TDIU11: Operating Systems

Mass Storage Systems

SGG9: chapters 10
SGG10: chapters 11

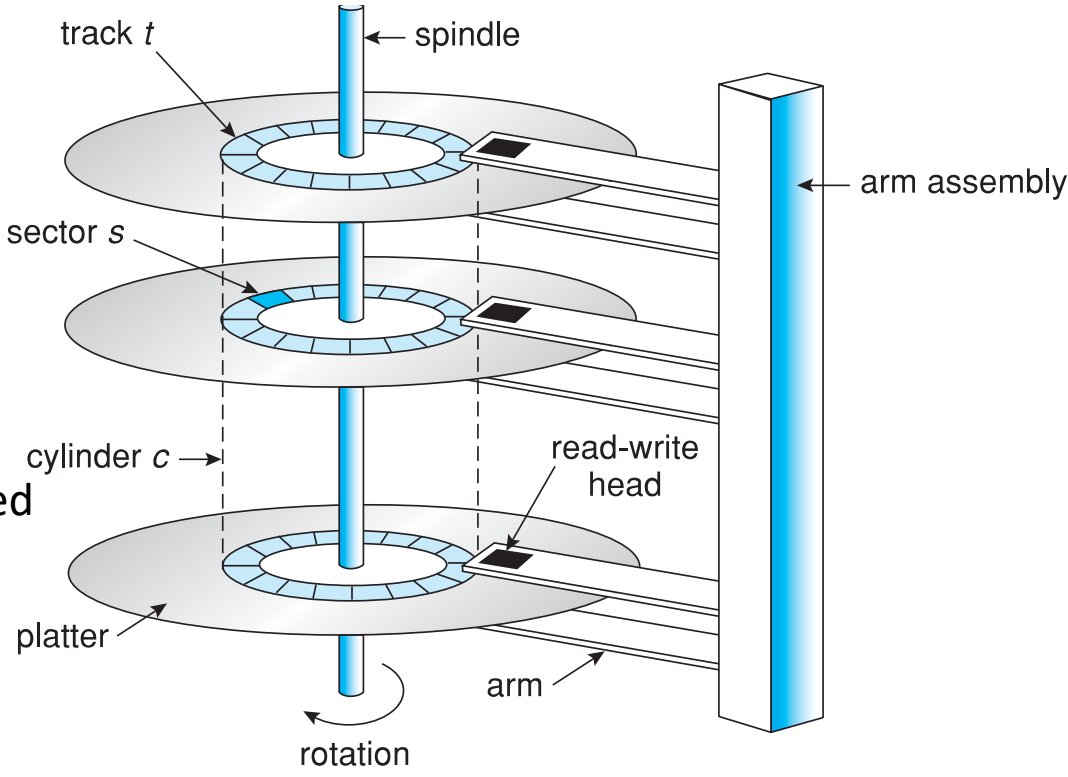
- hard disks, structure, disk scheduling

Ahmed Rezine, Linköping University

Overview of Mass Storage Structure

Magnetic disks bulk of secondary storage:

- Drives rotate at 60 to 250 times per second
- **Transfer rate** is rate at which data flow between drive and computer
- **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)



Hard Disks

- Platters from .85” to 14”: commonly 3.5”, 2.5”
- Up to 10TB per drive
- Performance
  - Transfer Rate – around 1Gb/sec
  - Seek time from 3ms to 12ms – 9ms common for desktop drives
  - Average seek time measured or calculated based on 1/3 of tracks
  - Latency based on spindle speed
    - 1 / (RPM / 60) = 60 / RPM
  - Average latency = ½ latency

Spindle [rpm]	Average latency [ms]
4200	7.14
5400	5.56
7200	4.17
10000	3
15000	2

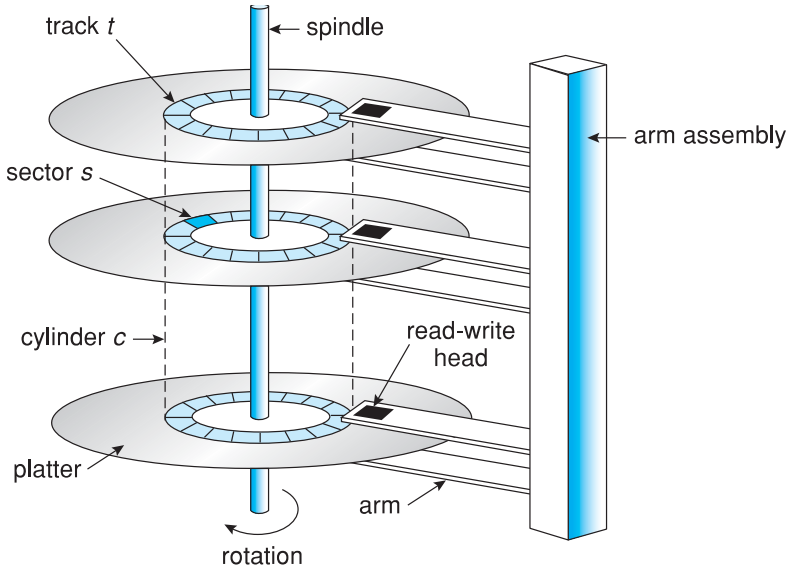
Hard Disk Performance

- **Access Latency** = **Average access time** = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead
- For example, to transfer a 4KiB block on a 7200 RPM disk with a 5ms average seek time, 1Gib/sec transfer rate with a .1ms controller overhead =
  - 5ms + 4.17ms + 0.1ms + transfer time =
  - Transfer time = (2^12 B) / (2^27 B/s) = 2^(-15) sec = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

Solid-State Disks

- Nonvolatile memory used like a hard drive
- Can be more reliable than HDDs
- May have shorter life span
- More expensive per MB
- But much faster
- No moving parts, so no seek time or rotational latency

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

Disk Scheduling

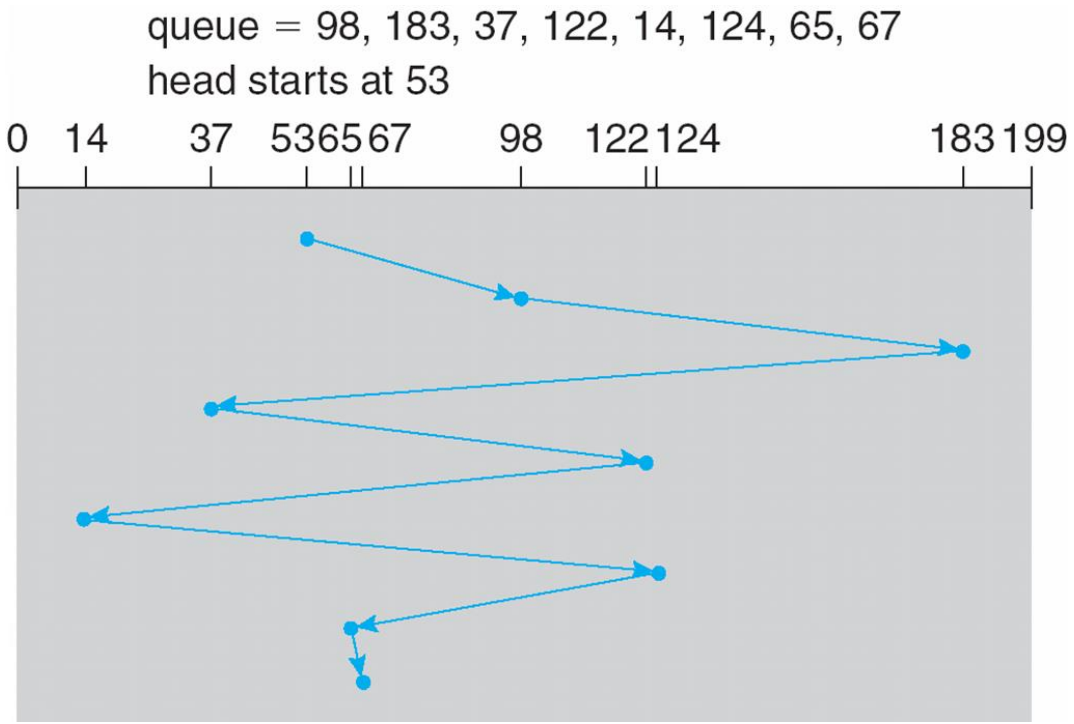
- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time ≈ seek distance
- Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Disk Scheduling (Cont.)

- There are many sources of disk I/O request
  - OS, System processes, Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists

Disk Scheduling (Cont.)

- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters

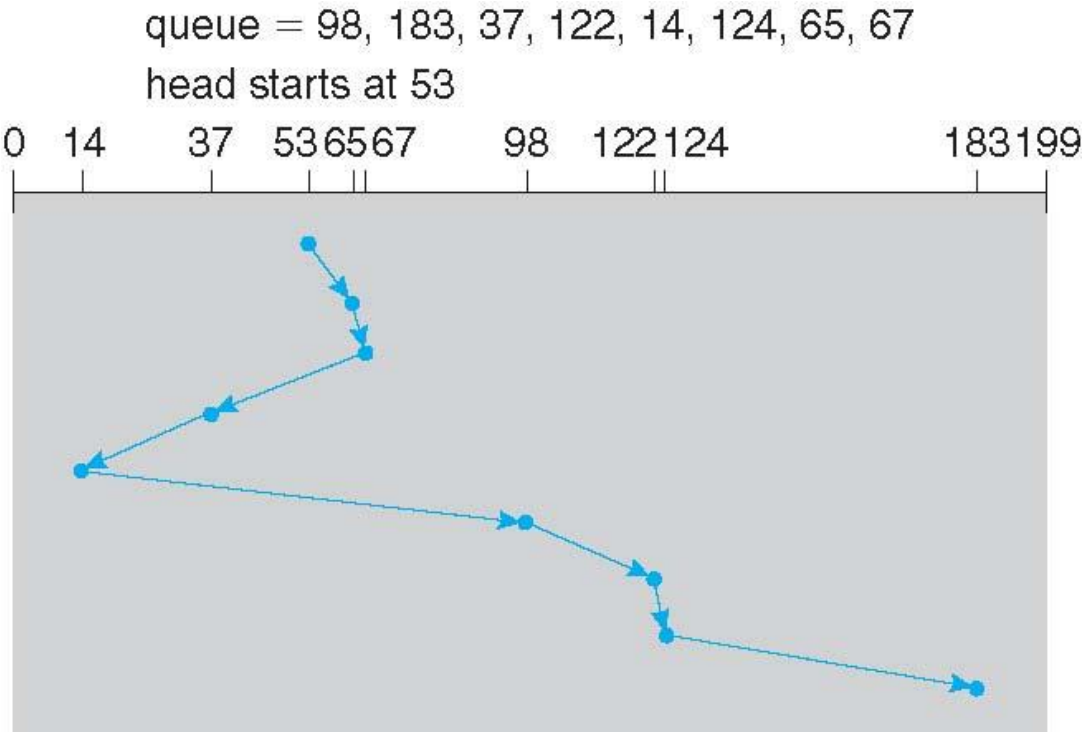


FCFS

Total head mvt: |98-53|+|183-98|+|37-183| + … + |65-67|= 640 cylinders

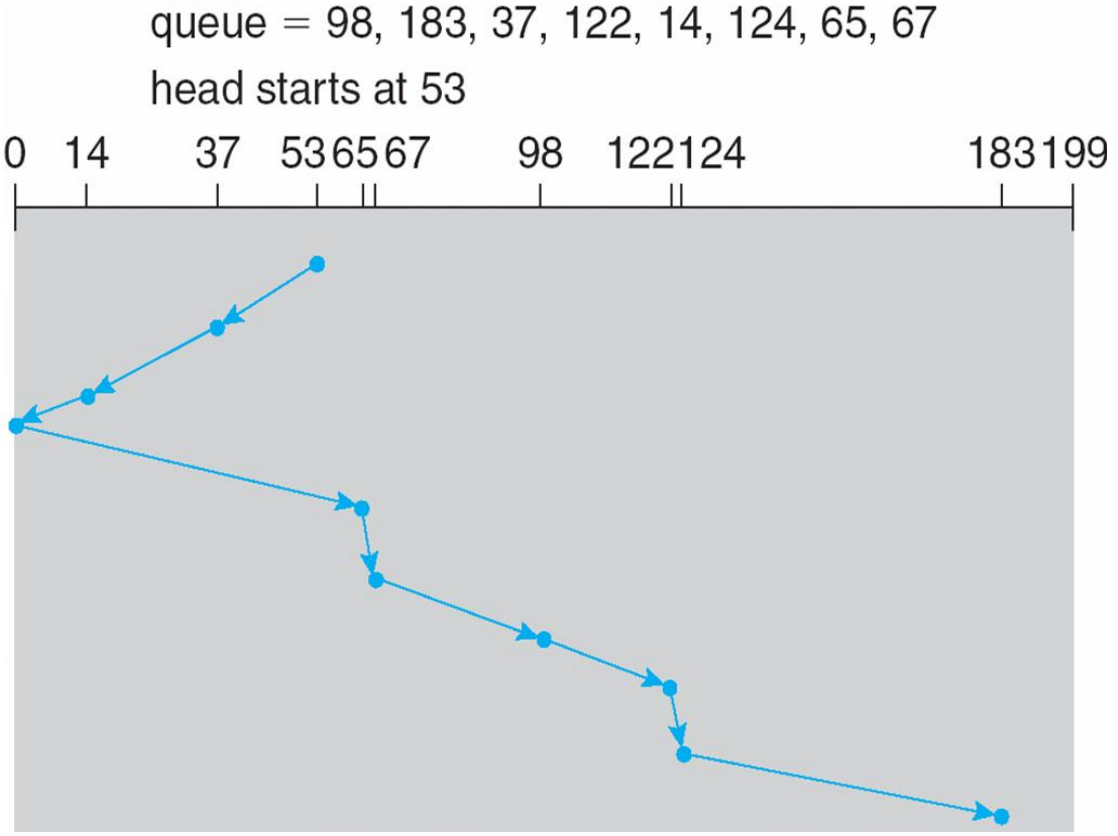
SSTF

- 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
- Illustration shows total head movement of 236 cylinders



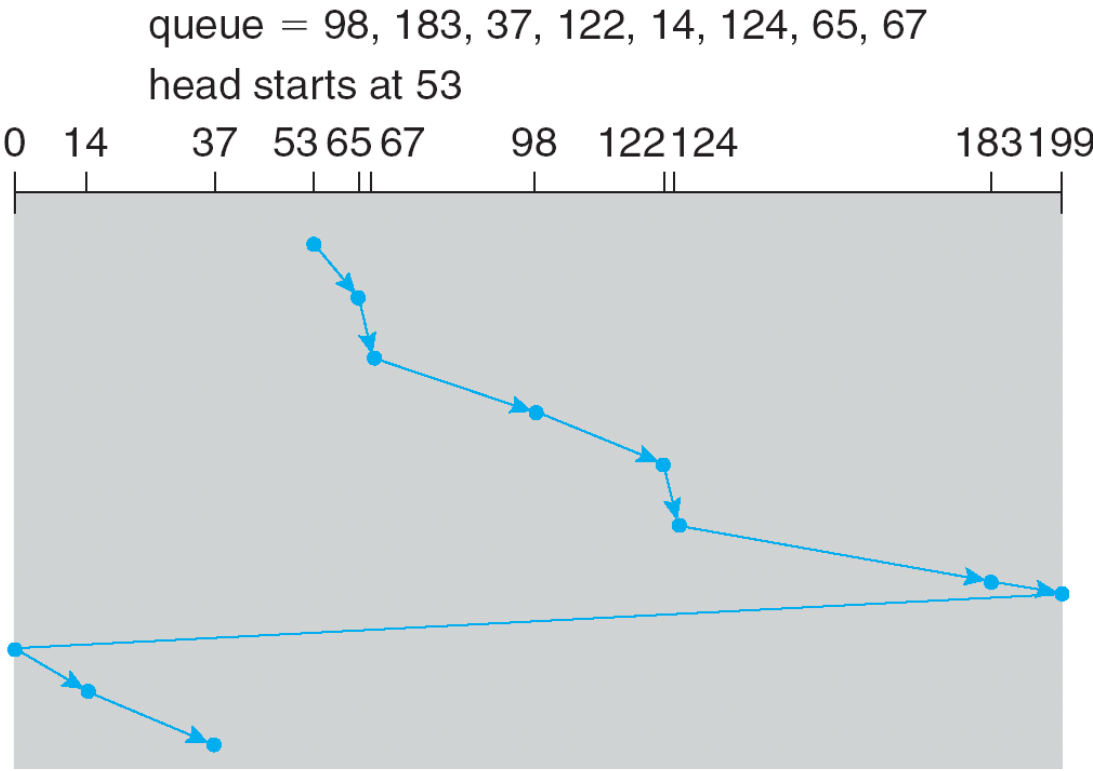
SCAN

- 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 236 cylinders if head at 53 and moving to 0.



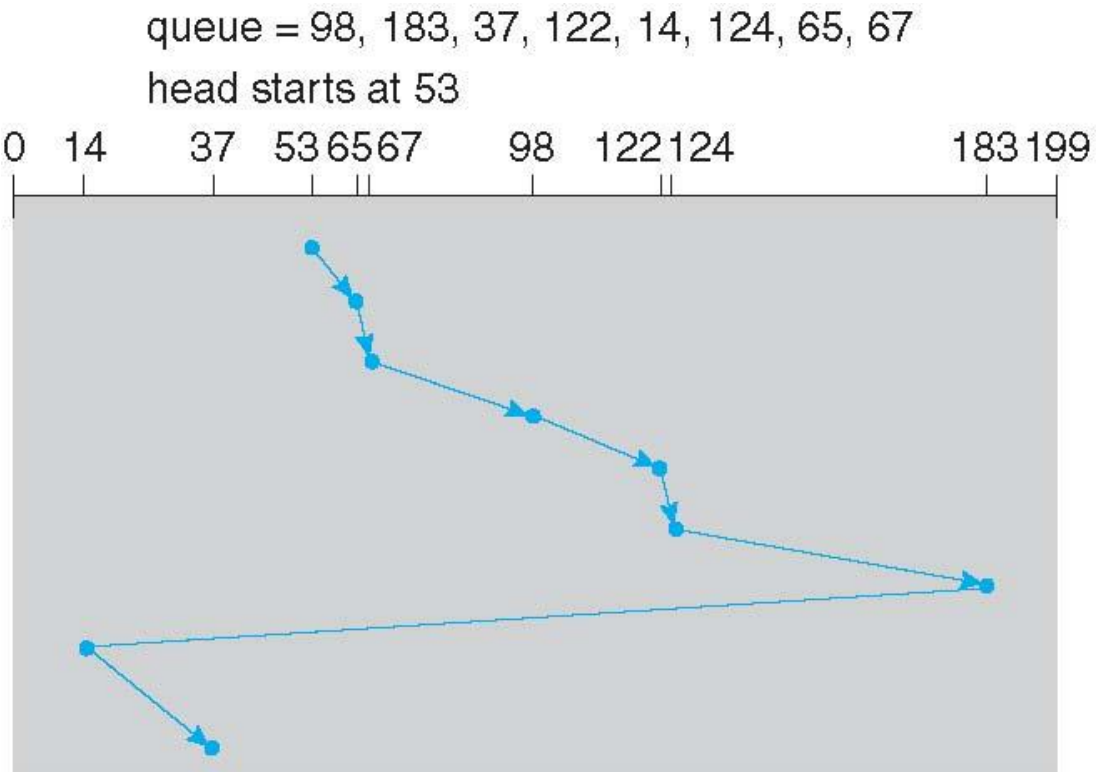
C-SCAN

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other:
  - When it reaches the other end, however, it immediately returns to the beginning of the disk
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one
- Illustration shows a mvt of 382 cylinders if head was at 53 moving to 199.



C-LOOK

- LOOK a version of SCAN, C-LOOK a 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
- Total number of cylinders?
- 322 if head at 53 and moving towards 199



Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk: Less starvation
- Requests for disk service can be influenced by the file-allocation method and metadata layout
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for the default algorithm
- Difficult to calculate how rotational latency disk-based queueing effect OS queue ordering efforts

TDIU11: Operating Systems

Recapitulation

- Interrupts, syscalls, dual mode
- Processes, Context Switch, Scheduling
- Memory allocation, segmentation, paging
- Demand paging, replacement algorithms
- Directories, inodes, allocation methods, free space
- Access matrix, security threats

Ahmed Rezine, Linköping University

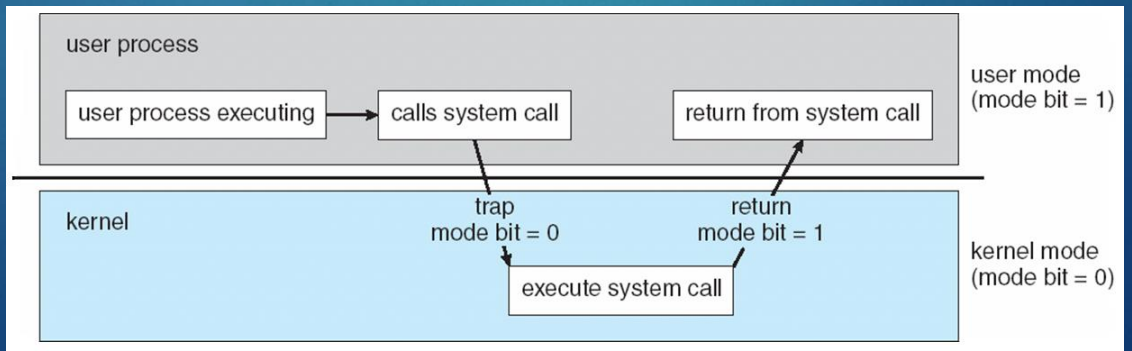
PintOS and TDIU16

- PintOS is a real and “simple” operating systems for x86
- PintOS was developed at Stanford for teaching OS courses
- Supports kernel threads, loading and running user programs and a file system.
- Stanford’s documentation is available at:
  - https://web.stanford.edu/class/cs140/projects/pintos/pintos.html
- In TDIU16, you will add functionalities to TDIU16’s PintOS:
  - https://www.ida.liu.se/~TDIU16/current/lab/index.sv.shtml
  - You will add several system calls that range from halting the system to synchronizing file manipulation

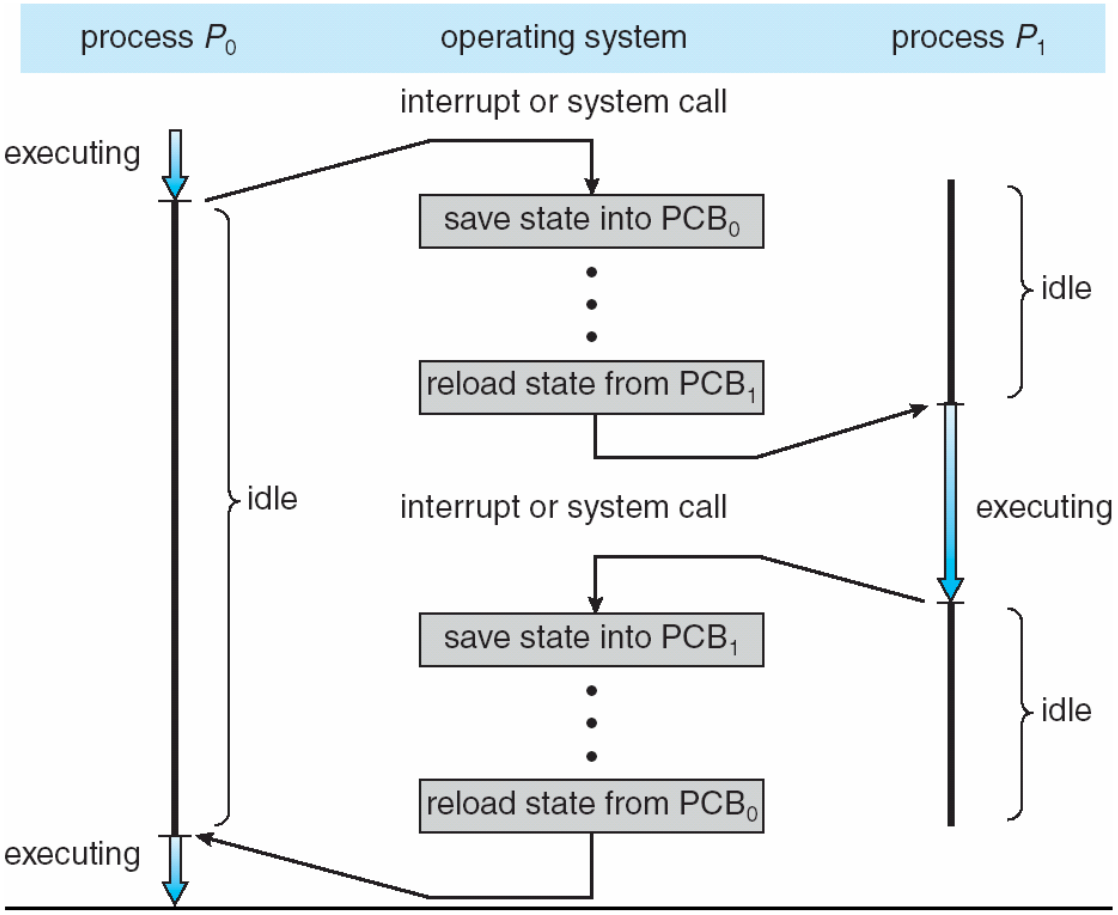
Transition from User to Kernel Mode

Timer to prevent infinite loop / process hogging resources

- Timer is set to interrupt the computer after some time period
- Keep a counter that is decremented by the physical clock.
- Operating system set the counter (privileged instruction)
- When counter zero generate an interrupt
- Set up before scheduling process to regain control or terminate program that exceeds allotted time



CPU Switch From Process to Process

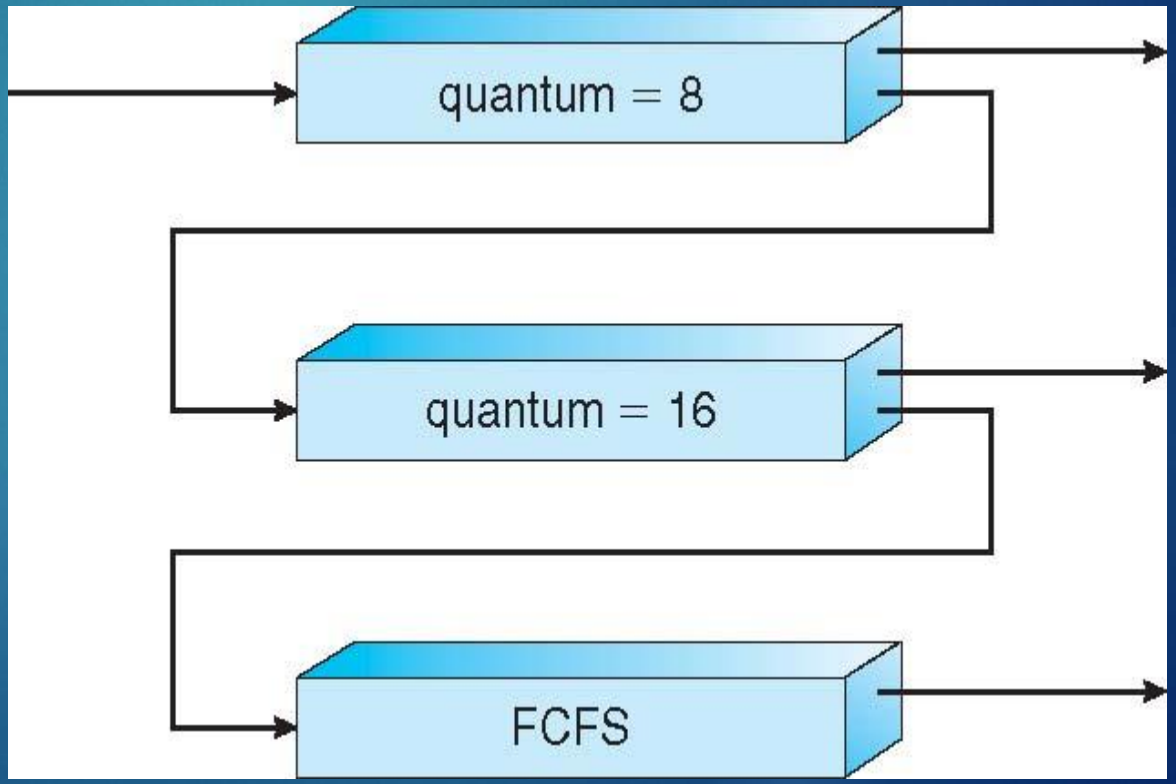


Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process: from submission to completion, including waiting to get to memory, ready queue, executing on CPU, I/O,…
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

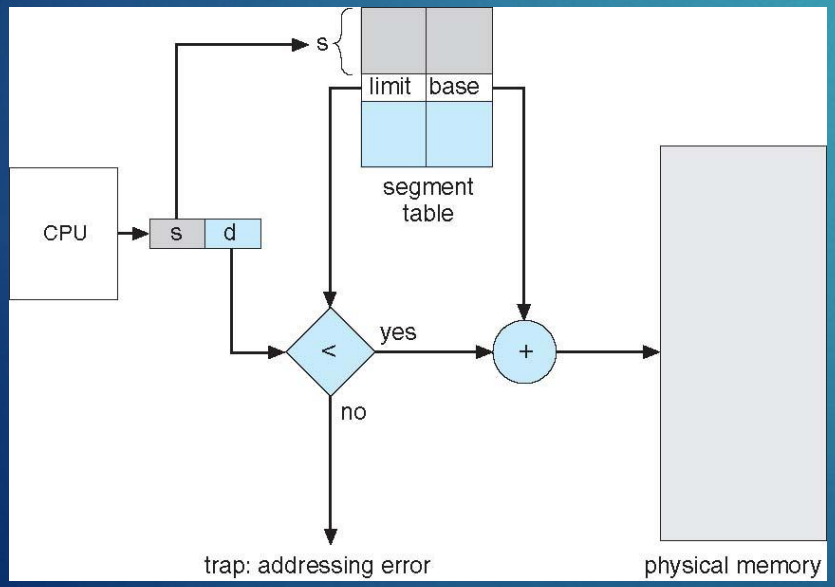
Example of Multilevel Feedback Queue

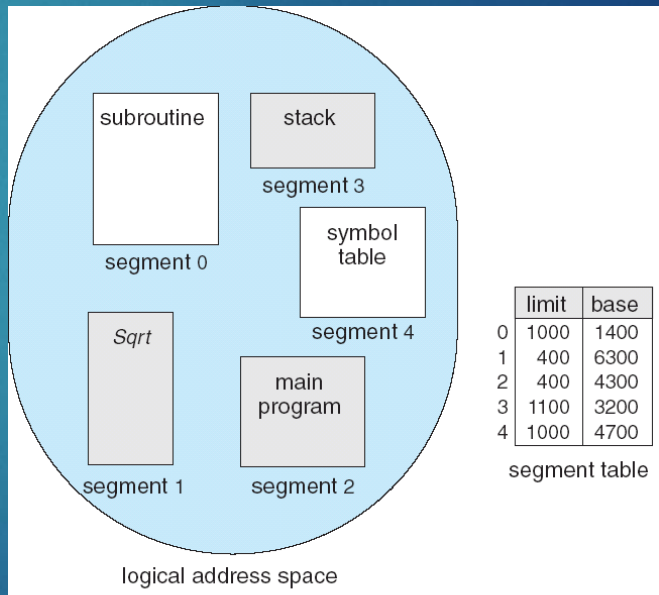
- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS



Segmentation Architecture

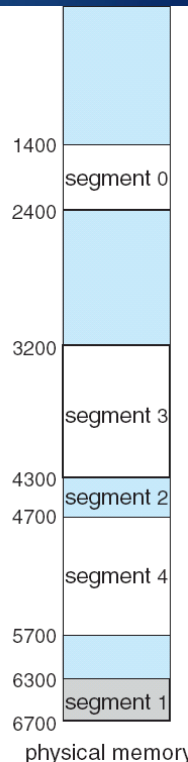
- Protection: with each entry in segment table associate:
  - validation bit = 0 ⇒ illegal segment
  - read/write/execute privileges
- Protection bits associated to segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem





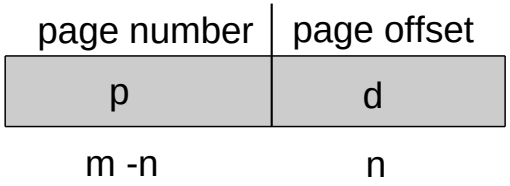
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

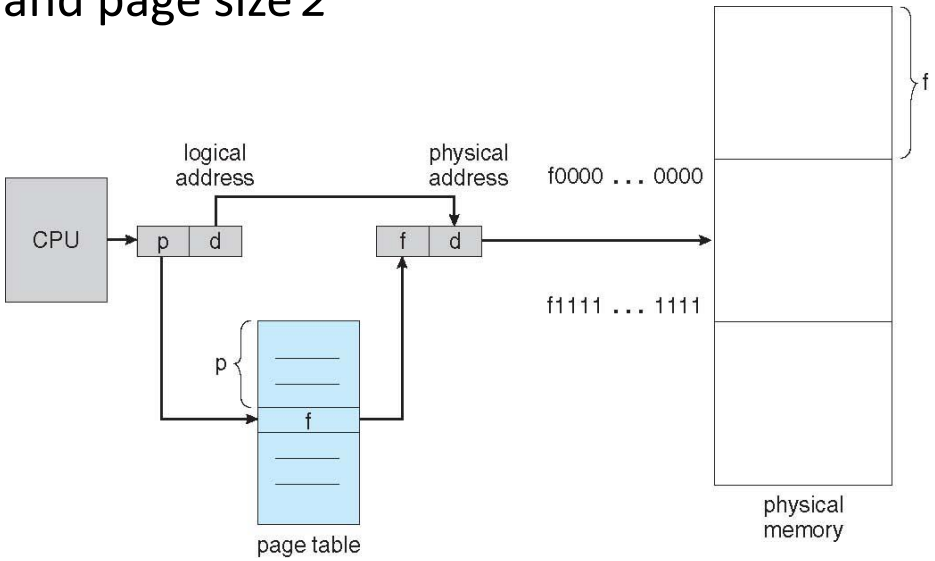
segment table



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

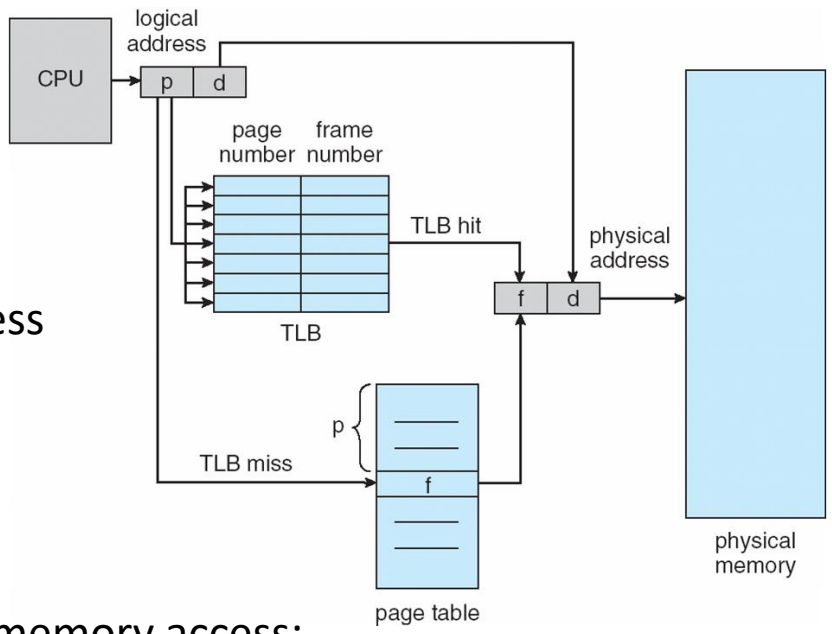




Effective Access Time

- Associative Lookup = ε time unit. Can be < 10% of memory access time

- Hit ratio = α. Percentage of times a page number is found in the associative registers



- If α = hit ration, ε = TLB search, t = memory access
- **Effective Access Time** (**EAT**):

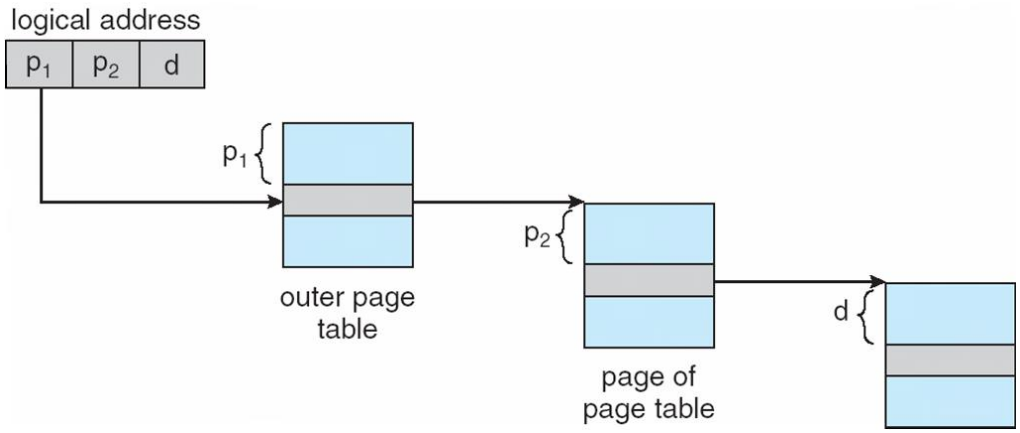
  $EAT = (t + \varepsilon)\,\alpha + (2t + \varepsilon)(1 - \alpha)$

  $= 2t + \varepsilon - \alpha\, t$

- If α = 80%, discarding ε = TLB search, 100ns for memory access:
  - EAT = 2 x 100 - 0.80 x 100 = 120ns
- Consider more realistic hit ratio -> α = 99%, 100ns for memory access.
  - EAT = 0.99 x 100 + 0.01 x 200 = 101ns

Two-Level Paging Example

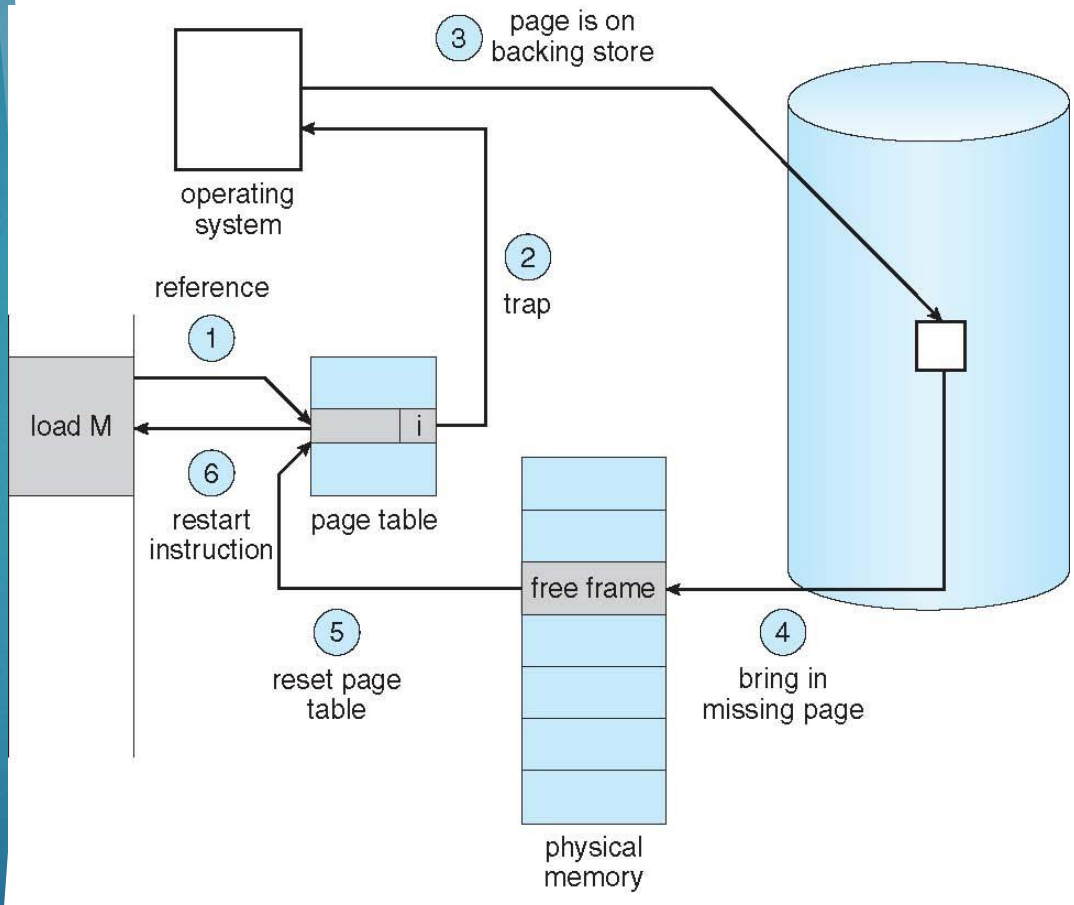
- A logical address (on 32-bit machine with 4KiB page size) has:
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits
- Since the page table is paged, the page number is further divided into p_1 and p_2.
- Where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table (**forward-mapped page table)**



page number		page offset
p_1	p_2	d
10	10	12

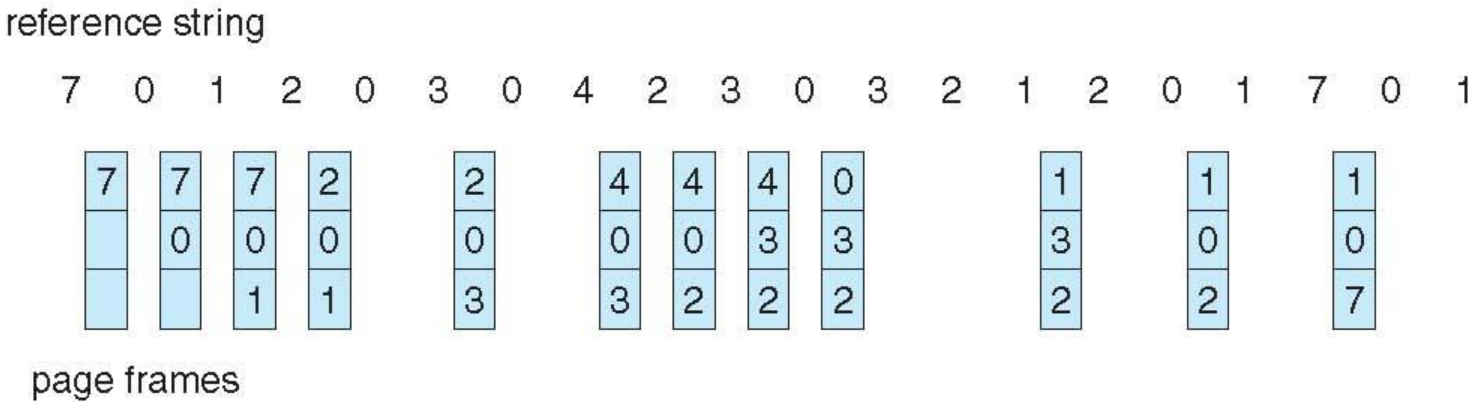
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:
- **page fault**
- Operating system looks at another table to decide:
  - Invalid reference ⇒ abort
  - Just not in memory
- Find free frame
- Swap page into frame via scheduled disk operation
- Update tables to indicate page now in memory
  Set validation bit = **v**
- Restart the instruction that caused the page fault



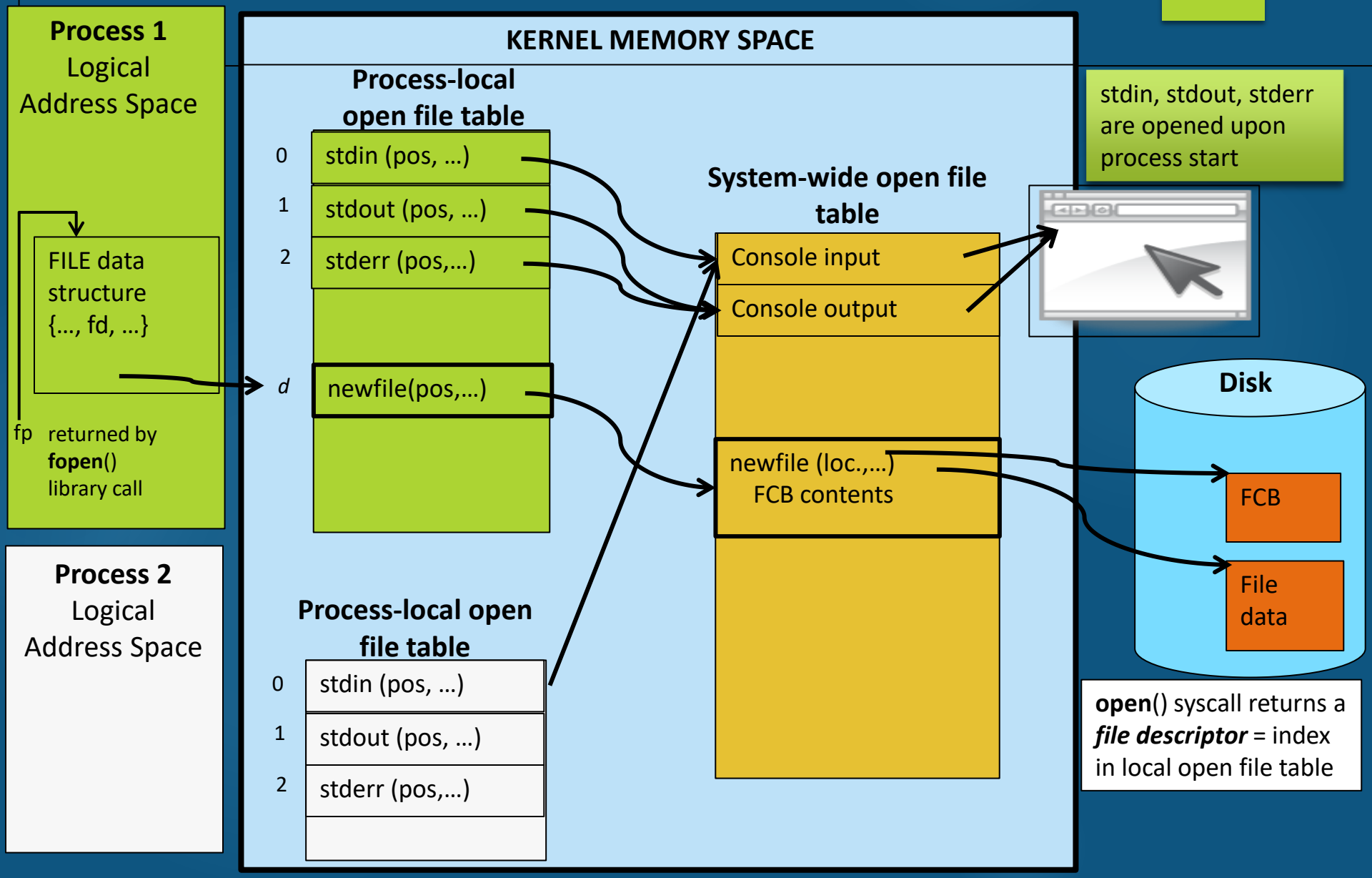
Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page
- 12 faults – better than FIFO but worse than OPT



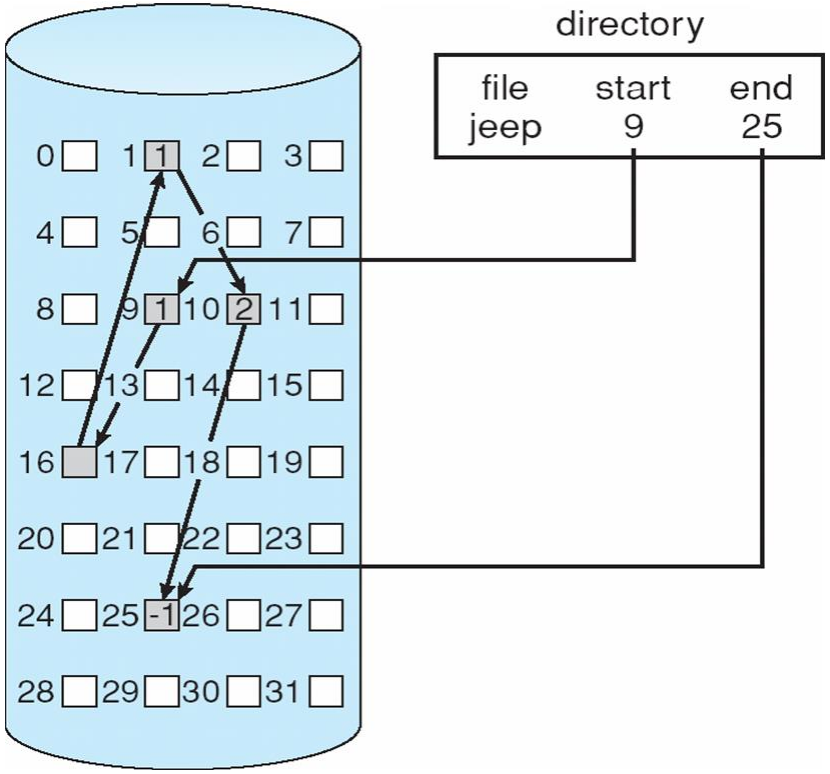
- Generally good algorithm and frequently used
- But how to implement?

File descriptors and open file tables



Allocation Methods - Linked

- **Linked allocation** – each file a linked list of blocks.
  - No external fragmentation. No need for compaction.
  - Each block contains pointer to next block
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation



Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
- **Bit vector** or **bit map** (***n*** blocks):
- CPUs have instructions to return offset within word of first "1" bit
- Bit map requires extra space:

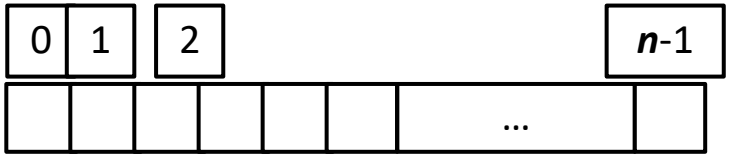
block size = 4KB = 2^{12} bytes

disk size = 2^{40} bytes (1 terabyte)

n = $2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

if clusters of 4 blocks -> 8MB of memory

- Easy to get contiguous files



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

Principles of Protection

- **Principle of least privilege**
  - Programs and users are given just enough **privileges** to perform their tasks
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**
- **"Need to know" principle**: at any time, a process should only be able to access those ressources it currently requires.

Domain Implementation (UNIX)

```
ll /bin:
  ...
  28 -r-s--x--x   1 root    lp      28092 Jan 23  2005 lp*
```

- Domain = user-id
- Domain switch accomplished via file system
  - Each file has associated with it a domain bit (setuid bit)
  - When file executed and setuid on, then user-id is set to owner of the file
  - When execution completes user-id is reset
  - setgid, sticky bit
- Domain switch accomplished via passwords
  - su command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
  - sudo command prefix executes specified command in another domain (if original domain has privilege and password given)

Authentication

- **Unix**: Passwords are kept in files
  - earlier /etc/passwd, now separate file e.g. /etc/master.passwd or /etc/shadow
- Each password is encrypted with a one-way crypto + a ”salt”
  - salt configures the en-/decryption algorithm → extends the password
- To verify a password:
  - Lookup the salt for the user
  - Encrypt the submitted password
  - Compare result with what is stored
- Attack: Steal the password file, generate passwords brute force, or use a dictionary…