Disk Scheduling

- **Idea**
  - Put N incoming disk requests in the queue
  - Execute them in the optimal order
- **Goal**
  - Optimizing disk I/O performances
- **Disk Scheduling placement:**
  - External:
    - In the kernel
    - In the disk driver
  - Internal
    - In the disk drive itself (command queuing)

Disk Scheduling

- Seek based algorithms
- Positional based algorithms

Disk Scheduling

- **Full Knowledge based Algorithms**
  - **Based on CHS**
  - **Algorithms include:**
    - Zones
    - Defect management
    - Reserved spaces
    - Rotational capability
    - Disk drive caching
  - **Seek based algorithms**
  - **Position based algorithms**
  - **Disk caching**
    - **Included in the algorithm**
    - **excluded in the algorithm**

- **LBN-Based Algorithms**
  - **Only seek based algorithms**

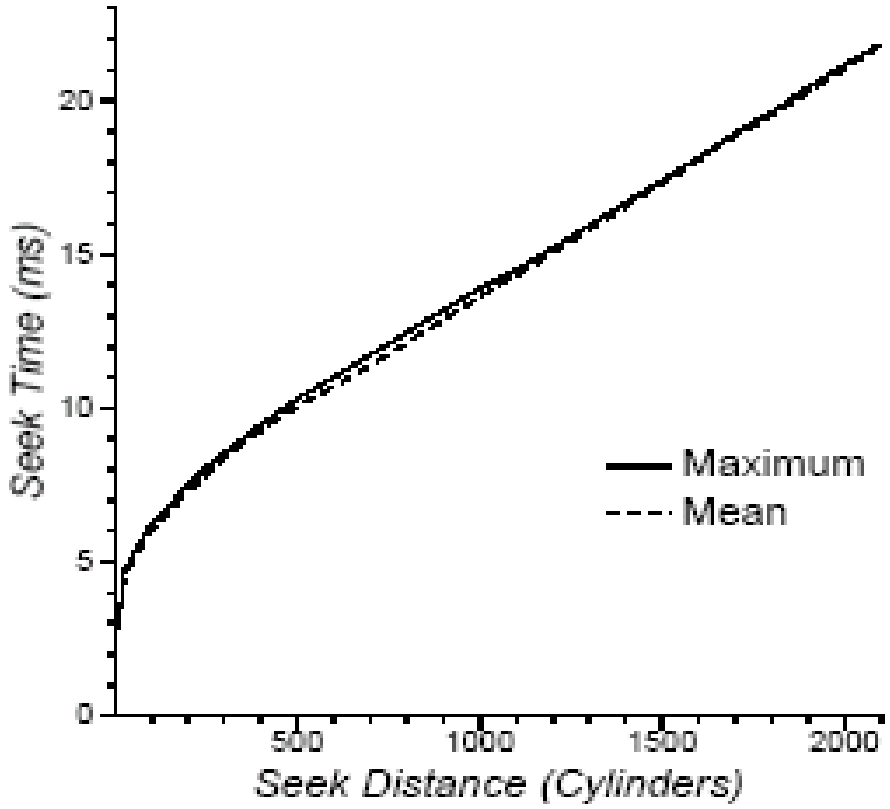
Seek based algorithms

- FIFO (FCFS)
- SSTF
- SCAN C-SCAN
- LOOK C-LOOK
- VSCAN(x.x)

Disk Scheduling-seek based

- The **operating system is responsible for using hardware efficiently —**
  - **for** the **disk drives**,
  - **this means having** a **fast access time** and **disk bandwidth**.
- **Access time has two major components:**
  - **Seek time**
    - **is** the **time**
    - **for** the **disk are to move** the **heads**
    - to the **cylinder containing** the **desired sector**.
  - **Rotational latency**
    - is the **additional time**
    - **waiting for** the **disk**
    - **to rotate** the **desired sector to** the **disk head.**
- **Minimize seek time**
- **Seek time ≈ seek distance =F(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**.

Seek time



$$T_s = \begin{cases} 3.24 + 0.40\sqrt{s} & \text{if } s \le 383 \\ 8.20 + 0.0075\,s & \text{if } s > 383 \end{cases}$$

Disk Scheduling (Cont.)

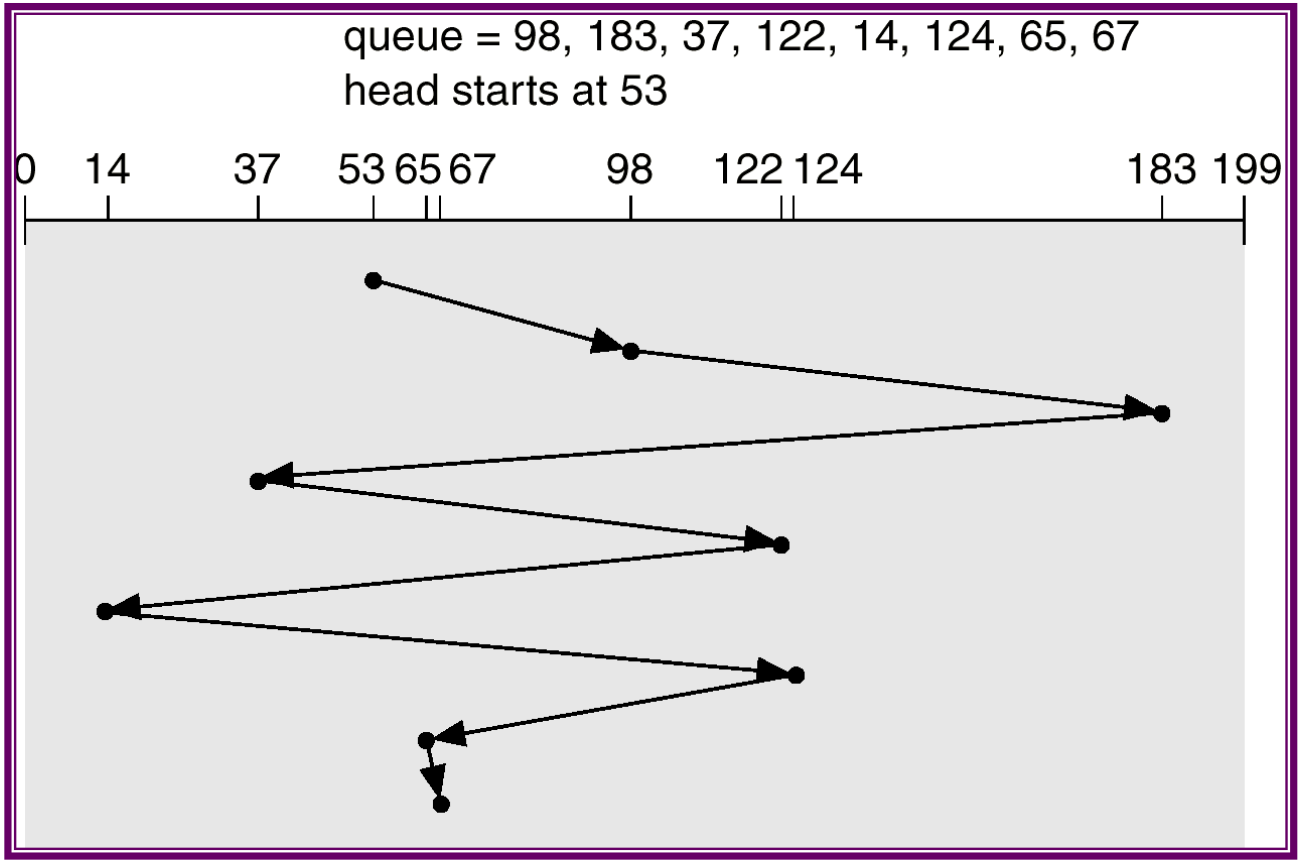
- **Several algorithms exist**
  - **to schedule**
  - the **servicing of disk I/O requests**.
- **We illustrate them with** a **request queue (0-199).**

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

Head pointer 53

FCFS (First Come First Served)

Illustration shows **total head movement of 640 cylinders**.



FCFS features

- **Performances**
  - ☞ **poor**
- **Starvation**
  - ☞ **Fairness, small starvation**

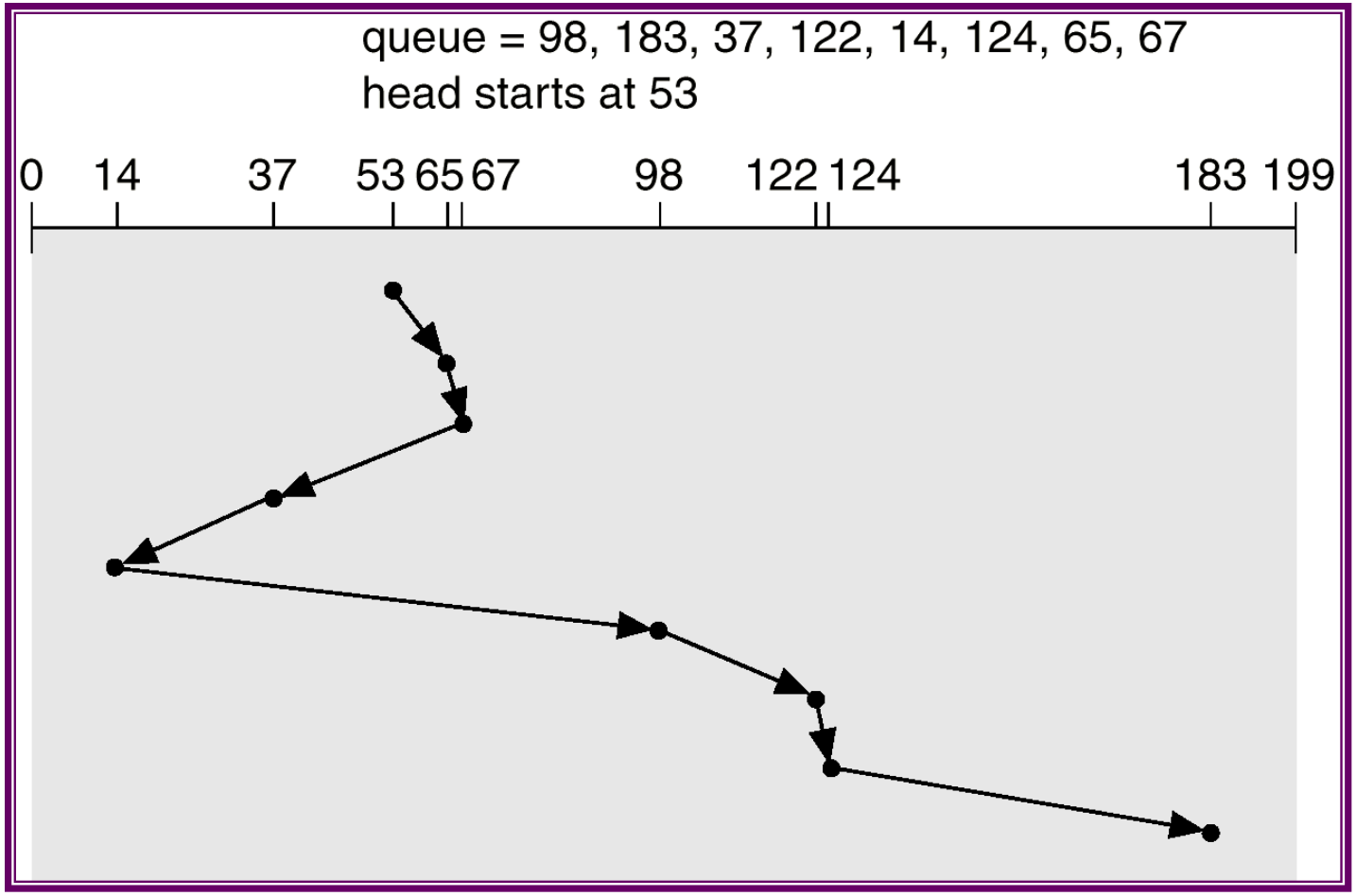
SSTF (Shortest Seek Time First)

- **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.**

SSTF (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

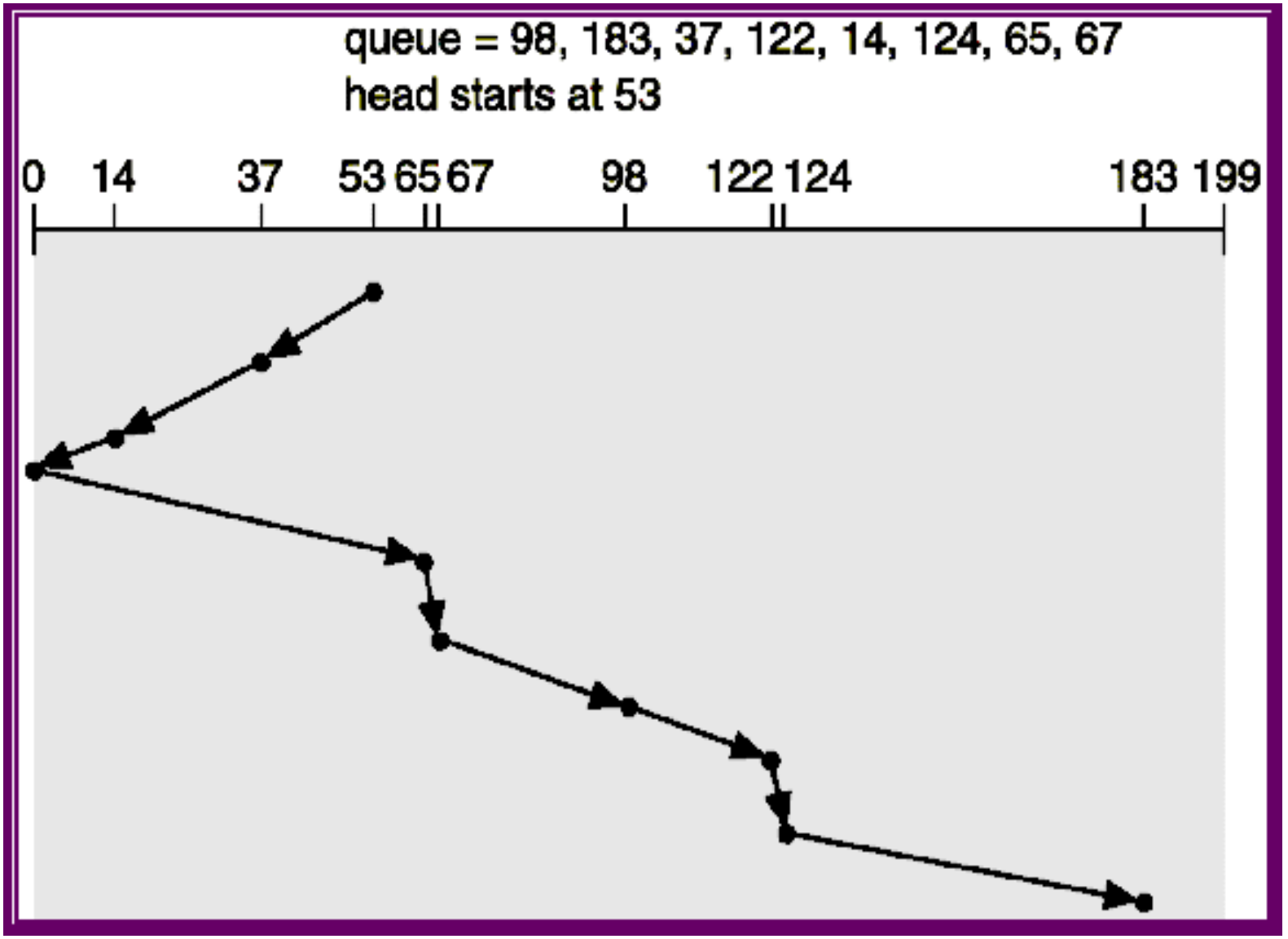
SSTF features

- **Performances**
  - Good to very good
- **Starvation**
  - Big starvation
  - Too long latency in small disk zones

SCAN - Elevator

- 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**.
- **Sometimes called** the **elevator algorithm**.
- **Illustration shows total head movement of 208 cylinders**.

SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

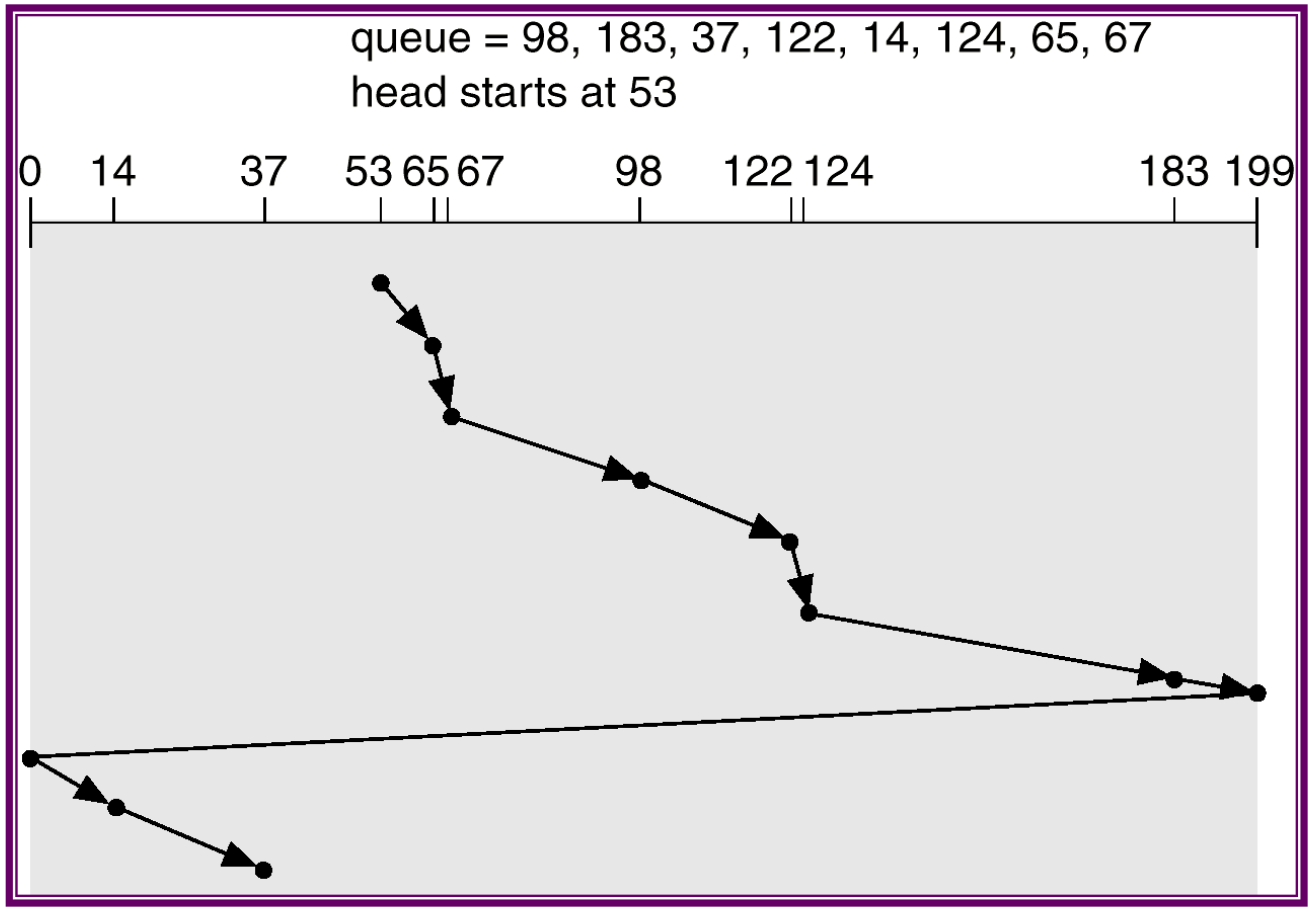
C-SCAN

- **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.**

C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

SCAN, C-SCAN features

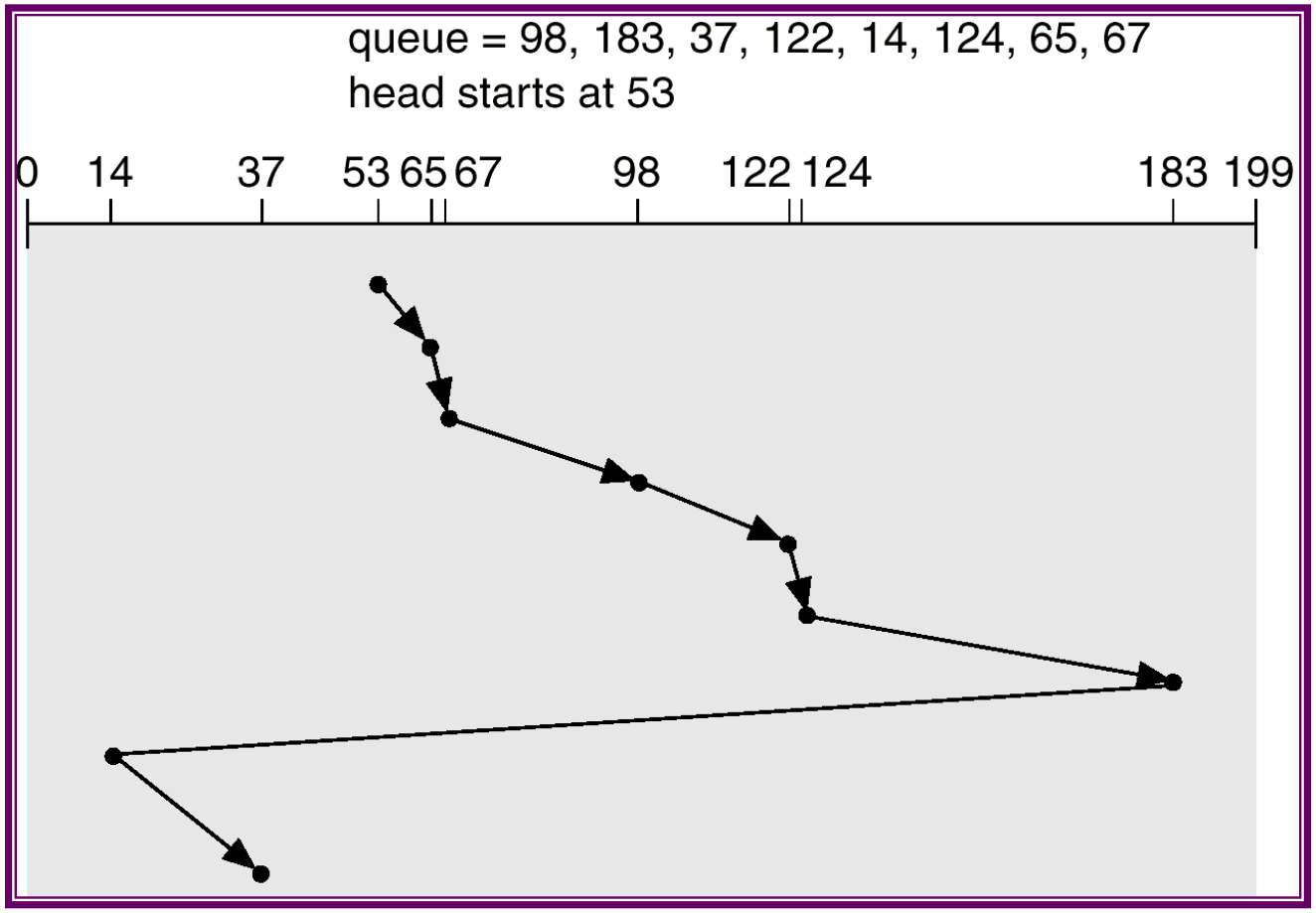
- **Performances**
  - SCAN good
  - C-SCAN: one full stroke without servicing
- **Starvation**
  - SCAN:
    - central zone is privileged related to outer zones
  - C-SCAN:
    - small starvation

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**.

C-LOOK (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

Look, C-Look features

- **Performances**
  - Look: good to very good
  - C-Look: one big stroke without servicing
- **Starvation**
  - Look:
    - central zone is privileged related to outer zones
  - C-Look:
    - small starvation

VSCAN(R)

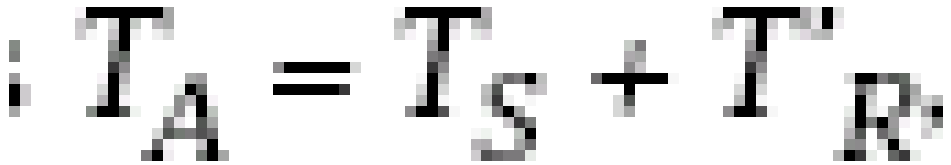
- **Geist and Daniel have proposed a continuum of algorithms called V(R), where *R* is a parameter**
- **SSFT----VSCAN(R)----SCAN**
- The **idea is to pick** the **next request according to SSTF**,
- **except**
  - **to add** a **penalty of R times**
  - the **total number** of **cylinders**
  - **for reversing direction.**
- **V(0) is SSTF**
- **V(1) is SCAN**
- **They suggest V(0.2)**
  - **as a good compromise**
  - **that performs better than SCAN**,
  - **but avoids** the **high variance and starvation difficulties of SSTF** .

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**.
- **Performance depends on** the **number** and **types of requests**.
- **Requests for disk service can be influenced by** the **file-allocation method.**
- 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.**

Position-based algorithms

- **Rotationally-sensitive scheduling algorithms**



- **Scheduling with Full Knowledge of LBA-CHS**
  - Zones
  - Defect management
  - Reserved spaces
  - Rotational capability
  - Disk drive caching
- **Names:**
- **SAFT = Shortest Access Time First**
- **SPTF = Shortest Position Time First**

Disadvantage of SPTF

- Full knowledge of LBA to CHS mapping
- Time intensive operation
- Dominant starvation (like SSTF)

Reducing starvation

- **Batch algorithms**
- **Aged algorithms**

Batch algorithms

- **Batch algorithms are ones**
  - that **prevent starvation**
  - by **temporarily preventing new requests**
  - **from joining the queue**
  - and
  - **thereby delaying old ones indefinitely**.
- The **batch algorithms described here can be used**
  - **continuously**,
  - or
  - in a **two-mode fashion**:
    - **invoked only when starvation has been observed**
    - **to bring it into check** and **prevent further occurrences**.
- The **two-mode behavior attempts**
  - **to benefit from** the **high throughput of SATF**,
  - **while limiting the damage**
  - **caused by starvation**
  - **by use of the batch technique**.

BSAFT

- **BSAFT = Batched Shortest Access Time First**
  - **simplest** batched algorithm (BSATF).
  - **“no new requests”** rule
  - **It operates by:**
    - **create** a **queue**
    - **processing all** the **requests**
    - **that are currently on** the **queue to completion**
    - **before admitting any more**.

LBSATF

- **LBSATF = Leaky Batched Shortest Access Time First**
- **relaxed** the "**no new requests**" **rule.**
- **Deadline** = **projected end time** for **complete of queue**
  - If a **new request arrives**,
    - **it is added to** the **batch**
    - **if** a **schedule can be found**
    - **that will complete it**
    - **as well as all the existing requests**
    - **before** the **existing deadline**.
  - **If not,**
    - the **request is put aside**
    - **until** the **next batch** **is taken**.

Aged algorithms

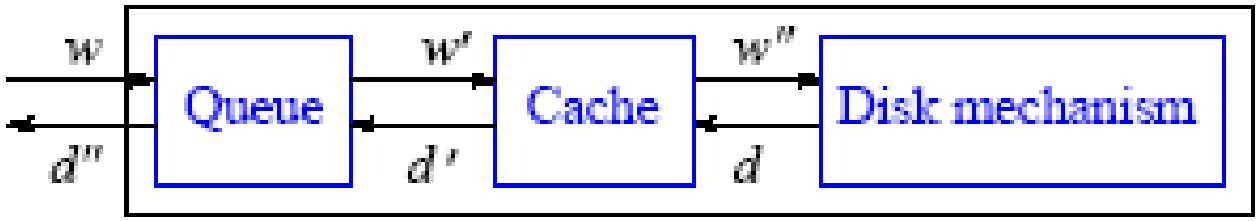
- **SPTF is highly susceptible to request starvation**.
- **ASPTF denoted as Aged Shortest Positioning Time First**.
- **ASPTF adjusts each positioning delay prediction** (**Tpos**)
  - **by subtracting** a **weighted value**
  - **corresponding to** the **amount of time**
  - the **request has been waiting for service** (**Twait).**
- The **resulting effective positioning delay** (**Teff**)
  - **is used**
  - **in selecting** the **next request:**

$$T_{eff} = T_{pos} - (W * T_{wait})$$

Full Knowledge with disk caching

- **modified versions**
  - **which track the contents of the on-board cache**
  - and
  - **estimate a positioning time of zero**
  - **for any request that can be satisfied (at least partially) from the cache.**

- **The resulting algorithms are denoted as**
- **SPCTF Shortest Positioning (w/Cache) Time First**
- **(ASPCTF) Aged Shortest Positioning (w/Cache) Time First.**



Comparison of algorithms

- **LBA based** and **full knowledge algorithms**
  - **have similar performances**
  - **in the case of seek based algorithms**
- C-LOOK,
  - **which always schedules requests** in **logically ascending order**,
  - **best exploits** the **prefetching cache** for **workloads**
  - **with significant read sequentially**.
- **For random workloads**,
- **C-LOOK has been shown** to **provide slightly inferior performance**
- **to other seek-reducing algorithms** (e.g., SSTF and LOOK).
- **In addition,** the **LBN-based C-LOOK algorithm**
  - **is straightforward** and
  - **relatively simple to implement**.

SPFT conclusions

- **SPTF algorithms achieve higher performance**.
- The **use** of **such algorithms requires**
  - **thorough knowledge** of the **disk's current state**
  - **as well as** the **management schemes**
    - **employed by the disk drive firmware**.
- The **computational cost is very high**

C-LOOK conclusions

- **Best algorithm for modern disk drives, probably**
- **Why?**
  - **always schedules requests in logically ascending order**
  - **best exploits** the **disk cache**
- **LBN C-Look**
  - **Easy implementation**

Technology trend – command queuing

- command queuing
- =
- put disk scheduling

- into disk drive, itself