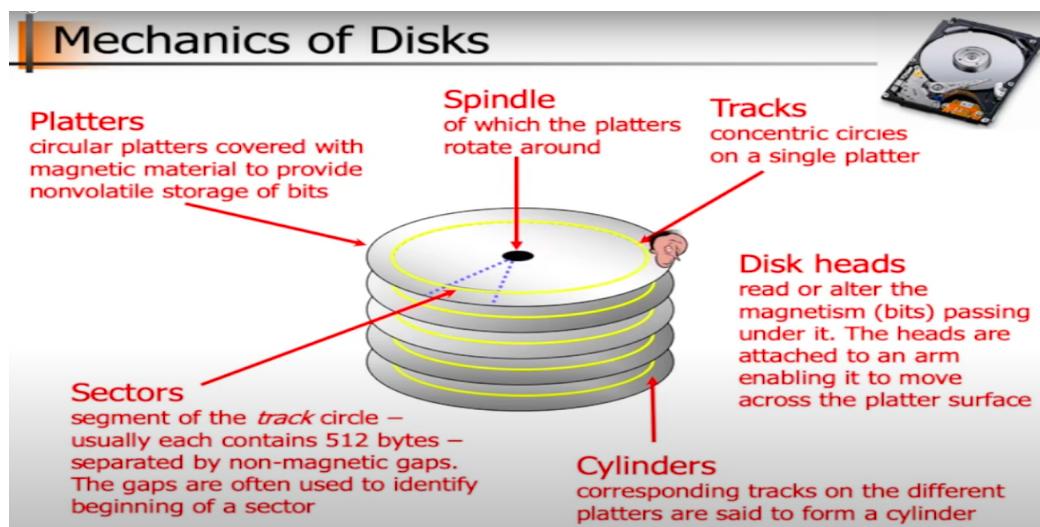


## Intro før vi går løs på lagring(Info om disk)

- Disk er persistente som vil si at, når strømmen går av på datamaskinen så vil disken kunne gjenopprette noe data som ble mistet når datamaskinen skrudde seg av. Disk er altså "non-volatile"/persistente.
- De er billigere sammenlignet med primærminne
- De har mer kapasitet til å lagre mer data enn primærminne og cache
- Problemet er dette med aksessstiden hvor det er utrolig tregt å aksessere ting fra disk
- Utover teksten så skal vi se på to viktige ressurser som er **lagringsplassen** og I/O bandwidth

## Mechanics of disks



### Platters/Plater

- Anses som runde magnetiske materiale hvor magnetene lagrer bits(0 eller 1)

### Spindle/Akse

- Noe som platene snurrer rundt på

### Tracks/Spor

- Deler platene inn i konsentriske sirkler kalt "spor"

### Sector/Sektorer

- Disse sporene vil da bli delt inn i sektorer, som oppbevarer 512 byte hver.

### Cylinders/Flere plater

- Dette er da flere plater satt oppå hverandre som er de korresponderende konstentriske sirklene som ligger i samme sted, samme avstand fra "spindle" i midten.

### Disk head/Disk hode

- Vi har en diskhode som leser fra spindle til den ytterste platen som leser bittene i disse sektorene som befinner seg i disken.

## Størrelsen til disk

For å finne størrelsen/lagringskapasiteten til en disk, så er vi avhengig av følgende

- Antall plater som befinner seg i disken
- Hvorvidt plattene brukes på en side eller begge sider
- Antall "konsentriske sirkler"/spor per overflate i disken(Destør flere spor, desto flere bytes)
- Gjennomsnittlig nummer av sektorer i hvert sport
- Antall nummer av bytes per sektor

### Example (Cheetah X15.1):

- 4 platters using both sides: 8 surfaces
- 18497 tracks per surface
- 617 sectors per track (average)
- 512 bytes per sector
- **Total capacity** =  $8 \times 18497 \times 617 \times 512 \approx 4.6 \times 10^{10} = 42.8 \text{ GB}$
- **Formatted capacity** = **36.7 GB**

**Note:**  
there is a difference between  
formatted and total capacity. Some  
of the capacity is used for storing  
checksums, **spare tracks**, etc.

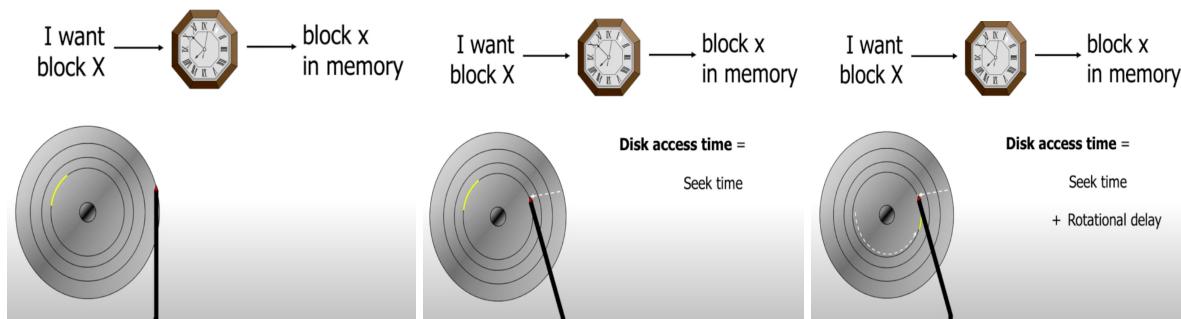
Ser vi bildet over, så ser vi et eksempel på en disk og hvilken betydning de ulike faktorene har å si for den totale størrelsen. Formatted capacity er egentlig den størrelsen som blir brukt, fordi den totale lagringsplassen inngår også noen sikkerhetsoperasjoner, sjekksum og diverse, som tar opp plass. I dette tilfellet så brukes 6 gb for slike operasjoner.

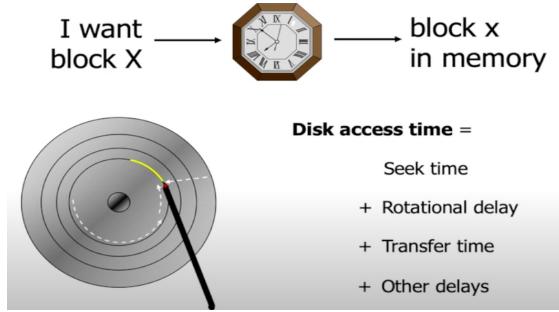
## Aksesstid

Hvordan er det data blir overført fra disken?

- Hvis vi husker diskhodet fra tidligere, så må vi få posisjonert diskhodet over platene, slik at når disken begynner å roterer rundt aksen, så skal diskhodet lese over bitene til disse platene mens den blir rotert.
- Tiden det tar mellom å foreslå en disk-request og tiden det tar når en blokk(operasjonen over) blir overført til minne, er kalt "disk latency" eller "disk access time".

(Pennen med rød tupp er diskhodet) (Diskhodet skal lese den gule linjen) (Venter til disken roterer så leses den gule linjen av diskhodet

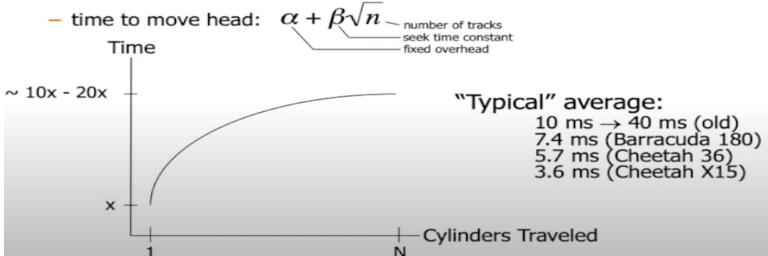




Diskaksess tiden er vist i bildet hvor vi har en rotasjonsforsinkelse(diskhodet må vente til disken roterer for å lese den gule linjen), overførselstid(Diskhodet må lese den gule-linjen bit for bit) og andre forsinkelser som ikke er, viktige å fokusere på.

### Disk Access Time: Seek Time

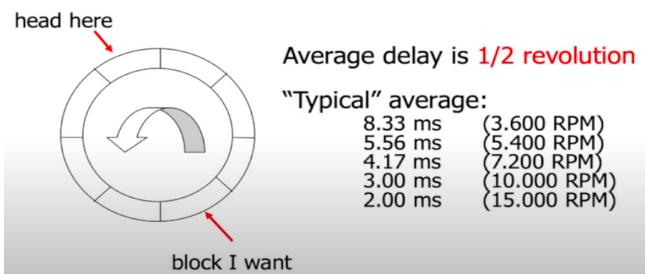
- Seek time is the time to position the head
  - time is used for actually moving the head – roughly proportional to the number of cylinders traveled
  - time to start and stop moving the head



Her ser vi et bildet som viser tiden det tar å posisjonere diskhodet under de konsentriske sirklene(spor) og det å bevege diskhodet frem og tilbake som den leser platene. Grafen viser da tiden tar å bevege diskhodet over de ulike cylinderne(flere plater som er satt på samme sted med de samme sporene, sjekk over hvis det var dårlig forklart under "Mekanikker"). Vi ser at når diskhodet har traversert gjennom alle sporene og platene, så vil antall plater synke og bli normal, akkurat som logaritmisk tid i IN2010. Tiden vil heller ikke øke noe mer siden det ikke er flere plater å traversere gjennom.

### Disk Access Time: Rotational Delay

- Time for the disk platters to rotate so the first of the required sectors are under the disk head



Rotasjonsforsinkelsen er tiden det tar å rotere sektoren med sporet vi ønsker å lese under diskhodet(eksempelet med den gule linjen). Litt avhengig av hvor fort man roterer, så har man vanlig "15 000"-omdreininger som gir 2.00ms, eller mindre avhengig av omdreininger.

## Disk Access Time: Transfer Time

- Time for data to be read by the disk head, i.e., time it takes the sectors of the requested block to rotate under the head
- Transfer time is dependent on **data density** and **rotation speed**
- Transfer rate =  $\frac{\text{amount of data per track}}{\text{time per rotation}}$
- Transfer time =  $\frac{\text{amount of data to read}}{\text{transfer rate}} = \frac{\text{amount of data to read} * \text{time per rotation}}{\text{amount of data per track}}$
- Transfer rate example
  - *Barracuda 180:*  
406 KB per track x 7.200 RPM ≈ 47.58 MB/s
  - *Cheetah X15:*  
306 KB per track x 15.000 RPM ≈ 77.15 MB/s
- If we have to change track, time must also be added for **moving the head**

**Note:**  
one might achieve these transfer rates reading continuously on disk, but time must be added for seeks, etc.

Tilslutt har vi overførerstiden hvor vi har regneoperasjonene vist over. Viktig å notere at noen ganger kan man ha for mye data å lese igjennom i et sport så det er ikke alltid at vi henter inn all dataen fra det gitte sporet. Det kan altså hende at vi må lese inn sporet på nytt som øker da overførerstiden på nytt.

## Disk Access Time: Other Delays

- There are several other factors which might introduce additional delays:
  - CPU time to issue and process I/O
  - contention for controller, bus, memory
  - verifying block correctness with checksums (retransmissions)
  - waiting in scheduling queue
  - ...

Dette er de andre faktorene som også har en rolle når det kommer til aksestiden. Ikke så viktig å kunne disse i detalj, men heller ha kjennskap til dem. F.eks at faktorer som skeduleringskøen eller prosesser som aksesserer CPU'en må da stoppe eller lignende.

## Disk-spesifikasjoner

Disk Specifications			
▪ Some existing (Seagate) disks:	<b>Note 1:</b> disk manufacturers usually denote GB as $10^9$ whereas computer quantities often are powers of 2, i.e., GB is $2^{30}$		
Capacity (GB)	Barracuda 180	Cheetah 36	Cheetah X15.3
Spindle speed (RPM)	181.6	36.4	73.4
#cylinders	7200	10.000	15.000
average seek time (ms)	24.247	9.772	18.479
min (track-to-track) seek (ms)	7.4	5.7	3.6
max (full stroke) seek (ms)	0.8	0.6	0.2
average latency (ms)	16	12	7
internal transfer rate (Mbps)	4.17	3	2
	282 – 508	520 – 682	609 – 891

Ikke viktig å kunne alle tallene her, men er viktig å kunne se sammenhengen mellom de ulike radene. F.eks så ser vi at kapasiteten og antall spor vi har(Cylinder) henger sammen hvor de med flere “Cylinder”, har større kapasitet.

	Barracuda 180	Cheetah 36	Cheetah X15.3
Capacity (GB)	181.6	36.4	73.4
Spindle speed (RPM)	7200	10.000	15.000
#cylinders	24.247	9.772	18.479
average seek time (ms)	7.4	5.7	3.6
min (track-to-track) seek (ms)	0.8	0.6	0.2
max (full stroke) seek (ms)	16	12	7
average latency (ms)	4.17	3	2
internal transfer rate (Mbps)	282 – 508	520 – 682	609 – 891

**Note 2:**  
there is a difference between internal and formatted transfer rate. **Internal** is only between platter. **Formatted** is after the signals interfere with the electronics (cabling loss, interference, retransmissions, checksums, etc.)

En annen ting er å sammenligne de tre markerte radene hvor “latency”/disk-aksess vil redusere om man har flere omdreininger (RPM) og høyere intern transfer-rate.

	Barracuda 180	Cheetah 36	Cheetah X15.3
Capacity (GB)	181.6	36.4	73.4
Spindle speed (RPM)	7200	10.000	15.000
#cylinders	24.247	9.772	18.479
average seek time (ms)	7.4	5.7	3.6
min (track-to-track) seek (ms)	0.8	0.6	0.2
max (full stroke) seek (ms)	16	12	7
average latency (ms)	4.17	3	2
internal transfer rate (Mbps)	282 – 508	520 – 682	609 – 891

Dette er da for søketiden, hvor vi ser hvordan disse henger sammen. Ikke så viktig da sammenlignet med neste trade-off.

	<i>Barracuda 180</i>	<i>Cheetah 36</i>	<i>Cheetah X15.3</i>
Capacity (GB)	181.6	36.4	73.4
Spindle speed (RPM)	7200	10.000	15.000
#cylinders	24.247	9.772	18.479
average seek time (ms)	7.4	5.7	3.6
min (track-to-track) seek (ms)	0.8	0.6	0.2
max (full stroke) seek (ms)	16	12	7
average latency (ms)	4.17	3	2
internal transfer rate (Mbps)	282 – 508	520 – 682	609 – 891

Her ser vi hvordan kapasiteteten og overførselses-raten har en trade-off, hvor de med lavere kapasitetet, har trege overførselsrater. Imens de med lave kapasitet, har høye overførsels-rater.

## Skriving og modifiserings-operasjoner

### Writing and Modifying Blocks

- A **write operation** is analogous to **read** operations
  - must potentially add time for block allocation
  - a complication occurs if the write operation has to be *verified* – must usually wait another rotation and then read the block again
  - **Total write time**  $\approx$  read time (+ time for one rotation)
  
- A **modification operation** is similar to **read and write** operations
  - cannot modify a block directly:
    - **read** block into main memory
    - modify the block
    - **write** new content back to disk
  - **Total modify time**  $\approx$  read time (+ time to modify) + write time

Alt over, inngår leseoperasjoner, men hva med tiden det tar å skrive og modifisere. For skriving så må vi først verifisere at dataene er korrekte. Når vi leser et gitt spor fra diskhodet, så må diskhode verifisere at det som har blitt lest inn, stemmer. La oss bruke eksemplet med den “gule-linjen, litt lengre oppe”, så først har vi runde 1, hvor vi da leser sporet og henter data fra den. Deretter må vi verifisere i runde “2” om det som ble lest inn, stemmer for da kan vi skrive dataen over til en minneblokk. Det er altså den totale skrive-tiden. Modifiseringsoperasjonen derimot, er tiden det tar å både utføre en leseoperasjon og skriveoperasjon.

## Kontrollere

For å håndtere de ulike delene i disken, dvs “cylinder”, “spor”, “disk-hode” osv, så har vi en “Disk-kontroller”. En “Disk-kontroller” er en liten prosessor som er i stand til å gjøre følgende:

- controlling the actuator moving the head to the desired track
- selecting which head (platter and surface) to use
- knowing when the right sector is under the head
- transferring data between main memory and disk

Kun viktig å huske at vi har en “Disk-kontroller”, og ha noe kjennskap til de ulike delene over.

## Effektivisering

### Efficient Secondary Storage Usage

- Must take into account the use of secondary storage
  - large gaps in access times between disks and memory, i.e., a disk access will probably dominate the total execution time
  - huge performance improvements if we reduce the number of disk accesses
  - a “slow” algorithm with few disk accesses will probably outperform a “fast” algorithm with many disk accesses
- **Several ways to optimize .....**

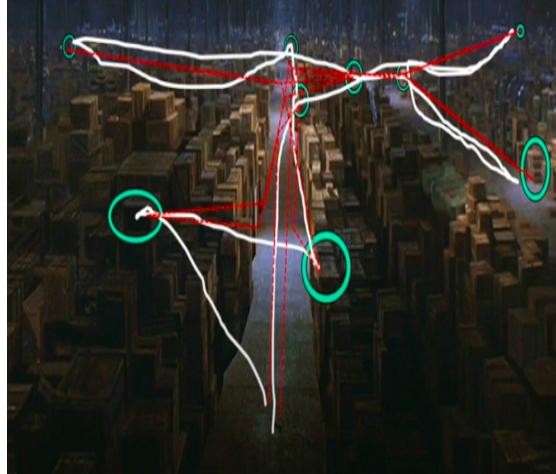
<ul style="list-style-type: none"> <li>– block size</li> <li>– disk scheduling</li> <li>– file management / data placement</li> <li>– memory caching / replacement algorithms</li> <li>– prefetching</li> <li>– multiple disks</li> <li>– ...</li> </ul>	<ul style="list-style-type: none"> <li>- 4 KB</li> <li>- SCAN derivate</li> <li>- various</li> <li>- LRU variant</li> <li>- read-ahead</li> <li>- a specific RAID level</li> </ul>
--	--

Vi vet at det å hente data fra disk kan ta lang tid hvor da den disk-aksessering i primærminnet er mer dominerende. Vi vil f.eks få større “perfomance improvements” hvis vi kan redusere antall disk-aksesser. Når det kommer til algoritmer, så kan det tenkes at en “treg” algoritme ikke nødvendigvis er så bra med tanke på ytelse. Men man vil helle ha en “treg” algoritme med mindre disk-aksesser enn en “rask” algoritme med mange disk-aksesser.

Vi har da ulike måter å optimalisere primærminnet på hvor det å øke blokk-størrelsen kan være nyttig. Dette skyldes av at ved paging og virtuelt minne, om vi har pages med lite antall størrelse, så er det en sjanse for at noen av blokkene til en prosess, må hentes fra disken siden page-størrelsen var liten. Så det å ha høyere blokkstørrelse er optimalt. Disk-scheduleringene kan også optimaliseres, noe vi kommer til om litt. Resten av punktene er også nytte-optimalisering hvor vi skal se litt nærmere på senere.

## Disk-Scheduling

- How to most efficiently fetch the parcels I want?



Forestill at vi har et lager med mange blokker. Disse blokkene vil da representer data-blokker i disken vår. Også har vi den mannen i bildet til venstre, som skal da hente pakkene markert med "grønn sirkel", og velge den mest effektive ruten. Rekkefølgen har en stor betydning når det kommer til å hente data-blokkene effektivt. Er rekkefølgen såpass treg, så vil det åpenbart føre til at det blir mindre effektivt å hente datablokkene. I bildet til høyre, så lagde Pål to ulike ruter hvor den "rød" var den dominerende siden den hadde færre distanse mellom de ulike blokkene sammenlignet med den "hvite".

- **Seek time is the dominant factor of the total disk I/O time**
- **IDEA:** Let the **operating system** or disk controller choose which request to serve next depending on the *head's current position and requested block's position on disk* (**disk scheduling**)
- Note that **disk scheduling  $\neq$  CPU scheduling**
  - a mechanical device – hard to determine (accurate) access times
  - disk accesses can/should *not be preempted* – run until they finish
- General goals
  - short response time
  - high overall throughput
  - fairness (equal probability for all blocks to be accessed in the same time)
- Tradeoff: **seek efficiency vs. maximum response time**

Utfordringen når det kommer til å søkeriden i disken, altså det å flytte diskhode fra spor-til-spor for å hente det sporet med dataen vi er ute etter, er det som tar tid. OS'et lager en rekkefølge hvordan vi skal håndtere situasjonen over med de "2" bildene hvor vi har datablokk til, hvor OS'et da lager rekkefølgen basert på hva den oppfattet som mest effektivt. For å finne hva som er mest effektivt, er igjen avhengig av systemet som man jobber med å arbeidsplassen eventuelt.

Viktig å notere at disk-scheduling er ulikt fra CPU-scheduling hvor den disk-scheduling har at disk-aksessering, ikke skal være “preempted”, man lar disken hente datablokkene og lar den kjøre til den er ferdig. Generelle mål ved disk-schedulering er at vi ønsker korte responstider, høy throughput og rettferdig aksesstid mellom hver datablokk. Vi har da ulike algoritmer ved disk-scheduling, akkurat som vi så med CPU-schedulering hvor vi har følgende algoritmer:

- Several traditional algorithms
  - First-Come-First-Serve (FCFS)
  - Shortest Seek Time First (SSTF)
  - SCAN (and variations)
  - Look (and variations)
  - ...
- A LOT of different algorithms exist depending on expected access pattern

### FCFS(First-Come-First-Serve)

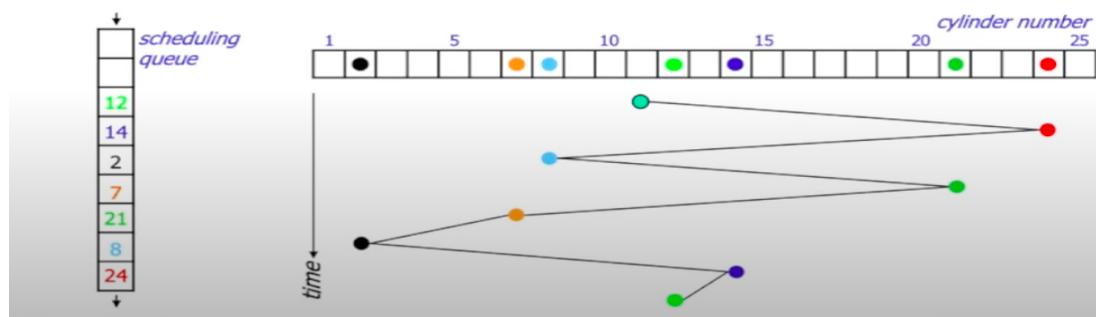
Denne opptrer som en FIFO, hvor da den første datablokk som kommer inn, er den første som blir utført. Denne opptrer med kort-responstid for alle diskblokker, dvs tiden det tar for en diskblokk som kommer inn i denne køen, blir utført. Siden det er en “first come, first serve” så vil de adressene som kommer inn i køen, bli utført like etter som tilsvarer kort responstid. Men problemet er at vi kan få lange søk siden det kan hende at en gitt spor som vi er ute etter, er langt unna diskhodet.

**FCFS** (FIFO) serves the first arriving request first:

- Long seeks
- “Short” response time for all

incoming requests (in order of arrival, denoted by cylinder number):

12 14 2 7 21 8 24



Hvis vi ser på bildet over så fungerer “FCFS” slik. Vi har da en skeduleringskø hvor hver forespørsler hvor nummeret angir en adresse i disken, her angitt hvilken “cylinder” det er. Vi ser at i grafen til høyre, så har vi at “24” blir utført, så kommer neste adresse er “8”. Se avstanden mellom disse to, søketiden er ganske lang siden vi må traversere gjennom flere

spor til å hente det gitte sporet i hvor “cylinderen” ligger. Kostnaden for søkingen vil altså bli stor.

## SSTF(Shortest Seek Time First)

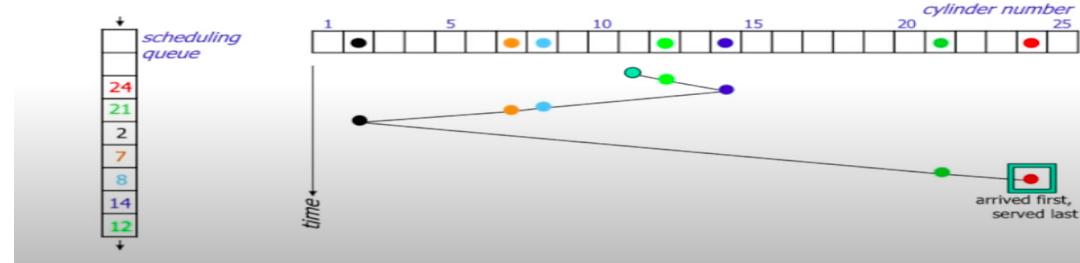
### Shortest Seek Time First (SSTF)

**SSTF** serves closest request first:

- short seek times
- longer maximum response times – **may even lead to starvation**

incoming requests (in order of arrival):

12 14 2 7 21 8 24



Denne er ganske selvforklarende hvor scheduleren vil kjøre de “sporene” fra adressen som har kortest søkerid mellom hverandre. F.eks så ser vi at vi kjører først “12”, deretter “14” siden den hadde kortest søkerid. Så fordelen ved denne algoritmen er at vi vil få kortere søkerid. Ulempen er at vi vil muligens få “starvation” siden om vi ser på “21 og 24”, så må disse vente utrolig lenge siden de er lengst unna de andre. Det tar altså lengre responstid for de “sporene” som vil ha lengst søkerid og dette er da ulempen ved denne algoritmen.

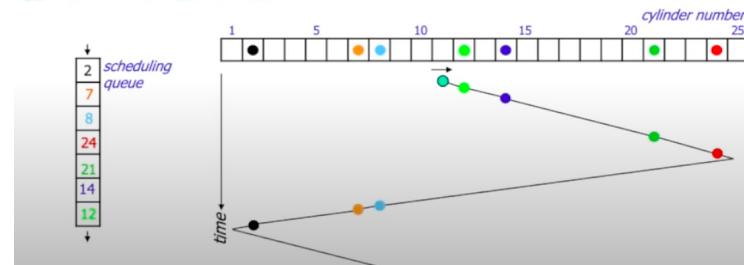
## SCAN(Elevator)

**SCAN (elevator)** moves head edge to edge and serves requests on the way:

- bi-directional
- compromise between response time and seek time optimizations

incoming requests (in order of arrival):

12 14 2 7 21 8 24



Den fungerer slik at diskhodet skanner fra kanten og inn mot aksen over diskplatene og utfører forespørslene underveis etterhvert som man passerer lokasjonen til en gitt forespørsel. La oss si at vi er i diskhodet starter fra “11”, og leser mot cylindernummerene mot høyre. Vi antar at de sporene med høye “cylindernummer” ligger inn mot aksen. Den vil da lese “12, 14, 21 og 24” siden de er da sporene som blir lest når diskhode går mot aksen.

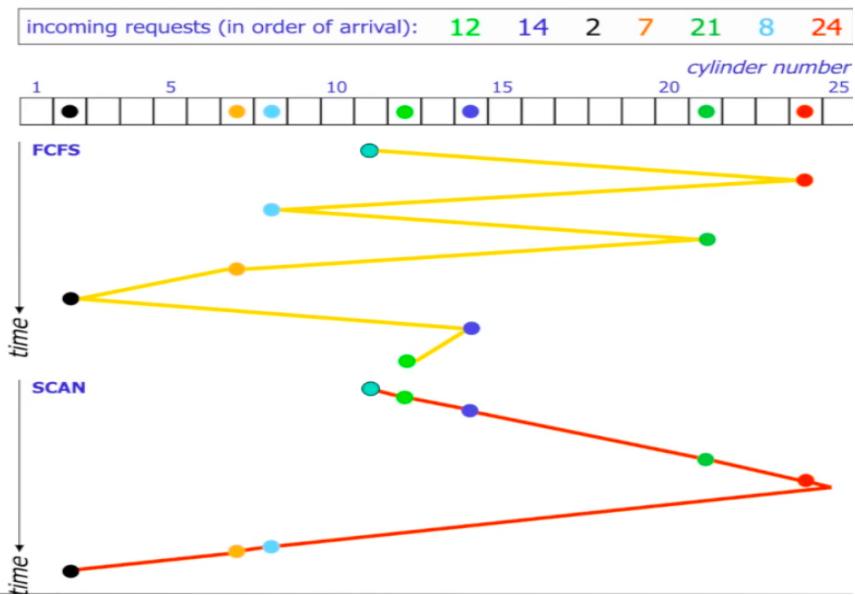
Når den skal lese tilbake fra aksen og mot ytterkanten, så vil den da lese de resterende "lave cylinder numrene".

## SCAN vs. FCFS

- Disk scheduling makes a difference!

- In this case, we see that SCAN requires much less head movement compared to FCFS

- here 37 vs. 75 tracks
- imagine having
  - 20.000++ tracks
  - many users
  - many files
  - ...



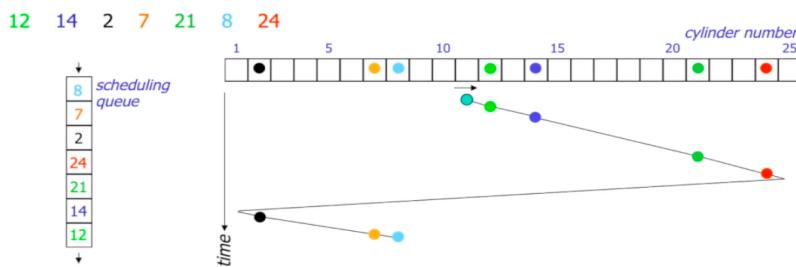
Her ser vi at "SCAN" reduserer antall disk-hode bevegelser siden den tar forespørslene som den traverserer gjennom diverse spor. Så hvis vi hadde hatt et system hvor søketiden var den dominerende faktoren, så ville dette vært åpenbart bedre enn "FCFS". En annen ting er om systemet hadde hatt mange brukere, mange filer og kanskje 20 000 spor å måtte gå igjennom. Da ville vel "FCFS" forårsake dårlig brukervennlighet siden søkeriden mellom de ulike sporene, ville tatt lang tid ettersom diskhodet må traversere gjennom flere spor, sammenlignet med "SCAN".

## C-SCAN

**Circular-SCAN** moves head from edge to edge

- optimization of SCAN
- serves requests on **one** way – uni-directional
- improves response time (fairness)

incoming requests (in order of arrival):



Dette er en optimalisering av "SCAN", hvor diskhode vil nå traversere fra kant, til kant. Som vist i bildet over, så vil diskhode da traversere helt til kanten mot "24", også traversere helt til den andre kanten, også traversere tilbake til kanten den kom fra og utføre søk slikt. Dette fører til bedre responstid. Man kan stille spørsmålet til hva er hensikten med dette, fordi diskhode kunne ha tatt forespørslene "8,7 2" når den var på vei tilbake. Vel grunnen er akselerasjon, noe som er et "must" for moderne disks når det kommer til søk.

- Why is C-SCAN in average better in reality than SCAN when both service the same number of requests in two passes?

- modern disks must **accelerate** (speed up and down) when seeking
- head movement formula:  $\alpha + \beta \sqrt{c}$



SCAN	C-SCAN
bi-directional	uni-directional
 requests: n, cylinders: x avg. dist: $2x$ (spread over both directions) total cost: $n \times \sqrt{2x} = (n \times \sqrt{2}) \times \sqrt{x}$	 requests: n, cylinders: x avg. dist: $x$ (over one direction only + one full pass) total cost: $\sqrt{n \times x} + n \times \sqrt{x} = (\sqrt{n} + n) \times \sqrt{x}$
if n is large: $n \times \sqrt{2} > \sqrt{n} + n$	

Total-kostnaden ved C-SCAN er lavere enn ved total-kostnaden ved SCAN. Hvis "n" derimot bli stor, så vil det være et billigere alternativ å utføre C-SCAN enn SCAN. Akselerasjon har derfor en betydning selvom SCAN hadde håndtert forespørslene bedre i det eksempelet over.

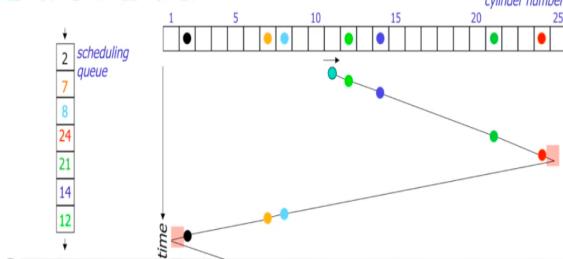
## LOOK and C-LOOK

**LOOK (C-LOOK)** is a variation of SCAN (C-SCAN):

- same schedule as SCAN
- does not run to the edges
- stops and returns at outer- and innermost requests
- increased efficiency
- SCAN vs. LOOK example:

incoming requests (in order of arrival):

12 14 2 7 21 8 24

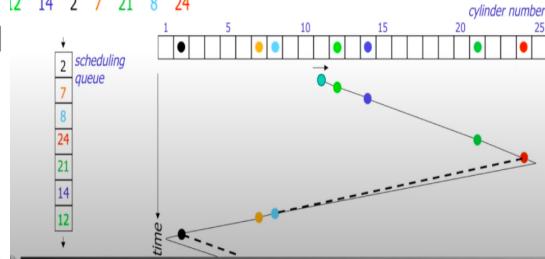


**LOOK (C-LOOK)** is a variation of SCAN (C-SCAN):

- same schedule as SCAN
- does not run to the edges
- stops and returns at outer- and innermost requests
- increased efficiency
- SCAN vs. LOOK example:

incoming requests (in order of arrival):

12 14 2 7 21 8 24



"LOOK" er en annen skeduleringsalgoritme som oppfører seg som "SCAN" men vi slipper å måtte gå helt til kanten. Grunnen er at dette ansees som bortkasta siden det ikke pleier å

ligge forespørsler helt mot kanten. Dette fører til at vi reduserer antall spor å traversere gjennom, samt at vi går altså fra den siste forespørselen til fra en kant og en annen kant som kan sees i bildet til høyre.