- Disk I/O time = seek + rotational delay + transfer.
- Transfer time: (bandwidth) time to move the bytes from the disk to memory.
- Rotational delay depends upon how fast the disk spins.
- Rotational delay: (latency) time for the sector to rotate underneath the head.
- Seek: (latency) position head over a track/cylinder. The seek time depends on how

To read or write a disk block:

---

Today: Secondary Storage

---

Approaches to Improving Performance:

- Increase physical memory to reduce amount of time paging and thereby improve CPU utilization.
- Improve CPU utilization.
- Increase the number of devices to reduce contention for a single device and thereby
- Offload computation from the main CPU by using DMA controllers.
- Reduce interrupt frequency by using large data transfers.
- Reduce data copying by caching in memory

Last Class: I/O Systems

---

I/O is expensive for several reasons:

- I/O is typically supported via system calls and interrupt handling, which are slow.
- I/O is typically supported via system calls and interrupt handling, which are slow.
- I/O is typically supported via system calls and interrupt handling, which are slow.
- I/O is typically supported via system calls and interrupt handling, which are slow.
If our sector size is too large, we will have lots of internal fragmentation.
* If the sector size is too small, we will have a low transfer rate because we will need
  * to perform more seeks for the same amount of data.
  * We should also pick our sector size carefully:
    - Place commonly-used files where on the disk?
    - Lay out data on disk so that related data are on nearby tracks.
    - Schedule disk operations to minimize head movement
    - Spin disk faster
    - Make disks smaller

* Key: to get the quickest disk response time, we must minimize seek time.

## Access Time

### and rotational latency:

- Key: to get the quickest disk response time, we must minimize seek time.

## Typical Disk Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer size</td>
<td>4 MB</td>
</tr>
<tr>
<td>Transfer rate</td>
<td>200 MB/sec</td>
</tr>
<tr>
<td>Average rotational latency</td>
<td>4.17 ms</td>
</tr>
<tr>
<td>Average seek time</td>
<td>6.5 ms</td>
</tr>
<tr>
<td>Throughput</td>
<td>512 KB/sec</td>
</tr>
<tr>
<td>Sectors per track</td>
<td>134</td>
</tr>
<tr>
<td>Tracks per surface</td>
<td>832</td>
</tr>
<tr>
<td>Platters per pack</td>
<td>10</td>
</tr>
<tr>
<td>Platters per pack</td>
<td>36.4 GB</td>
</tr>
<tr>
<td>Good PC disk</td>
<td>High-end disk</td>
</tr>
</tbody>
</table>
When would you expect this algorithm to work well?

- Distance of seeks:
- Order of seeks:

1. FCFS - Service the requests in the order that they come in

Example requests: 65, 40, 18, 78

**FCFS Disk Head Scheduling**

4. C-SCAN circular scan algorithm (0 to 100, 0 to 100, ...)

3. SCAN algorithm (0 to 100, 0 to 100, 0 to 100, ...). If there is no request between current position and the extreme (0 or N), we don't have to seek there.

2. Shortest seek time first (SSTF)

1. First-come, first-served (FCFS)

Idea: Permute the order of disk requests from the order that they arrive from the user to an order that reduces the length and number of seeks.

**Disk Head Scheduling**
Just as far as the last request:
- Simple optimization does not go all the way to the edge of the disk each time, but
- Requires a sorted list of requests.
- Distance of seeks:
- Order of seeks: assuming the head is currently moving to lower numbered blocks:

```
0       10      20     30     40      50     60      70     80      90   100
0       10      20     30     40      50     60      70     80      90   100

SCAN Disk Head Scheduling
```

Problems?
- Is it optimal?
- Is it efficient enough?
- Can implement this approach by keeping a doubly linked sorted list of requests.
- Distance of seeks:
- Order of seeks:

```
0       10      20     30     40      50     60      70     80      90   100
0       10      20     30     40      50     60      70     80      90   100

SSTF: always go to the next closest request.
```

```
SSTF Disk Head Scheduling
```
Improving Disk Performance using Disk Interleaving

- More uniform wait times for requests. Why?
  - Distance of seeks.
  - Order of seeks.

C-SCAN: Circular scan algorithm (0 to 100, 0 to 100, ...)

C-SCAN Disk Head Scheduling
- Is SCAN or C-SCAN faster?
- Is SCAN or SSTF fairer?
- Rank the algorithms according to their expected seek time.
- What property of disks can we use to make the insertion, deletion, and access to the

**Review Questions:**

and slow down the OS if we can gain improvement in I/O times.

For I/O systems, and disk, in particular, it is worthwhile to complicate

For most OS features, we are very concerned about efficiency.

Disks are slow devices relative to CPUs.

---

**Summary**

Is disk read-ahead any better?

decided that was difficult to do well since the future is difficult to predict.

We considered pre-fetching virtual pages into physical memory, but

used while you have them under the head.

**Goal:** reduce the number of seeks - read blocks that will probably be

buffer on disk controller.

**Idea:** read blocks from the disk ahead of user's request and place in

---

**Improving Disk Performance using Read Ahead**