UNIVERSITETET
I OSLO

Effektiv bruk av sekundærlager

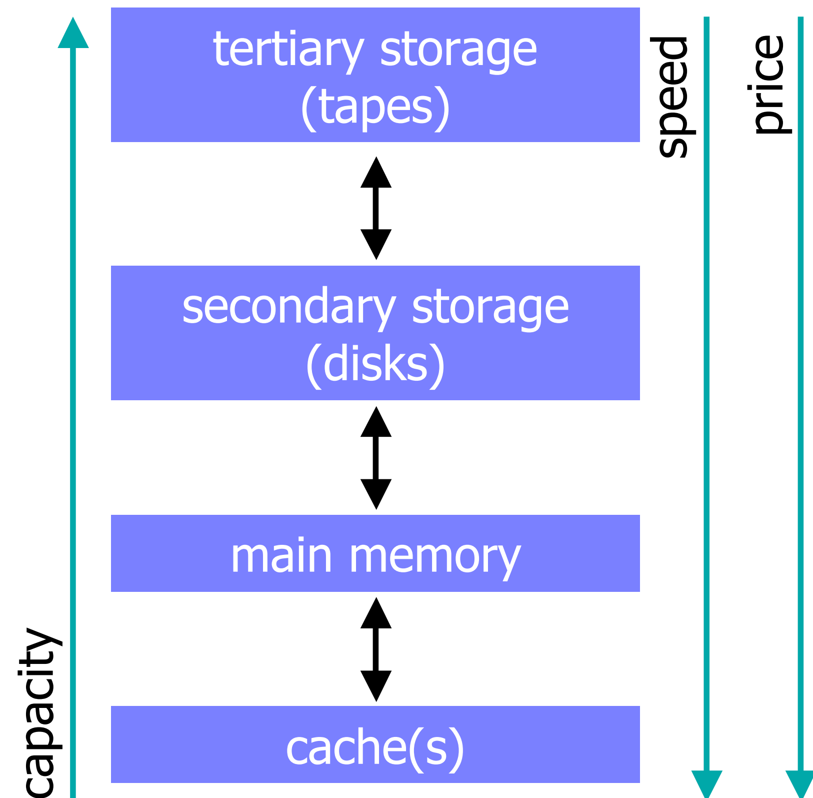
Inneholder lysark laget av:

Naci Akkök, Vera Goebel, Pål Halvorsen, Ketil Lund,
Hector Garcia-Molina, Ellen Munthe-Kaas

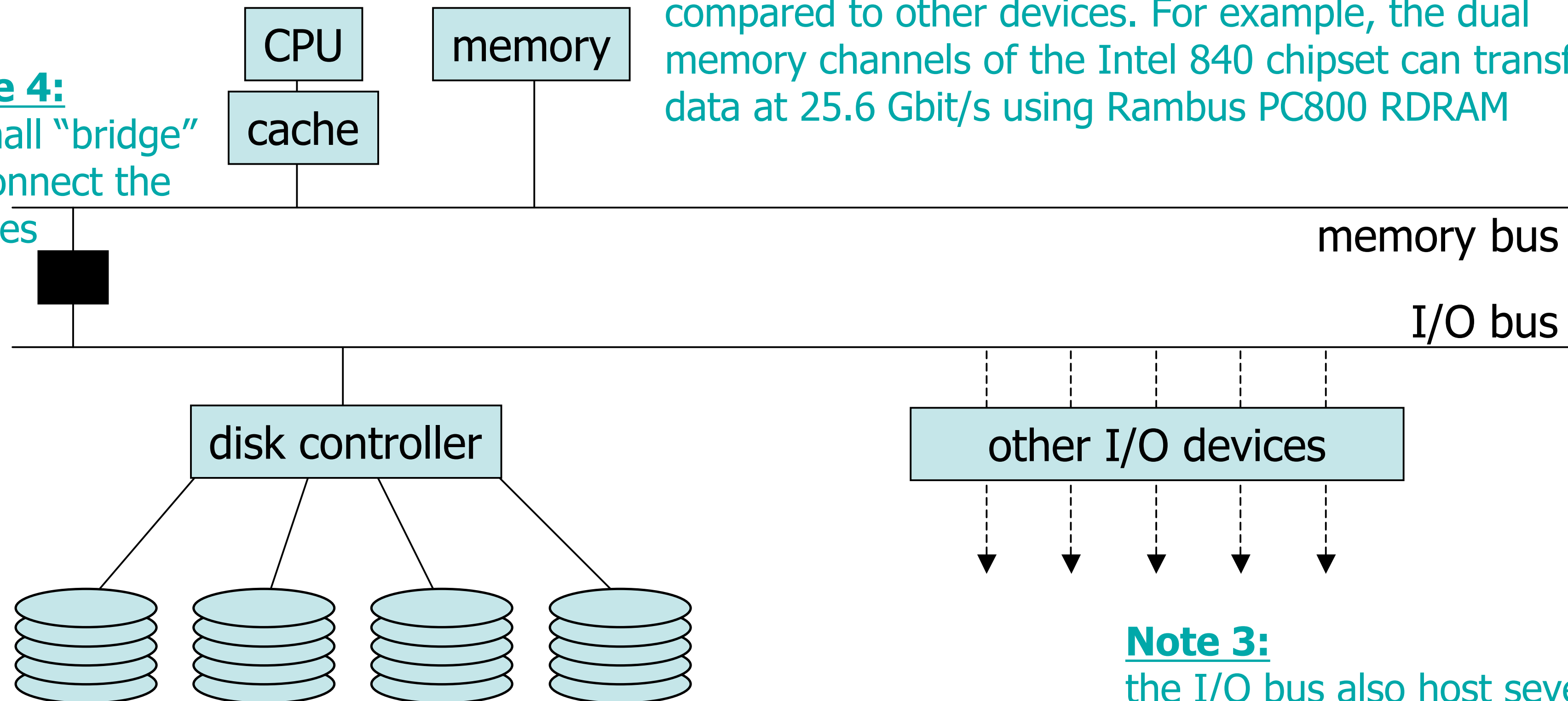
Bearbeidet av Ragnar Normann

Hukommelseshierarkiet

- Vi kan ikke aksessere disken hver gang vi trenger data
- Store datamaskinsystemer har derfor flere forskjellige komponenter hvor data kan lagres med
  - ulik kapasitet
  - ulik hastighet
- Mindre kapasitet gir raskere aksess og høyere kostnad per byte
- Høyere nivåer tjener som sikkerhetskopi av data på lavere nivåer
- Det laveste nivået ligger nærmest CPU-en



Hovedarkitekturen



Note 1:
the CPU and the memory is connected to a *memory bus* or *front end bus* due to speed mismatch compared to other devices. For example, the dual memory channels of the Intel 840 chipset can transfer data at 25.6 Gbit/s using Rambus PC800 RDRAM

Note 4:
a small "bridge" to connect the busses

Note 2:
secondary storage is connected to the *I/O bus*. For example, a 64 bit, 66 MHz PCI bus can at maximum transfer 4.2 Gbit/s

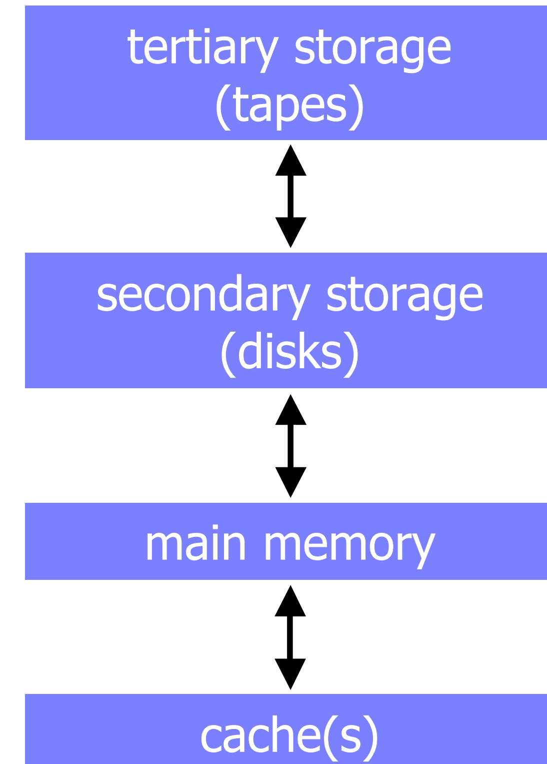
Note 3:
the I/O bus also host several other I/O devices such as network cards, sound cards, etc.

Dataflyt i hukommelseshierarkiet

- Hvis ønskede data ikke finnes på dette nivået, må vi se etter dem på nivået over (langsommere, men med mer kapasitet)
- Hvis vi henter data fra et høyere nivå, må vi kanskje overskrive data som allerede finnes på dette nivået
- Det finnes mange algoritmer/strategier for å velge et passende offer for utskiftning
- Hvis vi endrer data, må vi også endre dataene på de høyere nivåene for å opprettholde en konsistent database
- Her er det to hovedstrategier:
  - *Delayed write*
    Data på høyere nivåer oppdateres «når det passer»
  - *Write through*
    Data på høyere nivåer oppdateres umiddelbart; andre operasjoner må vente på oppdateringsoperasjonen

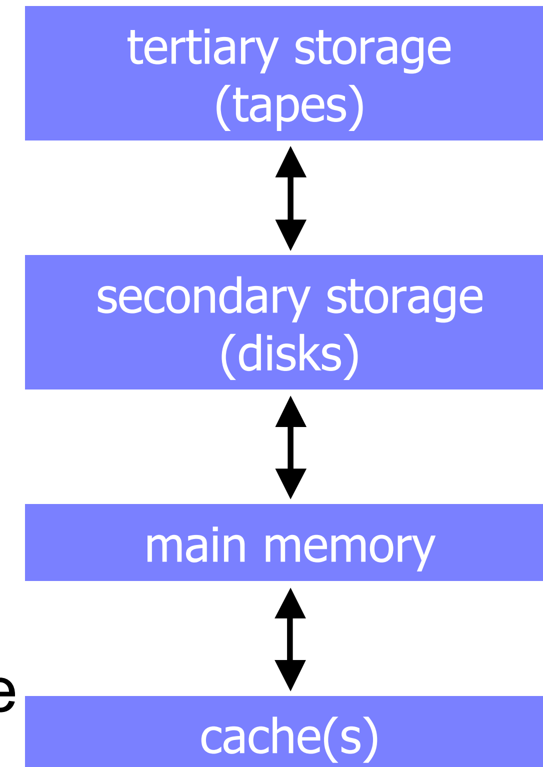
Cache

- Cacher er på laveste nivå
- Ofte har vi to nivåer (levels):
  - L1: på CPU-chipen
    - normalt 32 eller 64 KB
    - ofte delt i en data- og en instruksjonscache
  - L2: på en annen chip
    - normalt 1024 KB
    - typisk aksesstid noen få ns (10^{-9} sekunder)
- Hvis hver CPU har sin egen cache i et multiprosessor-system, må vi bruke *write-through* som oppdaterings-strategi



Hovedhukommelsen (Main Memory)

- Hovedhukommelsen er *random access*
- Tiden det tar å hente en byte, er nær konstant, men...
  - ... bruk av multiprosessorer kan gjøre at det tar ulik tid å aksessere ulike deler av hukommelsen
  - ... cachen gjør at det å aksessere data som ligger i nærheten, kan være mye raskere en å aksessere data langt borte



- Typiske tall :
  - størrelse fra noen hundre MB til flere GB
  - aksesstider fra 10 til 100 ns (10^{-8} – 10^{-7} sekunder)
- Memorymanageren bruker vanligvis *virtuell hukommelse*

Virtuell hukommelse

- Virtuell hukommelse lar prosessene bruke mer hukommelse enn det som er fysisk tilgjengelig
- Typisk vil prosessene få lov til å bruke 32-bits virtuelle byte-adresser, som gir et adresserom på 4 GB, noe som er mye mer enn den fysiske hukommelsen hver enkelt prosess får lov til å bruke
- Operativsystemet holder orden på hvilke data som er i fysisk hukommelse, og hvilke som ligger på disk
- Denne teknikken kalles *paging*
- Paging er detaljert behandlet i operativsystemkurset INF3151 og vil ikke bli nærmere gjennomgått her

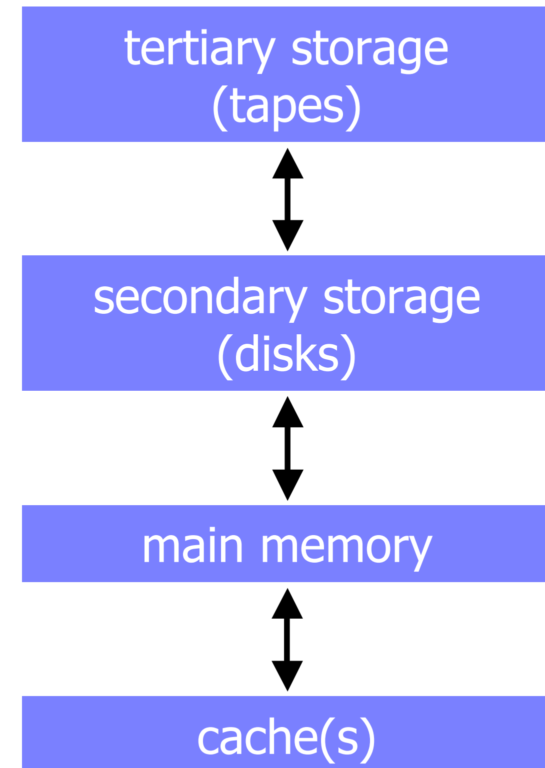
Hukommelseshåndtering og DBS-er

- Vanligvis prøver vi å holde databasen i hukommelsen for å øke ytelsen
- Vi vil gjerne unngå å bruke *write-through*-strategien mellom hukommelse og disk, men siden endringer i databasen skal være persistente, må alle oppdateringer skrives til disk og logges
- Databaser som er så små at de får plass i tilgjengelig virtuell hukommelse, kan, men trenger ikke, bruke operativsystemets hukommelseshåndtering
- Større databaser må håndtere sine data direkte på disk, og et slikt DBMS må derfor ha sin egen hukommelses-håndterer (Memory Manager)

Sekundærlager

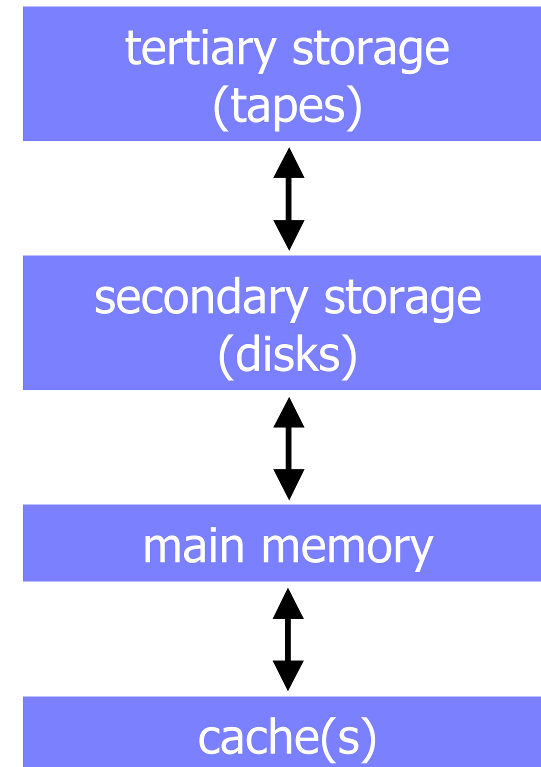
- Dette er et lager med vesentlig høyere kapasitet enn hovedhukommelsen
- Det finnes mange typer, men vi skal bare se på den viktigste, magnetisk harddisk
- Disker er rene sinker (ca. orden 10^6) i forhold til hovedhukommelsen
- De brukes både som persistent lager for data og som lager for programmets virtuelle hukommelse
- Ofte håndterer DBMS selv I/O mot disk

  Oppgavene og problemene er i hovedsak de samme som i et ordinært filsystem



Tertiært lager

- Noen systemer er for store til å kunne lagres på disker
- Tertiære lagre kan romme flere Pbyte (1 petabyte = 10^6 Gbyte = 2^{50} byte), men har aksesstider på flere sekunder, noen på flere minutter, dvs. minst 10^3 langsommere enn disker
- Det finnes mange typer som:
  - Ad-hoc tape-lager
  - Optiske juke-bokser
  - Tape-siloer
- Tertiære lagre er ikke noe sentralt tema i dette kurset

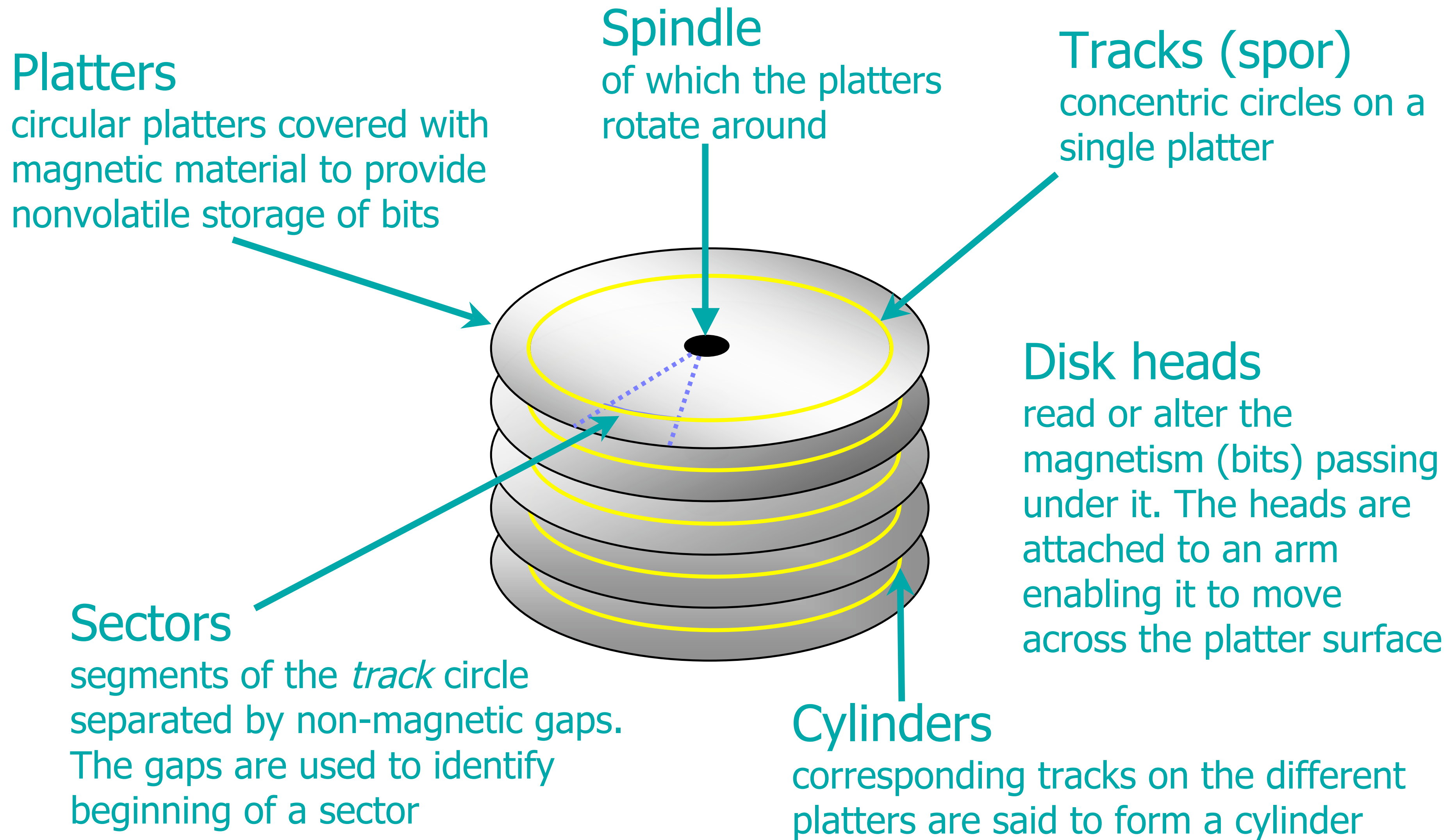


Disker

- Selv om det nå finnes mange persistente media som kan brukes som sekundærlager, er disker nær enerådende for større systemer (også for store databaser)
- På grunn av den store hastighetsforskjellen mellom disk og hovedhukommelse (en faktor på ca. 10^6) er det viktig å minimalisere antall diskaksesser
- Sekundærlageret er laveste nivå av persistente data

  Det er derfor viktig at ikke diskfeil fører til tap av data (Overføring til tertiallager gjøres vanligvis med *delayed write*, slik at tertiærlageret ikke alltid er oppdatert)
- I vår håndtering av disker har vi altså to hovedoppgaver:
  - Sikring mot feil
  - Effektiv bruk

Fysiske disker (Terminologi)



Kortfattet diskteknologihistorie

- Første disk var IBM350, introdusert 4.9.1956, med kapasitet 4,4 Mbyte, hastighet 1200 RPM, aksesstid nesten ett sekund, vekt 1,2 tonn
- Winchesterteknologi (disk og diskarm i samme støvtette boks) kom i 1973 (IBM3340 (vaskemaskinstørrelse))
- De første disker til PC-er kom i 1981 (5 Mbyte)
- To teknologier for diskkommunikasjon skilte seg ut:
  - SCSI («scussi») – Small Computer System Interface
    Eksempel: Seagates Barracuda- og Cheetah-disker
  - ATA – Advanced Technology Attachment
    Eksempel: IBMs disker

«State of the Art» diskteknologi ved årsskiftet 2007/2008

- Hitachi (tidligere IBM) leverer 1 Tbyte disk med SATA (Serial ATA) til bordmaskiner
- Seagate har gått over til SAS (Serial Attached SCSI)
- Det kan se ut til at SATA har et forsprang i konkurransen med SAS
- UiO/USIT satser på kabinetter med fjorten 500 Mbyte SATA-disker organisert som RAID 10
  (Vi kommer tilbake til RAID-teknologien)

  Kabinettene er koblet sammmen i et Fiber Channel SAN (Storage Area Network)

Diskkapasitet

- Diskstørrelsen er avhengig av
  - Antall plater, og om platene brukes på begge sider
  - Antall spor per overflate
  - Gjennomsnittlig antall sektorer per spor (moderne disker er *zoned*; dvs. at de ytre sporene har flere sektorer enn de indre)
  - Antall byte per sektor
- Det er forskjell på total og formatert kapasitet

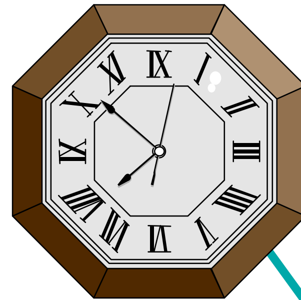
  Noe plass blir brukt til sjekksummer, reservespor o.l.

Aksessering av disk

- Hva må vi gjøre for å lese/skrive fra/til disk?
  – Plasser hodet over sylinderen (sporet) som inneholder datablokken vi vil lese eller skrive (den kan bestå av en eller fler sektorer)
  – Les eller skriv datablokken mens sektorene passerer under diskhodet
- Tiden fra en prosess ber om å få lese en diskblokk til blokken er i hovedhukommelsen kalles *diskens aksesstid*

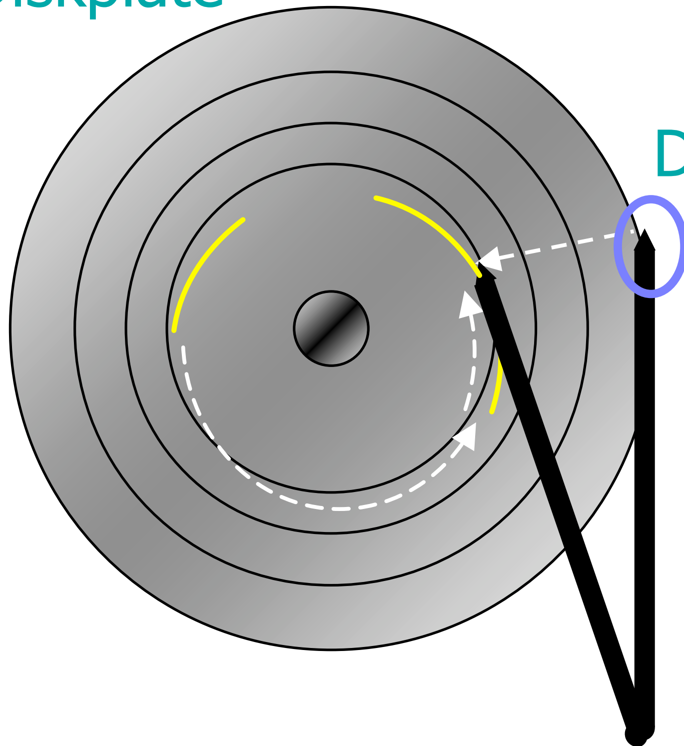
Diskers aksesstid

Jeg vil ha
blokk X



blokk X
i hukommelsen

Diskplate



Diskhode

Diskarm

Diskens aksesstid =

Søketid

\+ Rotasjonsforsinkelse

\+ Overføringstid

\+ Andre forsinkelser

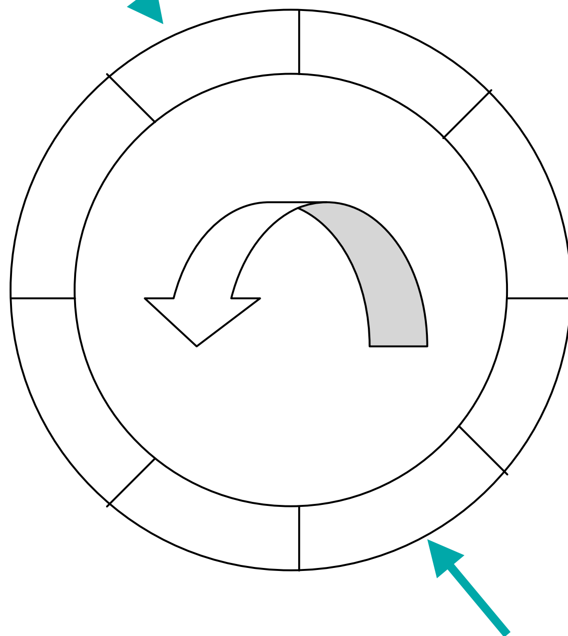
Søketiden

- Søketiden er tiden det tar å flytte diskhodet
  - Hodet trenger en minimumstid for å begynne og slutte å bevege seg
  - Tiden det tar å bevege hodet, er tilnærmet proporsjonal med antall sylindre det skal passerere
  - Typiske søketider er 4 - 10 ms

Rotasjonsforsinkelsen

- Tiden det tar for platene å rotere slik at første ønskede sektor er under diskhodet

Hodets posisjon



Ønsket blokk

Midlere forsinkelse er en halv omdreining

Noen vanlige verdier:

8.33 ms (3.600 RPM)
5.56 ms (5.400 RPM)
4.17 ms (7.200 RPM)
3.00 ms (10.000 RPM)
2.00 ms (15.000 RPM)

Overføringstid

- Tiden det tar for de ønskede data å passere under diskhodet
- Overføringstiden er foruten datamengden avhengig av datatettheten og rotasjonshastigheten
- Overføringshastigheten =
  datamengde per spor x rotasjonshastigheten
- Hvis dataene strekker seg over flere spor, må vi legge til tiden det tar å flytte hodet

- Eksempel:
  Gitt en disk med 250 KB per spor som spinner med 10 000 RPM

  Overføringshastigheten er:

  250 KB x 10 000 / min = (250 / 1024) MB x 10 000 / 60 s
  = 40,69 MB/s

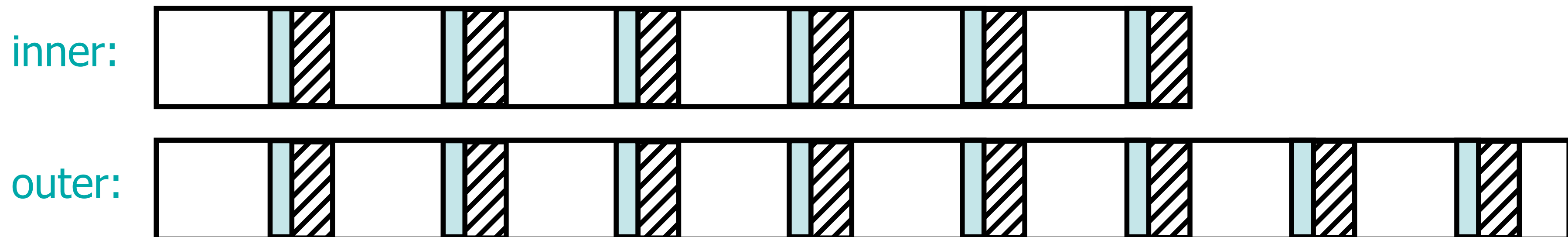
Andre forsinkelser

- En rekke andre faktorer medfører tidsbruk:
  - CPU-tid for å be om og prosessere I/O
  - Konkurranse om tilgang til diskkontroller, buss og hukommelse
  - Beregning av sjekksummer for å kontrollere at datablokker er korrekt overført (med mulige retransmisjoner)
  - ...

- Felles for alle disse er at de har neglisjerbar verdi sammenlignet med de øvrige faktorene (ns/µs versus ms)

To kompliserende faktorer

- Diskene er «zoned»: fordi de ytre sporene er lengre enn de indre, har de ofte flere sektorer
- I hver sektor lagres det en sjekksum for sektoren



Note 1:
transfer rates are higher on outer tracks

Note 2:
gaps between sectors

Note 3:
the checksum is read for each sector and used to validate the sector

Note 4:
the checksum is usually calculated using Reed-Solomon interleaved with CRC

Note 5:
for older drives the checksum is 16 bytes

Note 6:
SCSI disks may be changed by user to have other sector sizes

Skriving og endring av blokker

- Skriveoperasjoner er analoge med leseoperasjoner
- En komplikasjon oppstår hvis skriveoperasjonen må verifiseres – vi må da vente en rotasjon og så lese og verifisere blokken
- Total skrivetid ≈ lesetid + tid for en rotasjon
- Blokker kan ikke endres direkte på disk:
  - Les blokken inn i hukommelsen
  - Modifiser blokken
  - Skriv det nye innholdet tilbake til disk
  - (Verifiser skriveoperasjonen)
- Total modifiseringstid ≈ lesetid + tid til endring + skrivetid (tiden brukt på selve endringen er neglisjerbar)

Diskkontrollere

- Diskkontrolleren er en liten prosessor som:
  - Flytter diskhodene til riktig sylinder
  - Velger plate og overflate som skal brukes
  - Vet når riktig sektor er under hodet
  - Overfører data mellom hovedhukommelse og disk
- Nyere kontrollere er selv små datamaskiner
  - Både disk og kontroller har nå egen buffer som reduserer diskaksesstiden
  - Data på ødelagte diskblokker/sektorer blir flyttet til et reserveområde på disken – operativsystemet (OS) vet ikke om dette, så en blokk kan ligge et annet sted enn hva OS tror
  - Dermed blir OS' diskadresser virtuelle

Effektiv bruk av sekundærlager

- Siden bruken av sekundærlageret er totalt dominerende når det gjelder effektiviteten av et DBS, er det her det er lønnsomt å optimalisere

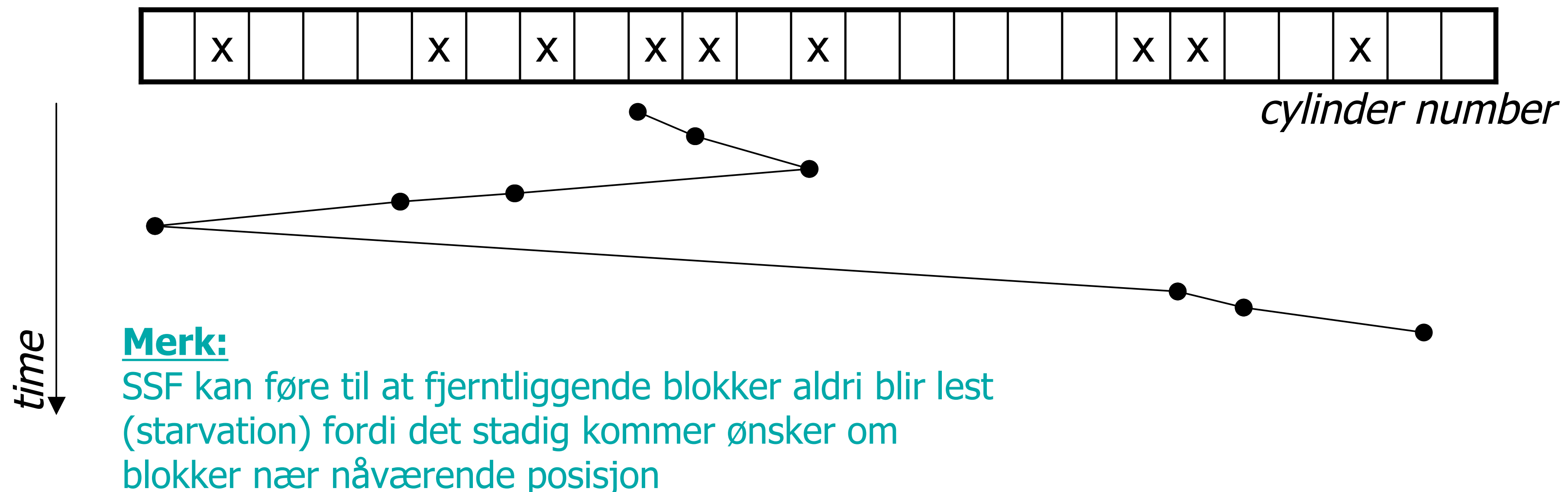
- Men det er mange måter å optimalisere på

Betydningen av blokkstørrelsen

- Forutsatt en tilfeldig plassering av blokkene på disken vil en dobling av blokkstørrelsen halvere antall diskaksesser
  - Total overføringstid blir den samme
  - Søketid og rotasjonsventetid blir halvert
- Men for store blokker er heller ikke bra
  - Hver blokk bør ligge innenfor ett spor (i hvert fall innen en sylinder)
  - Små dataelementer vil bare fylle brøkdelen av en blokk
- Valg av blokkstørrelse er avhengig av datastørrelse og aksessmønster
- Ny teknologi med større og raskere disker gjør at trenden er større blokkstørrelser

Søkestrategier – I

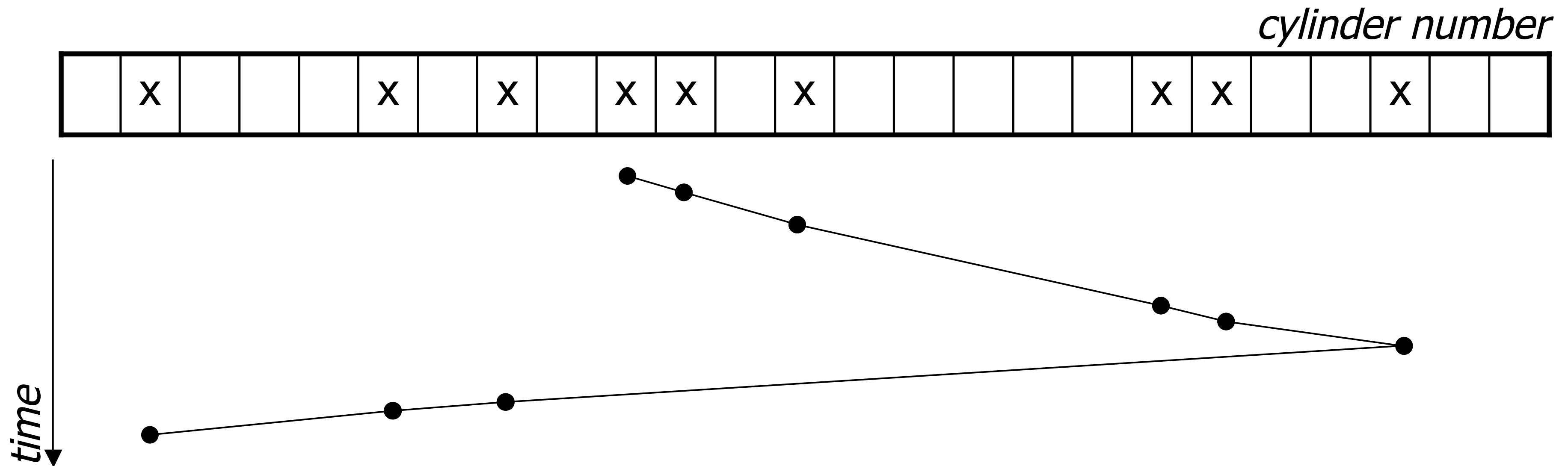
- Søketid er den dominerende faktor for total disk I/O tid
- Det finnes flere algoritmer
  - First-Come-First-Serve
  - Shortest Seek First (SSF):
    La diskkontrollerens valg av hvilken prosess den vil betjene avhenge av hodets posisjon og ønsket blokks posisjon (minimaliser hodebevegelsen)



Merk:
SSF kan føre til at fjerntliggende blokker aldri blir lest (starvation) fordi det stadig kommer ønsker om blokker nær nåværende posisjon

Søkestrategier – II

- Elevator (SCAN) algoritmen:
  La hodet gå fra innerste til ytterste sylinder og stopp hvis det passerer en ønsket blokk, snu når siste blokk i denne retningen er prosessert



- Flere andre algoritmer .....

Multippel bufring

- Hvis vi kan forutse aksessmønsteret på disken, kan vi øke effektiviteten ved å bruke flere bufre
- Det enkleste er dobbelbufring:
  - Les data inn i buffer nr 1
  - Prosesser data i buffer nr 1
    samtidig som data leses inn i buffer nr 2
  - Prosesser data i buffer nr 2
    samtidig som data leses inn i buffer nr 1
  - osv.

Beregningsmodeller

- **RAM-modellen** (RAM = Random Access Model):
  - Antar at alle data får plass i primærminnet
  - Tiden det tar å aksessere et data item er uavhengig av hva innholdet i itemet er eller hvor det ligger i minnet
- **I/O-modellen**:
  - Når ikke alle data kan få plass i primærminnet samtidig
  - Antall blokkaksesser (skriving/lesing) er det som avgjør hvor god en algoritme er
    (Tidsbruk i primærminnet er neglisjerbar)

Sortering

- Små mengder data (alt får plass i primærminnet):
  - RAM-modellen
  - Variant av **Quicksort**
- Store mengder data (kan ikke unngå diskaksesser):
  - I/O-modellen
  - **Two-phase, multiway merge-sort (TPMMS)**

Merge-sort

- **Merge-sort** er en algoritme for primær-minne-sortering (RAM-modellen)
- Anta n elementer skal sorteres
  - *Basis:* En liste med ett element
    - Listen er allerede ferdigsortert
  - *Induksjon:* Gitt en liste med mer enn ett element
    - Del listen i to like store lister
    - Sorter hver liste med merge-sort
    - Flett de resulterende listene til en sortert liste

TPMMS – I

TPMMS – Two-Phase Multiway Merge-Sort – er en algoritme for sortering ved store datamengder (I/O-modellen)

- Fase 1: Data grupperes i bolker som hver får plass i primærminnet
  - Sorter hver gruppe (bruk f.eks. Quicksort) og legg tilbake på disk
- Fase 2: Flett alle sorterte sublister:
  - Sett av én inputbuffer i primærlageret pr. subliste pluss én outputbuffer, hver buffer rommer én blokk
  - Last opp første blokk fra hver av sublistene
  - Finn minste element i inputbufferne, flytt det til outputbufferet (bruk f.eks. lineært søk, det tar kort tid sammenliknet med henting av blokker fra disk)
    Gjenta med gjenværende elementer
    - Når en inputbuffer er tom, hentes neste blokk i tilhørende subliste fra disk til inputbuffer
    - Når outputbufferet er fullt, skrives det til disk

TPMMS – II

- Fase 1:
  - Blokkene kan leses i rekkefølge slik de ligger lagret på disk
  - Hver blokk leses fra og skrives til disk en gang
- Fase 2:
  - Blokkene leses i uforutsigbar rekkefølge fra disk
  - Hver blokk leses nøyaktig en gang
  - Hver record flyttes til outputbufferet nøyaktig en gang, og skrives til disk en gang
  - Totalt skrives like mange blokker som det som blir lest

TPMMS – III

- Når "sprekker" TPMMS?
  La

  B – blokkstørrelse

  M – plass avsatt i primærminnet til bufring

  R – recordstørrelse

  (alle i antall bytes)

  - Hver subliste fra fase 1 er maks M bytes lang, dvs. inneholder maks M/R records
  - Antall sublister som kan håndteres i fase 2, er maks (M/B-1)
    (antall bufre er M/B, en av dem må brukes til output)
  - Totalt antall records som kan sorteres, er derfor
    $(M/R) * ((M/B)-1) \approx M^2/RB$

Eksempel:

B= 2^{14} = 16384 bytes

M = 100 MB = 100 * 2^{20} bytes = 6400 blokker

R = 160 bytes

M^2/RB = 4.2 * 10^9 records = 0,67 terabytes

– stort nok til praktisk talt alle realistiske formål

TPMMS – IV

Hva hvis TPMMS likevel ikke klarer å behandle alle records i løpet av de to fasene?

1. Bruk TPMMS til å sortere grupper av M^2/RB records, hver blir en sortert subliste
2. Bruk en tredje fase til å flette opp til (M/B-1) slike lister, totalt kan sorteres ca. $(M^2/RB)(M/B) = M^3/RB^2$ records

Eksempelet fra forrige foil:

$M^3/RB^2 = 27 * 10^{12}$ records = 4,3 petabytes

Diskfeil – I

- Moderne disker har en MTF (mean-time-to-failure) på over 50 år (da er ca. 50 % av diskene ødelagt), men
  - Mange disker feiler i løpet av noen måneder (produksjonsfeil)
  - Hvis det ikke er noen produksjonsfeil, vil disken trolig virke i mange år
  - Gamle disker har igjen en høyere risiko for feil pga. slitasje, støv o.l.

Diskfeil – II

Vi deler diskfeil inn i tre grupper:

- Opprettbare (intermittent failures)
  Midlertidige feil som kan rettes ved å lese blokken på nytt
  (f.eks at støv på disken gir en bit-feil)
- Uopprettbare (media decay/write errors)
  Permanente feil hvor bit er ødelagte
  (f.eks skader/sår i den magnetiske overflaten)
- Disk-kræsj
  Hele disken er permanent uleselig

Sjekksummer – I

- Diskblokker lagres med noen ekstra bit, kalt sjekksum
- De brukes til å validere en lest eller skrevet blokk:
  - Les blokk og lagret sjekksum
  - Beregn sjekksummen på lest blokk
  - Sammenlign lest og beregnet sjekksum
- Hvis lest og beregnet sjekksum er forskjellige, les blokken på nytt
  - Lykkes leseoperasjonen, så returner riktig innhold
  - Hvis antall leseforsøk går over grensen, returner feilmeldingen "bad disk block"

Sjekksummer– II

- Sjekksummer brukes bare for å *oppdage* feil
- Det er mange måter å beregne sjekksummer på
- 1-bit paritet er den enkleste: tell ener-bit i blokken
  - Likt antall gir paritetsbit 0
  - Odde antall gir paritetsbit 1

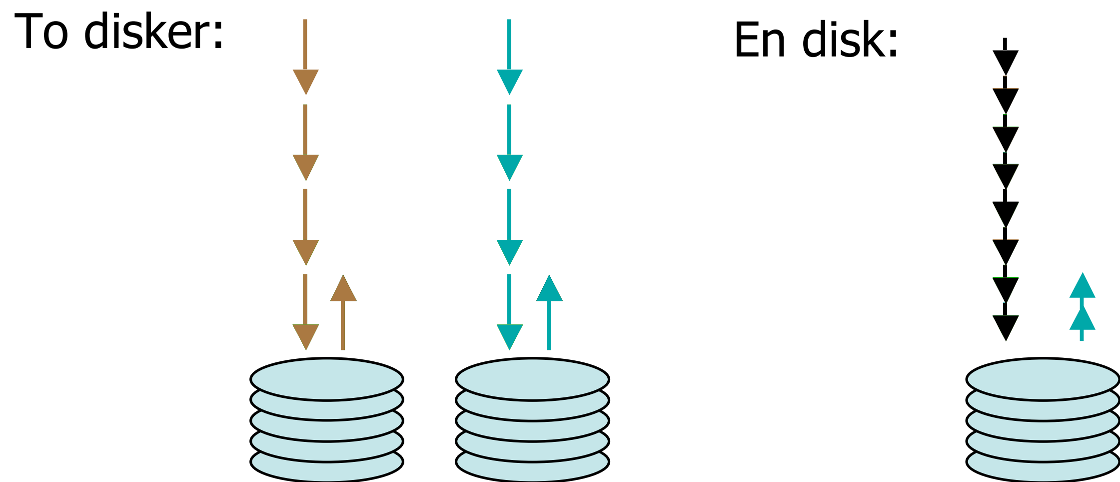
  Stor fare for å ikke oppdage feil
- 8-bit paritet: ett paritetsbit per bit i en byte:
  Tell 1-ere i mest signifikante bit, ... , minst signifikante bit
  Faren for å overse feil reduseres med faktoren $1/2^7$
- Polynomiske koder – CRC (cyclic redundancy check) :
  - Generer en sjekksum med modulo-2 aritmetikk (XOR)
- Flere andre ....

Håndtering av diskkræsj

- Det er ingen annen måte å gjenopprette data etter et diskkræsj enn å ha en kopi på et annet medium som magnetbånd eller en speilet disk
- Det finnes en rekke strategier for å redusere faren for tap av data ved permanente diskfeil
  - De fleste baserer seg på en utvidet paritetssjekk
  - De vanligste går under betegnelsen RAID-strategier (RAID = Redundant Arrays of Independent Disks)

Striping

- Dette er en teknikk for å øke lese- og skrivehastigheten til og fra disk
- I stedet for å skrive alle data til samme disk splittes dataene og skrives til flere disker i parallell til n disker
- Dette reduserer selve skrive/lesetiden med en faktor n (søke- og rotasjonsforsinkelsen blir omtrent som før)



Speiling

- Speiling betyr at vi har flere identiske kopier av hver disk, som oftest to

- Fordeler:
  - Raskere svartider ved lesing
  - Overlever diskkræsj – feiltoleranse
  - Balanserer lasten ved å fordele leseoperasjonene likt mellom de speilede diskene

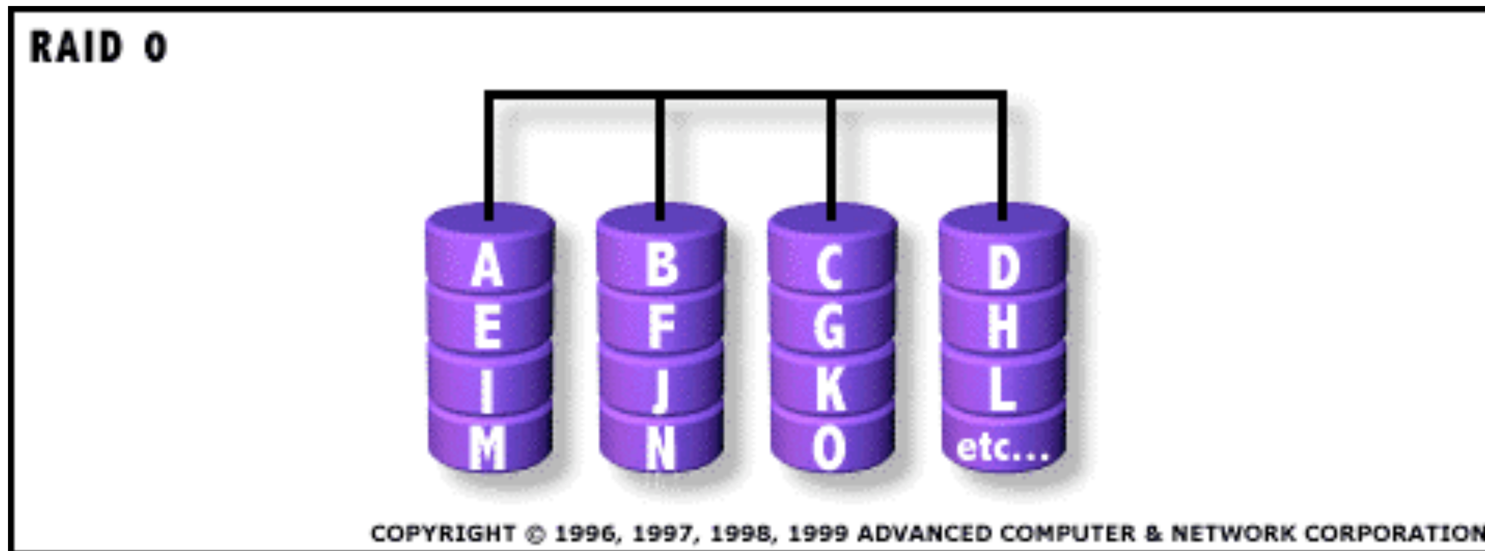
- Ulemper:
  - Øker lagringsbehovet

RAID (Redundant Arrays of Independent Disks)

- RAID level 0: non-redundant
- RAID level 1: mirrored
- RAID level 2: Hamming error correcting code (ECC)
  (No commercial implementations exist)
- RAID level 3: bit-interleaved parity
- RAID level 4: block-interleaved parity
- RAID level 5: block-interleaved distributed-parity
- RAID level 6: multiple redundancy

RAID 0 (ingen redundans) – I

- RAID 0: Stripet diskarray uten feiltoleranse
- Striping betyr at data brekkes opp i logiske blokker som hver skrives til en separat disk

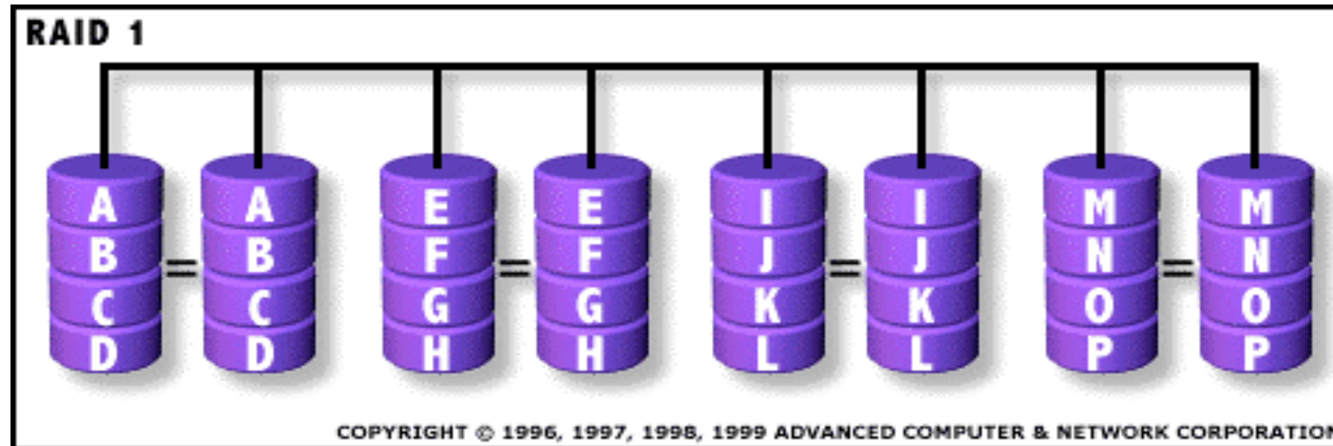


RAID 0 (ingen redundans) – II

- Fordeler
  - Ytelsen blir mye bedre ved å spre I/O på mange kanaler og disker
  - Den beste ytelsen får man når data stripes på flere kontrollere med bare en disk per kontroller
- Ulemper
  - Ikke en ekte RAID fordi den ikke er feiltolerant
  - Hvis en disk feiler, vil alle data i diskarrayen gå tapt
  - Må aldri brukes i feilkritiske situasjoner

RAID 1 (speiling)

- Data er duplisert og ligger på to disker



- Fordeler
  - En skriving eller to samtidige lesinger per speilet par
  - 100% redundans betyr at ingen rekonstruksjon trengs etter en diskfeil, bare en kopi til erstatningsdisken
- Ulempe: Ekstremt høy disk-overhead (100%)

RAID 2 og RAID 3 (Kursorisk)

- RAID 2: Hamming ECC
  - Alle databit skrives til én datadisk (ekstrem striping)
  - Alle dataord får sitt Hammingkodede ECC-ord lagret på egne ECC-disker
  - ECC-koden verifiserer korrekte data og retter feil på en enkelt disk
  - NB! Ingen kommersielle implementasjoner finnes
- RAID 3: Parallell dataoverføring med bit-paritet
  - Datablokkene stripes og skrives på datadiskene
  - Stripepariteten genereres ved skriving og lagret på en paritetsdisk
  - Pariteten sjekkes ved lesing

Modulo-2-summer

- En vanlig måte å lage en korreksjonsblokk (paritetsblokk) på er å beregne modulo-2-summen av datablokkene (Modulo-2-sum blir ofte kalt XOR (exclusive OR))
- Modulo-2-summen beregnes ved å la bit k i summen være
  - 1 – hvis et odde antall blokker har 1 i posisjon k
  - 0 - hvis et like antall blokker har 1 i posisjon k
- Eksempel

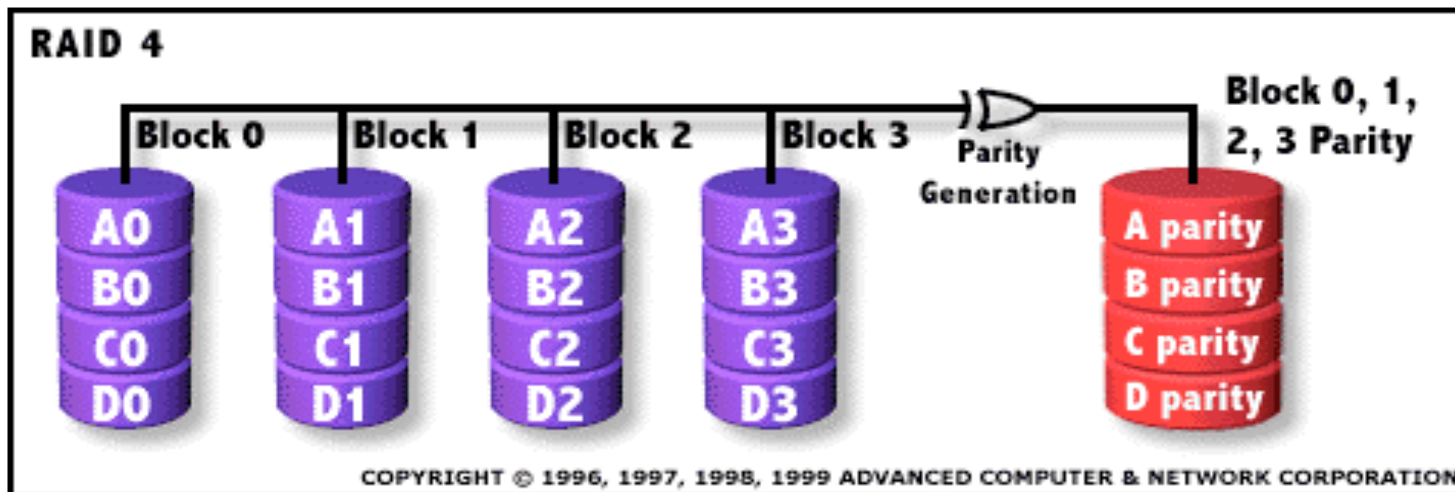
blokk 1	1	1	1	1	0	0	0	0
blokk 2	1	0	1	0	1	0	1	0
blokk 3	0	0	1	1	1	0	0	0
modulo-2-sum	0	1	1	0	0	0	1	0

Regneregler for modulo-2-summer

- La $\oplus$ betegne modulo-2-sum-operatoren
- Da gjelder:
  - Den kommutative loven: $x \oplus y = y \oplus x$
  - Den assosiative loven: $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
  - 0 er identiteten: $x = 0 \oplus x = x \oplus 0$
  - $\oplus$ er sin egen invers: $x \oplus x = 0$

RAID 4 (Blokkvis paritet) – I

- Uavhengige datadisker med en felles paritetsdisk
- Hver datablokk skrives i sin helhet til én datadisk
- Pariteten beregnes ved skriving og lagres på en felles paritesdisk
- I figuren har vi at: A parity = A0 ⊕ A1 ⊕ A2 ⊕ A3



RAID 4 – II

- Ved skriving av en datablokk beregnes den nye verdien av den tilhørende paritetsblokken ved formelen:

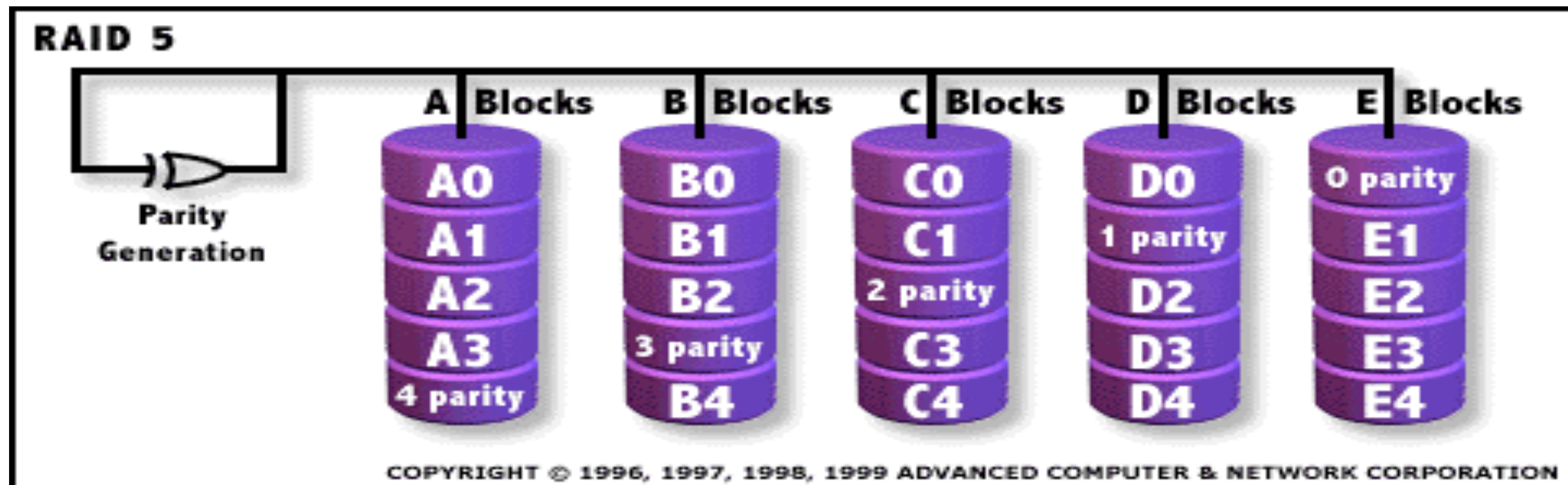
  gammel paritetsblokk $\oplus$ gammel datablokk $\oplus$ ny datablokk

- Hvis en enkelt disk kræsjer, det er det samme om det er en datadisk eller paritetsdisken, kan den regeneres ved å ta modul-2-summen av alle de andre diskene

- En ulempe ved RAID 4 er at paritetsdisken kan bli en flaskehals:
  – Den må skrives ved hver oppdatering

RAID 5 (Blokkvis fordelt paritet)

- Logisk identisk med RAID 4, men paritetsblokkene er fordelt på alle diskene, noe som fordeler lasten jevnt
- Anta at vi har $n + 1$ disker nummerert fra 0 til n
  Da lagres paritet på sylinder i på disk j hvis $j = i \bmod (n+1)$
  Da blir neste paritetssylinder sylinder $i + 1$ på disk $j + 1$
- RAID 5 er en mye brukt teknologi



RAID 6

- Dette er egentlig en familie metoder som tåler at mer enn en disk kræsjer samtidig
- De mest avanserte bruker n logiske paritetsdisker (fysisk fordelt som i RAID 5) for å tåle n samtidige diskkræsj
- Vi skal se nærmere på Hamming-kodet RAID 6 som tåler 2 samtidige diskkræsj blant 2^k - 1 disker hvorav k er paritetsdisker (og 2^k - k -1 er datadisker)
- Diskene nummereres fra 1 til 2^k -1 og de identifiseres med sitt nummer som tolkes som et binærtall
- Diskene som bare har ett bit satt (toerpotensene), blir paritetsdisker, og de defineres som modulo-2-summen av de datadiskene som har dette bitet satt

RAID 6 – Hammingkodeeksempel – I

disknummer				
paritet	*1*	0	0	1
	2	0	1	0
	4	1	0	0
data	*3*	0	1	1
	5	1	0	1
	6	1	1	0
	7	1	1	1

- Vi har 7 disker
- De tre diskene 1, 2 og 4 er paritetsdisker
- Disk 1 er modulo-2-summen av diskene 3, 5 og 7
- Disk 2 er modulo-2-summen av diskene 3, 6 og 7
- Disk 4 er modulo-2-summen av diskene 5, 6 og 7

RAID 6 – Hammingkodeeksempel – II

disknummer			
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

paritet: disk 1, 2, 4
data: disk 3, 5, 6, 7

- Anta at disk 4 og 5 kræsjer
- Siden disk 5 trenges for å regenerere disk 4,
  må vi først regenerere disk 5
- Disk 1 er modulo-2-summen av diskene 3, 5 og 7, så disk 5 er modulo-2-summen av diskene 1, 3 og 7
- Nå kan vi regenerere disk 4 som modulo-2-summen av diskene 5, 6 og 7

RAID 10

- RAID 10 er en blanding av RAID 0 og RAID 1
- I hvert disk-array (kabinett) ordnes diskene i par som speiles (RAID 1)
- Dataene blir så stripet over parene (RAID 0)
- Dermed kombineres hastigheten i RAID 0 med sikkerheten i RAID 1
- Det er dagens store og billige disker som har gjort RAID 10 konkurransedyktig
- RAID 10 er i ferd med å bli den vanligste driftsformen for store datasentre