UNIVERSITETET
I OSLO

Dagens temaer

- Virtuell hukommelse (kapittel 9.9 i læreboken)
- Input-Output

2005

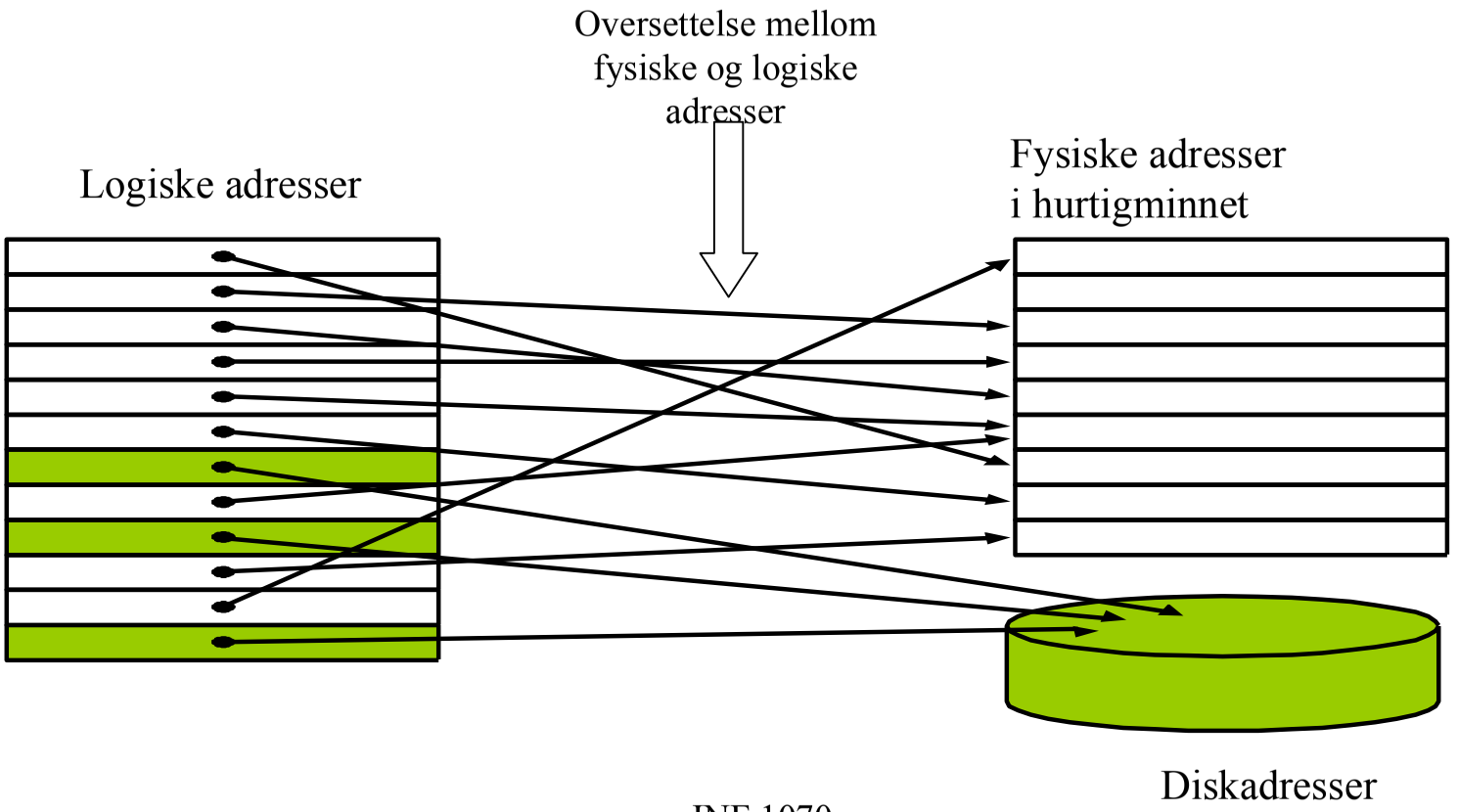
UNIVERSITETET
I OSLO

Virtuell hukommelse

- Ofte trenger et program/prosess mer RAM enn det som er tilgjengelig fysisk i maskinen
- Et program deler RAM med andre programmer og prosesser, bl.a:
  - Operativsystemet
  - Driver-rutiner
  - 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 både i RAM eller på harddisken brukes begrepet *logisk adresse* istedenfor *fysisk adresse* om de lokasjoner som et program aksesserer.

Virtuell hukommelse (forts.)

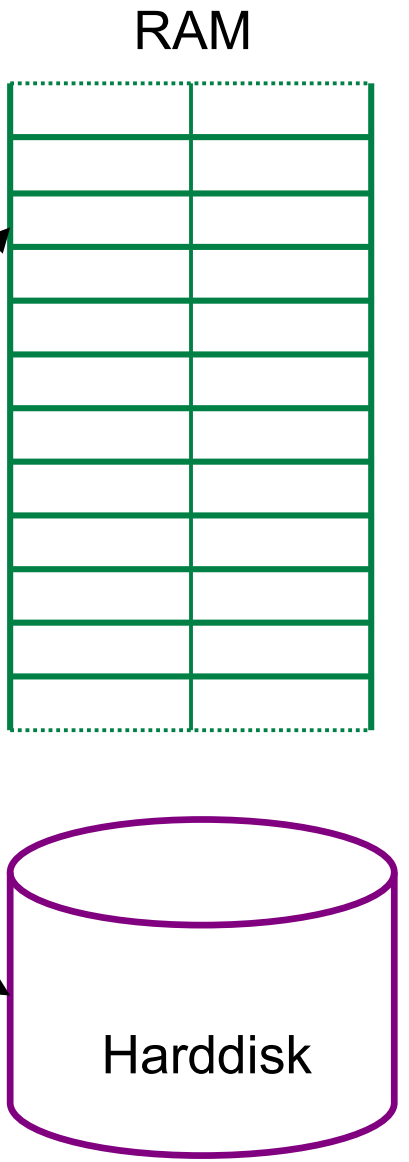
- Maskinen må ha en mekanisme som gir en kobling mellom logiske (virtuelle) adresser og fysiske adresser



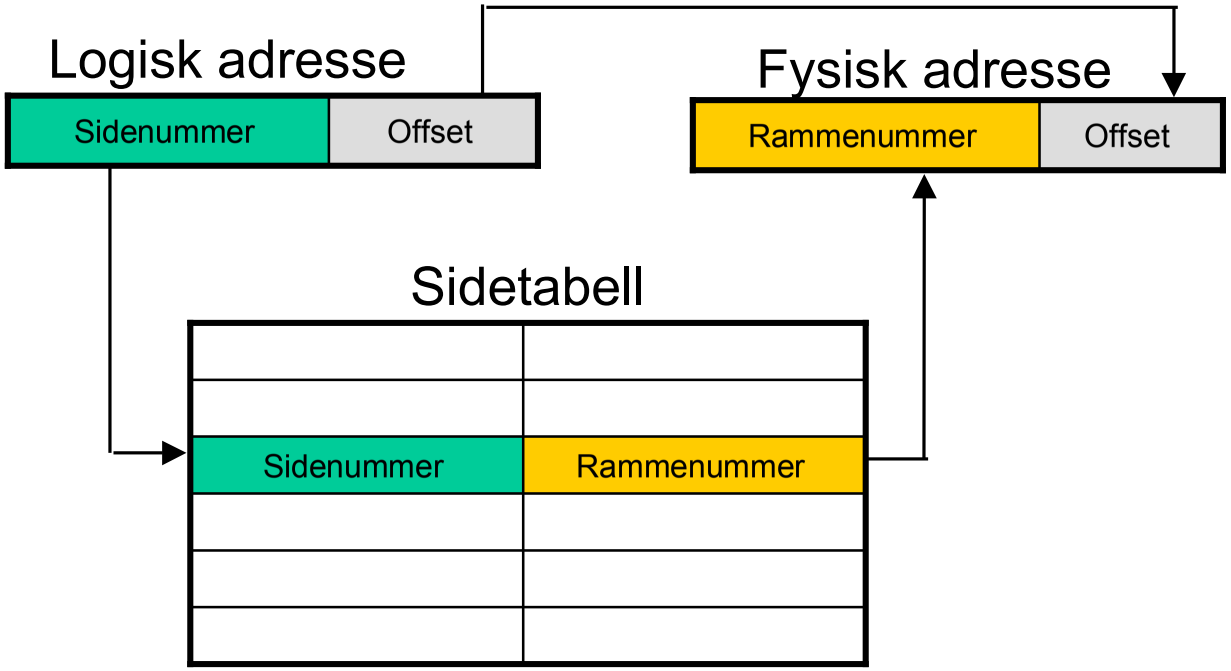
Sidetabell

- 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).

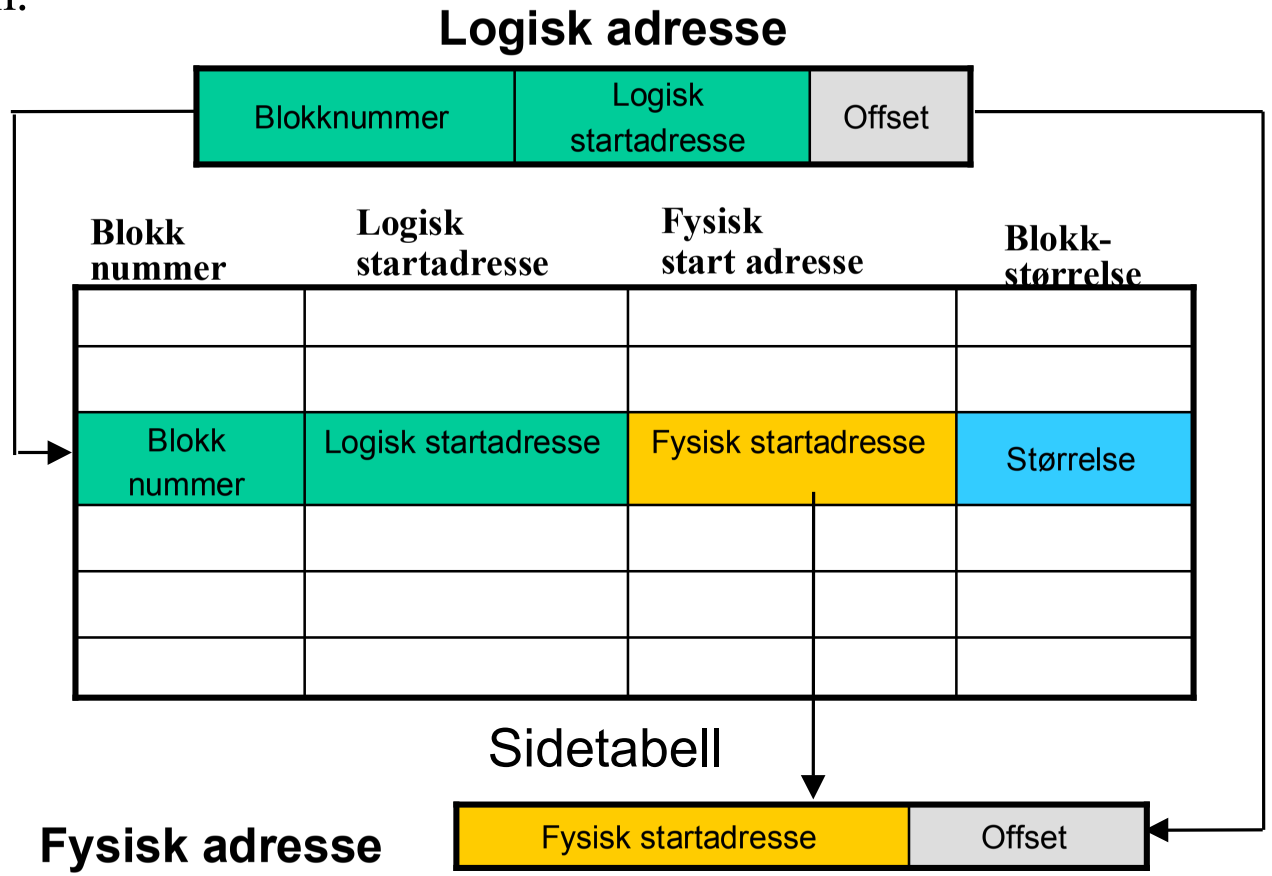
Logisk adresse	Fysisk adresse
0x00000	0x1000
0x02000	0x5000
0x04000	hdsd-0
0x06000	hdsd-2
0x08000	hdsd-6
0x10000	0x7000
0x12000	hdsd-3
0x14000	hdsd-4
.....	.....



- 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.



UNIVERSITETET
I OSLO

Segmentation versus paging

• Begge støtter virtuell hukommelse, dvs isolerer logisk og fysisk adresserom, slik at den fysisk hukommelse kan realiseres på ulike måter uten at dette er synlig for prosessoren.

• Med segmentation er det lettere for operativsystemet å holde ulike prosesser sine 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.

UNIVERSITETET
I OSLO

Plassering 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 hurtigminnet) 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'.

UNIVERSITETET
I OSLO

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 gi mulighet for å finne
  - Hva den fysiske adressen er gitt den logiske
  - 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 kan angis med ett enkelt *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.

UNIVERSITETET
I OSLO

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.

UNIVERSITETET
I OSLO

Hvordan velge ut 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 (f.eks ren programkode) kan siden bare overskrives når den skal erstattes.
- Hvis den er modifisert, må man først skrive den siden som skal erstattes tilbake til disken før den nye kan hentes inn.

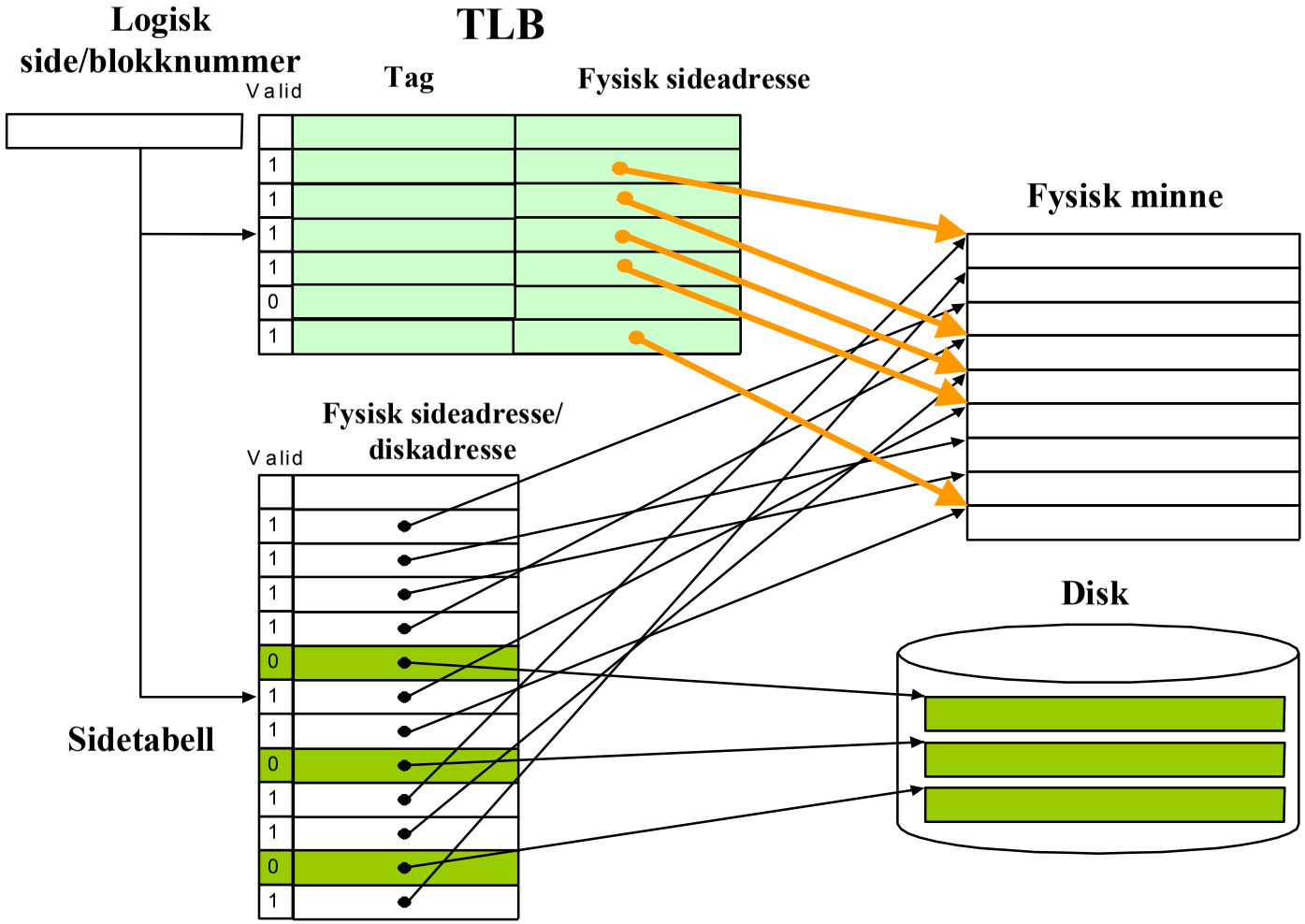
UNIVERSITETET
I OSLO

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 skrivet 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 virtuelt 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 set-assosiativ.



UNIVERSITETET
I OSLO

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 ifra
  - 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.

UNIVERSITETET
I OSLO

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.

UNIVERSITETET
I OSLO

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.

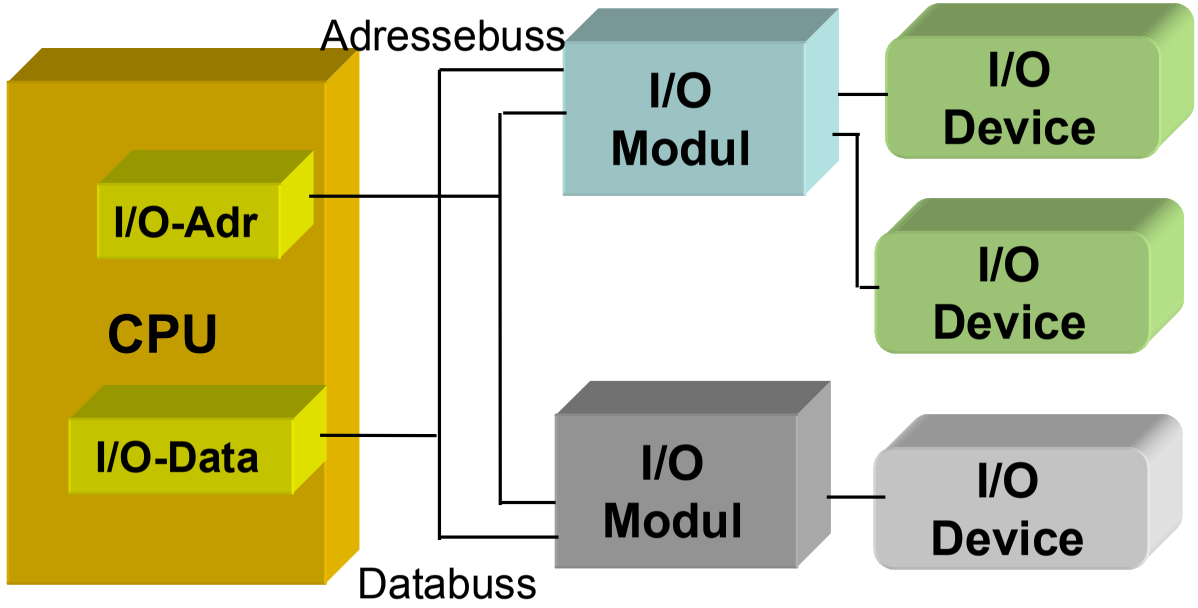
UNIVERSITETET
I OSLO

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
- Disse kjørereglene kalles for ***protokoller*** og er tilpasset forskjellige 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

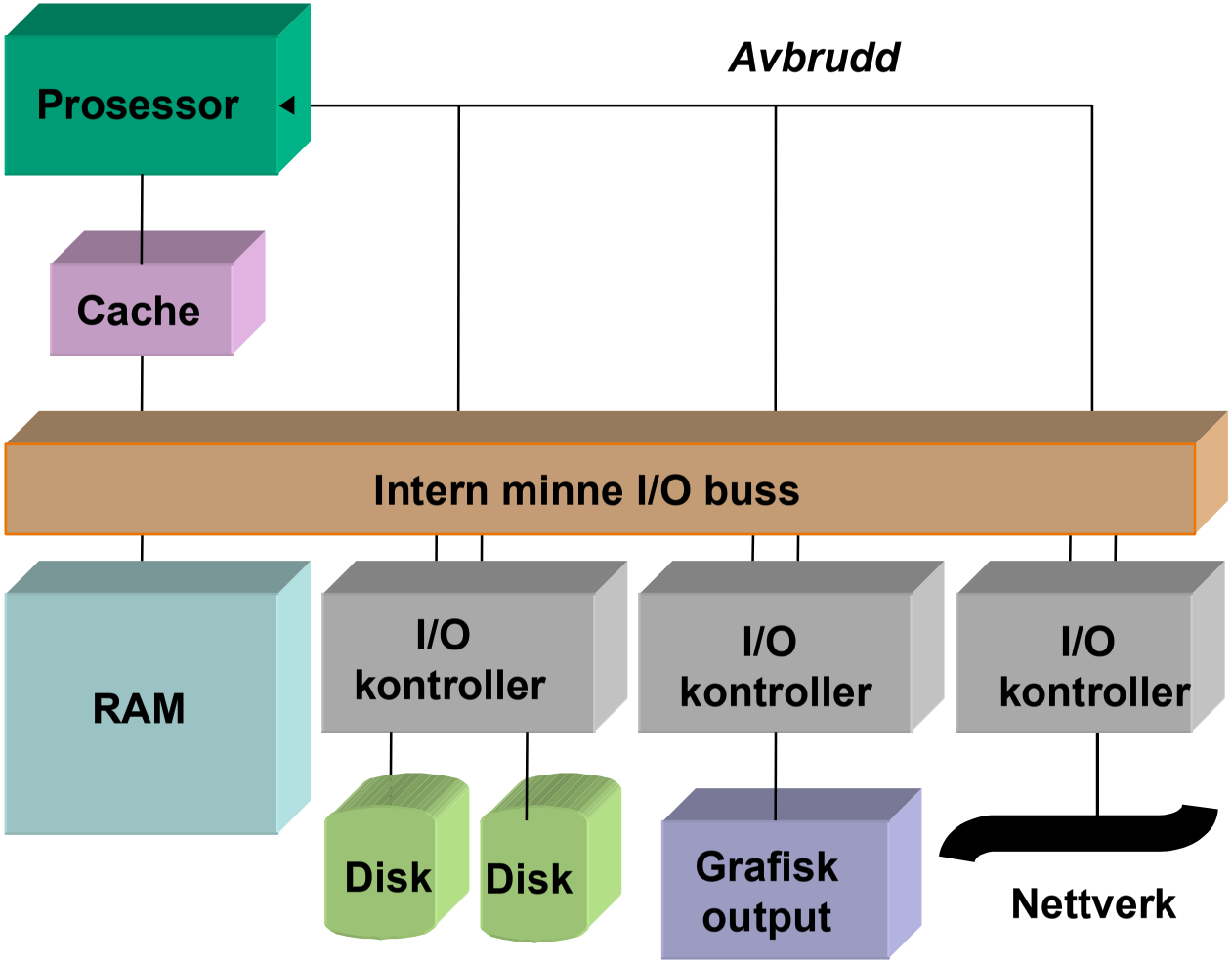
Programmerbar I/O

- Enkleste formen 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

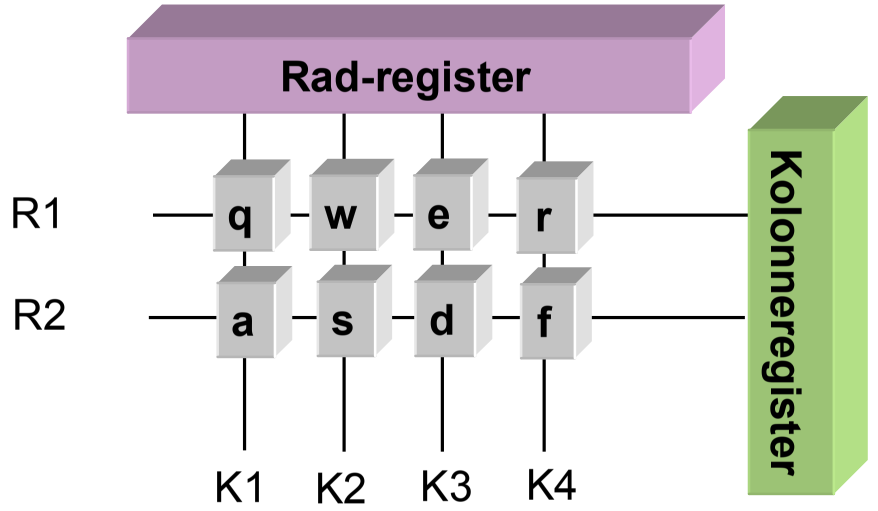
- Mellom bussen og de ulike enhetene som kommuniserer over bussen sitter en ***I/O-kontroller*** som tar seg av bl.a protokollhåndtering.
- De fleste enheter kan bruke ***avbrudd*** for å signalisere til prosessoren at noe har skjedd som krever spesiell behandling av prosessoren.



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 krever 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



UNIVERSITETET
I OSLO

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 verd).

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

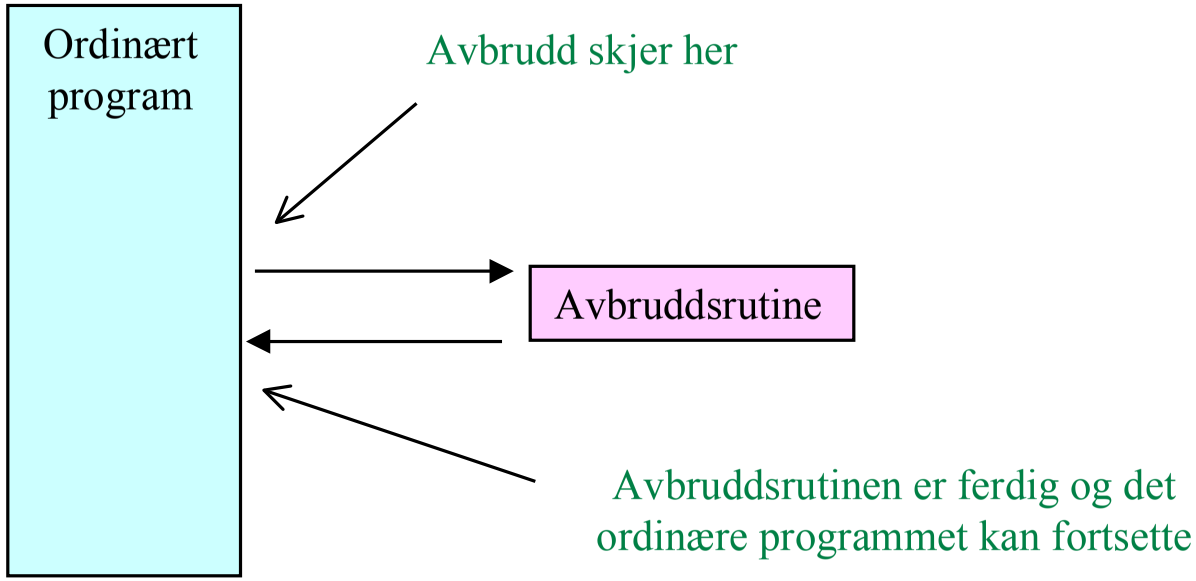
UNIVERSITETET
I OSLO

Avbrudd (1)

- Isteden 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 Kolonne-registrene
- Fordel: Prosessoren kan løse andre oppgaver enn kun 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 å si fra et en tast er trykket ned.
- Avbrudd er en generell mekanisme som finnes i alle maskiner og brukes til bla
  - 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)

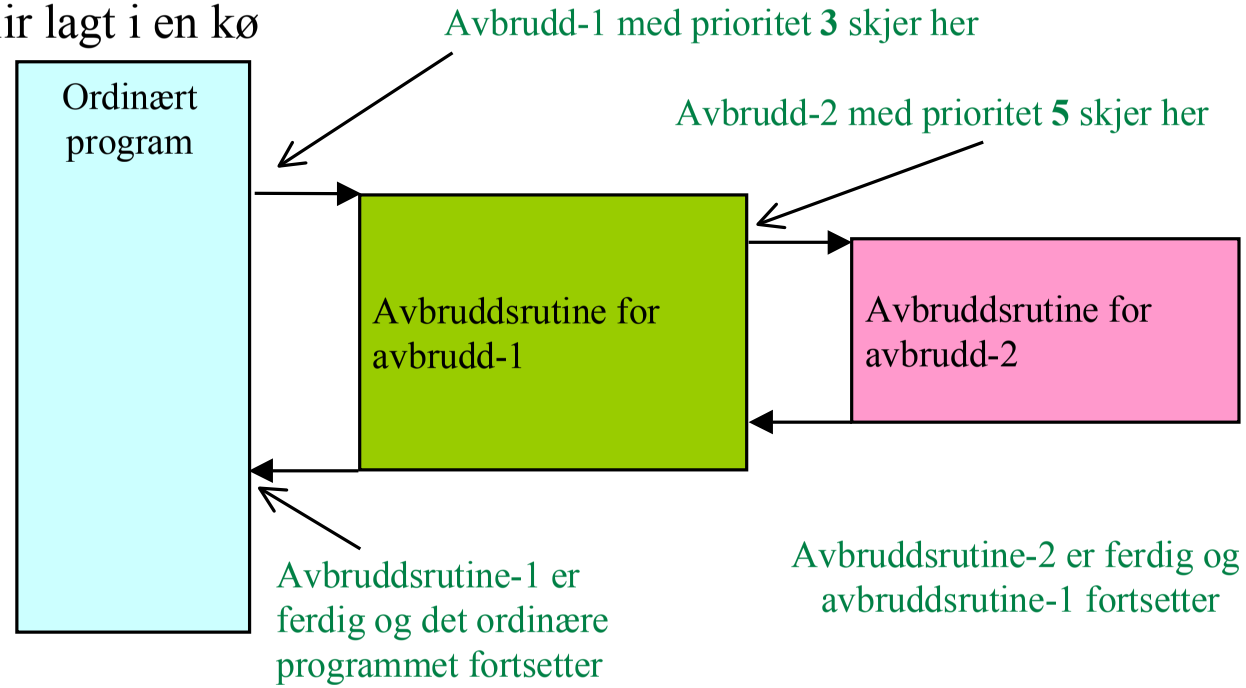
Avbrudd (2)

- 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.



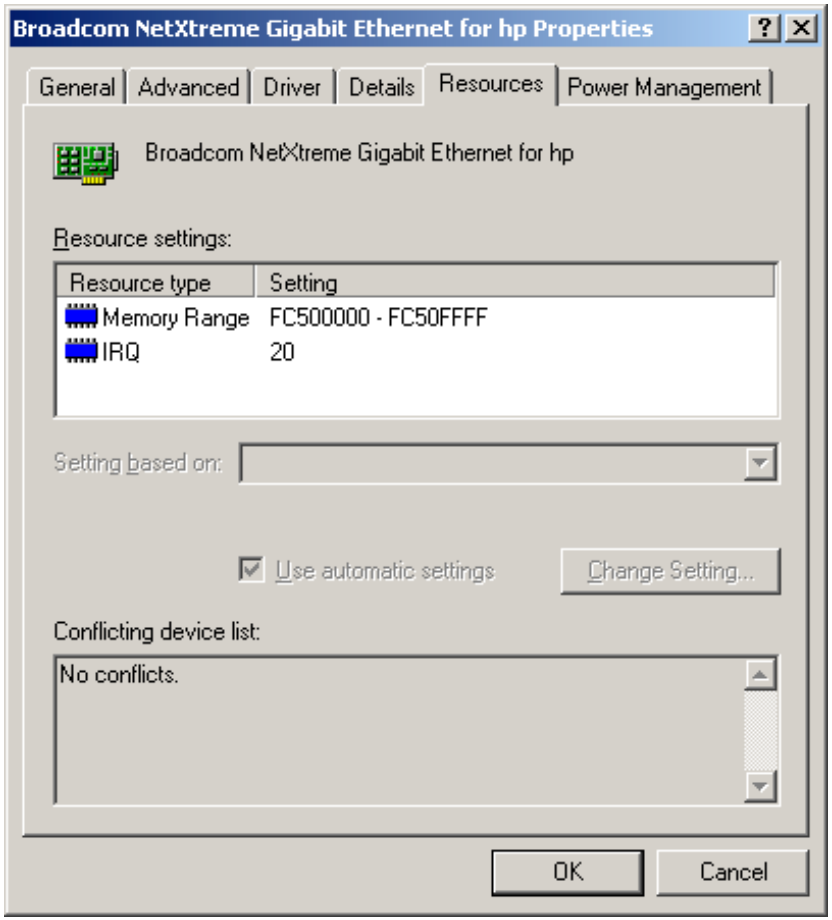
Avbrudd (3)

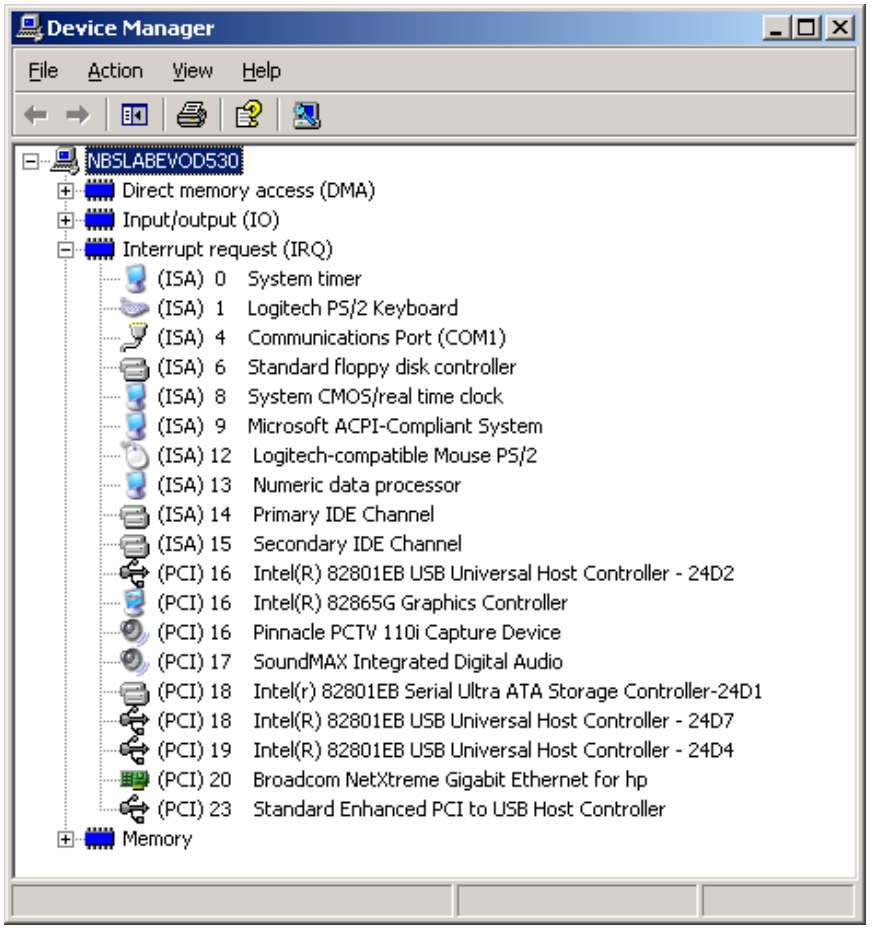
- Fordi hendelser og begivenheter kan ha varierende betydning og viktighet, tilbyr prosessorer flere *avbruddsnivåer* med ulik *prioritet.*
- Et avbrudd med høy prioritet kan avbryte behandlingen av (dvs avbruddsrutinen til) et avbrudd av lavere prioritet.
- Hvis avbrudd fra to ulike kilder har samme 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 (4)

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





UNIVERSITETET
I OSLO

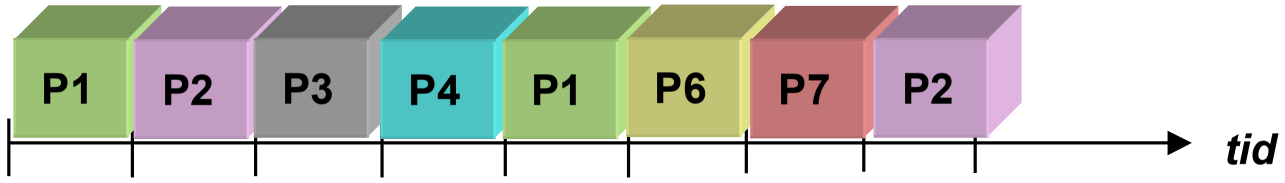
Bruk av avbrudd (1)

- Signalisering av ekstern hendelse
  - I kontrollsystemer overvåker og styrer en datamaskin temperatur, trykk, strålingsnivå etc. Avbrudd kan da 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/avslutnings-signal
  - 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 en 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



UNIVERSITETET
I OSLO

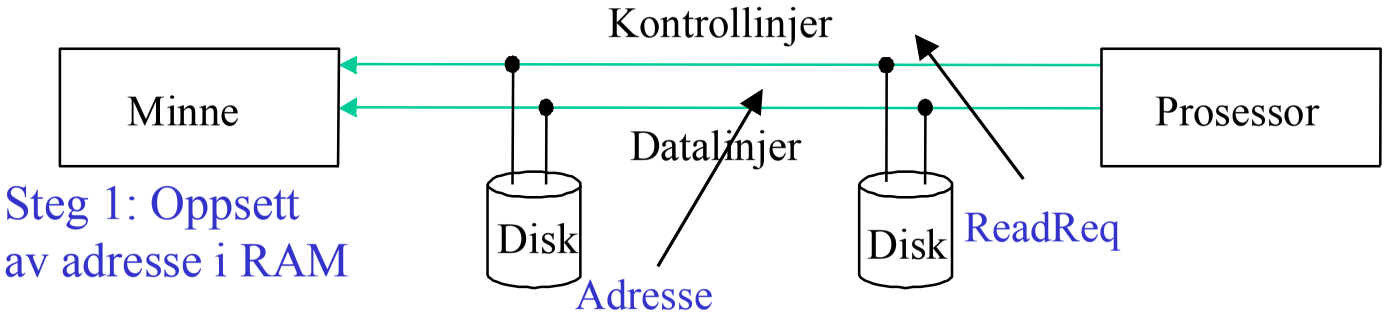
Implementasjon av tidsdeling

- Kan implementeres slik:
  - Med faste intervaller genereres et avbrudd som signaliserer at operativsystemet skal *suspendere* prosessens som kjører i øyeblikket. Registre, peker til stakk-området og statusregistre som prosessen brukte blir lagret unna på et sikkert sted
  - Operativsystemet tar over kontrollen og plukker ut hvilken prosess som skal få kjøre (skedulering).
  - Registre, stakkområde etc til prosessen som nå skal kjøres lastes inn i CPU'en av operativsystemet.
  - Operativsystemet gir fra seg 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

UNIVERSITETET
I OSLO

Direct Memory Access (DMA)

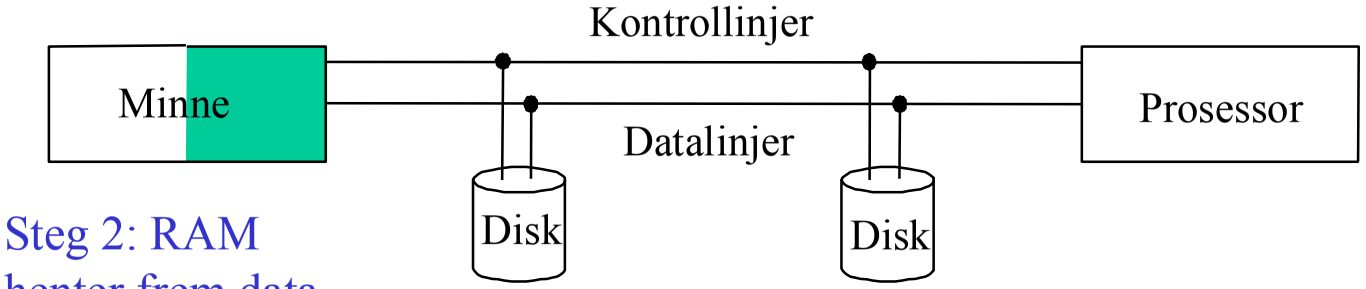
- Ved DMA flyttes data mellom ulike minne-enheter uten at prosessoren er involvert i annet enn start og stopp i overføringen.
- Avbrudd brukes til å gi beskjed til prosessoren når overføringen er ferdig
- Eksempel :Overføring av data fra RAM til disk (se neste foil)

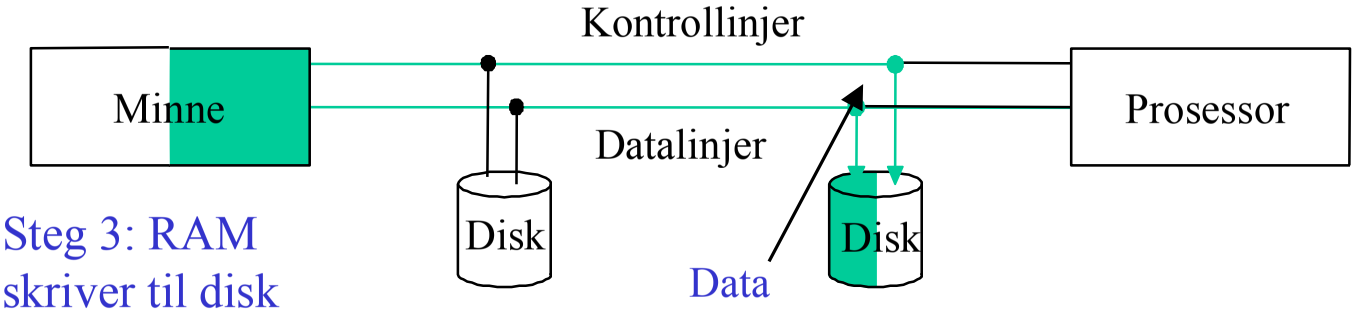


Eksempel :

Overføring av data

fra RAM til disk





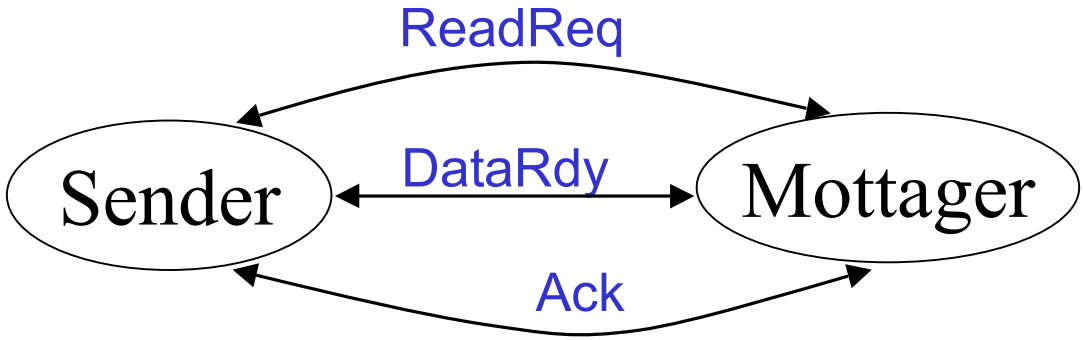
UNIVERSITETET
I OSLO

Synkrone og asynkrone busser

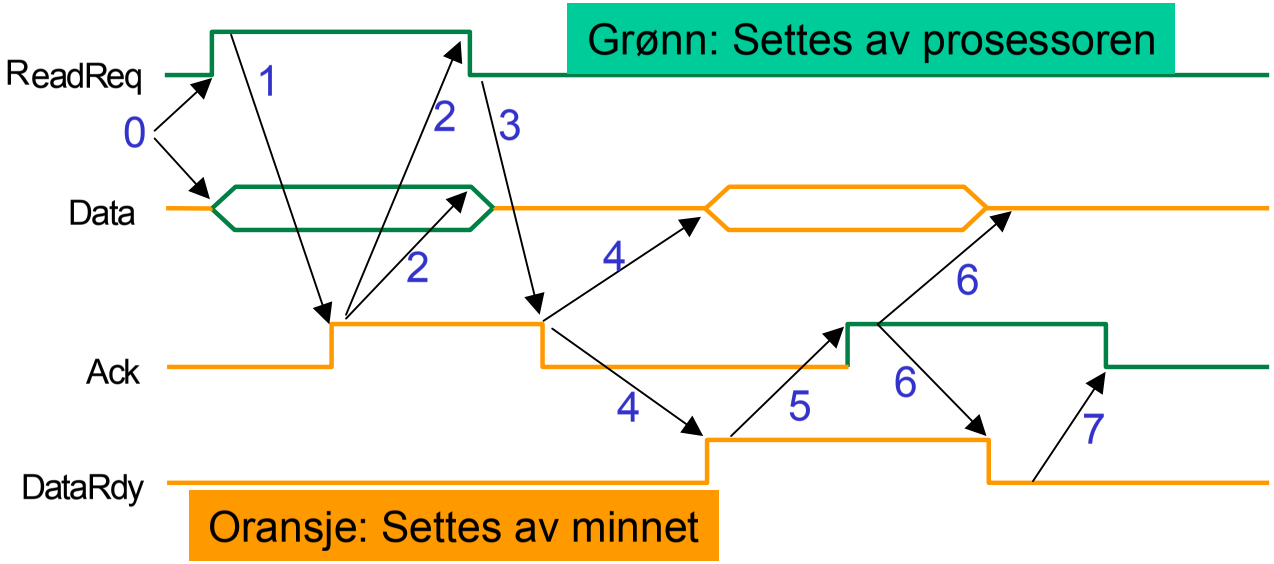
- Busser er enten ***synkrone*** eller ***asynkrone***:
  - **Synkron**: Endringer på bussen skjer etter en fast protokoll, relativt til et klokkesignal som er en del av kontrollinjene.
  - **Asynkron**: Intet klokkesignal blant kontrollinjene. Overføring av data skjer etter regler avtalt mellom enhetene ('handshaking') for hver gang det skal overføres en enhet data (bit, byte osv)
- Egenskaper og bruk av synkrone busser:
  - Som regel meget rask fordi det mindre protokoll-overhead
  - Knytter sammen enheter med samme klokkehastighet
  - Enhetene må ligge nær hverandre fysisk fordi lange avstanderkan gi avvik i klokkesignalet
- Egenskaper og bruk av asynkrone busser:
  - Knytter sammen enheter med ulik hastigheter
  - Gir færre begrensninger i busslengde fordi man ikke er avhengig av et felles klokkesignal
  - Mer komplisert protokoll for synkronisering av aktivitene og derfor mindre nyttetrafikk

Handshaking i asynkrone busser (1)

- Handshaking brukes for å koordinere overføring av av data mellom en sender og mottager(e).
- Gitt et enkelt system med tre kontrollinjer i tillegg til datalinjer:
  - 1) ReadReq: Brukes for å indikere en forespørsel om lesing fra minne. Adressen legges på datalinjene samtidig.
  - 2) DataRdy: Indikerer at data er klare på datalinjene.
  - 3) Ack : Brukes for å bekrefte at ReadReq eller DataRdy er mottatt fra den andre enheten.
- De tre kontrollinjene brukes for å utveksle informasjon om hvor langt de to enhetene har kommet.



Handshaking i asynkrone busser (2)



0) Prosessoren setter ReadReq='1', og legger adressen ut på datalinjene.

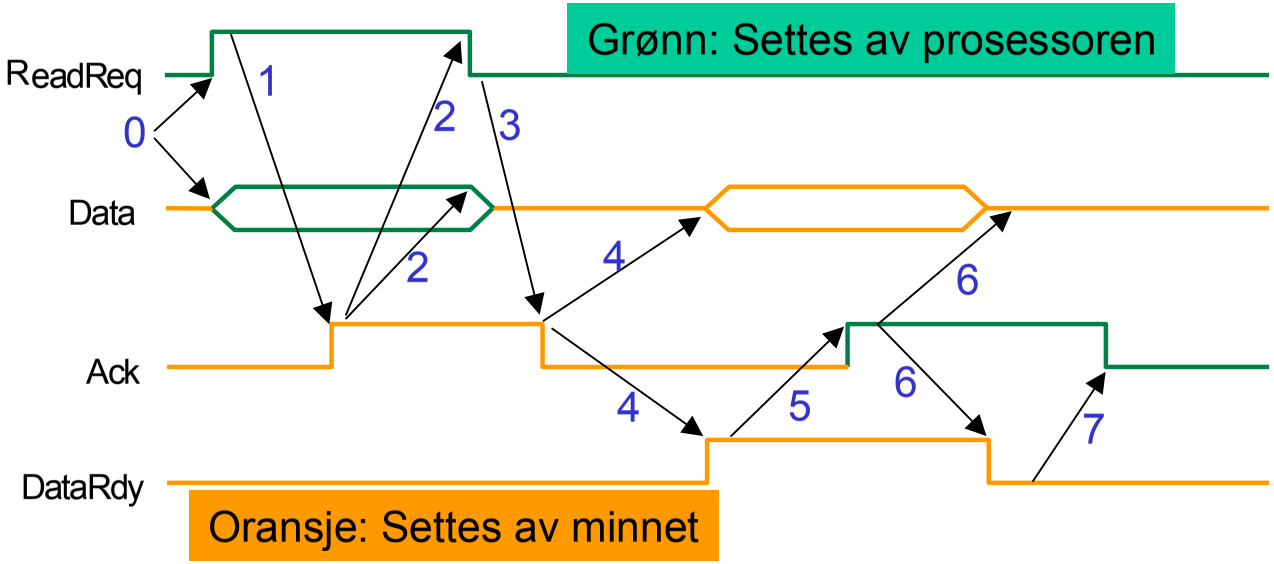
1) Minnet ser ReadReq = '1', leser adressen og setter Ack='1' for å indikere at adressen er lest

2) Prosessoren ser Ack='1', og setter ReadReq='0' og frigir datalinjene.

3) Minnet ser ReadReq='0' og setter Ack='0' for å bekrefte at ReadReq-signalet er mottatt.

4) Når minnet har data klart for overføring, plasseres data på datalinjene, minnet setter DataRdy='1'for å indikere at det er gyldige data på bussen.

Handshaking i asynkrone busser (3)



5) Prosessoren ser at DataRdy='1', leser data fra bussen, og indikerer at den har lest ferdig ved å sette Ack='1'

6) Minnet ser at Ack='1', setter DataRdy='0', og frigir datalinjene.

7) Prosessorne ser at DataRdy='0', og setter Ack='0' for å indikere at transmisjonen er ferdig.

Hvis det er mer data som skal overføres, gjentas punkt 0) til 7)