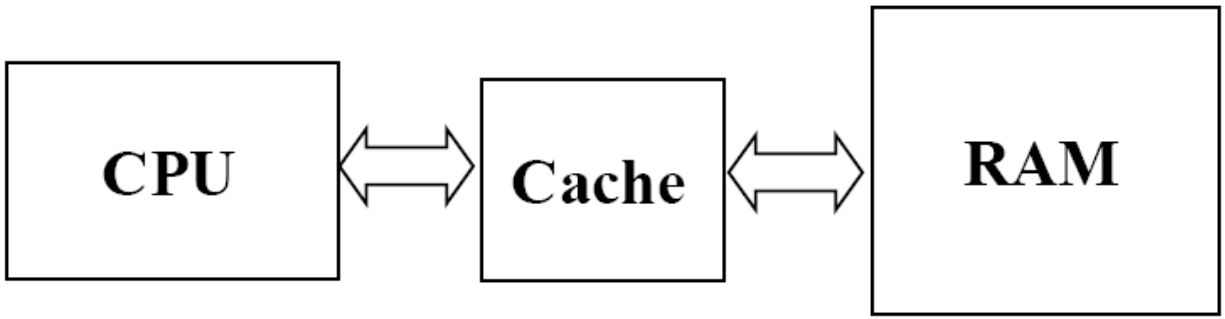
Dagens tema

- Mer om cache-hukommelse
  - Kapittel 6.5 i ”Computer Organisation and Architecture”)
- RAM
  - Typer, bruksområder og oppbygging
- ROM
  - Typer, bruksområder og oppbygging
- Virtuell hukommelse (kapittel 9.9 i læreboken)
- Input-Output (I/O)

Cache (repetisjon)

- Formål: Ønsker så mye og rask hukommelse som mulig tilgjengelig for et program under eksekvering (instruksjoner og data)
- Cache-minnet er raskere enn RAM (ca 10-100 ganger), men mye dyrere slik at man ikke kan ha like mye av det som RAM
- Cache inneholder kopier av områder i RAM
- Cache er integrert på CPU'en og/eller plassert rett ved siden av



Cache (repetisjon)

- Siden cache har mindre kapasitet enn RAM består mesteparten av administrasjonen av cache i å velge ut hvilke instruksjoner/data som kan ligge i cache, og hvilke som må ligge i RAM.

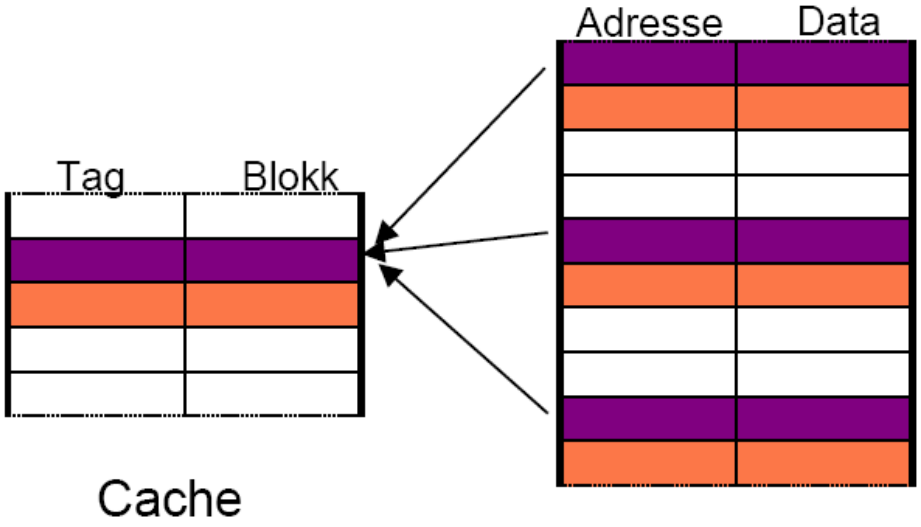
- Cache benytter seg av **lokalitetsprinsippet**:
  - Instruksjoner/data aksesseres som regel sekvensielt, dvs. fra samme område i hukommelsen
  - Konsekvens: De samme instruksjonene/data aksesseres også nær hverandre i tid

- Cache kopierer derfor hele blokker (størrelse fra 4 byte opp til noen kilobyte) av gangen.

Indeksering av cache og blokkoverskriving

- To gjenstående utfordringer ved design av cache:
  - Hvordan cache-kontrolleren indekserer (dvs. leter fram) riktig blokk ved lesing/skriving
  - Hvilken blokk som skal overskrives/kastes ut når cache er full
- Det finnes tre ulike indekseringsteknikker for blokker i cache:
  - Direkte-avbildet
  - Set-assosiativ
  - Full assosiativ
- Metodene skiller seg fra hverandre ved:
  - Hvor enkle de er å implementere
  - Hvor raskt det går å finne en blokken/tag
  - Hvor god utnyttelse man får av cache
  - Hvor ofte man får cache miss.

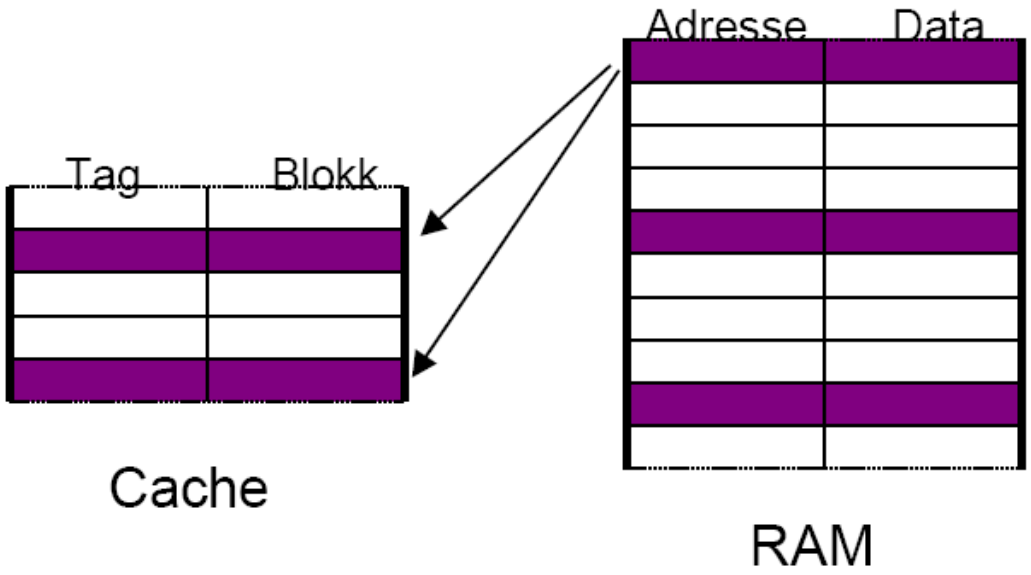
- Direkte-avbildet: En bestemt blokk fra RAM kan bare plasseres i en bestemt blokk i cache. Flere RAM-lokasjoner må “konkurrere” om samme blokk i cache.



- **Fordel:** Lett å sjekke om riktig blokk finnes i cache: Sjekk kun ett tag-felt og se om denne inneholder blokken man leter etter.

- **Ulempe:** Høy miss-rate (selv om det er ledig plass andre steder, kan en blokk fra RAM kun plasseres ett bestemt sted)

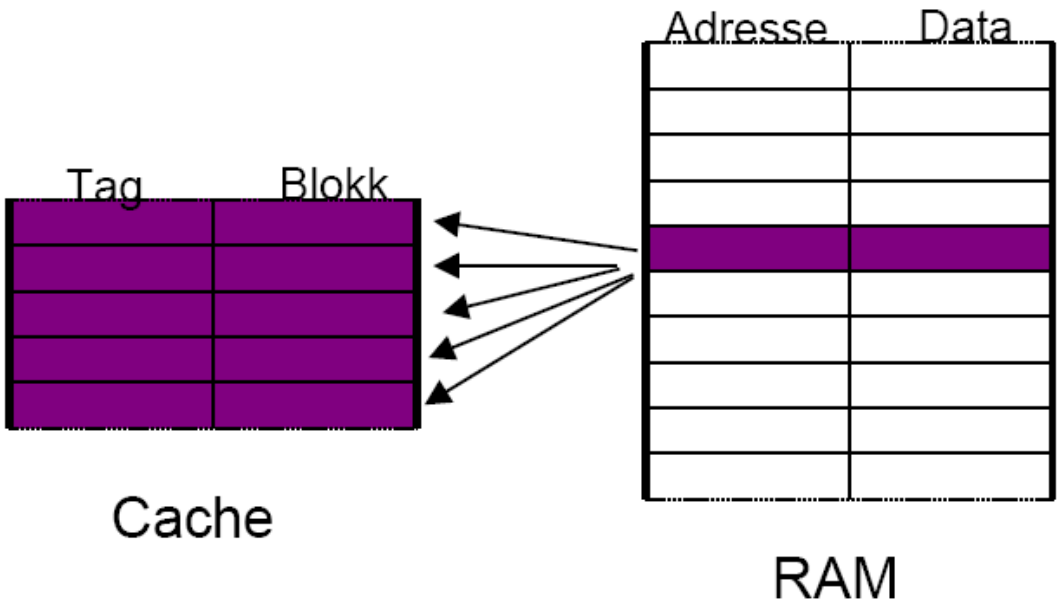
- Set-assosiativ: En bestemt blokk fra RAM kan plasseres i et begrenset antall blokk-lokasjoner i cache. En tilgjengelig lokasjon kallen en “way”



- **Fordel:** Lett å sjekke om riktig blokk finnes i cache: Søk gjennom et begrenset antall tag-felt og se om blokken man leter finnes i cache. Bedre utnyttelse av cache fordi en blokk kan plasseres flere steder

- **Ulempe:** Lenger søketid enn ved direkte-avbildet (med mindre man søker i parallell gjennom tag-feltene).

- **Full assosiativ:** En bestemt blokk fra RAM kan plasseres hvor som helst i cache



- **Fordel:** Cachen utnyttes meget godt, og har minst sjanse for cache miss av de tre metodene

- **Ulempe:** Søket for å finne en blokk kan bli tidkrevende, og man lager mekanismer for å forenkle søk, f.eks hashing. Dette kompliserer cachekontrolleren og krever ekstra hardware. Brukes ved cache < 4KB

- Ved cache miss må noen ganger en eksisterende blokk kastes ut for å gi plass til en ny blokk. Hvilken blokk som fjernes kan ha stor betydning for hastigheten til programmet som eksekveres.

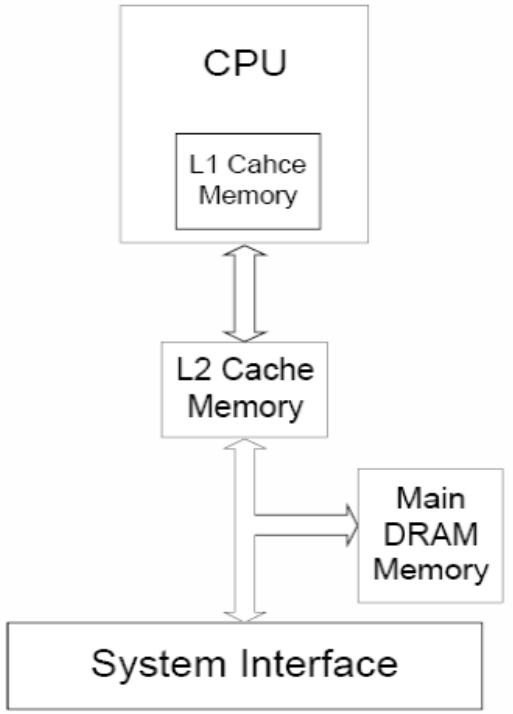
- **LRU (Least Recently Used):**
  - Blokken som har ligget lengst i cache *uten å ha blitt skrevet til eller lest* skrives over, pga. lokalitetsprinsippet
  - Ren LRU benyttes sjelden fordi det medfører mye administrasjon som i seg selv er tidkrevende (bl.a må et tidsstempel oppdateres hver gang en blokk aksesseres)

- **Random**:
  - Kaster man ut en tilfeldig blokk når man trenger å frigi plass til en blokk.

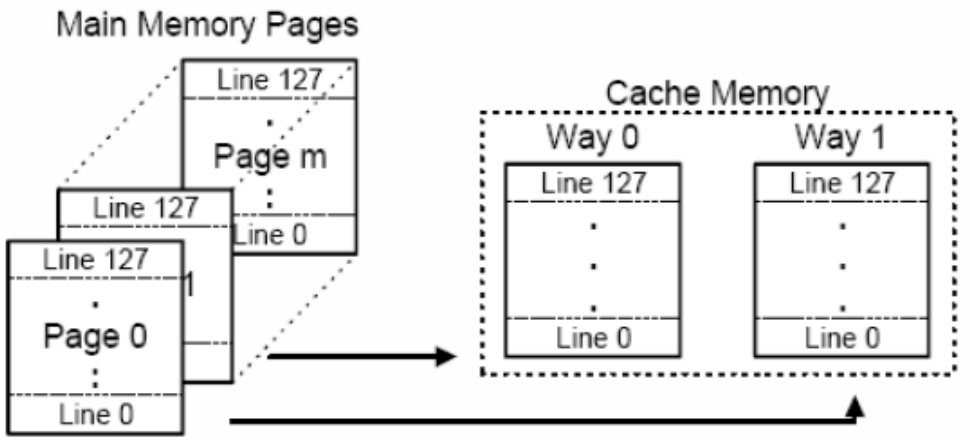
- **Hybrid**:
  - Deler inn blokker i tidsgrupper, og kaster så ut en tidfeldig valgt fra den gruppen som har ligget lengst i cache uten å bli brukt.

Cache i Intels Pentium III-arkitektur (1)

- Cache i Pentium har to nivåer, L1 (on-chip) og L2 (off-chip)
- Nivået angir rekkefølgen de aksesserer, ikke hvor de fysisk er plassert
- Bussbredden til L1 er 256 bit, mens den er 64 bit til L2 (endret i senere versjoner)
- Separat data- og instruksjons-cache er hver på 8KB

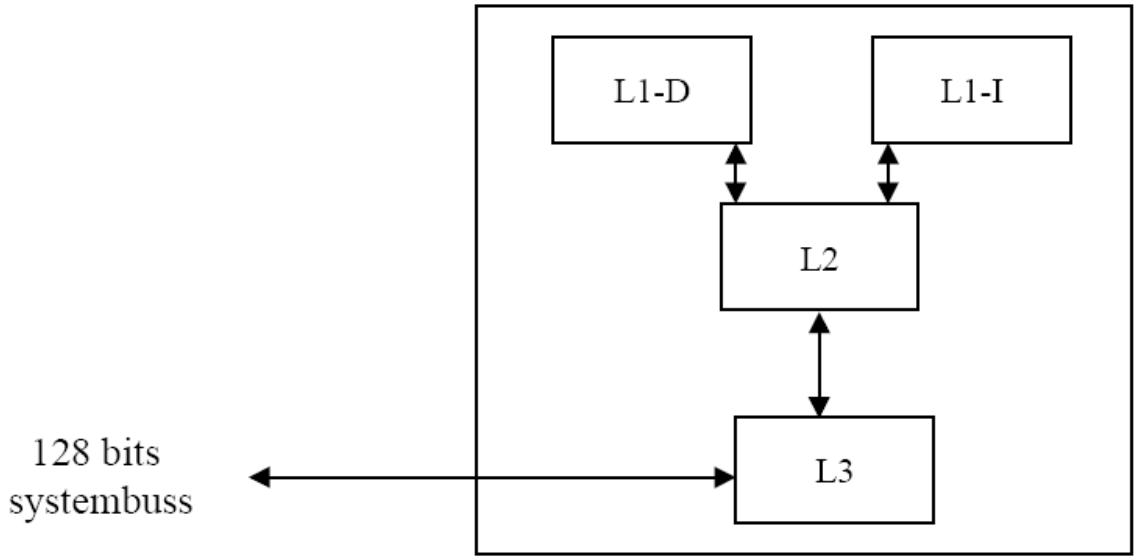


Cache i Intels Pentium III-arkitektur (2)



- Både L1 og L2 er 2-veis assosiativ
- Cache linjebredden er 256 bits, og fylles ved 4 påfølgende lese-operasjoner
- Sidestørrelsen er 4K eller 128 linjer (128 ways = sider i set-assosiativ cache)
- Mulig å styre i software om cache skal være ”write-back” eller ”write-through”
- Mulig å disable L1 cache i software
- L1 er look-through, mens L2 er look-aside

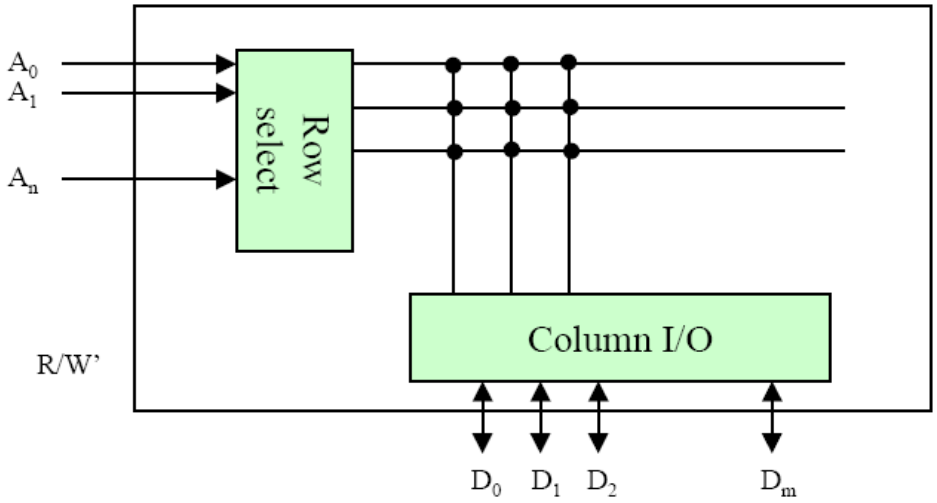
Cache i Intels Itanium I-arkitektur (1)



- L1-D og L1-I er 4-veis set-assosiativ og hver på 16 KB
- Kan håndtere 2 load og 2 store samtidig
- L1-D håndterer ikke flyttall, dette skjer i L2 som er på 256 KB og 8-veis set-assosiativ
- L3 er 1.5 eller 3 MB, og pipelinet for å takle opptil 8 load/store operasjoner

RAM-typer

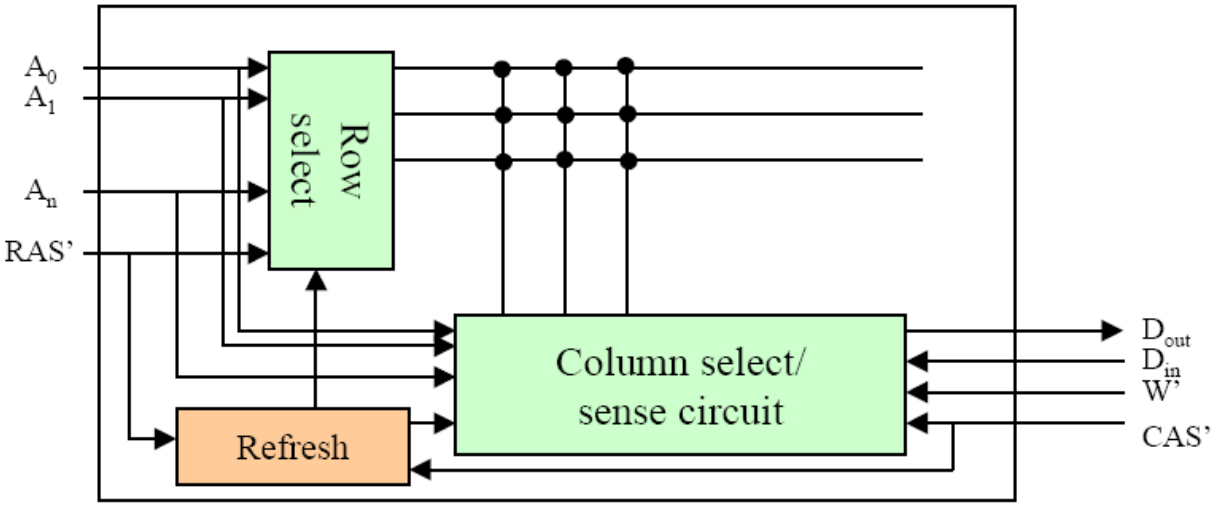
- RAM kan implemeteres på ulike måter, avhengig av bruksområde
- De to vanligste hovedtypene er **Statisk RAM** og **Dynamisk RAM**
- **Statisk RAM**: Hver enkelt lagercelle er laget av en RS-flipflop med transistorer
- **Fordeler**:
  - Kort aksesstid
  - Holder lagret verdi så lenge det er spenning på chip'en
- **Ulemper**:
  - Hver celle trenger 5-10 ganger mer plass enn dynamisk RAM
  - Dyrere per bit



2008

RAM-typer (forts.)

- **Dynamisk RAM**: Hver enkelt lagercelle er laget av en kondensator
- **Fordeler**:
  - Mer kompakt enn SRAM
  - Billigere per bit enn SRAM
- **Ulemper**:
  - Innholdet blir borte etter kort tid (15ns)
  - Trenger ekstra logikk for å friske opp innholdet slik at lagringen blir permanent



ROM

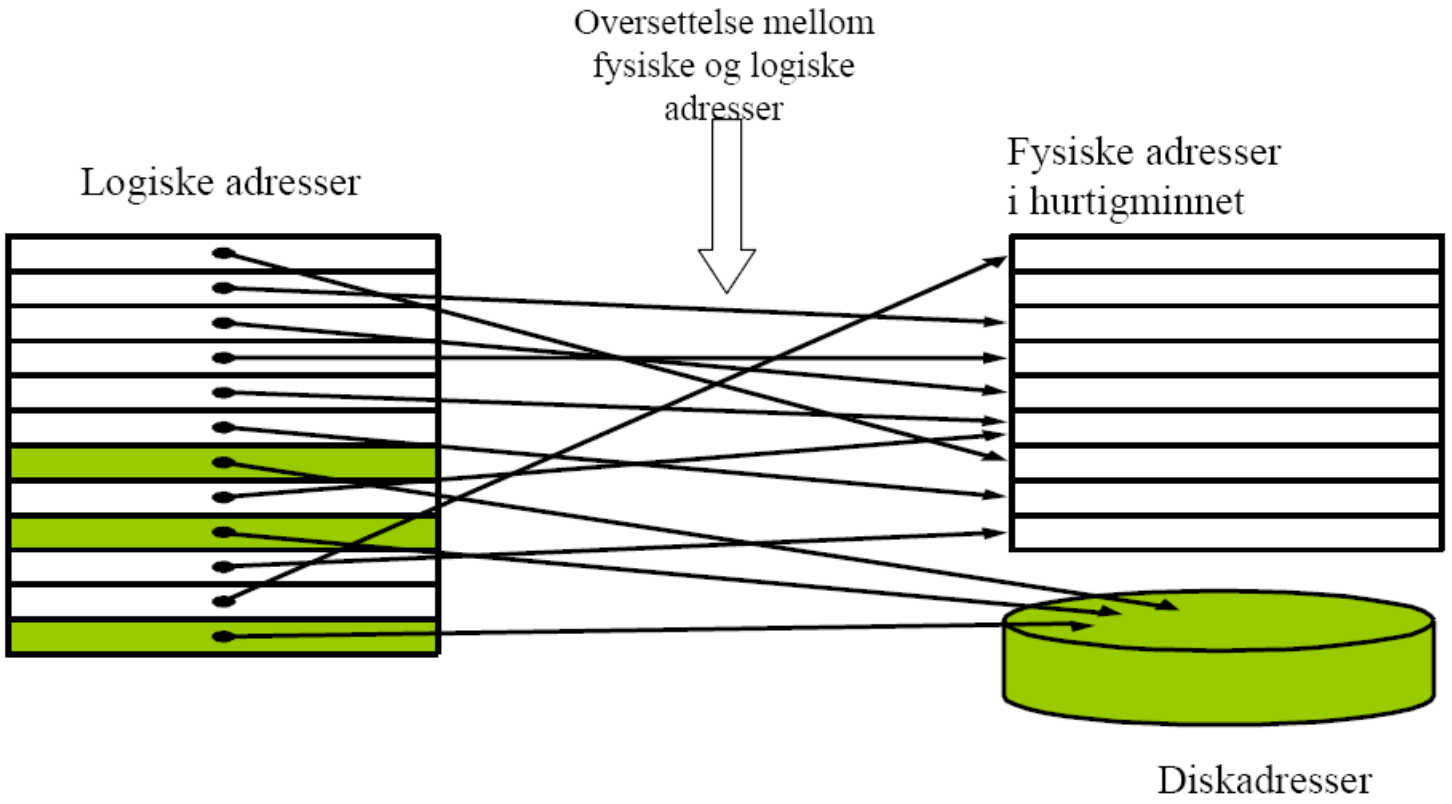
- I motseting til RAM kan Read-Only Memory (ROM) kun skrives til et begrenset antall ganger
- Den vanligste formen er ROM som enten
  - prefabrikeres (hard-wired) med et fast innhold i hver celle
  - programmeres én gang av brukeren (PROM)
- EPROM er en type som kan nullstilles og programmeres igjen
  - F.eks ultrafiolett lys kan brukes til å nullstille koblingspunkter
- EEPROM bruker elektrisk spenning istedenfor UV-lys for å nullstille
- NVRAM er svært utbredt i dag
  - Kan re-programmeres ca 100 000 ganger
  - Skriving er flere størrelsesordener langsommere enn lesing
  - Brukes i bl.a. MP3-spillere og mobiltelefoner

Virtuell hukommelse

- Ofte trenger et program/prosess mer RAM enn det som er tilgjengelig fysisk
- Et program deler RAM med andre programmer og prosesser, bl.a:
  - Operativsystemet
  - Driverrutiner
  - Bakgrunnsjobber (f.eks utskrifter, filoverføring)
  - Brukerprogrammer som kjøres samtidig
  - Programmer som tilhører forskjellige brukere
- ***Virtuelt minne***: RAM utvides med plass på harddisken, slik at et program kan adressere et større område som RAM enn det som faktisk er tilgjengelig.
- Siden data kan plasseres i RAM og/eller på harddisken brukes begrepet logisk adresse i stedet for fysisk adresse om de lokasjoner som et program aksesserer.

Virtuell hukommelse (forts.)

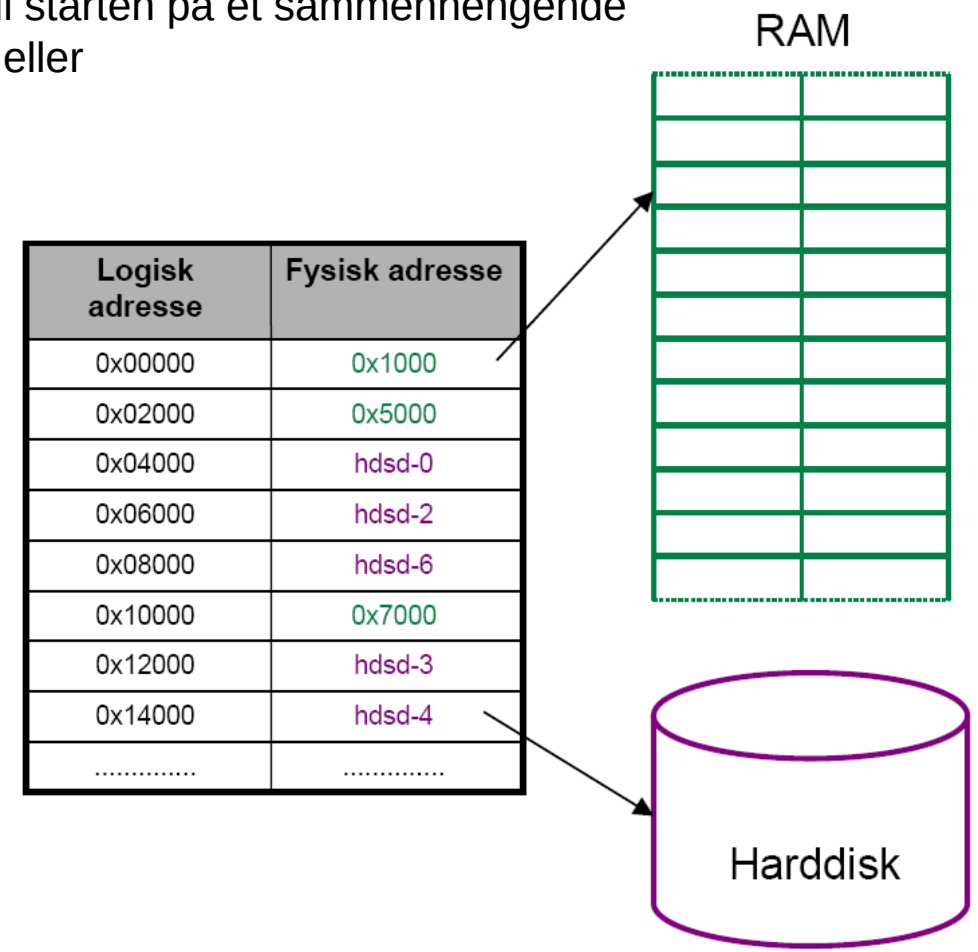
- Maskinen må ha en mekanisme som gir kobling mellom logiske (virtuelle) adresser og fysiske adresser



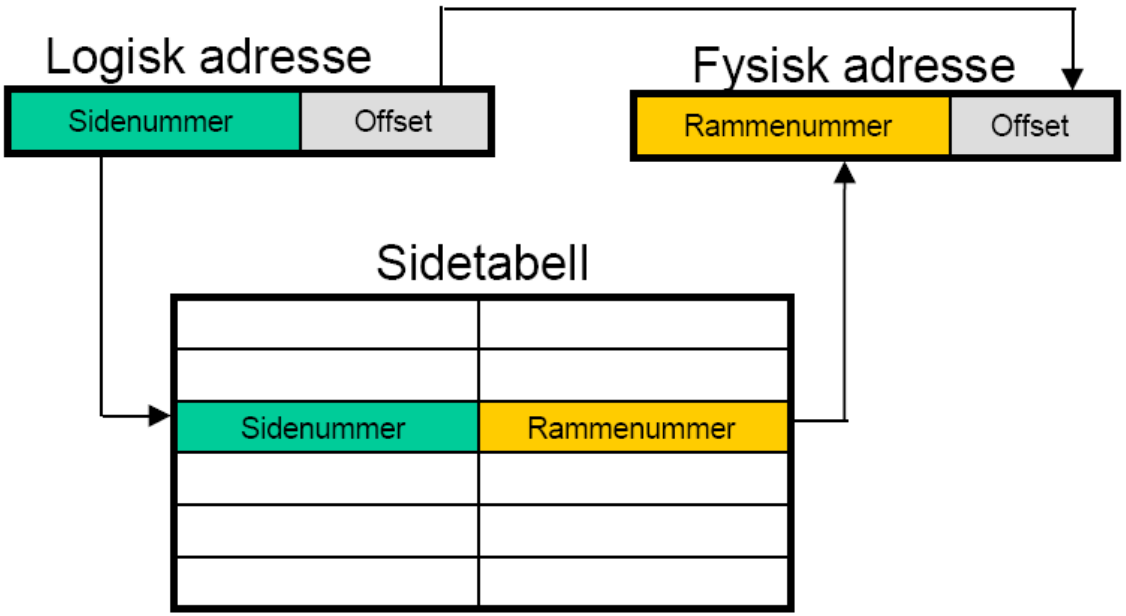
Sidetabell

- Sidetabellen brukes for oversettelse mellom logisk og fysisk adresse:
- Sidetabellen inneholder adressen til starten på et sammenhengende minneområde med en kjent lengde eller en fildeskriptor (indeks på disken).

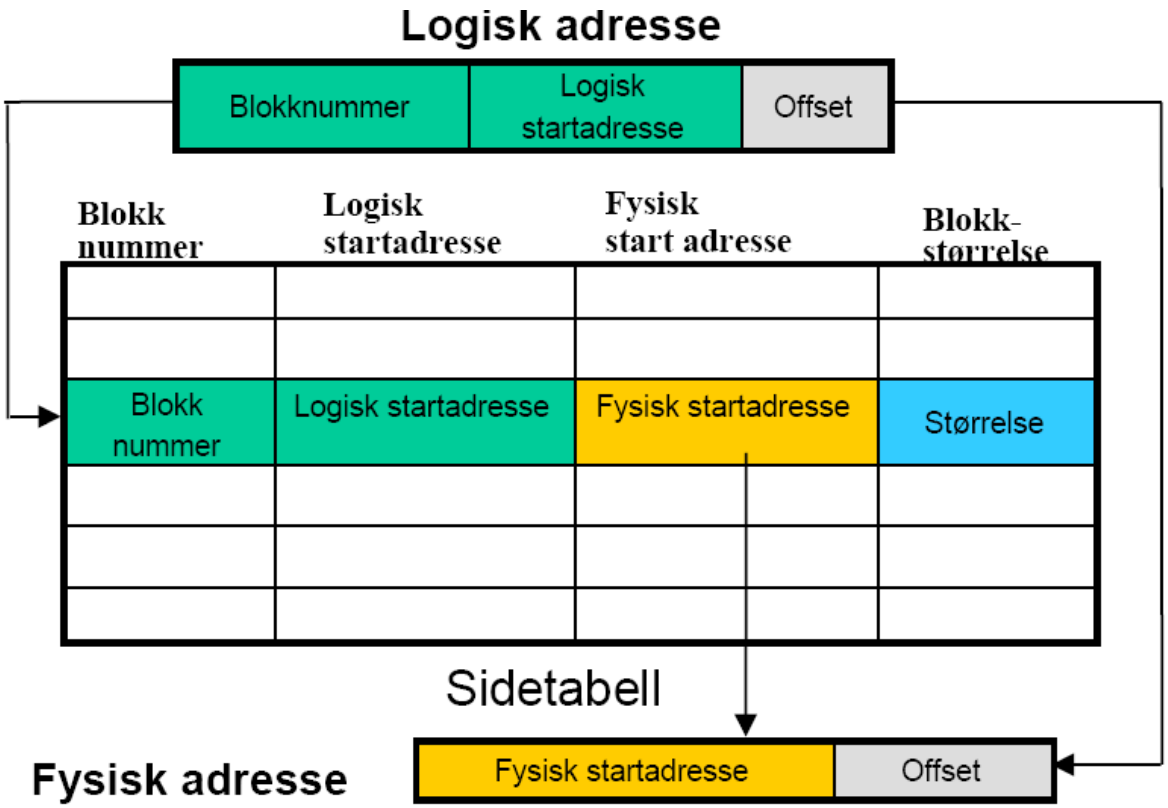
Logisk adresse	Fysisk adresse
0x00000	0x1000
0x02000	0x5000
0x04000	hdsd-0
0x06000	hdsd-2
0x08000	hdsd-6
0x10000	0x7000
0x12000	hdsd-3
0x14000	hdsd-4
.....	.....



- Virtuelt minne kan implementeres enten som paging eller segmentation
- **Paging**: Hukommelsen organisert som en én-dimensjonal array hvor hver adresse i sidetabellen refererer til starten av et minneområde med fast lengde.
- En logisk adresse består da av to felter: Et *sidenummer* og et *offset*:



- **Segmentation** (to-nivå paging): Hukommelsen organisert som en to-dimensjonal array hvor den logiske adressen består av et blokknummer og en startadresse. Sidetabellen gir overgangen fra logisk til fysisk startadresse, og størrelsen på blokken.



Segmentation kontra paging

- Begge støtter virtuell hukommelse, dvs. isolerer logisk og fysisk adresserom, slik at fysisk hukommelse kan realiseres på ulike måter uten at dette er synlig for prosessoren.
- Med segmentation er det lettere for operativsystemet å holde ulike prosessers hukommelsesområder adskilt fra hverandre
- Segmentation er mer fleksibelt fordi ulike prosesser kan få tilpasset størrelsen på sine minneområder uavhengig av sidestørrelser (med paging er sidestørrelsen fast, med segmentation settes den uavhengig for hver side).
- Segmentation er mer komplisert å implementere, og krever mer hardware.

Plassèring av sider i RAM eller på disk

- Virtuelt minne har samme utfordring som cache: Man må bestemme hva som skal ligge i RAM (liten og rask), og hva som skal ligge på disk (stor og langsom).
- Forskjellen mellom aksesstid for RAM og harddisk er enda større enn mellom RAM og cache; opptil 10 000 til 100 000 ganger (nanosekunder vs millisekunder)
- Kostnaden ved *page fault*, (at en side ikke finnes i RAM) er svært stor.
- For å redusere omfang og konsekvens av page fault må man:
  - Bruke stor nok sidestørrelse (f.eks 4 KB). Små sidestørrelser gir hyppigere page fault
  - Bruke full asossiativ plassering av sider i det fysiske adresserommet slik at minnet utnyttes best mulig
  - Bruke 'write-back' istedenfor 'write-through' for å redusere anntall skrivinger til disk

Gjenfinning av sider

- For å redusere sjansen for ’page fault’ brukes nesten alltid full assosiativ plassering: Når en ny blokk hentes inn fra harddisken kan den plasseres på en vilkårlig ledig plass i RAM.
- For å finne og plassere en bestemt blokk fysisk bruker man en komplett sidetabell slik at man slipper et fullt søk hver gang.
- Sidetabellen gir mulighet for å finne:
  - Hva den fysiske adressen er, gitt den logiske adressen
  - Informasjon om innholdet på en gitt fysisk adresse tilsvarer den logiske (for å finne ut om man må lese inn en ny side fra disk eller ikke). Dette angis med et valid-bit som ved oppstart er ’0’
- Sidetabellen ligger i en egen enhet kalt MMU (Memory Management Unit) og er som regel bygget opp av statisk RAM og må ha en linje for hver side som finnes.

Hva skjer ved page-fault

Hvis riktig side ikke finnes i det fysiske minnet skjer følgende:

1. Programmet som har bedt om å få lese fra en logisk adresse som ikke finnes RAM gir fra seg kontrollen til operativsystemet.
2. Operativsystemet leser en intern datastruktur for å finne ut hvor den aktuelle siden ligger på harddisken (hvilken side den skal lese finnes i sidetabellen).
3. Hvis det ikke er ledig plass, må en fysisk side kastes ut fra RAM (eventuelt skrives tilbake til disk) på tilsvarende måte som ved cache
4. Sidetabellen oppdateres slik at den peker til riktig fysisk side.
5. Det fysiske minnet oppdateres med riktig innhold fra harddisken.
6. Kontrollen gis tilbake fra operativsystemet til programmet som gav det fra seg slik at eksekveringen kan fortsette.

Hvordan velde sider som kan overskrives

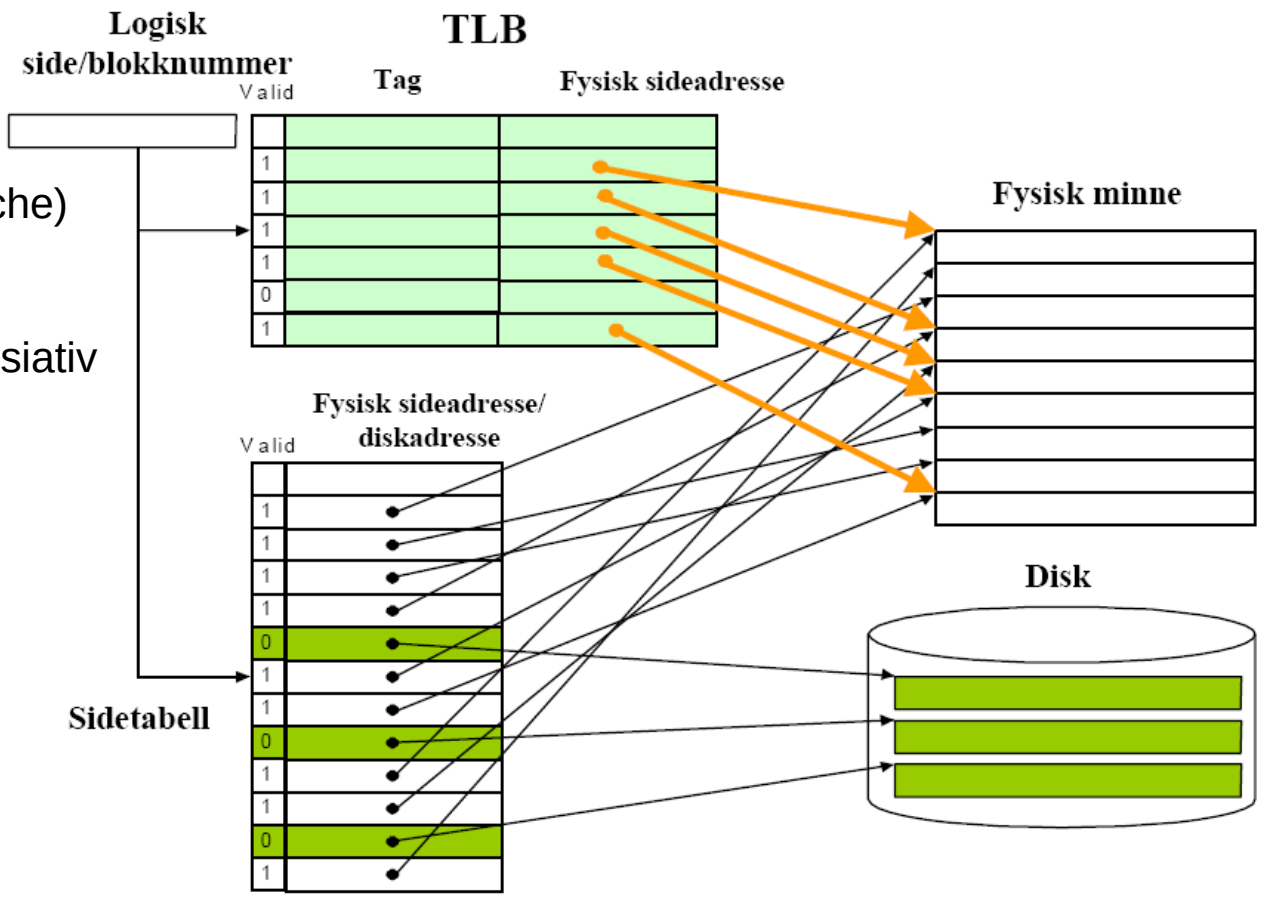
- Mest vanlig: Least Recently Used (LRU), fordi sjansen er størst for at denne ikke lenger er i bruk
- LRU implmenteres ved et eget bit som settes når blokken aksesseres, og som nullstilles ved jevne mellomrom for at sider ikke skal ligge evig.
- Operativsystemet inneholder en tabell over sidene sortert etter når LRU-bitet sist ble nullstilt.
- Listen over sider som er kandidater for å bli kastet ut kan også sorteres etter FIFO-prinsippet, dvs etter hvor lenge siden de ble hentet inn fra disk, uavhengig av når de sist ble aksessert.
- Hvis en side kun er lest, kan siden bare overskrives når den skal erstattes.
- Hvis den er modifisert, må man først skrive siden som skal erstattes tilbake til disk før den nye kan hentes inn.

Write-back eller Write-through?

- 'Write-through': Brukes sjelden fordi skriving til harddisk er meget langsomt og vil gir dårlig ytelse (hver skriving til RAM vil også medføre skriving til harddisk)
- Mest vanlig er 'write-back' som oppdaterer siden på harddisken kun ved en 'page fault'
- For raskt å finne ut om en side er skrevet til eller ikke, brukes et eget bit (dirty bit) i sidetabellen. Settes til '1' hvis det er skrevet til siden, '0' ellers.
- Siden sidetabellen lagres i vanlig RAM, medfører oppslag i RAM to oppslag: Først i sidetabellen, og deretter i det fysiske minnet (ser bort fra eventuell page fault).
- For å øke hastigheten bruker man en egen cache som innehlder de mest brukte sidene fra sidetabellen. Denne cachen kalles ofte for Translation Lookaside Buffer (TLB).

Eksempel på vituelt minne med TLB

- TLB må også inneholde 'dirty bit' for å indikere at det må gjøres write-back. (TLB er som vanlig cache)
- TLB er enten fullt assosiativ (ved små TLB) eller setassosiativ.



Input - Output (I/O)

- Med I/O menes de enheter og mekanismer som gjør det mulig å transportere data inn og ut av en data-maskin, en CPU osv.
- I/O er spesialisert og skreddersydd til ulike anvendelses-områder.
- I/O-enheter klassifiseres gjerne ut fra:
  - Type (input, output eller begge deler)
  - Datarate
  - Byte eller blokk-orientert
  - Responstid
- Sentralt i alle systemer med I/O er en eller flere (delte) busser, som kan være enten synkrone eller asynkrone.

Input - Output (I/O)

- En datamaskin kommuniserer gjennom mange ulike enheter:
  - Harddisk
  - CD-ROM/DVD
  - Hurtigminne
  - Mus
  - Tastatur
  - Skjerm
  - Nettverk
- Deler kommunikasjonen inn i to hovedgrupper:
  - Kommunikasjon mellom enheter internt i maskinen og mellom en datamaskin og direkte tilkoblet utstyr.
  - Kommunikasjon mellom ulike datamaskiner som er knyttet sammen i nettverk.

Input - Output (I/O)

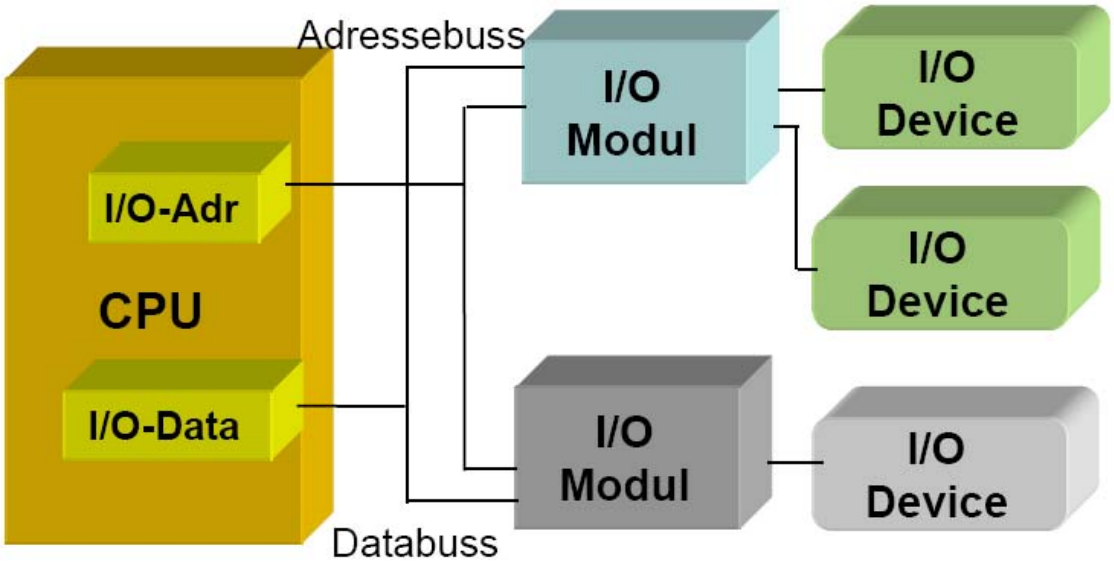
- Ytelsen til I/O- systemer avhenger av flere faktorer:
  - Prosessoren
  - Hukommelseshierarkiet
  - Bussen(e) som kobler sammen maskinen
  - Kontrollenheter for I/O og enhetene som er tilknyttet bussen.
  - Hastigheten til operativsystemet
  - Programvarens bruk av I/O
- De to viktigste parametrene for ytelse til I/O er:
  - **Throughput**: Båndbredde eller gjennomstrømning av data per tidsenhet.
  - **Responstid**: Forsinkelse fra start til svar.
- Internt er det som regel flere uavhengig busser som er spesialiserte, f.eks buss mellom CPU og RAM, mellom CPU og cache, system-buss.

Input - Output (I/O)

- Bussen er ofte en flaskehals i systemet, fordi mange enheter konkurrerer om å få bruke den og man må derfor ha kjøreregler
- Kjørereglene kalles **protokoller** og er tilpasset ulike behov og bruksområder
- Eksempler på protokoller er
  - PCI
  - TCP/IP
  - Ethernet
  - USB/Firewire
  - ATM
  - Bluetooth
- Kjennskap til protokoller og datakommunikasjon er like viktig som programmering

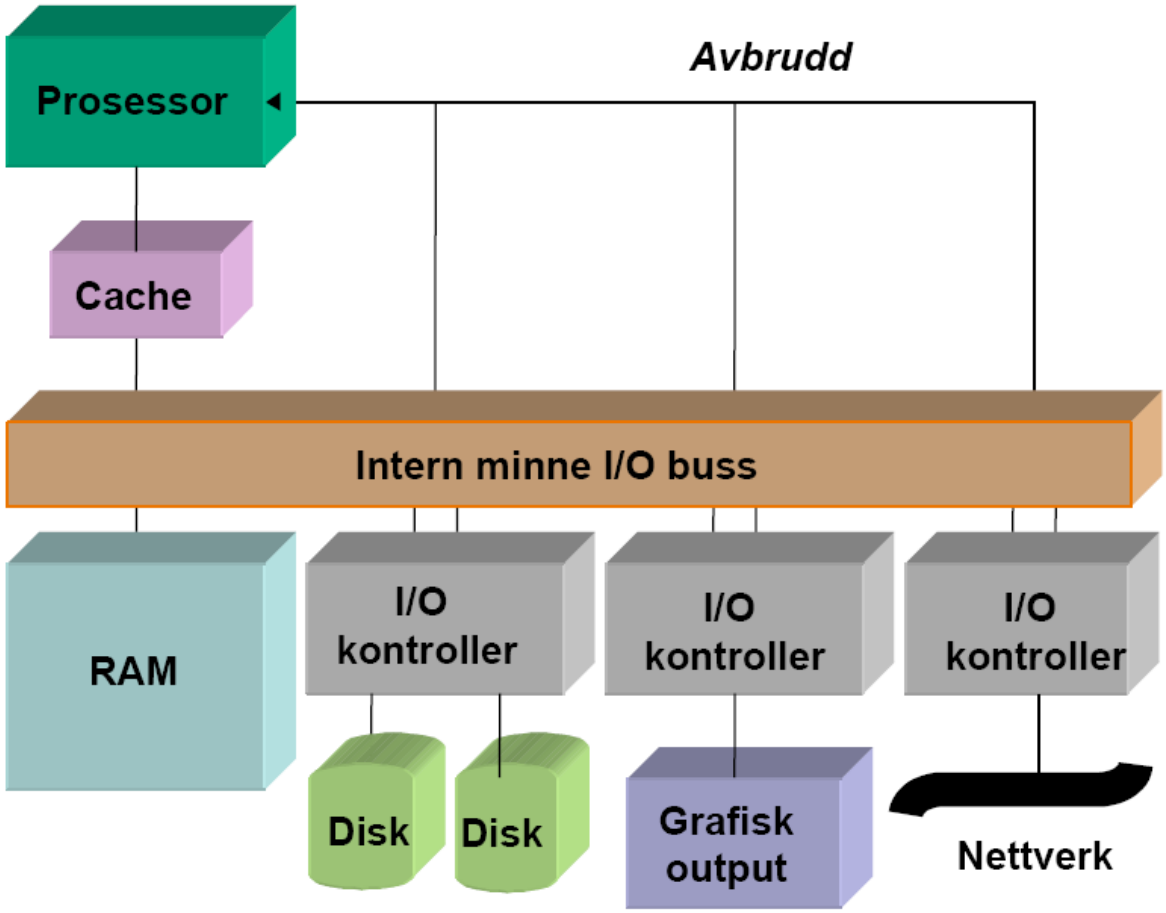
Programmerbar I/O

- Enkleste form for I/O og brukes i systemer uten store krav til hastighet eller ytelse.
- CPU'en kommuniserer med omverdenen (enten input eller output) via to registre I/O-DataReg og I/O-AdrReg.
- Det første inneholder data som skal skrives eller leses, mens I/O-AdrReg inneholder adressen til enheten som enten sender eller mottar data:



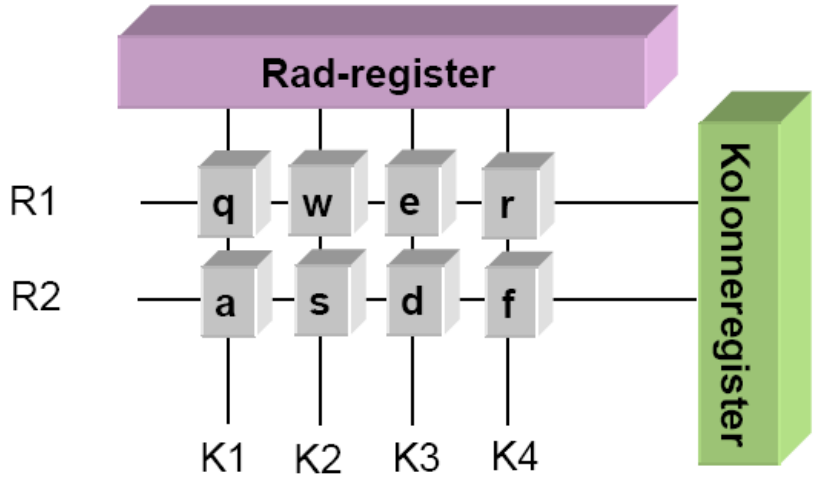
Intern kommunikasjon

- Mellom bussen og de ulike enhetene som kommuniserer over bussen sitter en I/O-kontroller som tar seg av bl.a. protokollhåndtering.
- De fleste enheter kan bruke avbrudd for å signalisere til prosessoren at noe har skjedd som krever spesiell behandling av prosessoren.



Eksterne hendelser (1)

- I noen tilfeller krever en ekstern hendelse eller begivenhet at prosessoren foretar seg noe bestemt (dvs eksekverer en bestemt subrutine eller funksjon).
- For at prosessoren skal finne ut at noe har skjedd kreves det signalering mellom den ytre enheten og prosessoren.
- Eksempel: Avlesning av tastetrykk på tastatur
- Når en tast trykkes ned, blir det kontakt mellom en rad og en kolonne. Trykkes tasten merket ‘e’ ned, blir det kontakt mellom R1 og K3. Dette registreres i Rad og Kolonneregisteret



Eksterne hendelser (2)

- Prosessoren kan lese innholdet av rad og kolonnergisteret for å finne ut hvilken tast som er trykket ned.
- Problem: Hvordan finne ut når en tast er trykket ned?
- Dette kan løses på to måter: Polling og avbrudd

Polling

- Prosessoren kan med jevne mellomrom avlese inn-holdet av Rad- og Kolonne-registrene (f.eks hvert 10. millisekund) og sjekke om det er en *endring* fra forrige gang.

- Fordel: Enkelt å implementere (gå i evig løkke og les av registrene og sjekk mot forrige verdi).

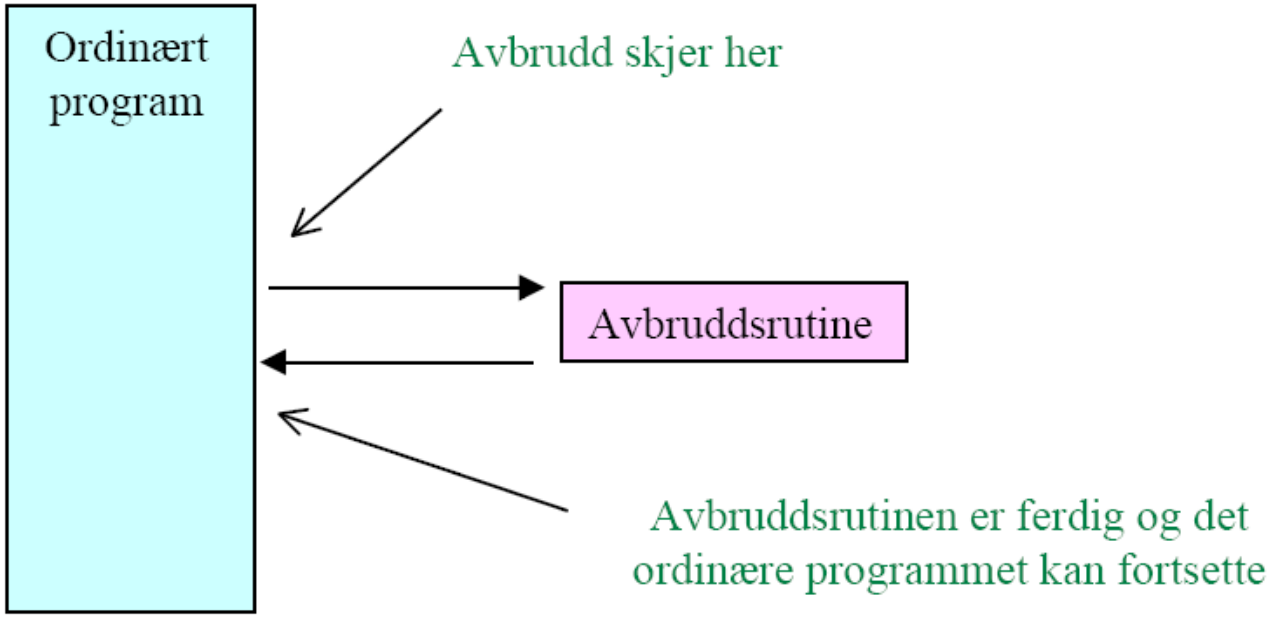
- Ulempe: Prosessoren får ikke gjort så mye annet enn å sjekke disse registrene hele tiden!

Avbrudd

- I stedet for å sjekke jevnlig, kan tastaturet selv si i fra at en tast er trykket ned. Prosessoren finner ut hvilken tast ved å sammenligne gammelt og nytt innhold i Rad- og Kolonneregistrene
- Fordel: Prosessoren kan løse andre oppgaver enn bare å sitte og vente på at en tast skal trykkes ned
- Ulempe: Det kreves mer av hardware; må ha egne signaler inn til prosessoren som kan brukes til f.eks. å si fra et en tast er trykket ned.
- Avbrudd er en generell mekanisme som finnes i alle maskiner og brukes til bl.a.
  - signalisere at en ekstern hendelse har skjedd
  - Markere avslutningen på en operasjon
  - Allokere CPU-tid (context switching)
  - Signalisere at en uventet eller ulovlig situasjon har oppstått (exception)

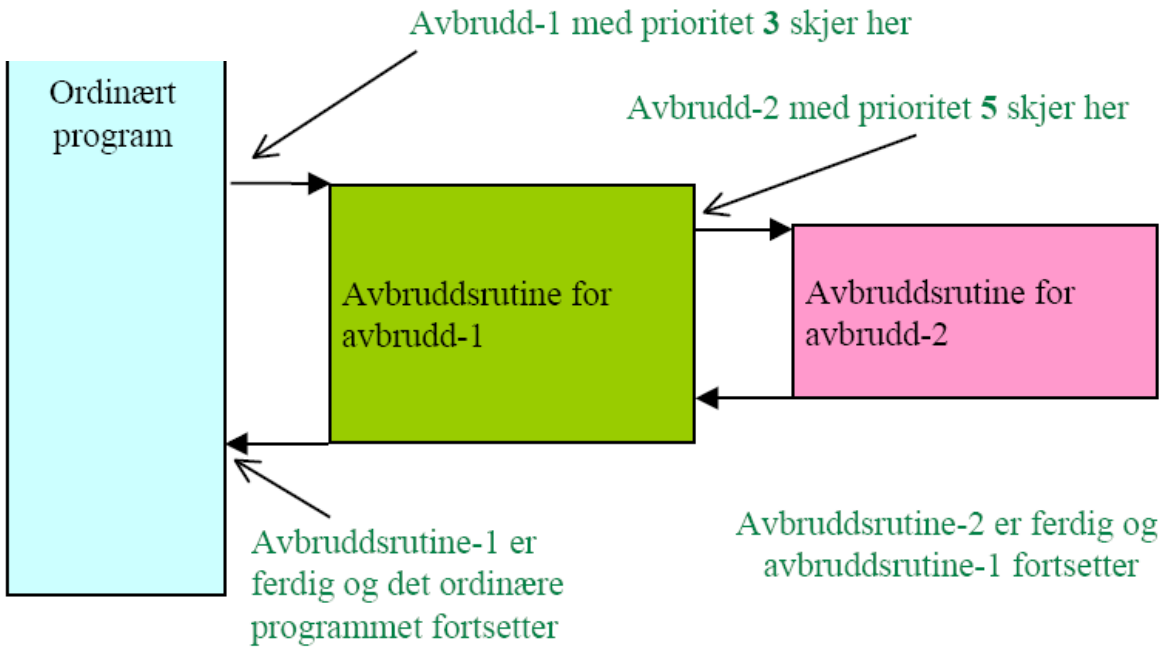
Avbrudd – Hva skjer?

- Prosessoren avslutter den instruksjonen den holder på med å eksekvere
- Alle registre som er i bruk må lagres unna
- Avhengig av hvilken kilde som genererte avbruddet vil det bli startet opp en avbruddsrutine som prosesserer avbruddet.
- Når avbruddsrutinen er ferdig, gjenopprettes registrene som ble lagret unna, og prosessoren fortsetter å eksekvere det programmet den kjørte før avbruddet skjedde.



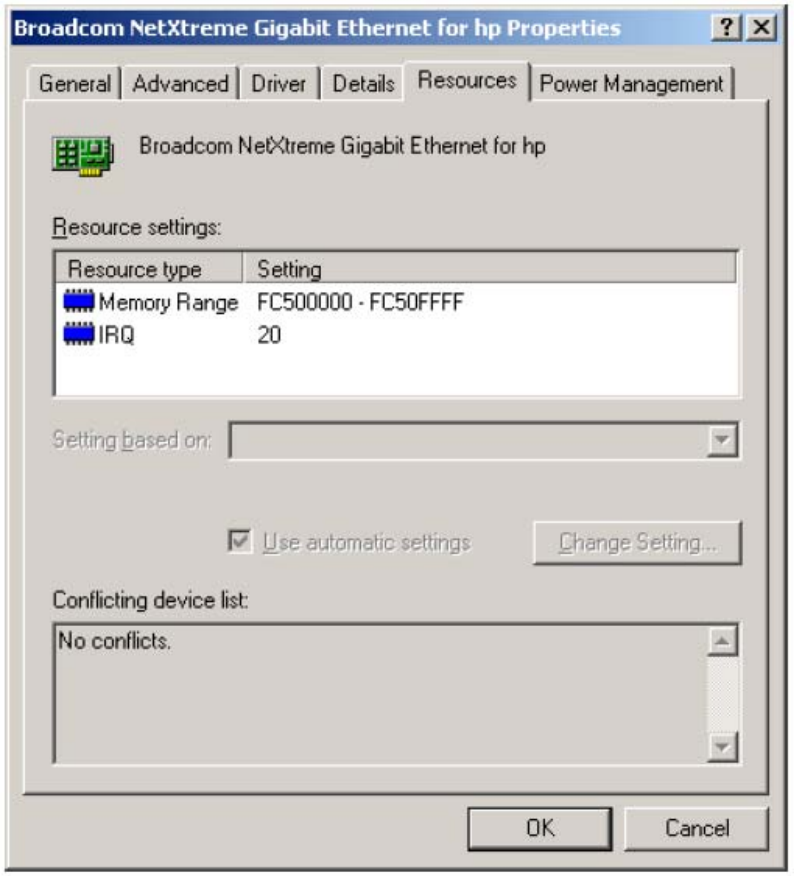
Avbrudd – Hva skjer?

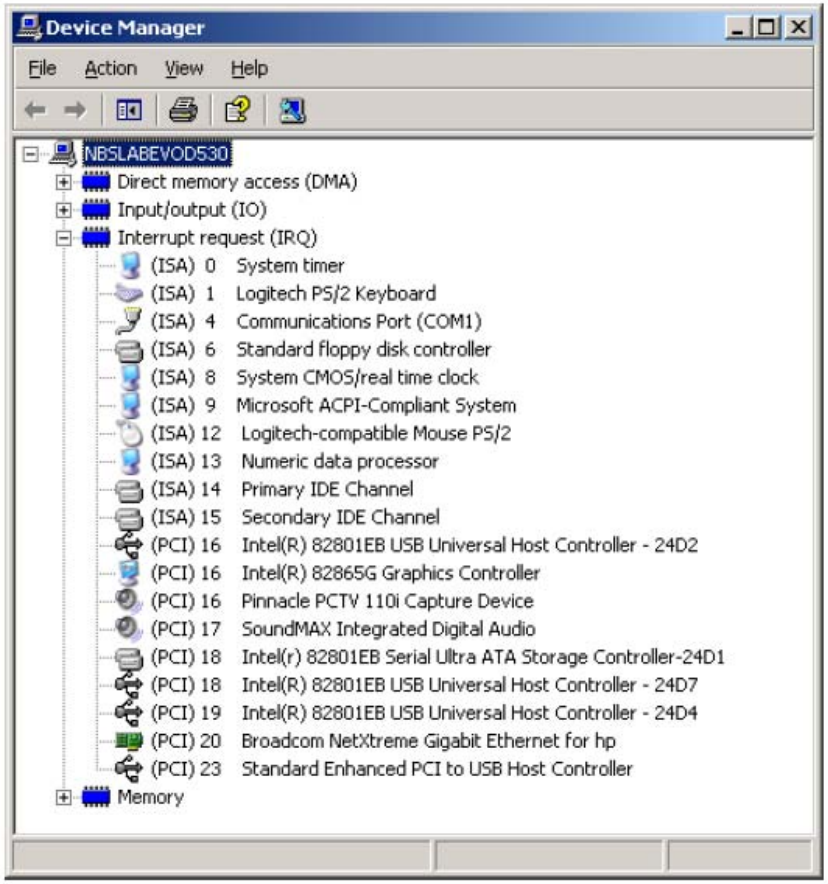
- Fordi hendelser og begivenheter kan ha varierende betydning og viktighet, tilbyr prosessorer flere avbruddsnivåer med ulik prioritet.
- Et avbrudd med høy prioritet kan avbryte behandlingen av (dvs. avbruddsrutinen til) et avbrudd av lavere prioritet.
- Hvis avbrudd fra to ulike kilder har samme prioritet behandles de ferdig i den rekkefølge de kom, og informasjon om andre avbrudd (med samme eller lavere prioritet) blir lagt i en kø



Avbrudd i Windows

- Interrupt Request (IRQ) på Windows XP. Lavere tall = høyere prioritet



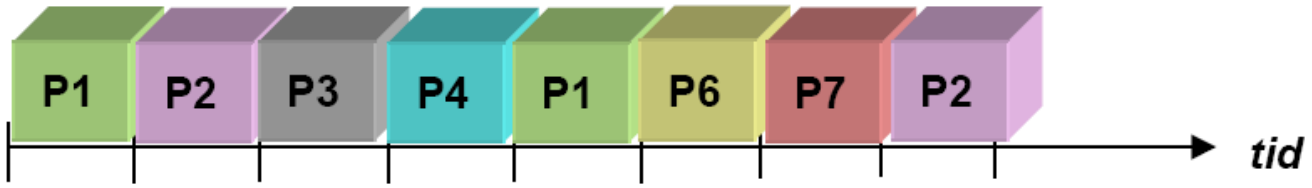


Bruk av avbrudd (1)

- Signalisering av ekstern hendelse
  - I kontrollsystemer overvåker og styrer datamaskiner temperatur, trykk, strålingsnivå etc. Avbrudd kan brukes til å signalisere at et kritisk nivå eller en grense er nådd som krever spesiell handling, f.eks iverksetting av alarm
  - Prosessering av tastetrykk er også eksempel på slike hendelser som krever spesiell prosessering (f.eks Ctrl--C som betyr at et program skal avsluttes)
- Synkroniserings-/avslutningssignal
  - Avbrudd kan brukes av f.eks printere for å be en prosessor om å få sendt over mer data hvis et internt buffer er tomt.
  - Avbrudd kan generelt brukes til flytkontroll for å signalisere start/stopp av transaksjoner, overføringer osv. (“send mer data”, “stopp å sende data”…)

Bruk av avbrudd (2)

- Signalisering av unormal hendelse
  - Dette er en viktig mekanisme og brukes både av hardware og software for å signalisere at en gitt unormal hendelse har inntruffet.
  - Hvis avbruddet genereres av software kalles det “execption” (unntak) og brukes enten for å gi beskjed om en ulovlig operasjon som (f.eks divisjon med null), eller for å angi at en instruksjon må behandles av en ekstern hardware-enhet (f.eks egen flyttalls-prosessor)
- Tidsdeling i operativsystemer
  - Operativsystemer simulerer parallellitet ved å dele prosessor-tiden opp i små tidsintervall, og lar hver prosess får bruke prosessoren i minst ett tidsintervall.



Implementasjon av tidsdeling

- Kan implementeres slik:
  - Med faste intervaller genereres et avbrudd som signaliserer at operativsystemet skal suspendere prosessens som kjører i øyeblikket. Registre, peker til stakkområdet og statusregistre som prosessen brukte blir lagret
  - Operativsystemet tar over kontrollen og plukker ut hvilken prosess som skal få kjøre (skedulering).
  - Registre, stakkområde osv til prosessen som skal startes, lastes inn i CPU'en av operativsystemet.
  - Operativsystemet gir kontrollen til neste prosess som kan fortsette å eksekveres
- På samme måte som avbrudd fra ulike kilder kan ha ulik prioritet, vil også prosesser ha ulik prioitet. Typisk vil operativsystemet ha høyere prioritet og få tilgang til CPU'en før et brukerprogram

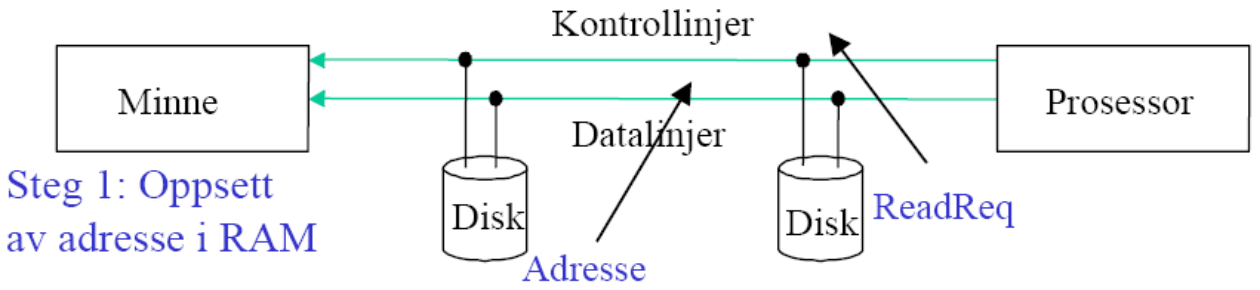
Direct Memory Access

- Ved DMA flyttes data mellom ulike minne-enheter uten at prosessoren er involvert i annet enn start og stopp i overføringen.
- Avbrudd brukes til å gi beskjed til prosessoren når overføringen er ferdig
- Eksempel :Overføring av data fra RAM til disk (se neste foil)

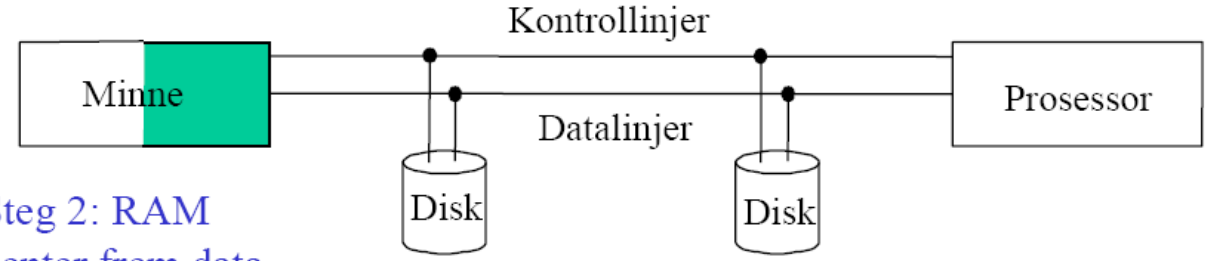
Eksempel :

Overføring av data

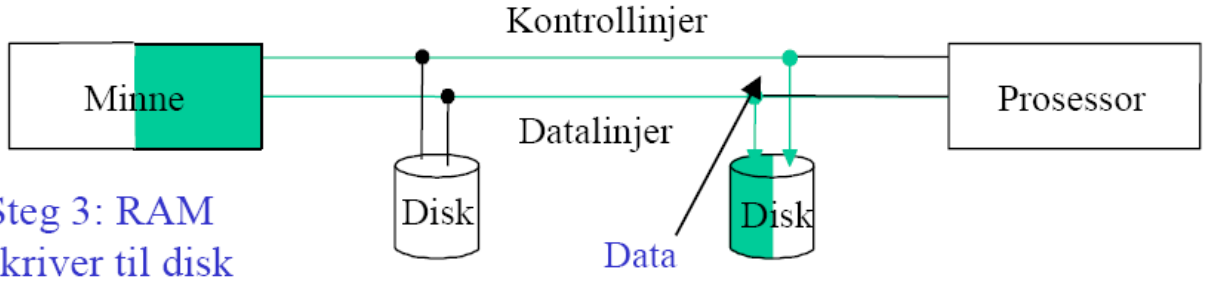
fra RAM til disk



Steg 1: Oppsett av adresse i RAM



Steg 2: RAM henter frem data



Steg 3: RAM skriver til disk

2008