

UiO • **Institutt for informatikk**

Det matematisk-naturvitenskapelige fakultet

**IN1020**

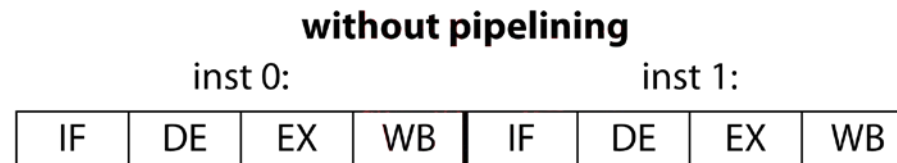
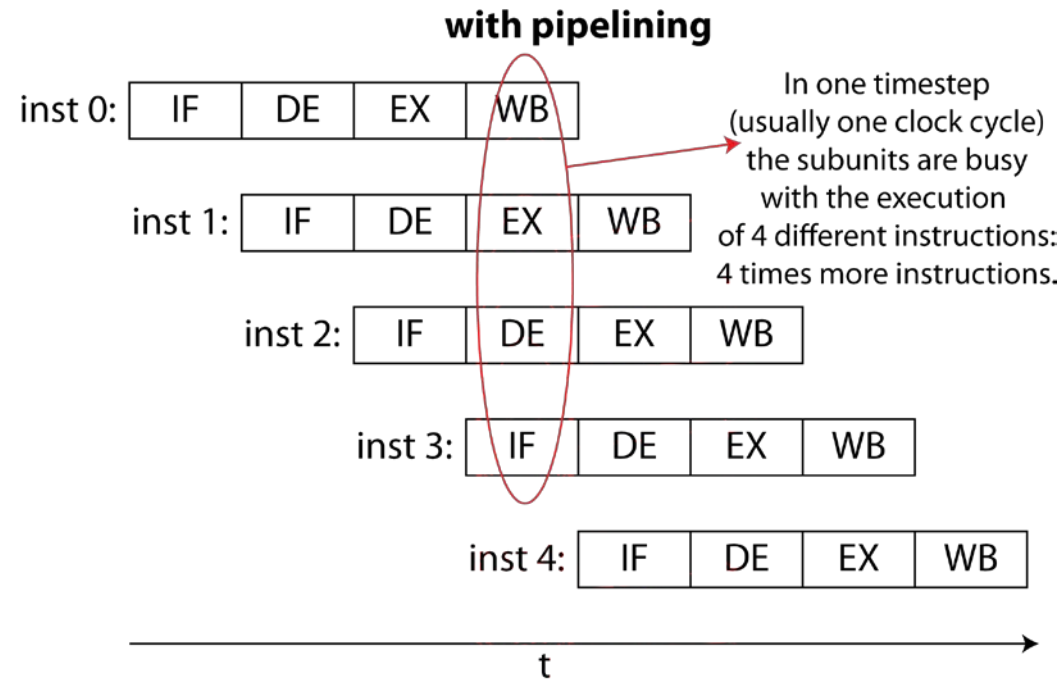
**Minnehierarki**



# Hovedpunkter

- Bakgrunn
  - Kort repetisjon
  - Motivasjon
- Teknikker for hastighetsøkning
  - Multiprocessor
  - Økt klokkehastighet
  - Raskere disk
  - Økt hurtigminne
    - Bruksområder
    - Lagringskapasitet
    - Aksestider
- Cache
  - Mapping strategier
  - Write strategier
  - Replacement strategier
  - Arkitektur

# Pipeline



# Sentrale temaer

- Speed-up
- Hazarder
  - Data
  - Control
  - Resource

# Teknikker for hastighetsøkning

- Pipelining er viktig for å øke antall instruksjoner som utføres pr. sekund
- Andre teknikker som øker hastighet:
  - Flere CPU i en maskin (multiprosessor)
  - Økt klokkefrekvens
  - Stor ordlengde på instruksjoner
  - Raskere disk
  - Hurtigere og større RAM
- Noen av disse er ren forbedring i teknologien, andre er forbedringer av selve arkitekturen.

# Multiprosessor

- Ide: Istedenfor å la én CPU utføre instruksjoner, lar man flere CPUer samarbeide om den samme jobben.
- Gir en teoretisk hastighetsøkning direkte proporsjonal med antall ekstra CPUer: det går  $n$  ganger raskere med  $n$  CPUer enn med én CPU.
- I praksis ikke mulig av flere årsaker:
  - Ikke alltid mulig å dele opp et problem i like store deler som kan løses uavhengig av hverandre.
  - Det kreves administrasjon og koordinering av programeksekveringen før, under og etter at jobben er fordelt på de ulike CPUene, dvs ekstra overhead
- Endel applikasjoner egner seg for lastdeling, f.eks server-applikasjoner, dvs enkelt å parallellisere oppgavene

# Økt klokkehastighet og større ordlengde

- Økt klokkehastighet reduserer tiden hver enkelt instruksjon tar (Resultatet av teknologisk utvikling.)
- Hastigheten til alle deler av en datamaskin øker ikke like raskt:
  - Interne data/adressebusser
  - Nettverk
- Klokkehastigheten til en CPU øker mye raskere enn lese/skrivehastigheten til hukommelsen, mao: Effektive hastighets-forbedring blir langt mindre.
- Brede datapath og ordlengde gjør det mulig å behandle større tall i en operasjon og å lese/skrive mer data fra hukommelsen i én operasjon.
- Det er en øvre grense for hvor mange bit som kan behandles i en operasjon

# Raskere disker

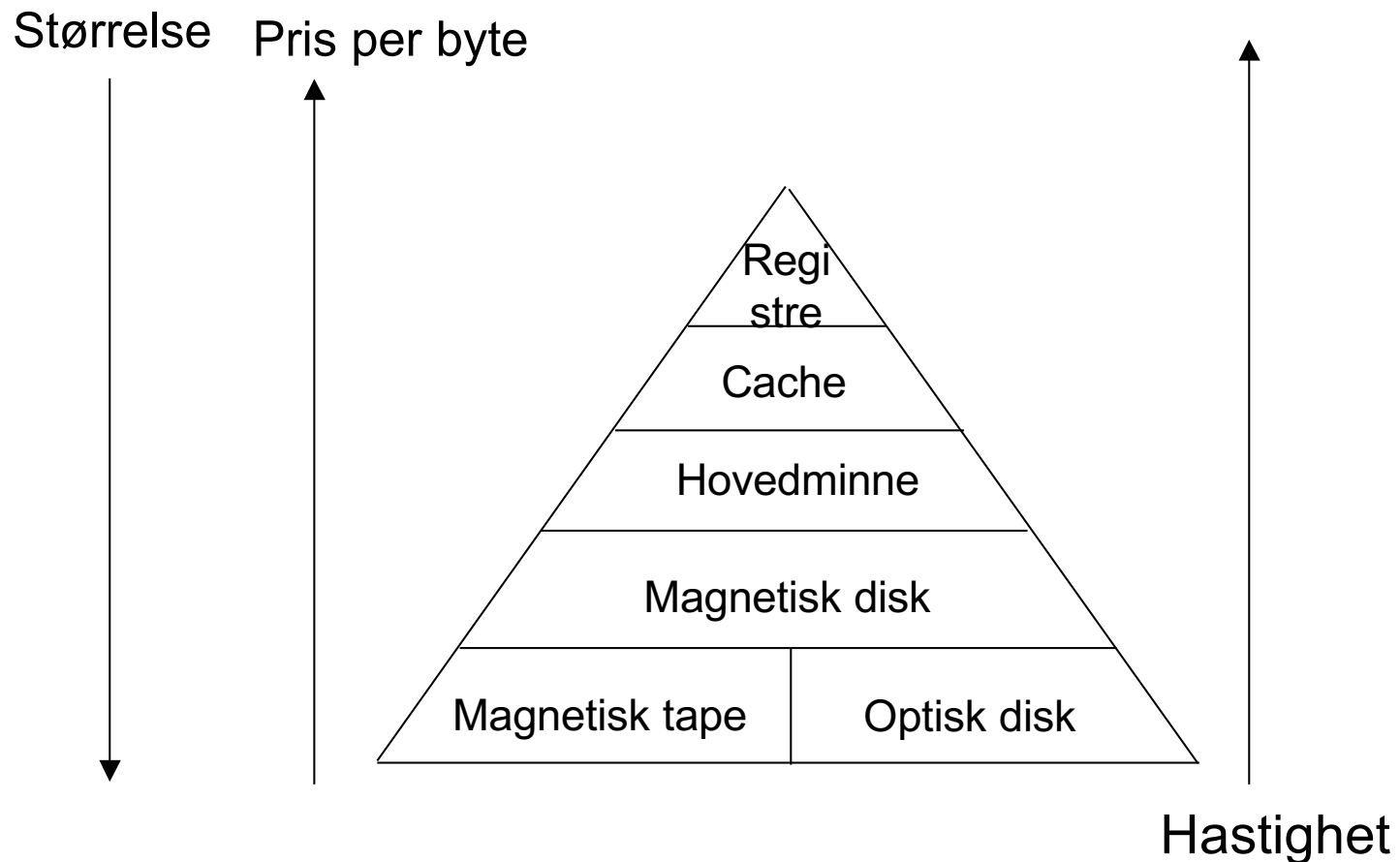
- Programmer trenger å aksessere disker av og til ved f.eks innlesning og skriving av data, ved oppstart og virtuell hukommelse
- En harddisk ca 100 000 ganger langsommere enn en CPU, dvs at en CPU kan gjøre 100 000 instruksjoner, mens en harddisk gjør én lese/skriveoperasjon.
- Teknikker finnes for å lese store blokker av data ad gangen, prøve å gjette hvilke data som skal leses neste gang og lagre disse i hurtigminne, etc
- Harddisker blir både raskere og får mer kapasitet, men allikevel vil hyppig diskaksess være et betydelig problem hvis hastighet er viktig.



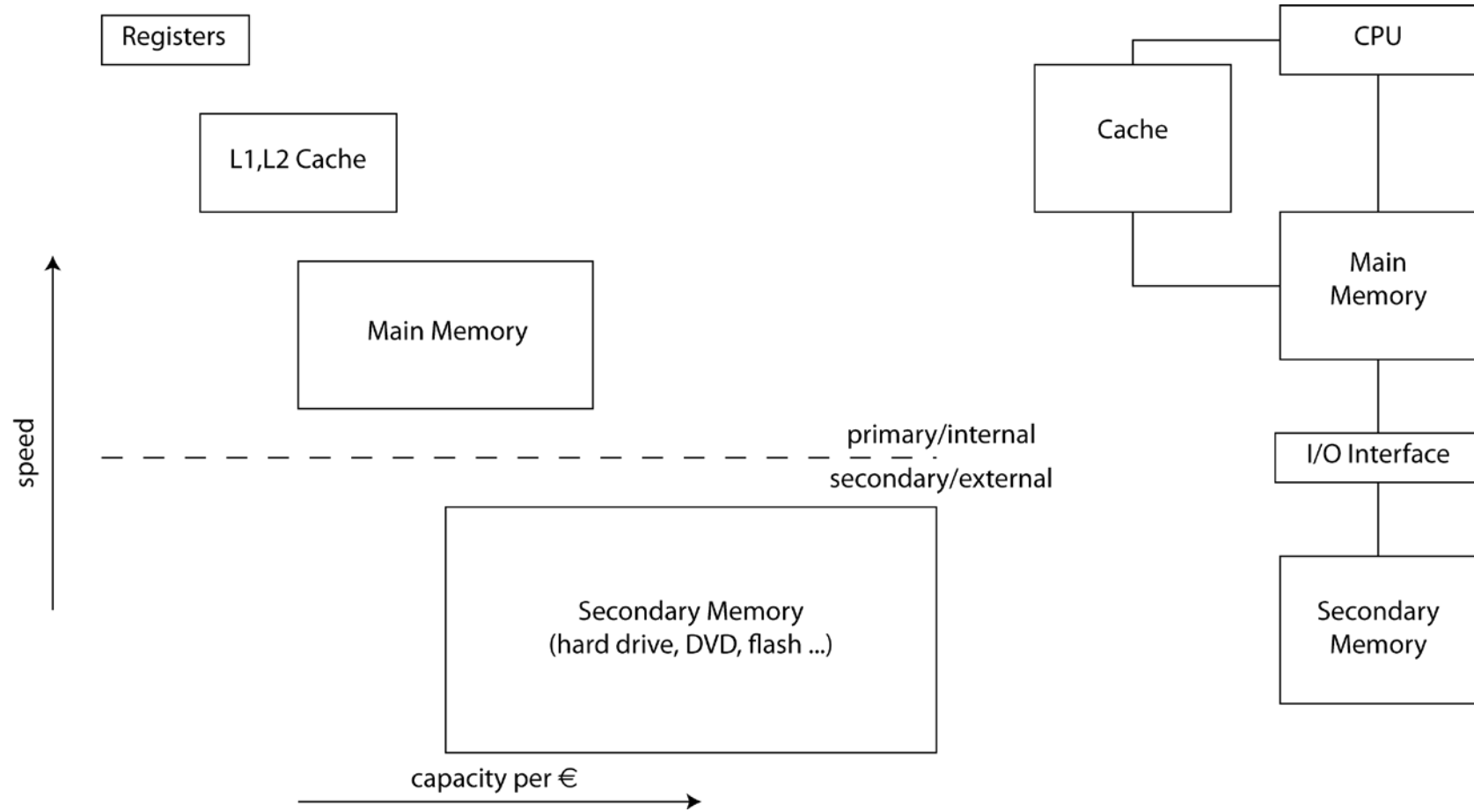
# Større og raskere hurtigminne

- Ønsker ubegrenset med minne som er like raskt som CPU'en.
- I praksis må man velge ut fra flere hensyn.
- Tre parametre klassifiserer minnetyper:
  - Antall byte tilgjengelig per enhet
  - Pris per byte
  - Aksesshastighet

# Minnehierarki



# Minnehierarki



# Minnehierarki – Bruksområder

- **Register**

*Intern kladdeblokk for CPU ´en med rask aksess til innholdet.*

- **Cache**

*Hurtig mellomlager for både instruksjoner og data for å jevne ut hastighetsforskjellen mellom CPU ´en og hurtigminnet.*

- **RAM**

*Buffer mellom eksternt lagringsmedium og CPU ´en med rask både lese- og skrive aksess.*

- **SSD/Flash/Harddisk**

*Høykapasitetsmedium for program/data*

# Minnehierarki – Lagringskapasitet

- **Register**

*Integrert på CPU ´en, relativt få (32-128 stykker)*

- **Cache**

*Mellomlager internt (L1) eller i nærheten (L2,L3) av CPU ´en, typiske kapasiteter er fra 10 KiloByte (L1) til 1 MegaByte (L2) og flere MegaByte (L3).*

- **RAM**

*Internt på hovedkortet i nærheten av CPU ´en, størrelser opptil flere GigaByte.*

- **SSD/Flash/Harddisk**

*Ekstern eller intern lagringsenhet i maskinen med kapasitet opptil flere TeraByte.*

## Minnehierarki – aksesshastighet

Registers	< 1ns	≈ 100 Byte
L1 (på CPU) cache	≈ 1ns	≈ 10 KB
L2,L3 (utenfor CPU) cache	2-10ns	≈ 1 MB
Hovedminne (RAM)	20-100ns	≈ 1 GB
SSD/Flash	100ns-1us	≈ 1 TB
Harddisk	1ms	≈ 1 TB

# Cache

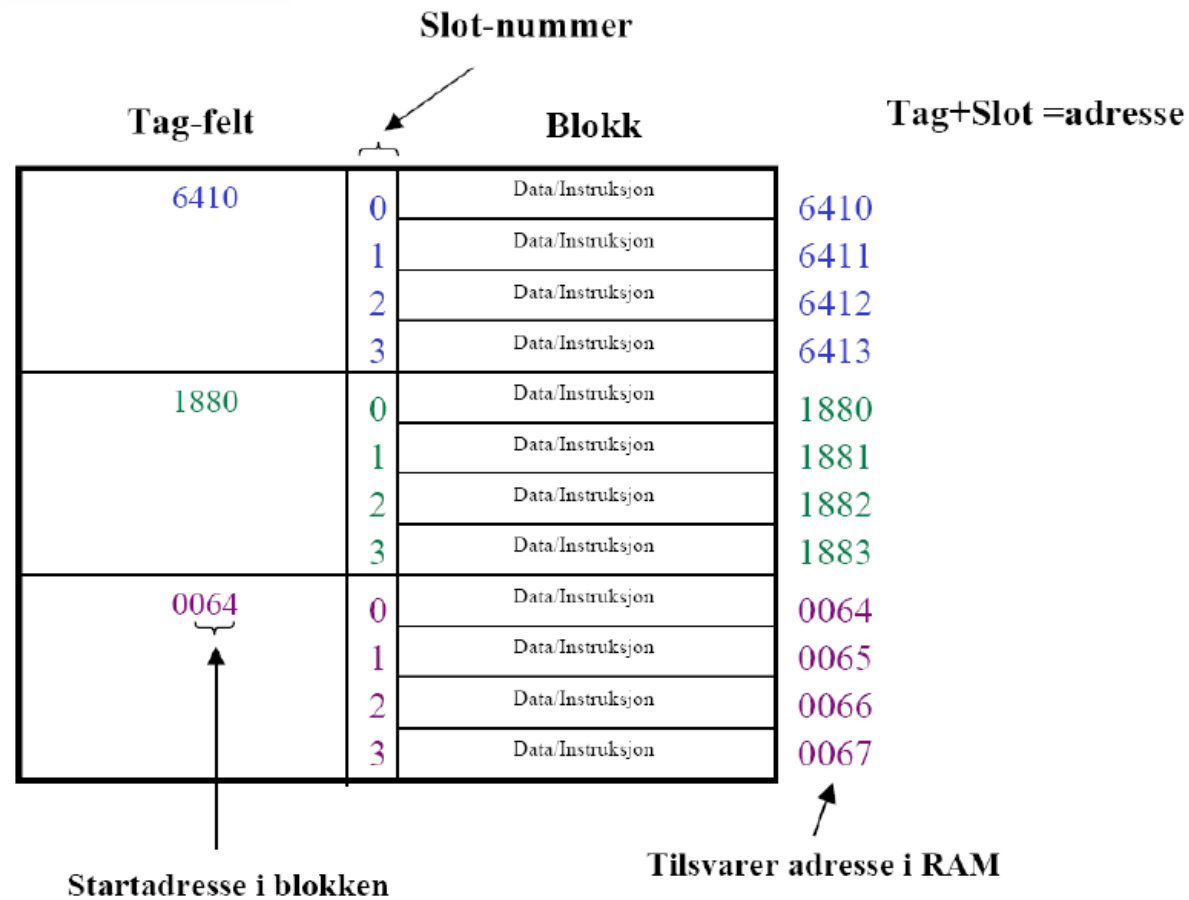
- Cache forbedrer von Neumann arkitektorens flaskehals med hensyn på minneaksessering.
- Cache ligger nærme CPU for å øke hastighet, men små størrelser for å redusere produksjonskost og derfor er det et trade-off diskusjon mellom størrelse og hastighet
- Cache inneholder en kopi av et subsett av hurtigminne (RAM)
- CPU´en henter data og/eller instruksjoner fra cache istedenfor RAM.

# Cache

- Siden cache er mindre enn RAM, må det bestemmes hvilke data/instruksjoner som skal ligge i cache
- Cache basere seg på at instruksjoner/data aksesseres:
  - Nær hverandre i tid
  - Nær hverandre i rom



# Cache



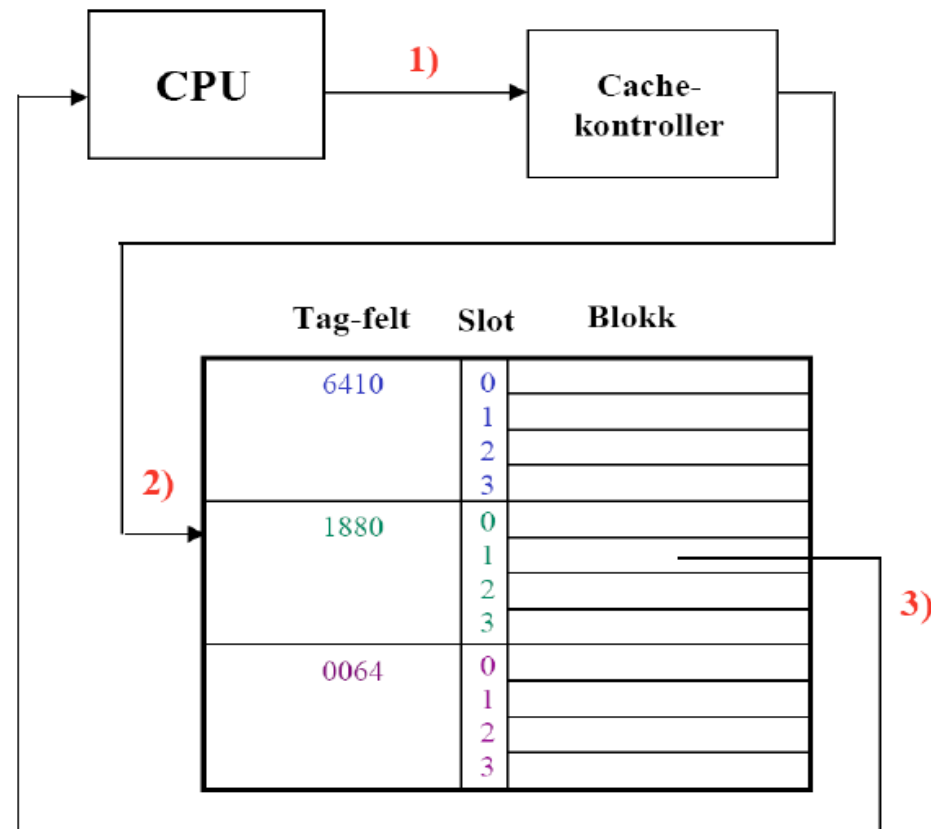
# Cache

- Hvis data/instruksjon CPU'en trenger ligger i cache, kalles det for ***cache hit***.
- Hvis data/instruksjon CPU'en trenger ikke ligger i cache, kalles det for ***cache miss***.
- En essensiell del for utnyttelsen av cache er å kunne raskt sjekke for hit eller miss.

# Cache hit

- Når en prosessor skal hente en instruksjon eller lese/skrive data, vet den ikke om den skal hente fra RAM eller cache. Hvis det prosessoren ber om ligger i cache, kalles det en cache hit

Eks: hent innholdet  
i loaksjon 1881.



# Cache miss

- Ved **cache miss** må data/instruksjon først kopieres fra hurtigminnet over i cache før CPU'en kan bruke det.
- Cache fylles fra hurtigminnet første gangen en bestemt lokasjon blir referert

# Cache miss

X4
X1
Xn - 2
Xn - 1
X2
X3

Før referanse til Xn

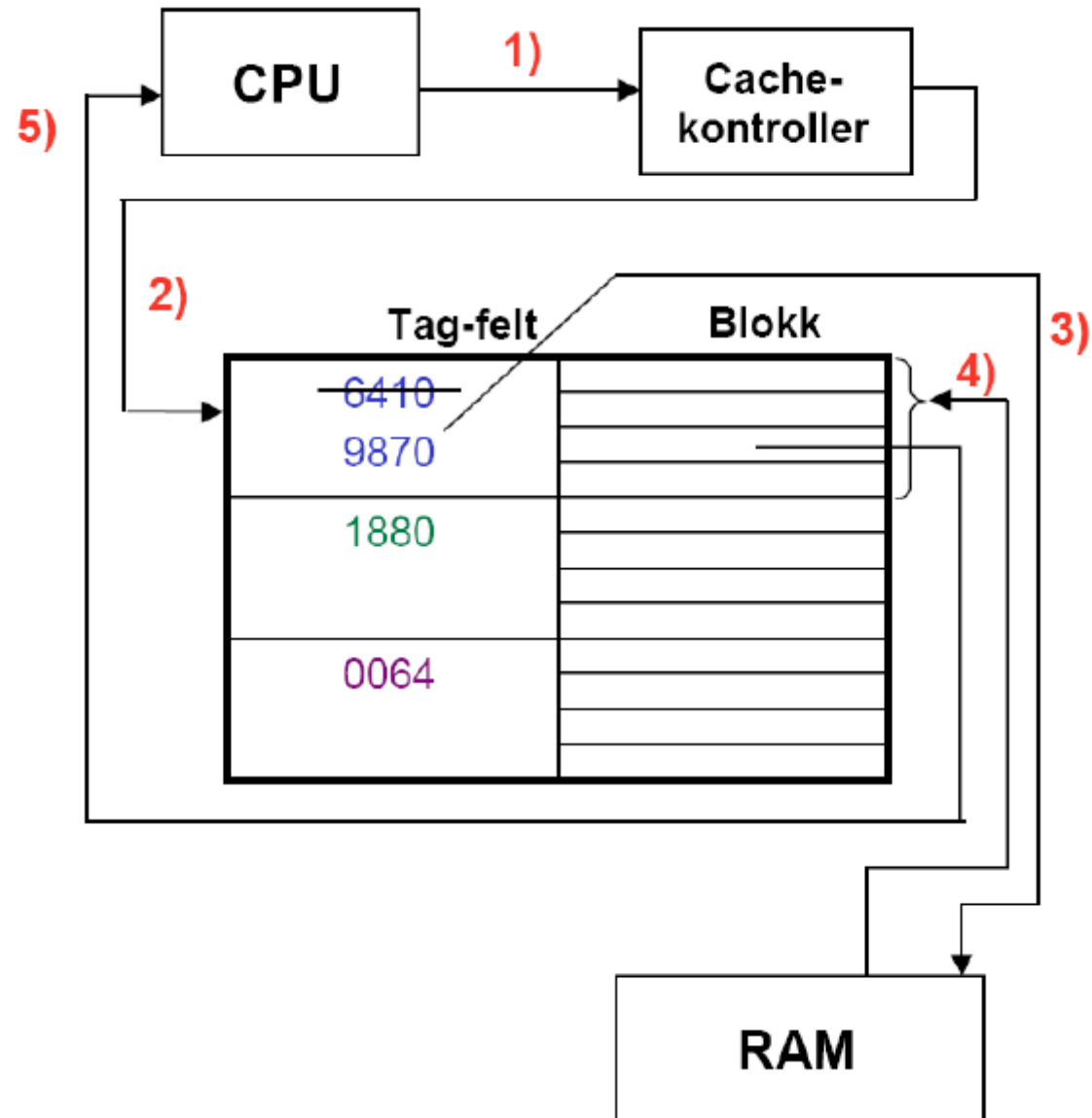
X4
X1
Xn - 2
Xn - 1
X2
Xn
X3

Etter referanse til Xn

# Cache miss

- Hvis data ikke ligger i cache, får man en cache miss

- (1) CPU ber om innholdet i lokasjon 9872
- (2) Cachekontrolleren finner ut at denne ikke finnes i cachen
- (3) Blokken som starter på 9870 leses inn fra RAM
- (4) Blokken som starter på 6410 overskrives for å gi plass til 9870



# Indeksering av cache og blokkoverskrivning

**(1) Hvordan** cache kontrolleren indekserer  
(dvs. leter fram) riktig blokk ved  
lesing/skriving

**(2) Hvilken** blokk som skal overskrives(kastes  
ut når cache er full

# Mapping strategier for cache

1. Direct-mapped cache
2. Set-associative cache
3. Full associative cache

Metodene skiller seg fra hverandre ved:

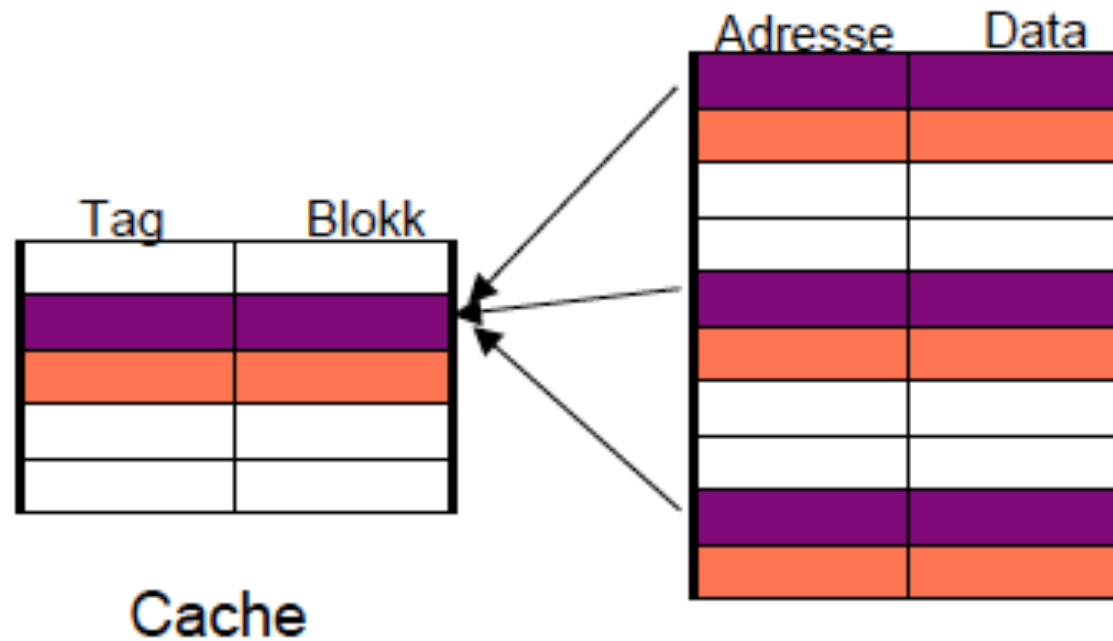
- Hvor enkle de er å implementere
- Hvor raskt det går å finne en blokk
- Hvor god utnyttelse man får av cache
- Hvor ofte man får en cache miss



# Direct-mapped Cache

- 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 (selvom det er ledig plass andre steder, kan en blokk kun plasseres ett bestemt sted)

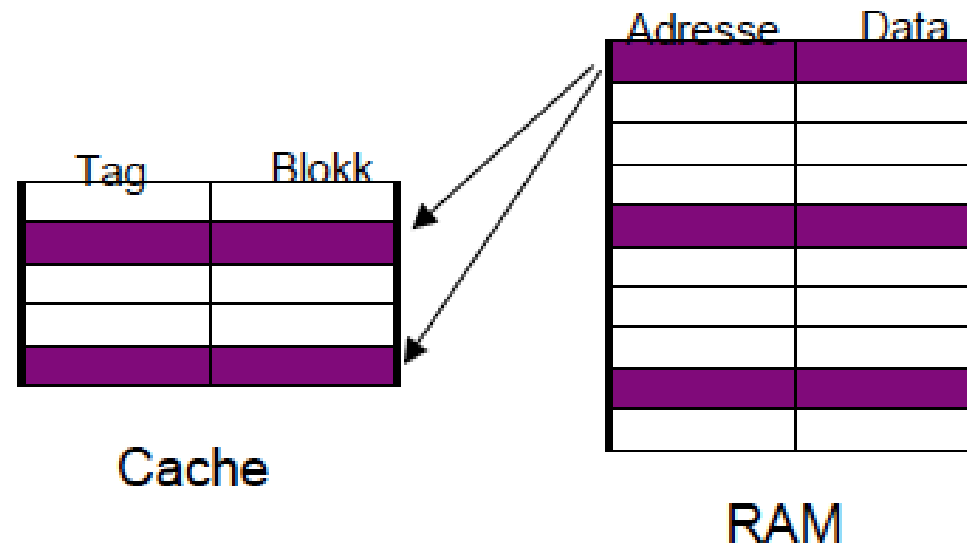
# Direct-mapped Cache



## Set-associative

- En bestemt blokk fra RAM kan plasseres i et begrenset antall blokk-lokasjoner i cache.
  - 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.
  - Ulempe: Lenger søketid enn ved direkte-mappet (med mindre man søker i parallell gjennom tag-feltene)

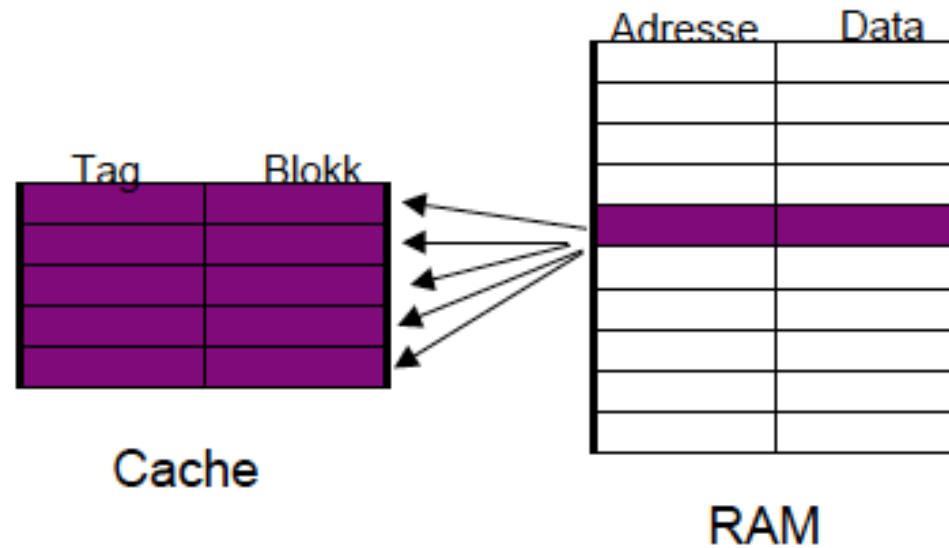
# Set-assosiative



## Full assosiative

- En bestemt blokk fra RAM kan plasseres hvor som helst i cache
  - Fordel: Cachen utnyttes meget godt, har minst sjanse for cache miss av alle de tre metodene
  - Ulempe: Søket for å finne en blokk kan bli tidskrevende. Man kan lage mekanismer for å forenkle søk, f.eks. hashing, men dette kompliserer cache-kontrolleren og krever ekstra hardware.

# Full-assosiativ



# Write strategier

- **Write hit**
  - Det skal skrives til hukommelsen og blokken med lokasjonen det skal skrives til ligger i cache
  - Data skrives til riktig lokasjon i blokken i cache
- **Write miss**
  - Det skal skrives til hukommelsen og blokken med lokasjonen det skal skrives til ligger ikke i cache
  - Cache-kontrolleren må bestemme
    - Hvilken blokk i cache som kan/skal overskrives
    - Kopiere inn riktig blokk fra RAM til cache
    - Besørge skrijving til riktig lokasjon i cache
- Ved begge write-operasjoner vil ikke innholdet i RAM og innholdet i kopien av blokkene i cache være identiske -> **ikke OK**

## Write inkohrens

- Hvis det bare skrives til cache og ikke RAM, vil ikke RAM inneholde gyldige data
- Hvis en blokk det er skrevet til kastes ut fordi cache er full, mister man data
- To strategier benyttes for å håndtere write-operasjonen korrekt:
  - 1) Write-through
  - 2) Write-back



# Write-through

- Etter hver gang det skrives til en blokk, skrives innholdet i blokken også tilbake til RAM
  - **Fordel:**
    - Enkelt å implementere i cache-kontrolleren
  - **Ulempe:**
    - Hvis det skal skrives flere ganger rett etterhverandre til samme blokk må prosessoren vente mellom hver gang

# Write-back

- Innholdet i en blokk i cache kopieres kun tilbake til RAM hvis blokken skal overskrives i cache og det har blitt skrevet til den i cache
- For raskt å detektere om en blokk har blitt skrevet til benyttes et kontroll-bit i cache (kalles dirty-bit'et)
  - **Fordel:**
    - Gir ingen hastighetsreduksjon ved flere påfølgende write-operasjoner til samme blokk siden CPU'en ikke trenger vente mellom hver skriving
  - **Ulempe:**
    - Hvis arkitekturen støtter direkte lesing/skriving mellom RAM og Input/Output-enheter uten å gå via CPU'en (kalt Direct Memory Access eller DMA), risikerer man inkonsistente data hvis ikke DMA-kontrolleren kommuniserer med cache-kontrolleren

# Replacement strategy

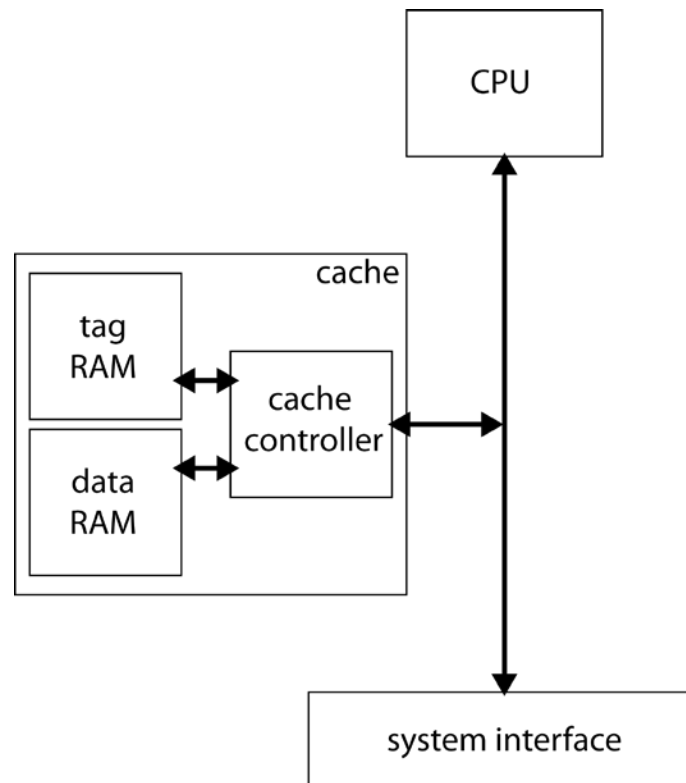
- Ved en cache miss må noen ganger en eksisterende blokk kastes ut for å gi plass til en ny blokk. Hvilken blokk som fjernes kan ha stor betydning for hastigheten til programmet som eksekveres.
- First-in-First-out (FIFO)
  - Blokkene er organisert som en ringbuffer.
  - Blokken som har ligget lengst i minne blir overstrevet (uavhengig om den er mye brukt)
- 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 tilfeldig valgt fra den gruppen som har ligget lengst i cache uten å bli brukt.

# Arkitektur

- Cache kan plasseres på (minst) to måter i forhold til CPU og RAM
  - (1) Look-aside
  - (2) Look-through
- De to organiseringene påvirker hvordan lesing håndteres ("read architecture")
- "Read architecture" og "write policy" er det viktigste karakteristika til cache
- Cache koherens: Hvorvidt innholdet i cache og RAM er identisk
- For å bevare koherens kan cache gjøre
  - "Snoop": Cache overvåker adresselinjer for å se etter transaksjoner som angår data den sitter på
  - "Snarf": Cache tar data fra datalinjer og kopierer dem inn
- Inkoherens kan være
  - "Dirty data": Data i cache er modifisert, men ikke tilsvarende data i RAM
  - "Stale data": Data i RAM er modifisert, men ikke tilsvarende data i cache.

# Look-aside arkitektur

- Cache er plassert med RAM
- Både RAM og cache leser adresselinjene samtidig



# Look-aside arkitektur

- Når prosessorer starter en lesing, sjekker cache adressen (snooping) for å se om det er en "read hit"
- Hvis "read hit":
  - Cache svarer prosessoren og stopper videre buss-syklus for å hindre lesing fra hovedminnet (RAM)
- Hvis "read miss":
  - Hovedminnet (RAM) vil besvare henvendelsen fra prosessoren. Cache "snarfer" data slik at de er i cache neste gang prosessoren ber om dem.
- Fordeler:
  - Mindre komplisert enn look-through
  - Bedre responstid (RAM og CPU ser samme buss-syklus)
- Ulempe
  - Prosessoren kan ikke aksessere cache hvis en annen enhet aksesserer RAM samtidig

# Look-through

- Cache sitter fysisk mellom CPU og internbussen
- Cache-kontrolleren vil avgjøre om buss-syklus skal slippe videre til RAM

