Operativ Systemer 3

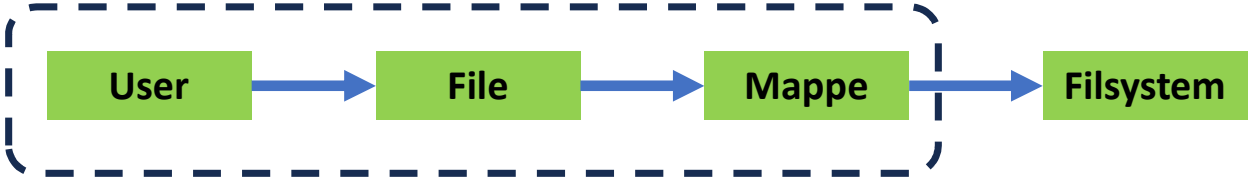
Lavet af: Vivek Misra

Filsystemer

Her får vi introduktion til filsystemer, hvor vi dykker nærmere inde i dem.

Filsystemer

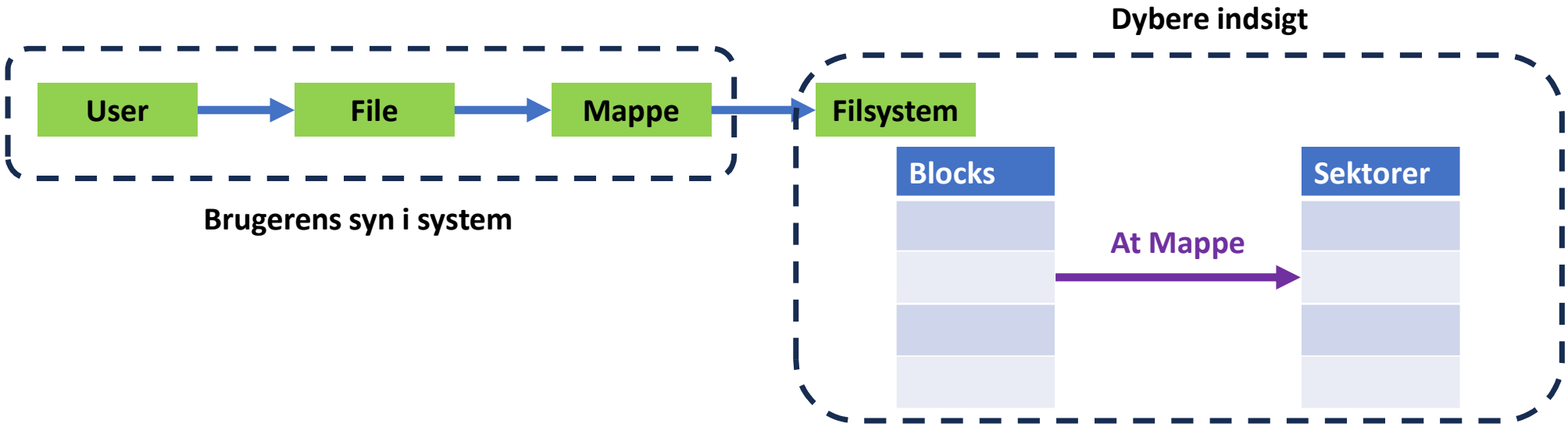
- Alle de store funktioner af en Operativ Systemer er Filsystemer.
- I Operativ Systemets Kernel finder man Filsystemet og det er en Software i OS.
  - Hensigten med Filsystemet er at kunne styre filerne i OS, altså en file-manager.
  - Filsystemet sørger for, at kunne holde styr på brugerens data i systemet og dermed gør det muligt for brugeren at kunne nå hen til filen.
  - Vi kan se, at brugeren danner en fil som i form af en fil. Filen befinder sig inde i en Mappe.
    - Man kan sige, at en mappen er samling af relaterede filer, som vi også kalder for Directory.



Brugerens syn i system

- Vi kan se, at det område som er markeret med sort er egentligt det område som brugeren har syn til filsystemet. Men når selve filet er gemt i mapperne, så er historien efter det anderledes for brugeren.

Filsystemer



- Vi kan se, at når vi går dybere inde i systemet så bliver filerne logisk divideret i blokke.
- Derefter bliver disse blokker ”mapped” inde i disken og her ”mapper” man ift. Sektorer.

Filens Attributter og Operationer

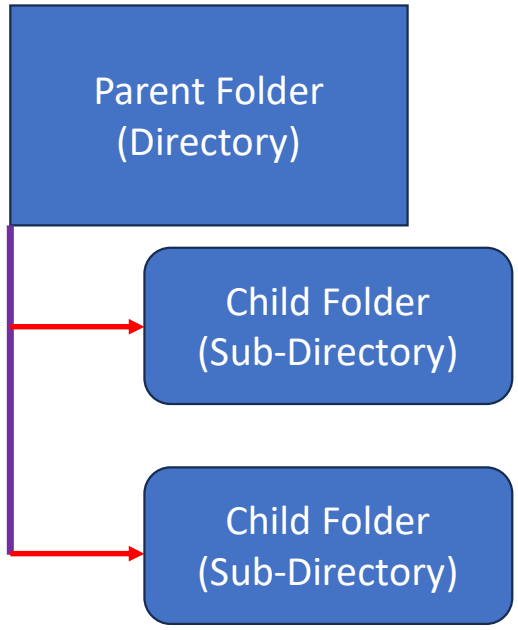
Operationer på filen:	Attributter på filen:
• Creating • Reading • Writing • Deleting • Truncating • Reportioning	• Name • Extension Type or .pdf .docs etc. • Identifier • Location • Size • Modified Date, Created Date • Protection & Permission • Encryption and Compression

KERNEL FILSTRUKTUR (HUSK)

- Hvis man kigger på filstrukturen, så kan man se at C:/ er kendetegnet for at være root-folderen.
- Root-folderen er den øverste folder af alle folderne i mappehierakiet.
- Hvorimod de andre folder som befinder sig under hinanden er kaldt for child-folder.
- Hvorimod folder som befinder sig ovenfor child-folderen er kaldt for parent-folderen.

Vigtige Ting om Filstruktur

- Det er vigtigt at understrege følgende fra figuren på sidste side:
  - Directory er en mappe, men under mappen kan der findes flere forskellige mapper.
  - Vi forbinder oftest associationer mellem Directory og Folders, hvilket teknisk set er det samme.



Allokeringsmetoder

- Hensigten med Allokeringsmetoder er, at kunne logiske dele filen i blokker og derved indsætte det i storrage som harddisk.
- Når man putter blokken i harddisk, så er det ikke bare harddisken man putter det i. Men det er mere præcist sektorerne i selve harddisk som man putter blokkene i.
- Når man putter selve blokkene i sektorer, så er der to måder man gøre det på:
  - Contigous Allocation
  - Non-Contingous Allocation
- EKSEMPEL PÅ CONTIGOUS ALLOCATION:
  - Det er der, hvor elev1 får plads 1 ved Eksamenslokalet, og de resterende elever får også den samme plads udefra deres tal.
- EKSEMPEL PÅ NON-CONTINGOUS ALLOCATION:
  - Det er der, hvor det ikke nødvendigvis er elev1 som får plads 1, men det kan være at et andet elev får plads 1 i Eksamenslokalet.

Allokeringsmetoder

- Nu kommer spørgsmålet, hvad får man egentligt ud af med Allokeringsmetoden?
  - Efficient Disk Utilization: Det er her, hvor vi udnytter GB eller Terabyte kapaciteten af disken på en effektiv måde.
  - Access: Det er der, hvor vi opsamler data på forskellige måder og hvor man så efterfølgende har adgang til dem på en (hurtigere) måde.
- Man kan teknisk set sige, at man skaber struktur på tingene.

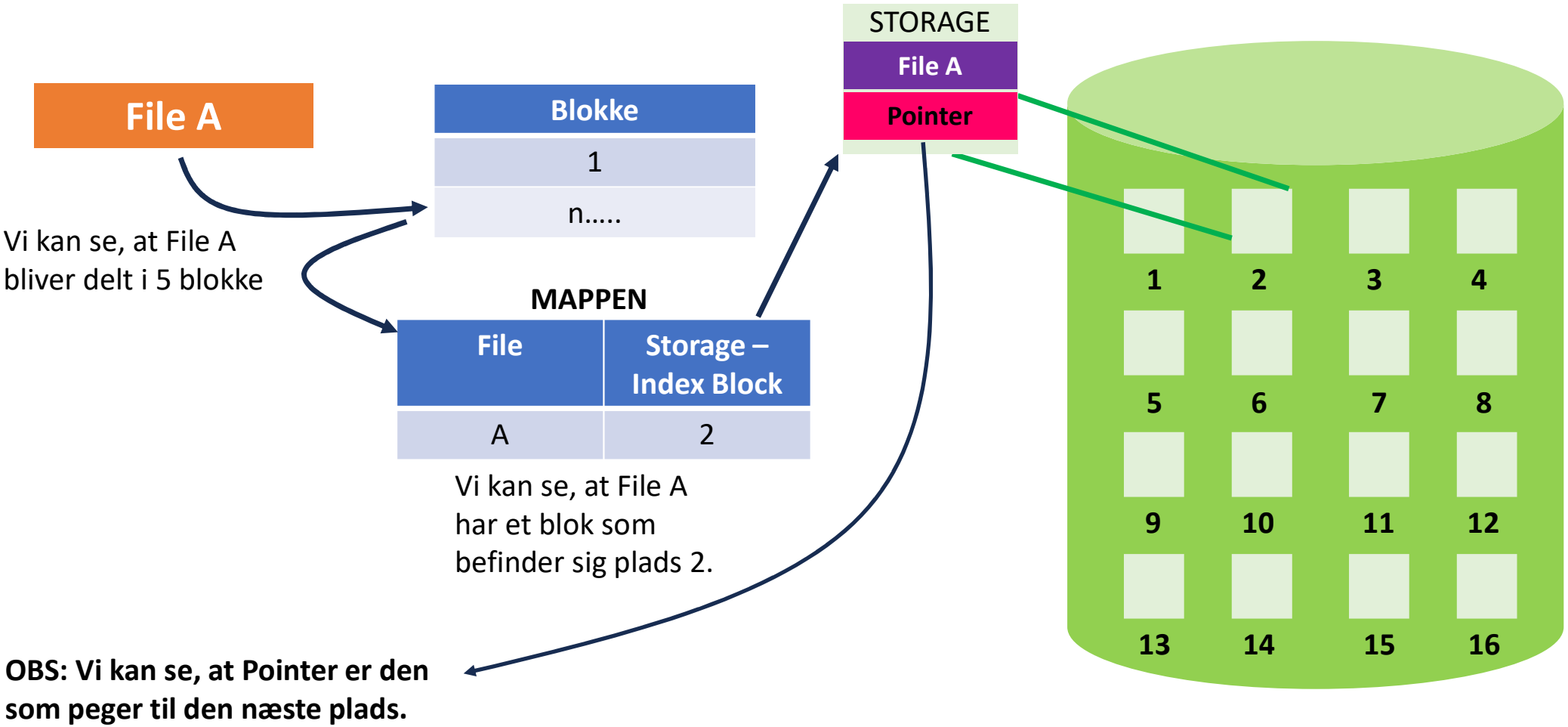
Sekventiel & Random Access

- Sekventiel Access: Det er en term som beskriver en gruppe af elementer (som eksempelvis data i en memory array eller en disk fil eller en magnetisk tape data storage) som der er adgang til i en defineret ordrede sekvens.
- Random Access: Det er der, hvor evnen til at kunne få adgang til et vilkårligt element i en sekvens på lige tid. Eller et element på hvilket som helst datum fra en population af adresserbare elementer omtrent lige så let og effektivt som ethvert andet, uanset hvor mange elementer de måtte være sættet.

Linked List Allokering

- Vi kan se, at inden vi putter en fil inde i storage, så plejer vi at inddele det inde i blokken.
  - Vi kan antage, at vi inddeler det inde i 5 blokke.
- Men når vi snakker om LinkedList Allokering, så plejer vi at referere til selve Non-Contingous som betyder at vi tilfældigt sætter blokkene inde i forskellige tal-pladser fra dem selv.
- Vi kan se, at I en LinkedList er tingene arrangeret i rækkefølge efter hinanden udefra placering. Vi kan derfor, se at når vi har en fil i mappen så ved vi på forhånd at vi eksempelvis har File 1 som er gemt ved Plads 2.
- Når man er ved Plads 2, så har man jo adgang til dataen af filen men man har samtidig også adgang til den næste plads hvor der er den næste fil.
  - DETTE ER MULIGT GENNEM ”POINTER” SOM PEGER TIL DEN NÆSTE FIL.

Linked List Allokering



Linked List Allokering

- Fordele:
  - No External Fragmentation: Det betyder, at vi er istand til at udfylde pladserne ud på en effektiv måde.
  - File Size Can Increase: Vi kan se, at vi allerede på forhånd har bestemt at vi inddeler filen i 5 blokke og indsat de ved de forskellige pladser i Alokkeringen. Men i tilfældet af, at hvis der nu sker en forøgning af blokkene så bliver det ikke sat inde ved pladserne ved anden omgang.
- Ulemper:
  - Larger Seek Time: Seek tid er den ønskede tid man ankommer til tracken på. Det betyder, at man skal køre hele tracken på en kontinuert måde som kan gøre at man skal læse pladserne igennem.
  - Random Access / Direct Access: Vi kender allerede den første blok's placering. Men hvad nu hvis vi kun ønsker at starte med finde block 4, så skal vi starte fra blok 1 og køre igennem til blok 4.
  - Overhead of Pointers: Det betyder, at hvis man i en fysisk blok i storage, så har man sammen med data i dette også nogle pointers som også fylder lidt i Memory.

Index List File Allokering

- Vi kan se, at index er en indeholdsfortegnelse som primært indeholder en liste over forskellige områder i en bog.
- Vi kan se, at vi har en Harddisk, hvor vi har en fil udenfor som er delt inde i blokke. Bagefter gemmer man blokkene inde i diskens storage.
- Vi ved, at jo formålet med at sætte blokkene inde i harddisken er ikke kun at gemme data, men det er også hensigten at kunne få fat i data'erne hvis brugeren får brug for dem.
- Vi kan se, at vi har en Directory som forklarer hvilken fil vi arbejder med og indeks blokke som fortæller hvor blokkene er filene er placeret henne fra den definerede plads som står i filen.

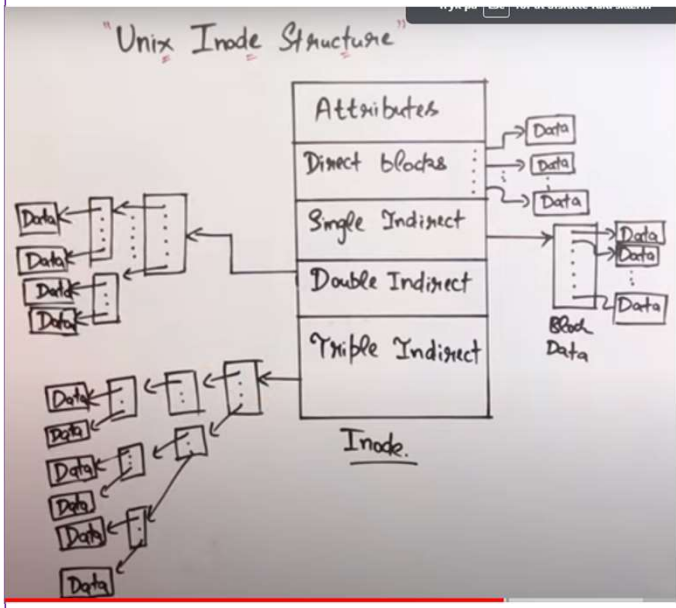
- ØNSKER AT DU KENDE MERE TIL DET, KAN DU BARE KIGGE PÅ LINKED LIST FILE ALOKKERINGSBILLEDE, SOM ER DET SAMME KONCEPT MED INDEX LIST.

Index List File Allokering

- Fordele:
  - Direct Access: Den støtter direkte adgang gennem index-block, hvor vi derfra kan få adgang til de forskellige index-blokke hvor de forskellige blokke af filer befinder sig af.
  - No External Fragmentation: Det betyder, at vi udnytter fuldt ud plads.
- Ulemper:
  - Pointer Overhead: Det er der, hvor vi har mange Pointers som vi skal bruge til at navigere rundt igennem indeks-blokkene og som også fylder blandt lidt plads i Memory'en.
  - Multilevel Index: Det er der, hvor vi har "booket" en plads hvor vi bruger det til at finde andre indekx-blokke. Men hvis nu vi har får mange blokke, så kan det være at vi skal "booke" flere indeks-blokke der er regnet til at føre hen til forskellige pladser.
    - Det kræver plads i Memory også!

UNIX-Inode Struktur

- Vi snakker om UNIX-Inode. Det kan ses, at ”I” betyder Indeks og ”Node” betyder blokke.
- Vi snakkede sidst om, at når filens størrelse bliver meget stort, så resulterer det også en større indeks-blokke i storage af Harddisken. Dette gør, at man ikke har mulighed for at gemme file-bloks i en blokken af harddisken.
- I dette tilfælde, anvender vi flere diskblokke hvilket gør at indeks-lokaliseringen bliver kompleks.
- Det vi gør så er, at vi laver en Hybrid Approach hvor vi gemmer attributter (metadata), direkte blokke (pointers til data-blocks), single indirect (sender til pointers, som leder til data-block). Så har vi dobbelt indirect (gør det samme som single indirect men bare 2 gange). Og det samme gælder for tripple indirect.

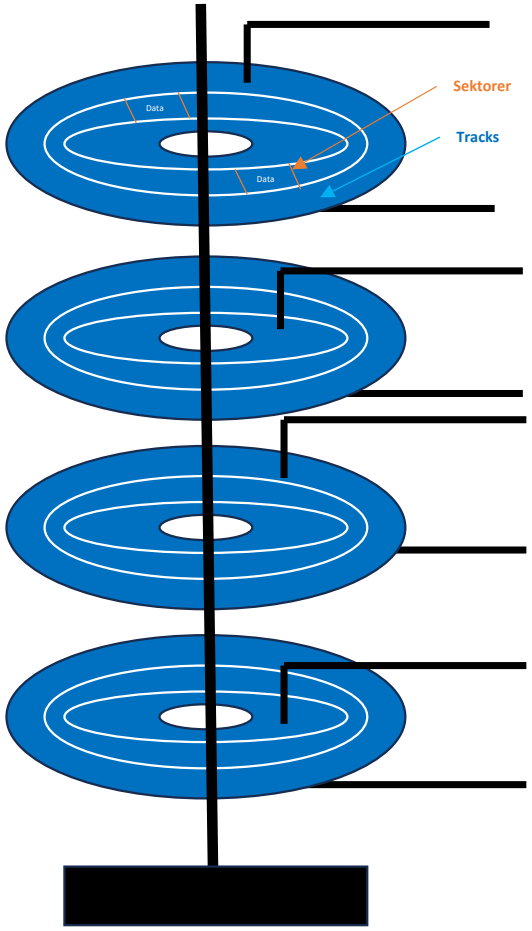


Disk og Hardware Arkitektur

Hardware-Arkitektur med fokus på CD-Disk, RAM, ROM, Cache Memory og andre Hardware Elementer.

Disk Arkitektur

- Til højre side kan man se, at vi har en 4 plader som er bundet sammen i en spindel.
  - Vi kan se, at pladen har en øvre overflade og nedre overflade.
  - Vi kan se, at vi har skrivepinde som er bundet til selve pladerne, både nedefra og ovenfra. Hensigten med disse skrivepinde er, at kunne læse data fra diskene.
  - Hvis man kigger yderligere nærmere, så kan det ses at der findes tracks ved pladerne.
    - Tracks findes både ovenpå overflade og under overfladen.
  - Vi kan se, at man på Tracks har forskellige Sektorer i form af streger i overfladen.
  - Det kan ses, at sektorer findes både ved øvre-overflade og nedre-overflade.
  - Imellem de forskellige sektorer findes der data, som bliver opsamlet af skrivepindene.



Hvad er ROM?

- ROM aka. Read Only Memory.
  - ROM er installeret i selve Motherboard.
  - ROM er en indbygget Software, som er dannet fra begyndelse og kan ikke ændres ved.
  - ROM er Non-Volatile, hvilket betyder at den fortsætter sit arbejde selvom hvis strømmen er gået.
    - Eksempel på dette kan være Vaskemaskinen, hvor den kører 20 minutter. Men når strømmen går og så kommer tilbage, så fortsætter ROM'en fra det tidspunkt den efterlod sidst og det er et eksempel på Nonvolatile. ROM glemmer aldrig!
    - Fjernbetjeningen er også et eksempel på brug af ROM. Her kan det ses, at alle knapperne er defineret på forhånd til at indskrive et tal ved aftrykning på knappen. Mikroovnen er også et eksempel på ROM.
- PROM aka. Programmable Read Only Memory.
  - Denne kan ikke ændre sig. Et eksempel på dette kan være starten af en Nokia-Telefonen, hvor to hænder mødes.
  - **MEN HVAD NU HVIS VI SKAL ÆNDRE LOGO'ET PÅ NOKIA ELLER LIGNENDE?**

Hvad er EPROM?

- Som vi snakkede sidst, hvad nu hvis virksomheden ønsker at ændre deres logo på Nokia telefonen? – Det er her, hvor EPROM kommer på spil.
- EPROM aka. Erasable Programmable Read Only Memory: Denne Memory kan ændres, men kompleksen er meget svær. Eksempelvis skal man gennem en hel masse stråling og sikkerhedskontro for at kunne fjerne data'et væk.
- **Med tiden udviklede EPROM videre og det gjorde, at vi fik EEPROM.**
- EEPROM aka. Elektromagnetic Erasable Programmable Read Only Memory.
- EEPROM er et eksempel på, hvor vi kan ændre selve dataen inde ved ROM'en på en bedre og mere nemmere måde.

Hvad er RAM?

- RAM aka. Random Access Memory. Det er denne her som laver alt vores arbejde.
- Random Access Memory er der, hvor vi trækker alt vores data inde ved RAM og udfører arbejdet på den.
  - Et eksempel på dette kan være forholdet mellem en Gryde og en Tallerken. Gryden er der, hvor alt mad bliver tilberedt men Tallerken er der hvor maden fra gryden bliver serveret på og spist på.
  - Alt filer fra Harddrive bliver rykket til RAM, for at kunne udføre arbejde.
  - RAM er Volatile, som betyder at når strømmet går ud så ender dataene med at forsvinde. Når strømmen forsvinder går data'en fra RAM til Harddrive.
  - **DET ER DERFOR VI ALTID GEMMER ELLER "SAVER" VORES ARBEJDER, NÅR VI ER FÆRDIG MED AT LAVET DET. PÅ DEN MÅDE BLIVER DET GEMT I HARDDRIVE OG VI KAN TILGÅ DET IGEN!**
  - ***DER FINDES TO FORMER RAM, SOM VI VIL FORKLARE PÅ NÆSTE SIDE!***

Hvad er RAM?

- Som nævnt findes der to former for RAM:
  - DRAM aka. Dynamic RAM
  - SRAM aka. Static Ram.
- Dynamisk RAM er der, hvor man laver alt arbejde fordi det er dynamisk!
- Men i Statisk RAM er der særlige ting som er gemt derhenne. Det kan ses, at ting som gemmes i SRAM gemmer ting på den måde som vi har lyst til at ændre på. Men det funktionere ift. SRAM opbygning.
  - Eksempel på brug af SRAM kan være kalender-uret, hvor det kan ses at når timerne rykker frem og tilbage i Vinteren og Foråret, så indstiller vi uret frem og tilbage. Men urets fortsatte drejning styres af SRAM.
- Der findes mange forskellige typer af RAM, som eksempelvis DDR1, DDR2, DDR3 og DDR4. Hvor DDR4 er den seneste udkommet.

Hvad er Harddisk?

- Harddisk Drive er en CD-formet men hård hardware.
  - Vi plejer, at kalde denne enhed for Harddrive (HDD).
  - Alle data som vi gemmer er lagret herhenne i HDD.
  - EKS: Når vi ønsker at se en video fra noget Harddisk, så henter vi data'et fra Harddisk og nedhenter det i RAM'en.
  - Harddisk er meget stort og kræver meget strøm i computeren.
  - Samtidig med dette er Harddisk meget risikabelt for Data-Loss, eftersom en lille "krads" på overfladen af Harddisk er med til at ødelægge data ved Harddisk som gør at brugeren gemte filer bliver ødelagt.
  - **I stedet for at være meget Fragile, fandt Ingeniørerne en anden metode til at gemme store data på og denne Hardware er SSD.**

Hvad er SSD?

- Solid State Drive aka. SSD.
- SSD har den samme arbejde som HDD, men forskellen er bare at den fysisk mere robust samtidig med mere hurtigt og effektiv til at levere data i RAM'en.
- Fordi SSD'en er meget lille ift. HDD er den ikke så meget energikrævende. Dette betyder, at den bruger ikke så meget elektricitet.
- Men det er meget vigtigt, at gøre obs på at SSD'en er meget dyrt sammenlignet med HDD'en.
- **Hvad er bedst og bruge og hvornår?**
- Det kan anbefales, at have både Harddrive og SSD i Computeren. Men der er anbefalinger:
- HDD: Alle filer som vi ser sjældent kan gemmes her.
- SSD: De applikationer som der udføres arbejde på, kan gemmes her.

Hvad er Cache Memory?

- Cache Memory er den Hardware som gemmer midlertidig data, der benyttes igen og igen.
  - Eksempler på Cache Memory kan være søgeresultater, hvor det kan ses at Cache Memory gemmer søge resultater og bringer det oppe på skærmen.
  - Ting som bruges gentagne gange, er gemt i selve Cache Memory.
  - Cache Memory er den som befinder sig tæt på CPU'en og har 3 lager i sig.
  - Cache Memory er den hurtigste Memory af dem alle.
  - Cache Memory er selve forbindelsen mellem RAM og Harddisken, fordi den hjælper med at "fetch" data fra Harddisk og over i RAM.

Kategorier af Memory

- Vi kan se, at vi har følgende Kategorier for Memory:
- Computer: (Primary Memory)
  - Cache Memory
  - ROM
  - RAM
- User (Secondary Memory)
  - HDD / SSD
  - Pandrive
  - Flopy

Hvad er Grafik-Kort?

- I Computeren har man en Grafikkort, som er styret af Grafik Processing Unit (GPU).
- Grafikkortet er den som bestemmer farve på computerskærmens farver i billeder og spil.
- Man har blæser i Grafikkortet, som er med til at holde den kold ved Motherboardet.
- Grafikkortet er i stand til at inddele farver i flere kategorier.

RAID og Storage

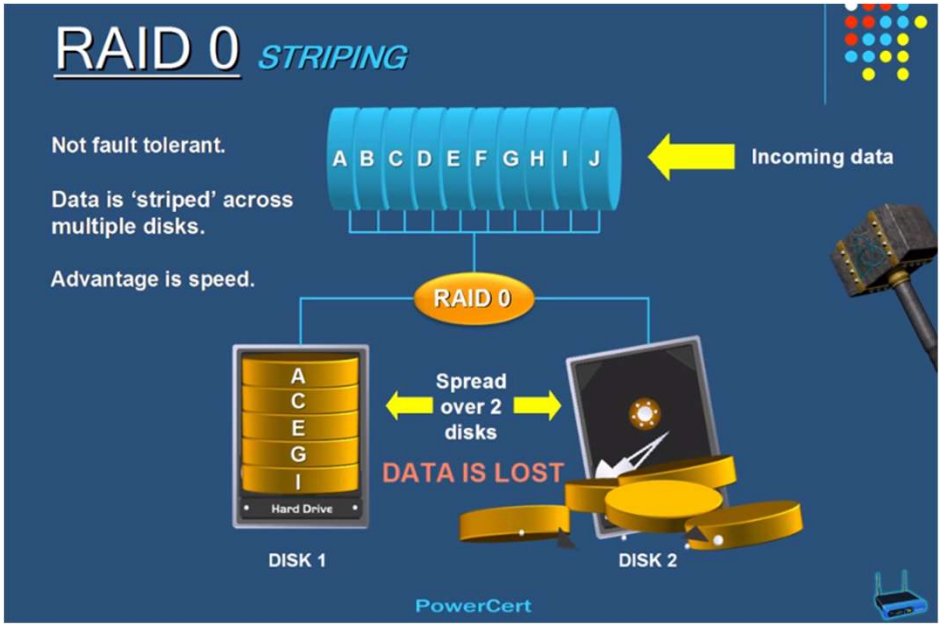
Nu kommer vi til at undersøge nærmere omkring, hvilken betydning RAIDs har for storage.

Hvad er RAID?

- Vi kender allerede, at Storage er en vigtigt del i at kunne lagre data i Memory.
- Det er også mening, at kunne sikre data således at det ikke er Volatilt til at forsvinde væk.
- Vi skal være opmærksomme på, at hvis en Disk fejler i Storage så skal vores Data ikke forsvindes væk!
- Den bedste måde, at forhindre data-tab er gennem RAID.
  - RAID står for (Redundant Array of Independent Disks).
- **Hensigten med RAID er, at kunne kopiere den enkelte data fra den enkelte disk over i forskellige diske. Så der i tilfældet af brydning hos den enkelte disk, ikke sker datatab, men omvendt dataene opbevares hos de andre diske.**
- Der findes forskellige former af RAID, som vi kommer til at beskrive på de næste slide.

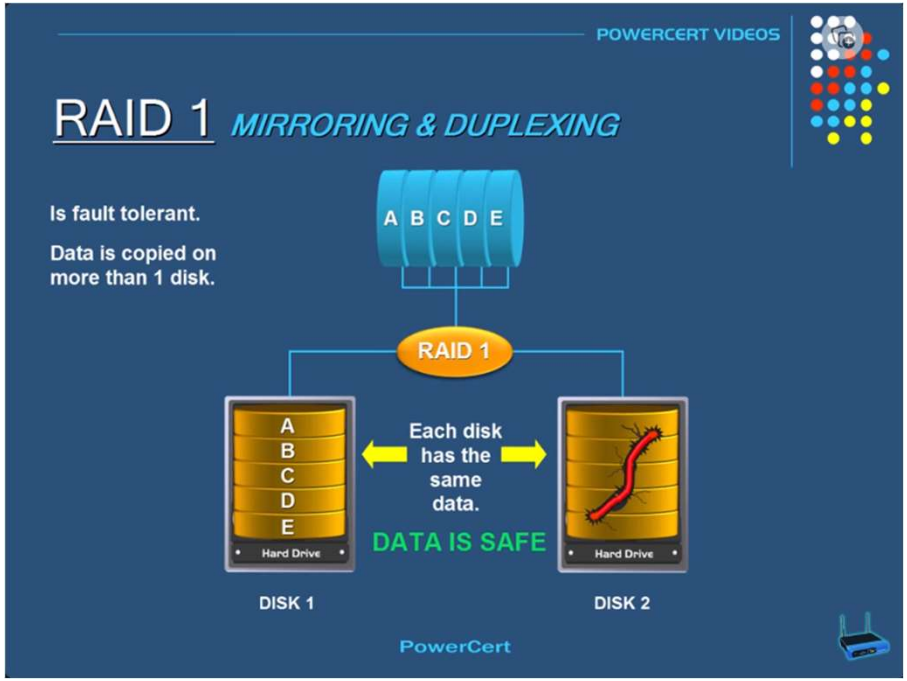
RAID 0

- **RAID 0 er den RAID som har størst risici for datatab.**
- **Vi kan se, at RAID 0 er der, hvor data er spredt i forskellige diske.**
- **Her kan vi eksempelvis se, at den indkommende data er spredt i Disk 1 og Disk 2.**
- **I tilfældet af, at Disk 2 bliver beskadiget, så bliver halvdelen af dataet tabt, da dataene befindende i Disk 2 er ikke kopieret over i Disk 1.**



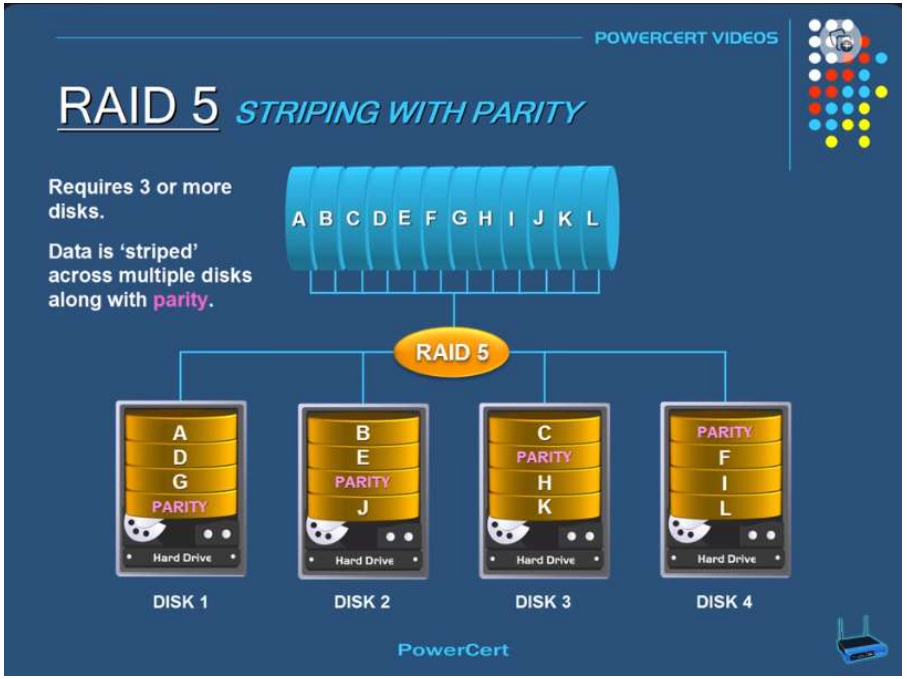
RAID 1

- **Ved RAID 1 er der mindre datatab, fordi vi kan se at den inde kommende data bliver kopieret over i to forskellige diske.**
- **Dette betyder, at hvis vi har indkommende data som er A,B,C,D,E og som er sat både inde i Disk 1 og Disk 2.**
- **Og i tilfældet af Disk 2 bliver ødelagt, så er dataene fra Disk 2 allerede på forhånd bevaret i Disk 1 – da der er en kopi af dem.**



RAID 4 -> RAID 5

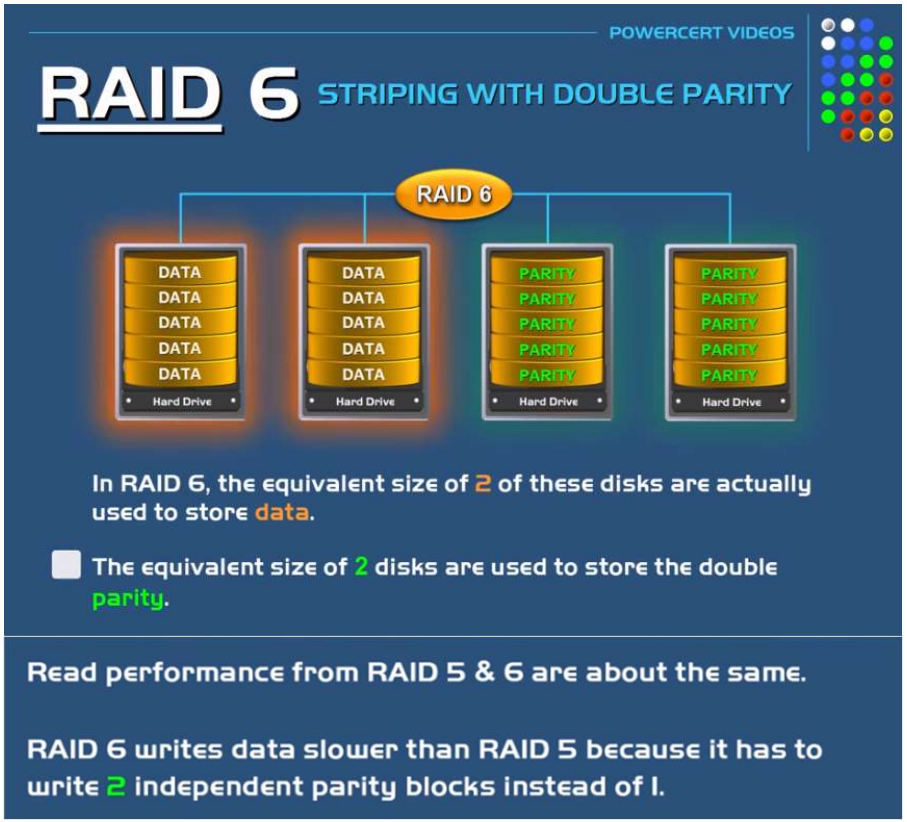
- **RAID 5 er den mest almindelige opsætning som er brugt, da den er hurtigt og kan gemme en stor mængde af data.**
- **Vi kan se, at dataene er spredt (ikke duplikeret) i forskellige Disk, men samtidig med det har vi også noget som er spredt i mellem dem og som er kaldt ”Parity”.**
- **Hensigten med Parity er, at kunne genoprette de mistede data i tilfældet af en Disk bliver ødelagt.**
- **RAID 5 er kun designet til at håndtere en enkelt Disk-Fejl og ikke 2. Hvis 2 Diske fejlers, så er alt dataet tabt.**



I RAID 4 lå alle ”Parity” eksempelvis i Disk 4, men så blev de spredt rundt i en linære orden hvilket gjorde at RAID 5 blev dannet.

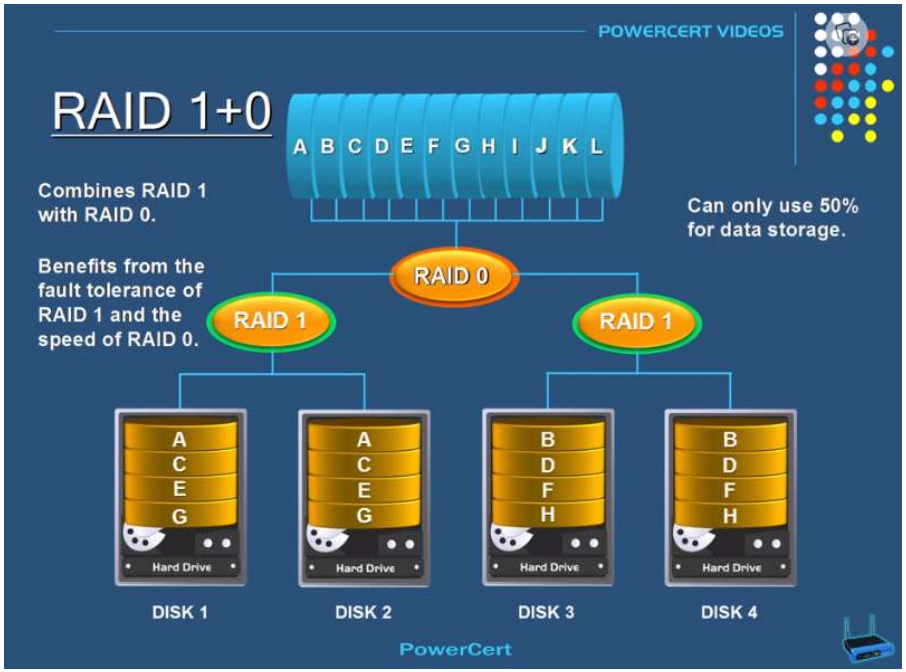
RAID 6

- **RAID 6 er ligesom RAID 5, men her er RAID 6 er istand til at kunne håndtere fejl ved 2 Diske.**
- **Hvis to Diske fejler i RAID 6, skal der bare indsætte nye diske på stede ligesom ved fejl i enkelt Disk hos RAID 5.**
- **Så skal de andre Diske bruge deres ”dobbelt Parity” til at kunne genetablere dataerne i de nye Diske.**



RAID 10

- **RAID 10 er den som kombinerer RAID 0 og RAID 1 sammen.**
- **Som vi kan se, at har RAID 0 en fordel ved at kunne sprede data i hurtigt igennem.**
- **Fordelen ved RAID 1 er, at den er god til at kopiere og beskytte data ovre i forskellige diske.**
- **Når de to RAID's bliver koblet sammen, så bliver dataudspredning ikke kun hurtig og effektiv, men det bliver også meget sikret!**
- **Men ulempen er kun, at man kan anvende kun 50% af den samlede lagring.**



SLUT 3

Lavet af: Vivek Misra