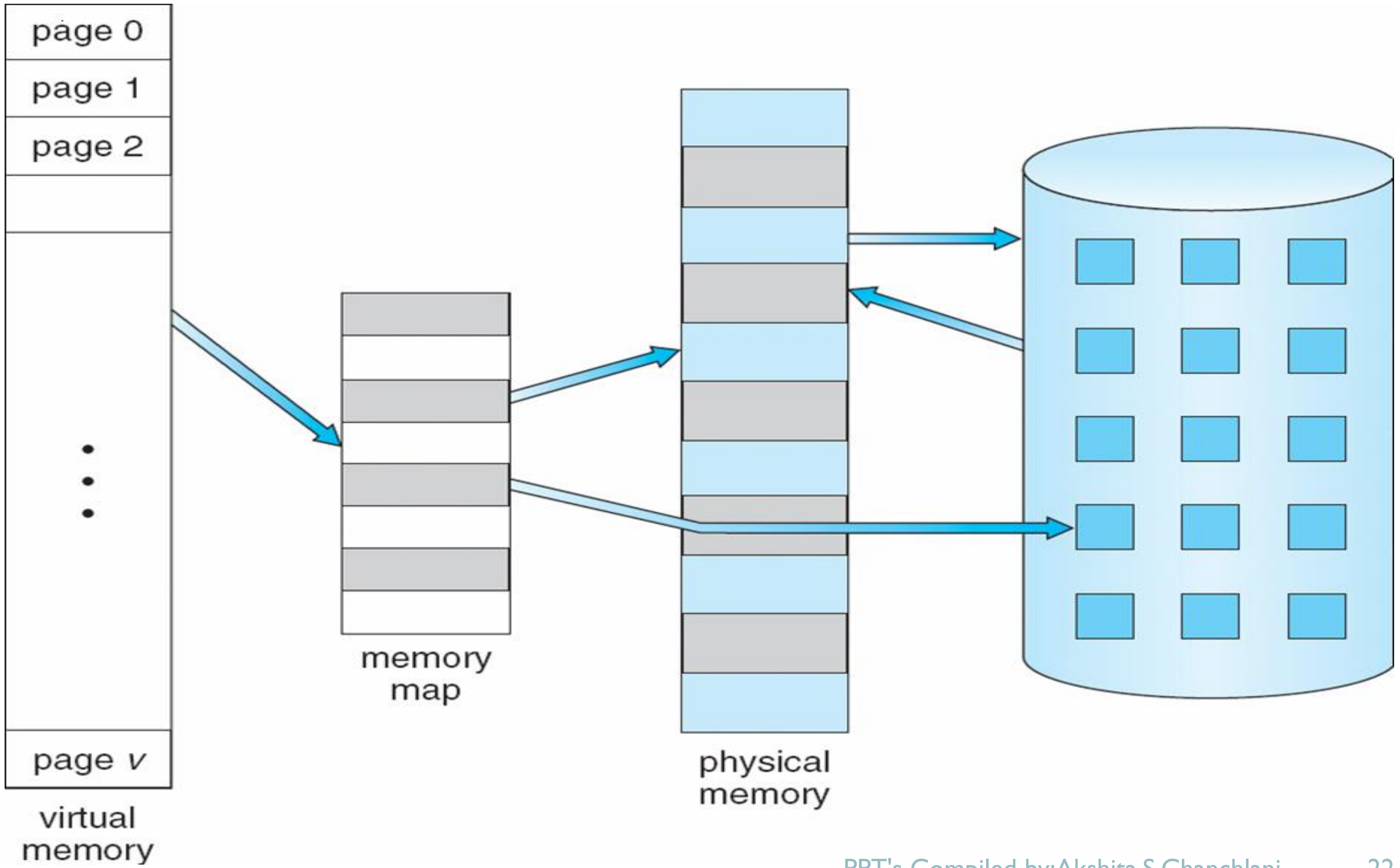
Virtual memory components



Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

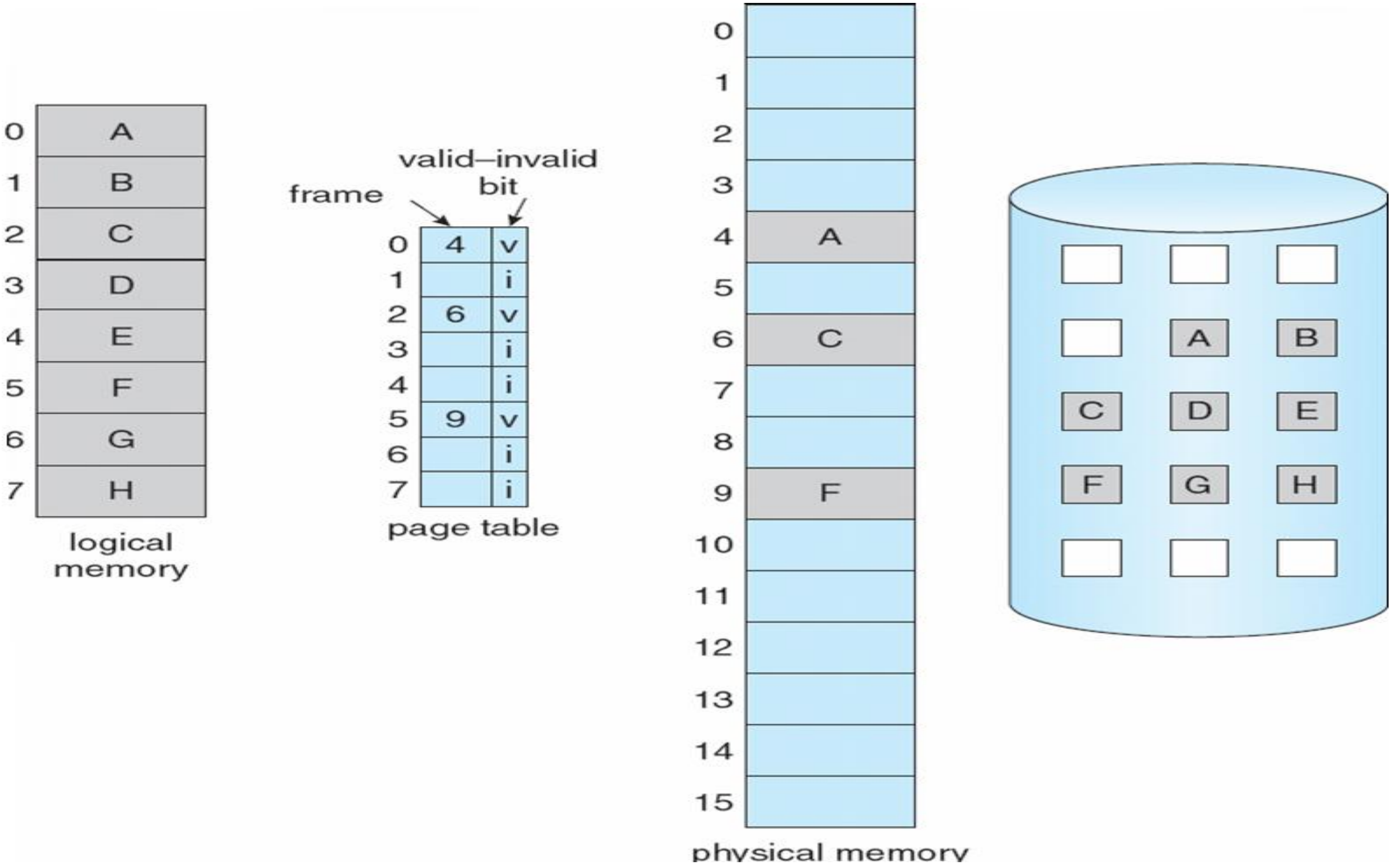
Virtual Memory that is larger than Physical Memory



Demand Paging

- The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.
- Bring a page into memory only when it is needed:
  - –Less I/O needed, no unnecessary I/O
  - –Less memory needed
  - –Faster response
  - –More users
- Page is needed $\Rightarrow$ reference to it:
  - –invalid reference $\Rightarrow$ abort
  - –not-in-memory $\Rightarrow$ bring to memory
- Similar to paging system with swapping.
- **Lazy swapper – never swaps a page into memory unless page will be needed; Swapper that deals with pages is a pager.**

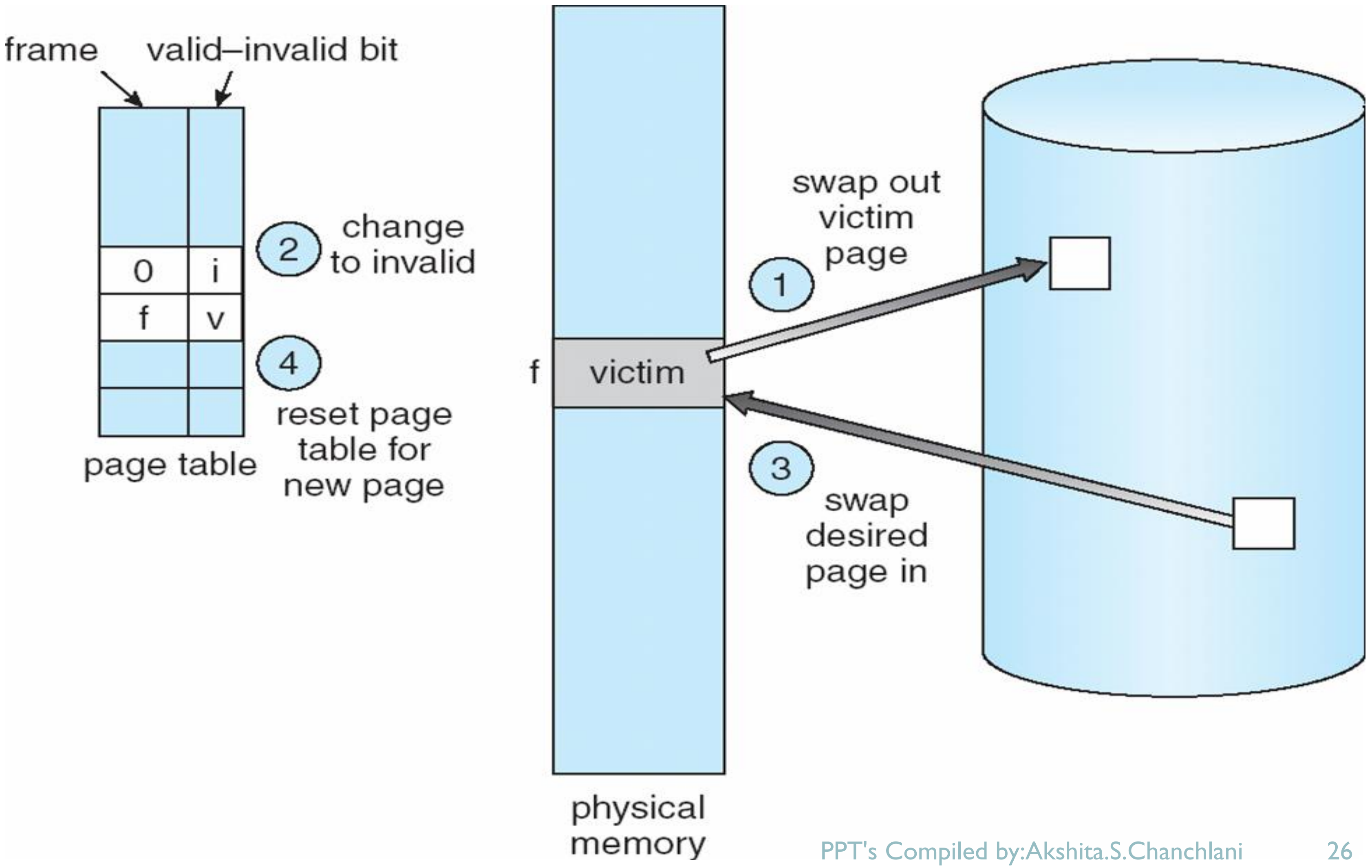
Page Table when some pages are not in Main Memory



What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.
- Need page replacement algorithm.
- Performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.

Steps in handling a Page Replacement



Steps in handling a Page Replacement

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

2.Find a free frame:

- –If there is a free frame, use it.
- –If there is no free frame, use a **page replacement algorithm to select a victim page.**

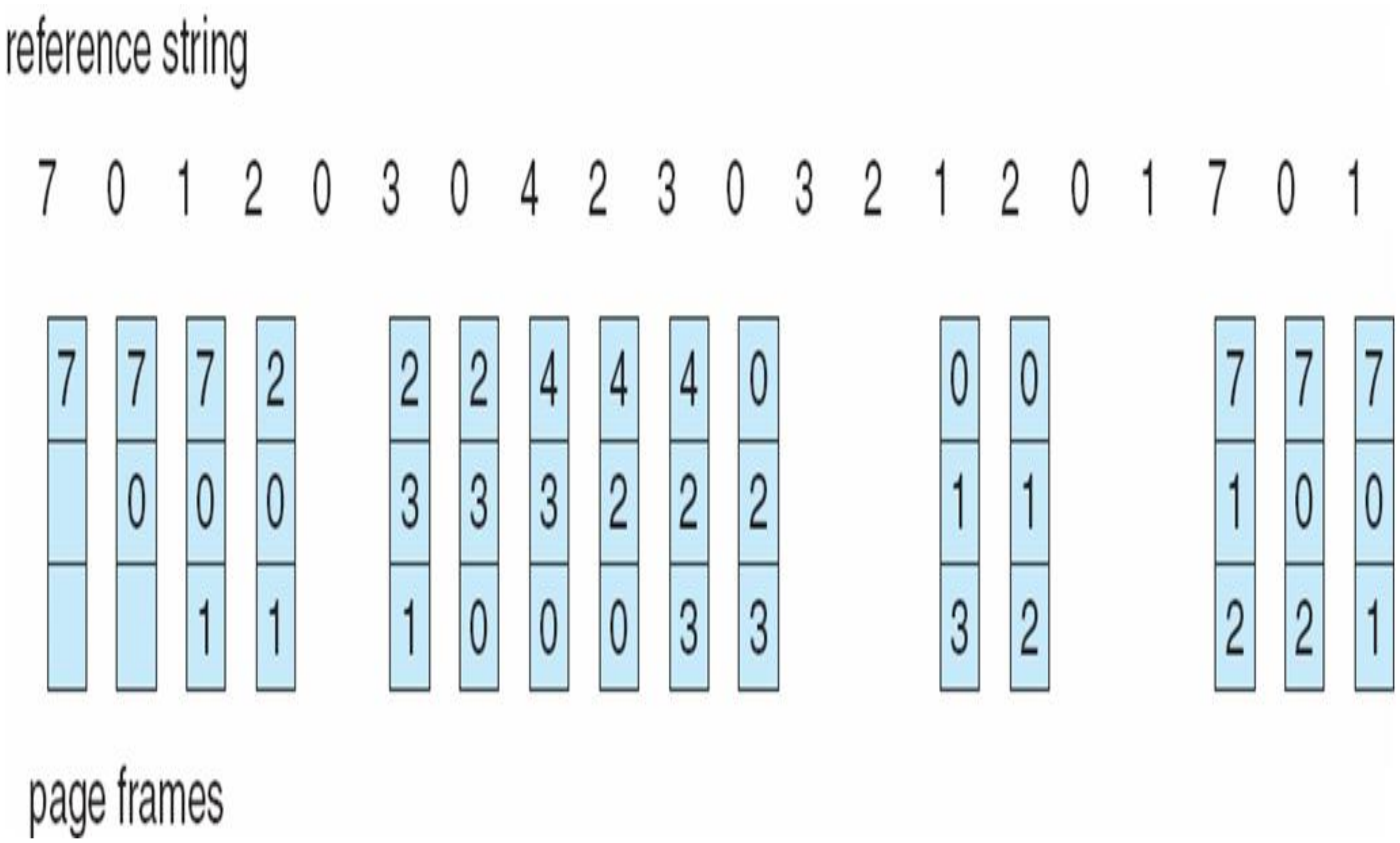
3.Bring the desired page into the (newly) free frame; update the page and frame tables.

4.Restart the process.

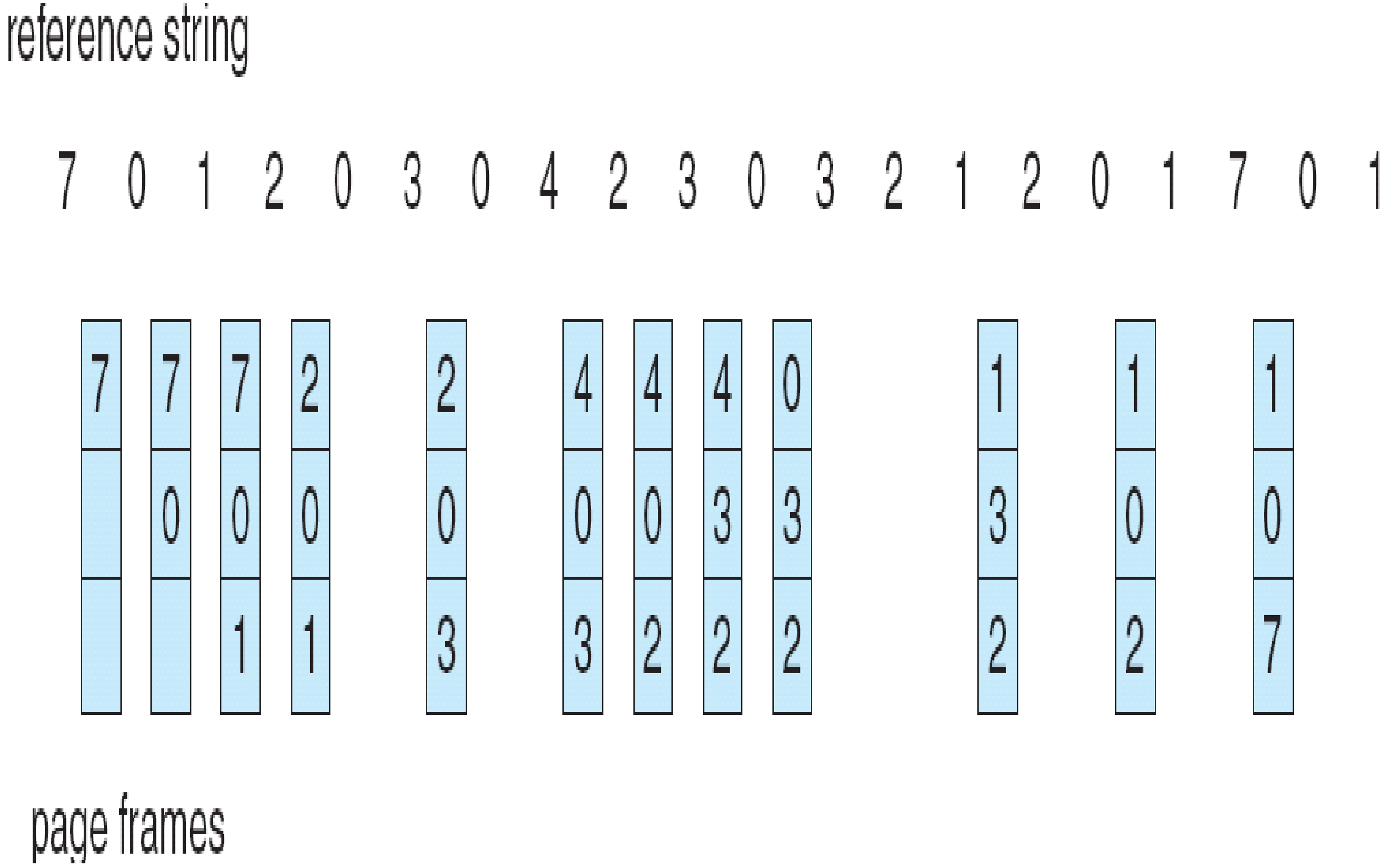
Page Replacement Algorithms

- FIFO,
- LRU,
- Optimal,

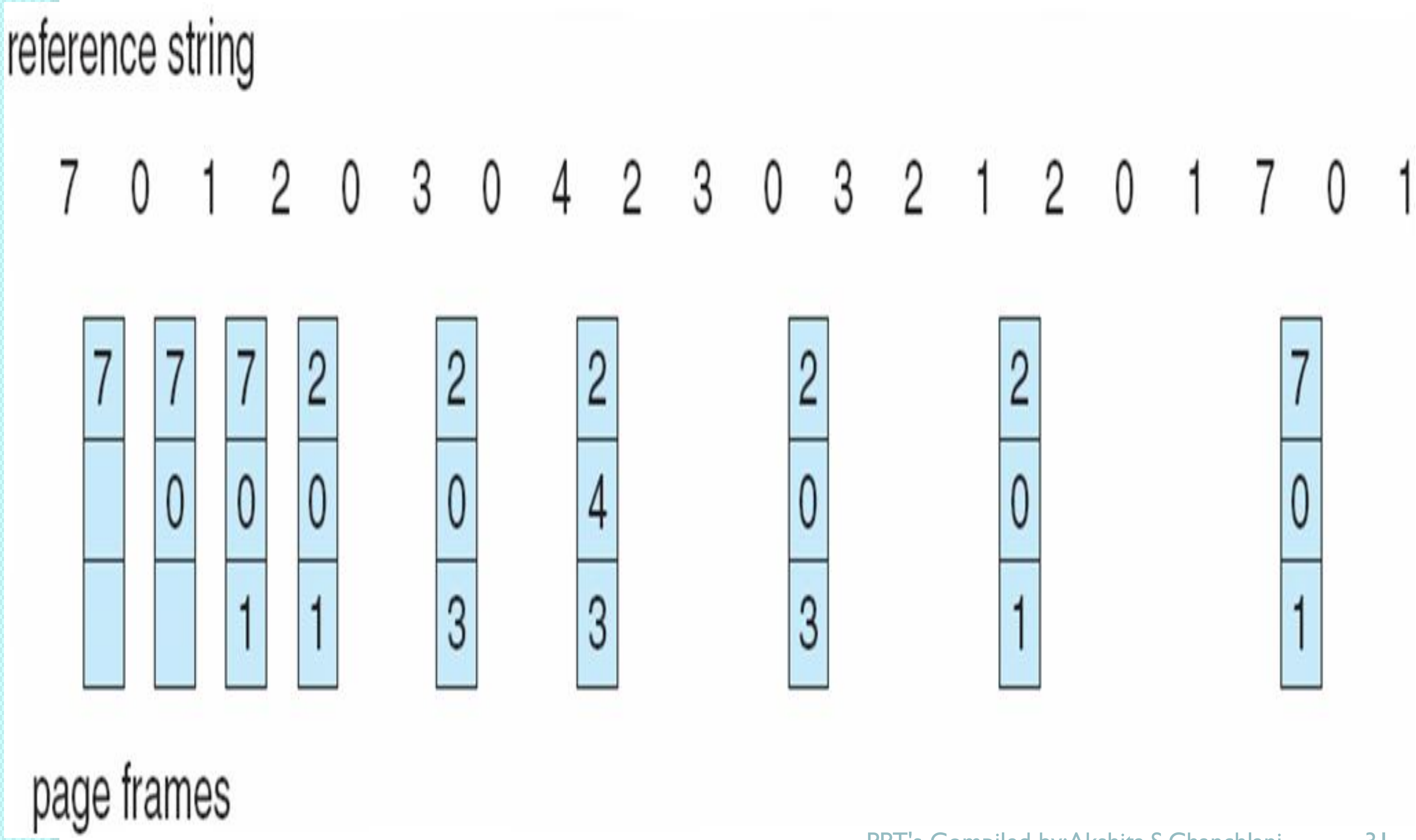
FIFO Page Replacement



LRU Page Replacement



Optimal Page Replacement



Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high
  - –low CPU utilization
  - –OS thinks it needs increased multiprogramming
  - –adds another process to system
- *Thrashing* is when a process is busy swapping pages in and out

File

- file is a collection of logically related data or information
- file is a stream of bytes/bits
- file is a basic storage unit
    - File = data+metadata
    - Data : Actual File Contents
    - Metadata : Information about file.
- File Attributes:
    - Name,type,location,size etc..
- File Operations
    - Create,delete,write,read

- **"file system“ is a way to store data onto the disk in an organized manner so that it can accessed efficiently**

What is an inode / FCB

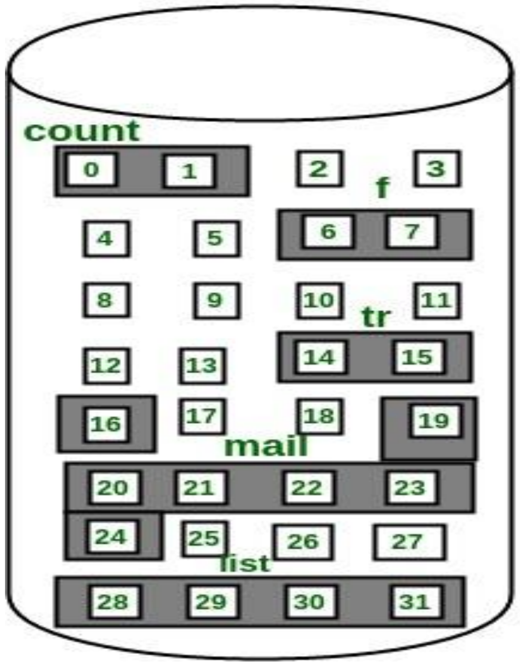
- An inode (index node) is a control structure that contains key information needed by the OS to access a particular file. Several file names may be associated with a single inode, but each file is controlled by exactly ONE inode.

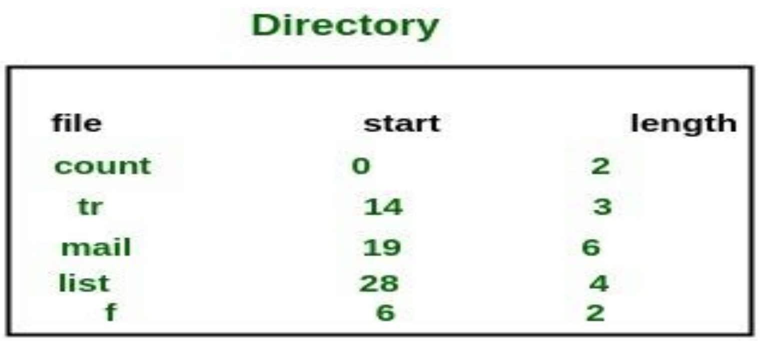
- On the disk, there is an inode table that contains the inodes of all the files in the filesystem. When a file is opened, its inode is brought into main memory and stored in a memory-resident inode table.
- Information about the file can be kept in one structure referred as "FCB" i.e. File Control Block/iNode
  - inode/FCB contains info about the file like:
    - name of the file
    - type of the file
    - size of the file
    - parent folder location
    - access perms for user/owner,
    - grp member and others etc

File Allocation on Disk

- Low level access methods for a file depend upon the disk allocation scheme used to store file data
  - Contiguous
  - Linked
  - Block or indexed

Contiguous Allocation



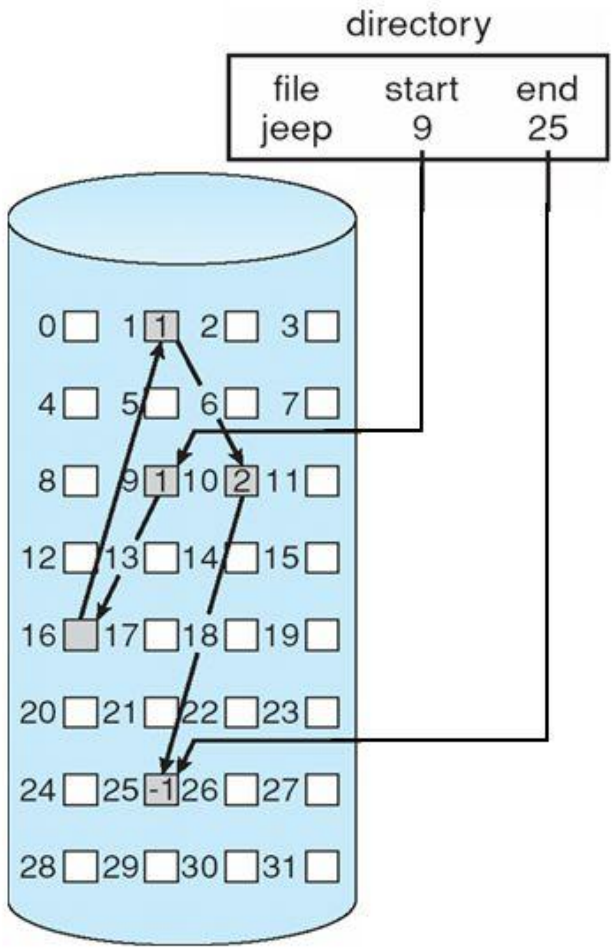

Directory

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

- File is allocated large contiguous chunks
- Expanding the file requires copying
- Dynamic storage allocation - first fit, best fit
- **External fragmentation occurs on disk**

Linked Allocation

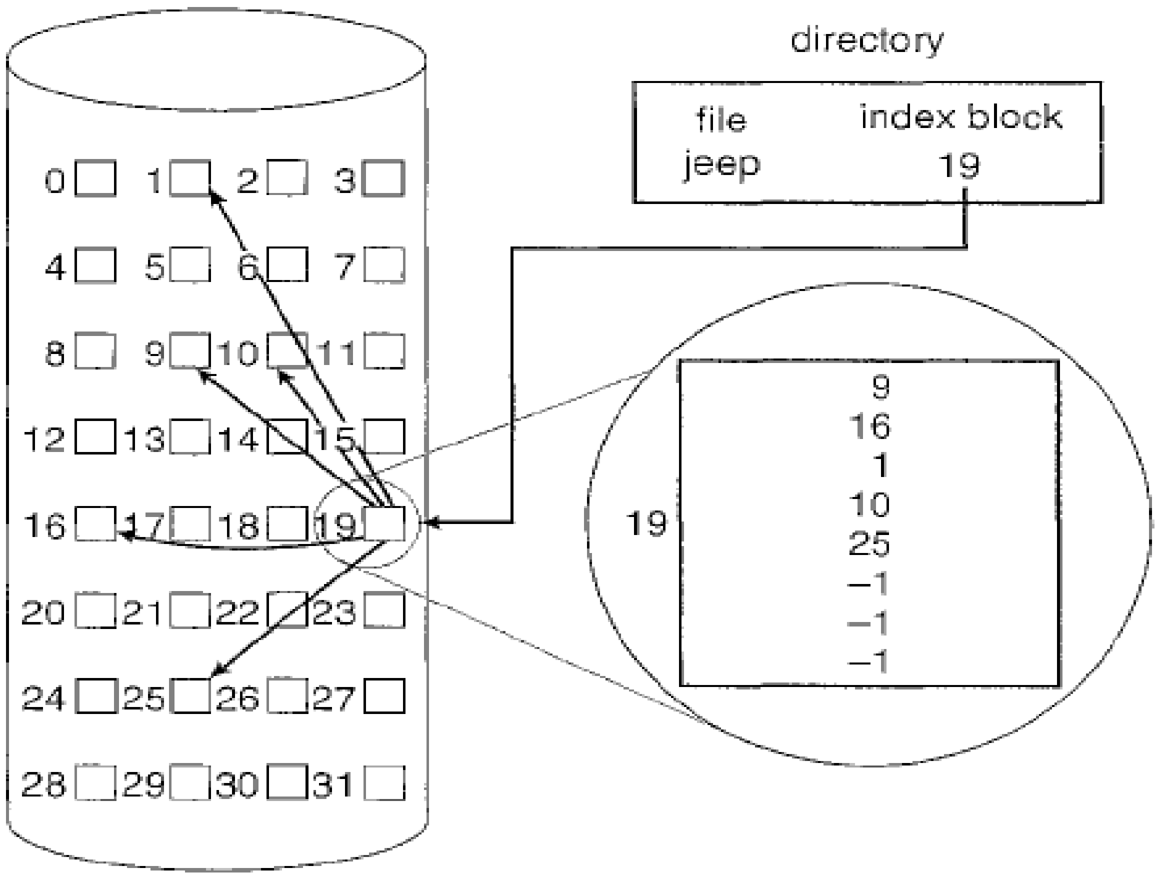
- Each file is a linked list of disk blocks, which may be scattered on the disk
- Directory contains a pointer to the first and last blocks, and each block contains a pointer to the next block
- **Advantages**:
  - No external fragmentation
  - Easy to expand the size of a file
- **Disadvantages**:
  - Not suitable for random access within a file
  - Pointers take up some disk space
  - Difficult to recover a file if a pointer is lost or damaged
- Blocks may be collected into clusters of several blocks
  - Fewer pointers are necessary
  - Fewer disk seeks to read an entire file
  - Greater internal fragmentation



Block / Indexed

- a special block known as the **Index block** contains the pointers to all the blocks occupied by a file.
- The ith entry in the index block contains the disk address of the ith file block.
- The directory entry contains the address of the index block.
- 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.

Indexed Allocation



Disk Scheduling

- **Disk scheduling** is done by operating systems to schedule I/O requests arriving for the disk.
- Disk scheduling is important because:
  - Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
  - Two or more request may be far from each other so can result in greater disk arm movement.
  - Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

Disk Scheduling Criteria

Seek Time

- Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write.
- minimum average seek time is better.

Rotational Latency

- The time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads.
- minimum rotational latency is better.

Transfer Time

- The time to transfer the data.
- It depends on the rotating speed of the disk and number of bytes to be transferred.

Disk Access Time

- **SeekTime+Rotational Latency +Transfer Time**

Disk Response Time

- The average of time spent by a request waiting to perform its I/O operation.
- minimum variance response time is better.

Disk Scheduling Algorithms

First Come First Serve

- FCFS, the requests are addressed in the order they arrive in the disk queue.

Shortest Seek Time First

- SSTF (Shortest Seek Time First), requests having shortest seek time are executed first.
- it decreases the average response time and increases the throughput of system.

Scan/Elevator

- SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path.

CSCAN

- disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Look

- similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only.

Clook

- CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request.

FCFS(First Come First Serve)

a request queue (0-199)

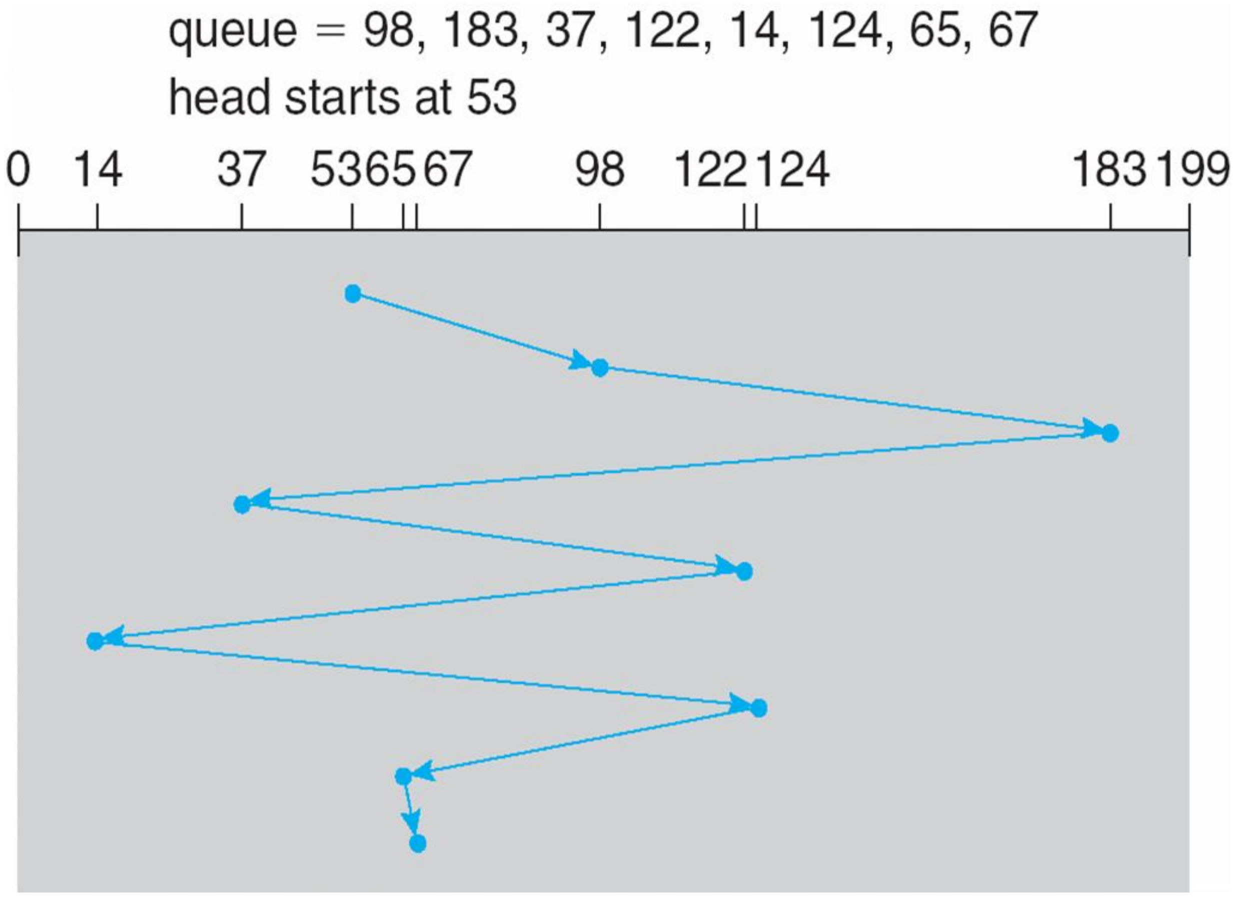


Illustration shows total head movement of 640 cylinders.

SSTF(Shortest Seek Time First)

•Selects the request with the minimum seek time from the current head position

•SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

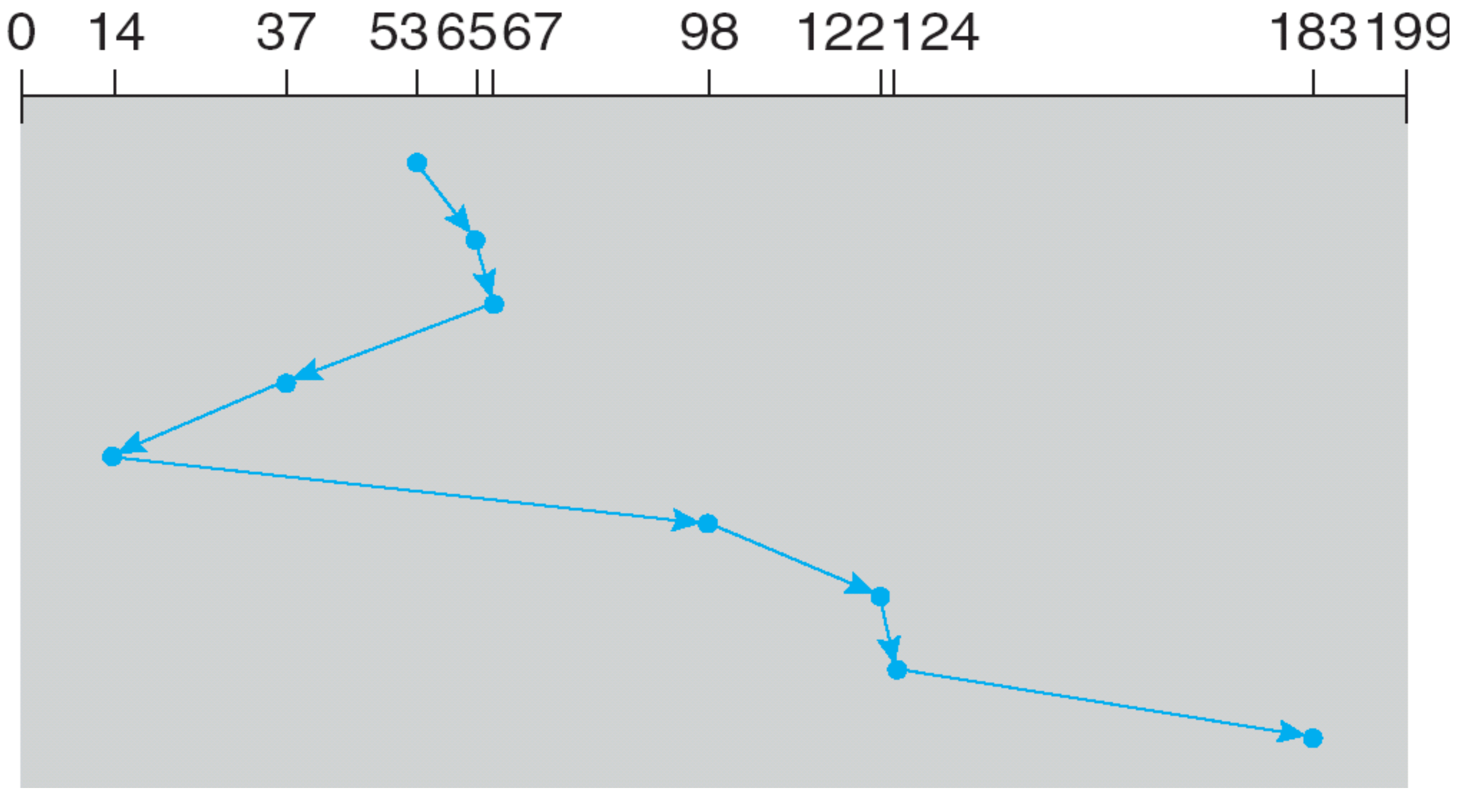


Illustration shows total head movement of 236 cylinders.

44

SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- SCAN algorithm Sometimes called the elevator algorithm

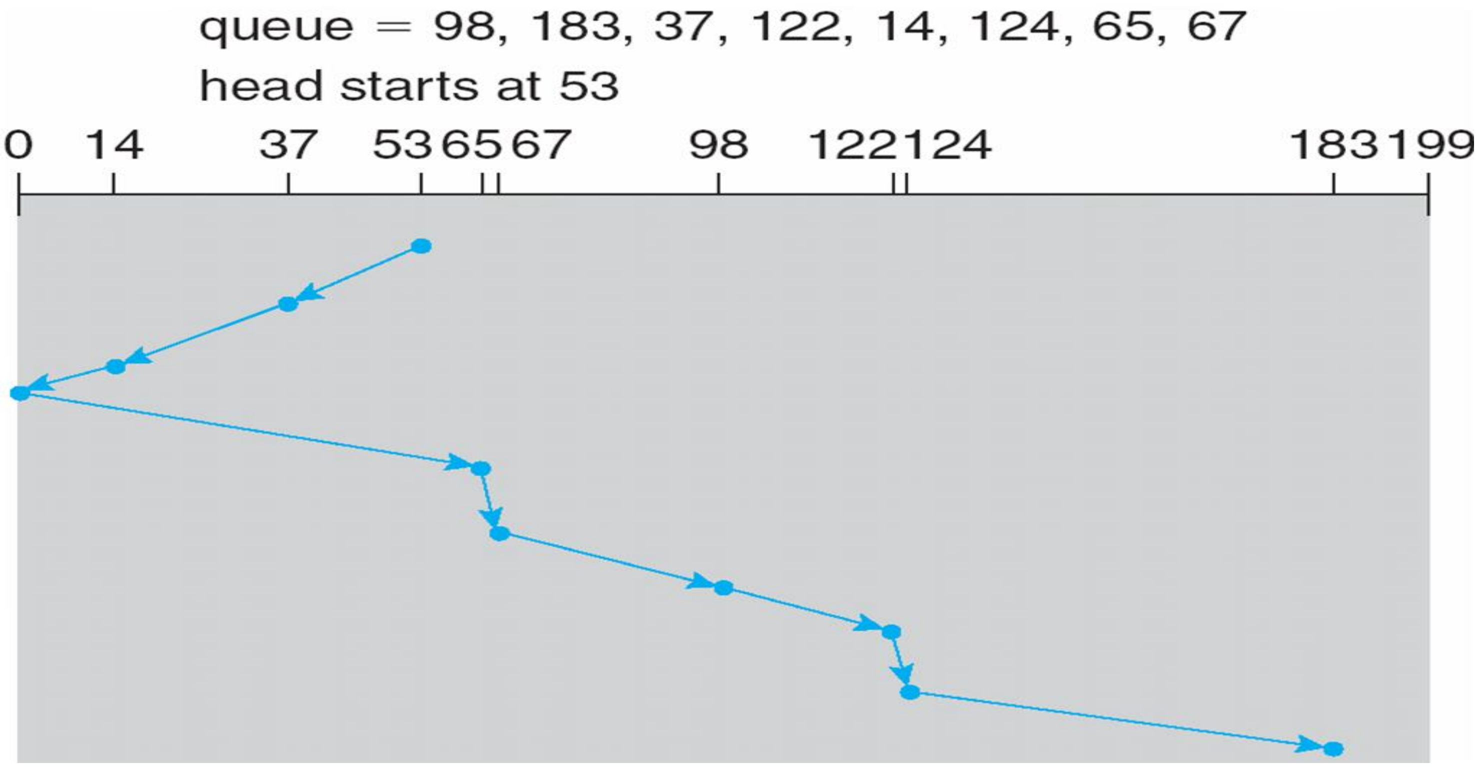


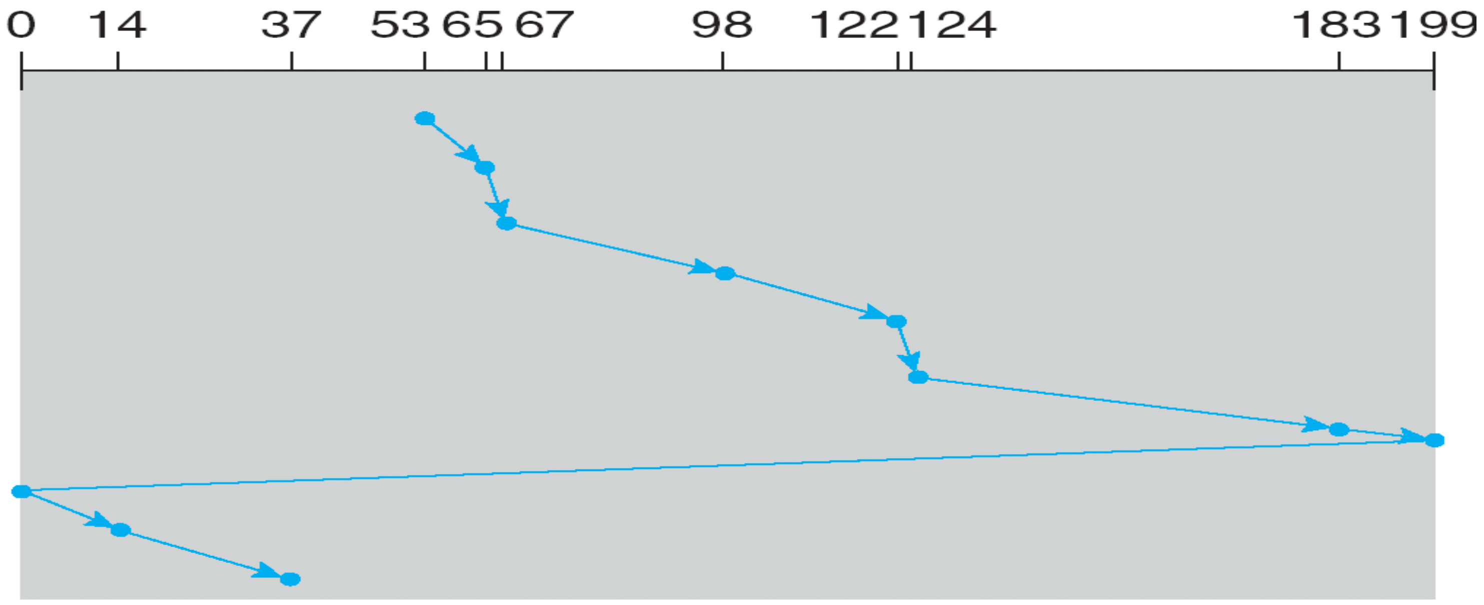
Illustration shows total head movement of 208 cylinders.

C-SCAN (Cont)

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
    - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

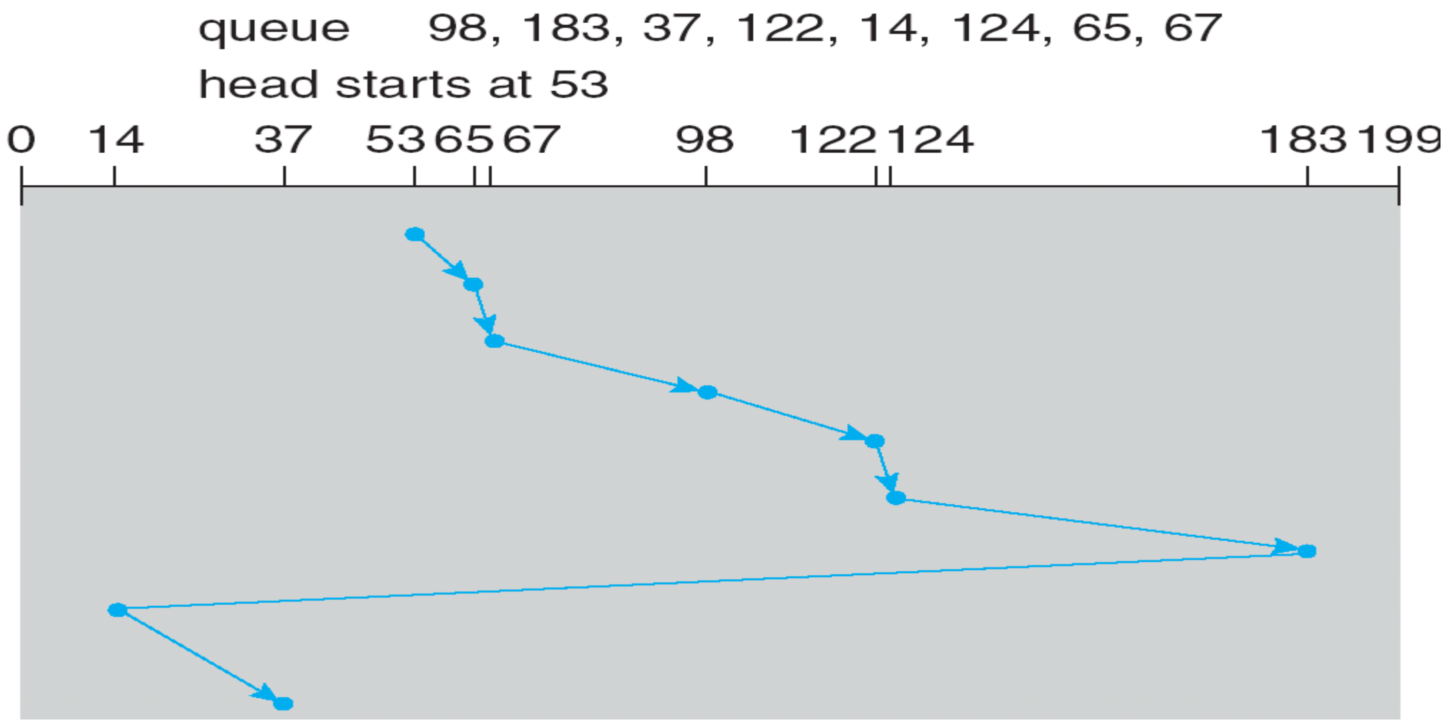
queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk



THANK YOU