UNIVERSITETET
I OSLO

Dagens tema

- Flere teknikker for å øke hastigheten
- Cache-hukommelse del 1 (fra kapittel 6.5 i ”Computer Organisation and Architecture”)
  - Hvorfor cache
  - Grunnleggende virkemåte
  - Direkte-avbildet cache
  - Cache-arkitekturer

2008

Teknikker for hastighetsøkning

- Pipelining er viktig for å øke antall instruksjoner som utføres per sekund.
- Andre teknikker som øker hastigheten ytterligere er:
  - Flere CPUer i en maskin (multiprosessor)
  - Økt klokkehastighet
  - Stor ordlengde på instruksjoner og i datapath
  - Raskere disker
  - Hurtigere og større RAM
- Noen teknikker er ren forbedring i teknologi, andre er forbedring av selve arkitekturen.

Multiprosessorer (parallellprosessorer)

- 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.
- Bredere 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 (typisk i dag er 64 eller 128 bit).

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 er omtrent 100 000 ganger langsommere enn en CPU, dvs. at en CPU kan gjøre 100 000 instruksjoner mens en harddsik gjør én lese/skriveoperasjon.

- Teknikker finnes for å lese store blokker av data av gangen, prøve å gjette hvilke data som skal leses neste gang og lagre disse i hurtigminne, etc.

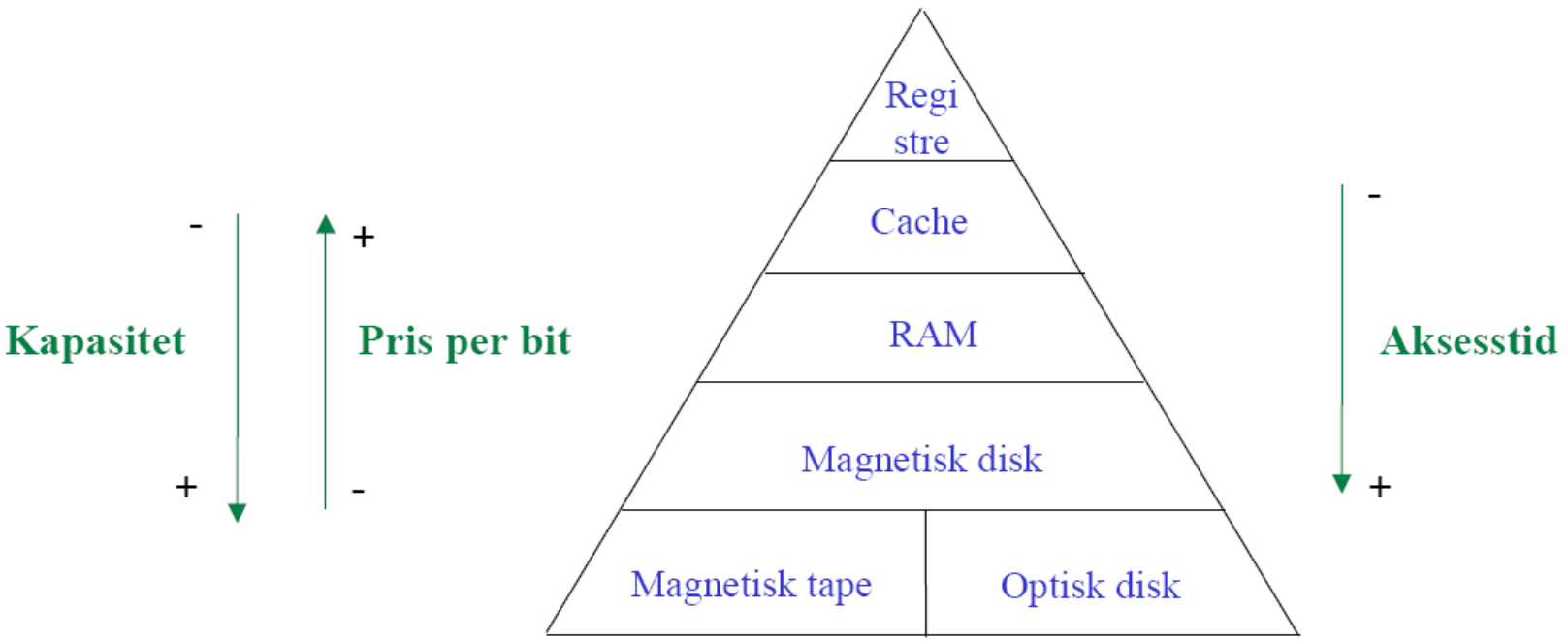
- Harddisker blir både raskere og får mer kapasitet, men allikevel vil hyppig diskaksess være et betydelig problem hvis hastighet er viktig.

- Solid State disker (SSD) begynner å komme på markedet. Disse diskene har ingen mekaniske deler og omtrent 100 ganger raskere aksesstid enn en tradisjonell harddisk. Dataoverføringshastigheten er omtrent lik

Større og raskere hurtigminne

- Det optimale ville være ubegrenset tilgang på billig hurtigminne like raskt som prosessoren selv. Henting av instruksjoner, og lesing/skriving av data ville gå like raskt som registeraksess.

- I praksis må man velge hva som er viktigst av pris, hastighet og kapasitet



2008

Bruksområde for de ulike hukommelsestypene

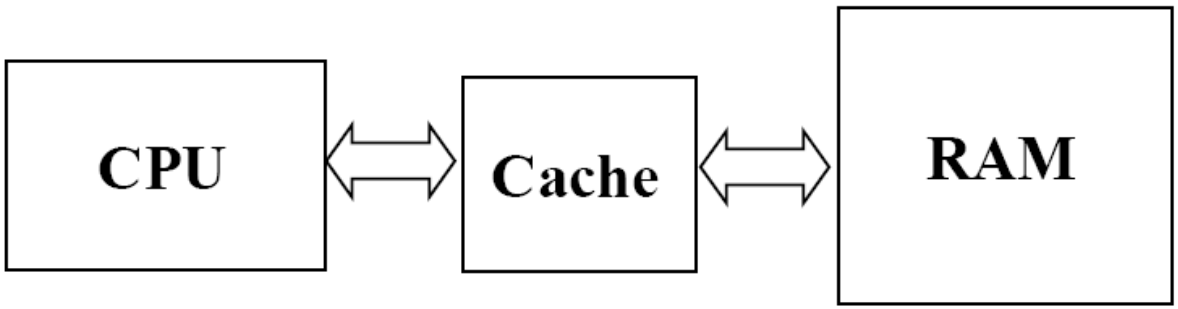
- **Registre**: Integrert på CPU'en, relativt få (32-64 stykker) med like mange bit i hver som maskinens ordbredde
- **Cache**: Mellomlager som ligger inne på (L1) eller i nærheten av (L2, L3) CPU'en, typisk kapasitet fra 8 KiloByte (L1) til 512 (L2) KiloByte, og noen MegaByte (L3).
- **RAM**: Internt på hovedkortet i nærheten av CPUen, størrelse opptil mange GigaByte.
- **Magnetisk disk**: Ekstern eller intern lagringsenhet i maskinen, med kapasitet opptil TeraByte.
- **Magnetisk tape**: Sekvensielt medium med opptil 10-talls sekunders aksesstid.
- **Optisk disk**: (CD-ROM, CD-RW og DVD). Billig eksternt lagringsmedium med kapasitet opptil flere GigaByte (DVD).

Aksesstider

- **Registre**: En klokkesykel, dvs. like raskt som resten av prosessoren (nanosekunder).
- **Cache**: Samme størrelsesorden som interne registre, men likevel noe langsommere (avhengig av L1 eller L2/L3).
- **Hovedminne**: Flere titalls nano-sekunder.
- **Magnetisk disk**: 10 000 til 100 000 ganger langsommere aksesstid, dvs. flere millisekunder. For å finne data må disken rotere til riktig posisjon
- **Solid State disk**: aksesstid på rundt 0.1mikrosekund (100 nanosekunder)
- **Magnetisk tape**: Sekvensielt medium med opptil flere titalls sekunders aksesstid (man må i verste fall spole gjennom hele tapen før man finner det man leter etter.
- **Optisk disk**: (CD-ROM, CD-RW og DVD): Opptil flere sekunders aksesstid. For å finne data må platen rotere til riktig posisjon.

Cache

- Ønsker så mye og rask hukommelse som mulig tilgjengelig for et program under eksekvering, både for instruksjoner og data
- Cache-minnet er logisk sett plassert mellom CPUen og RAM slik det er vist i figuren under:
- Innholdet i cache vil alltid være et subsett av innholdet i RAM
- Fysisk er cache enten integrert på CPU'en eller plassert rett ved siden av (eller begge deler)



Cache (forts.)

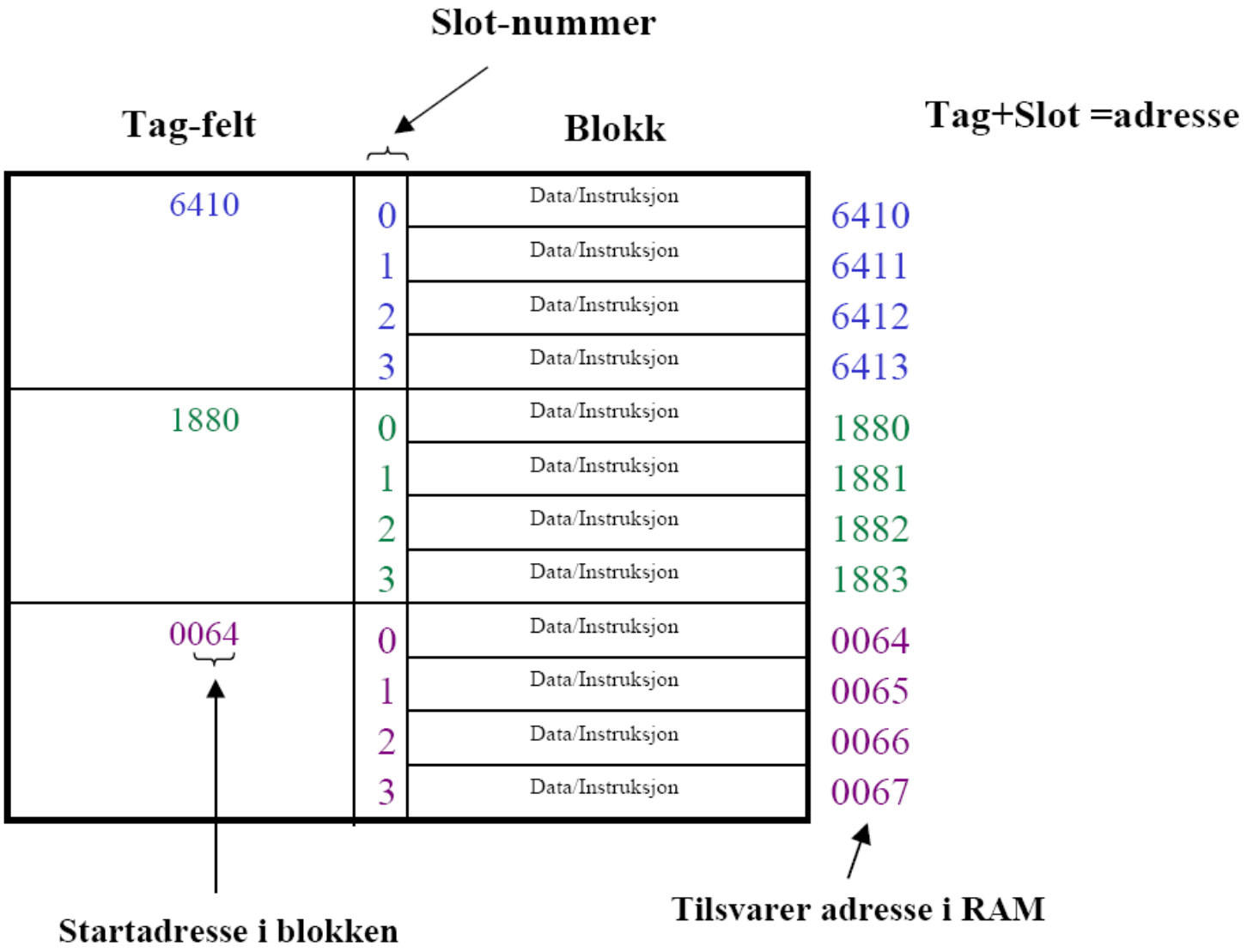
- Kapasiteten til cache ligger i mellom det interne i CPUen og RAM. Typiske verdier er fra 8 KB til 2 MB (Finnes i flere nivåer kalt L1, L2 og L3)
- Siden cache'n er nesten like rask som CPUen, trenger ikke CPU'en å stoppe opp og vente i mange klokkesykler på at RAM skal levere data.
- CPU'en har ingen "bevissthet" om at det finnes cache og ser ikke forskjell på om data eller instruksjoner ligger i cache eller i RAM
- CPUen får beskjed utefra (buss- eller cache-kontroller) når data er klare. Om CPUen venter 1 eller 100 000 klokkesykler spiller ingen rolle annet enn for hastigheten.

Cache (forts.)

- Siden cache'n er mindre enn hurtigminnet (RAM) består mesteparten av administrasjonen av cache i å velge ut hvilken del av programmet og hvilke data som skal ligge i cache, og hvilke som må ligge i RAM.

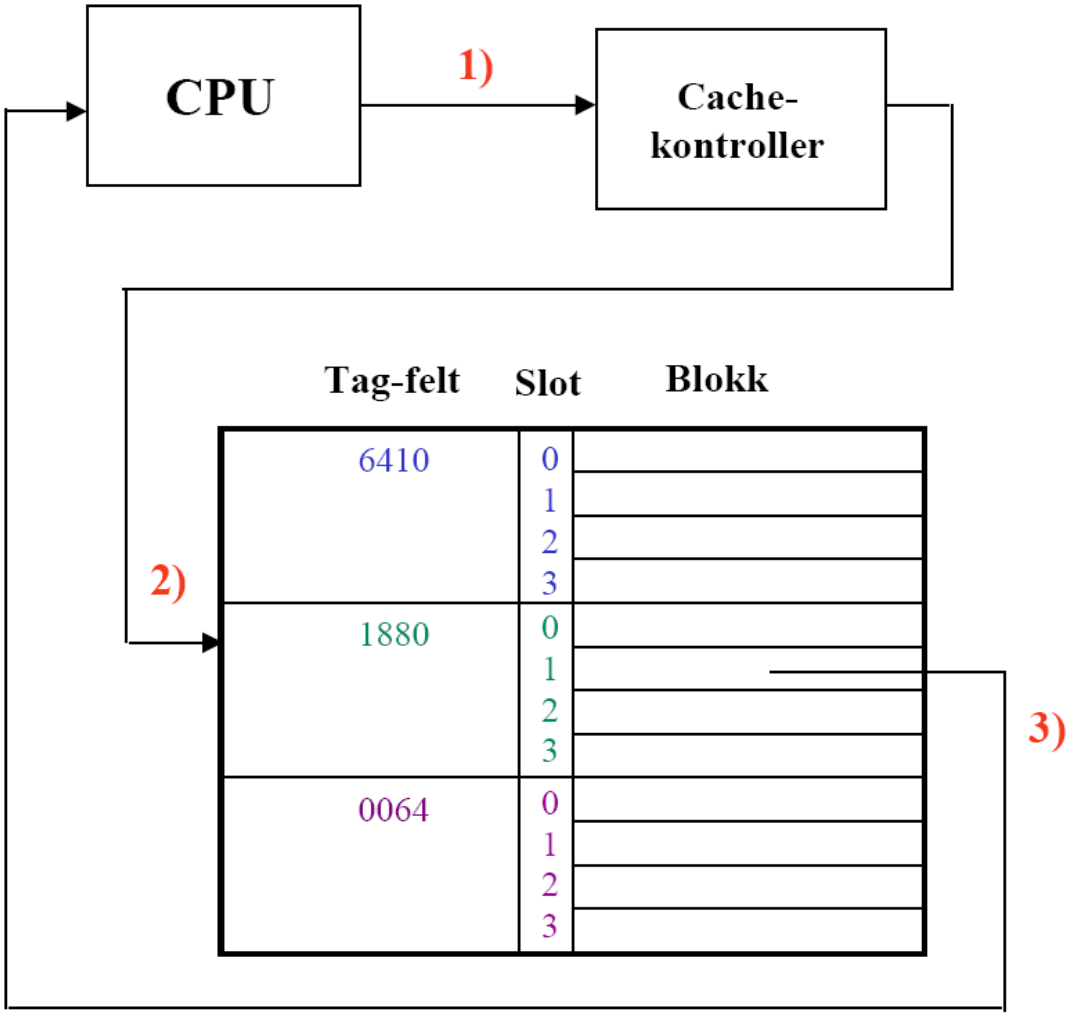
- Cache benytter seg av lokalitetsprinsippet:
  - Instruksjoner/data aksesseres som regel sekvensielt, dvs. fra samme område i hukommelsen
  - Som en konsekvens vil de samme instruksjonene/data også aksesseres nær hverandre i tid

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

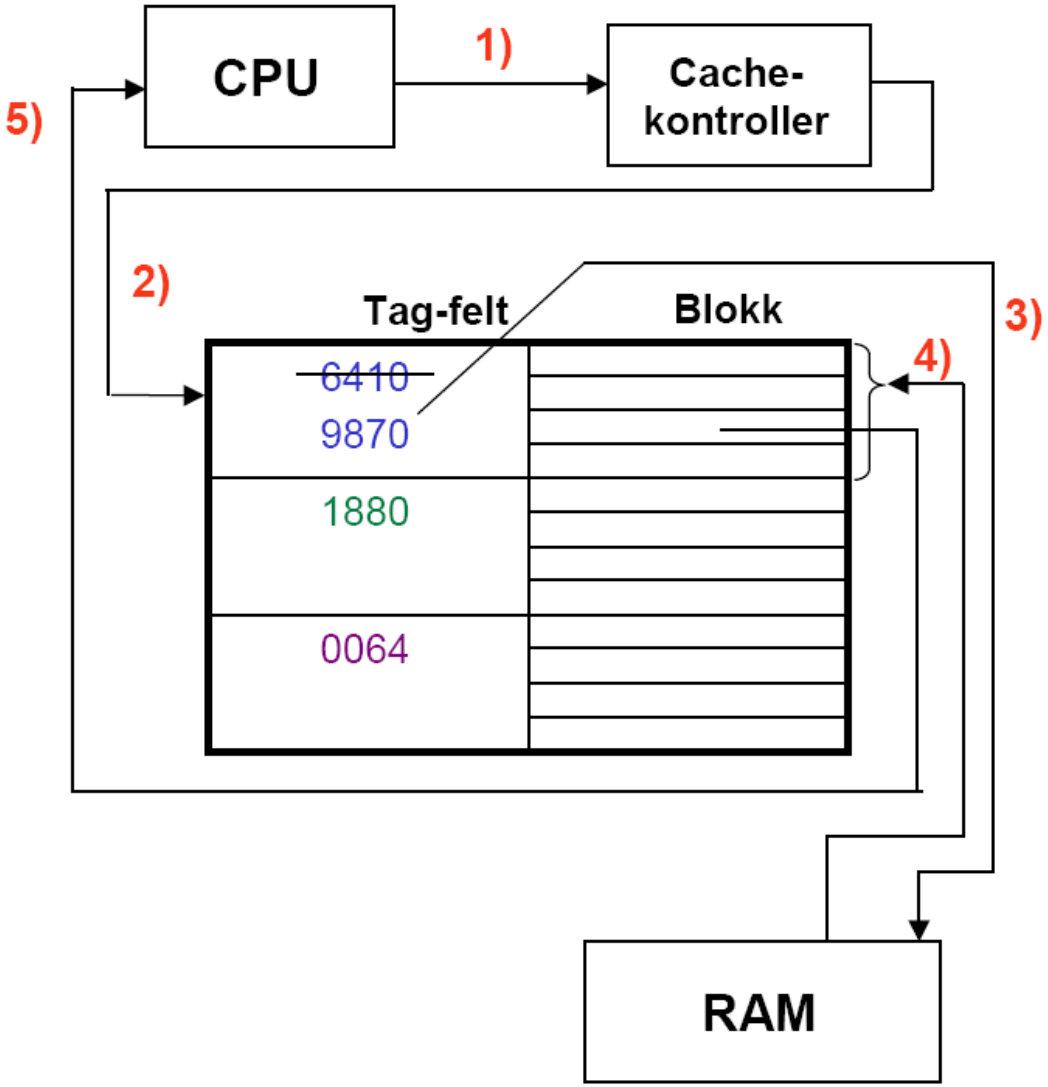

Slot-nummer
Tag-felt
Blokk
Tag+Slot =adresse
6410
1880
0064
0
1
2
3
Data/Instruksjon
6410
6411
6412
6413
1880
1881
1882
1883
0064
0065
0066
0067
Startadresse i blokken
Tilsvarer adresse i RAM

- 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 for cache hit:

- Figur til høyre: CPUen ber om å få innholdet i lokasjon 1881



- Hvis data ikke ligger i cache, får man en cache miss
- Figur til høyre: CPU'en ber om lokasjon 9872
- Cachekontrolleren finner ut at denne ikke finnes i cachen
- Blokken som starter på 9870 leses inn fra RAM
- Blokken som starter på 6410 overskrives for å gi plass til 9870



Read/write hit/miss

- Cache-kontrolleren må håndetere 4 tilstander av aksess av cache

- **Read hit**
  - Det skal leses fra hukommelsen og blokken med ordet befinner seg i cache
  - Data leveres med en gang fra cache til CPU'en

- **Read miss**
  - Det skal leses fra hukommelsen, men blokken det skal leses fra er ikke i cache
  - Cache-kontrolleren må bestemme
    - Hvilken blokk i cache som kan/skal overskrives
    - Kopiere inn riktig blokk fra RAM til cache
    - Levere ordet til CPU'en

- Ved begge read-operasjoner vil innholdet i RAM og innholdet i kopien av blokkene i cache være identiske → **OK**

Read/write hit/miss (forts.)

- **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 skriving 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**

Mer om write hit/miss

- 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:
  - **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 etter hverandre til samme blokk må prosessoren vente mellom hver gang

Mer om write hit/miss

- **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 det skal skrives flere ganger rett etter hverandre til samme blokk må prosessoren vente mellom hver gang.
    - 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

Cache-arkitektur

•Cache kan plasseres på (minst) to måter i forhold til CPU og RAM

• De to organiseringene påvirker hvordan lesing håndteres (”read architecture”)

• ”**Read architecture**” og ”**write policy**” er det viktigste karakteristika til cache

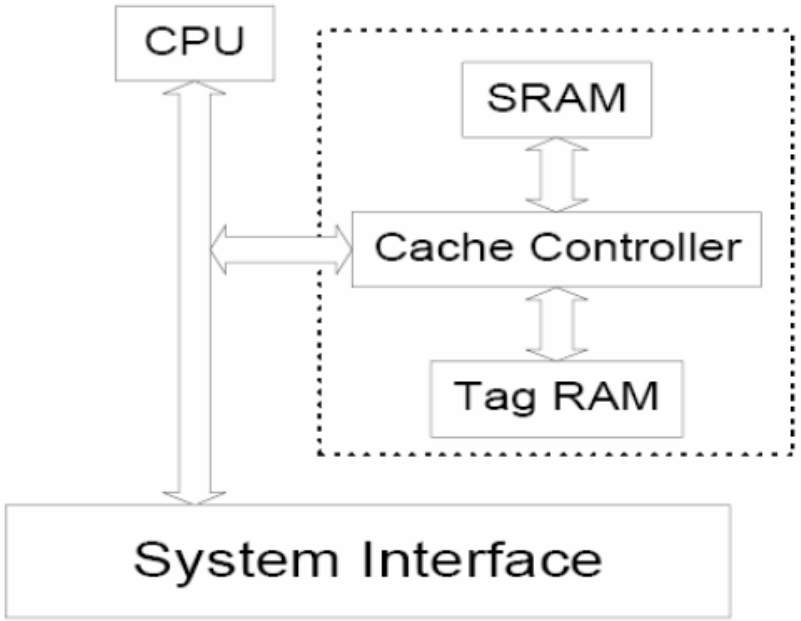
• Cache **koherens**: Hvorvidt innholdet i cache og RAM er identisk

• 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 (1)

- Cache er plassert parallelt med RAM (system-interface)
- Både RAM og cache leser adresselinjene samtidig



Look-aside arkitektur (2)

- Når prosessoren 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)

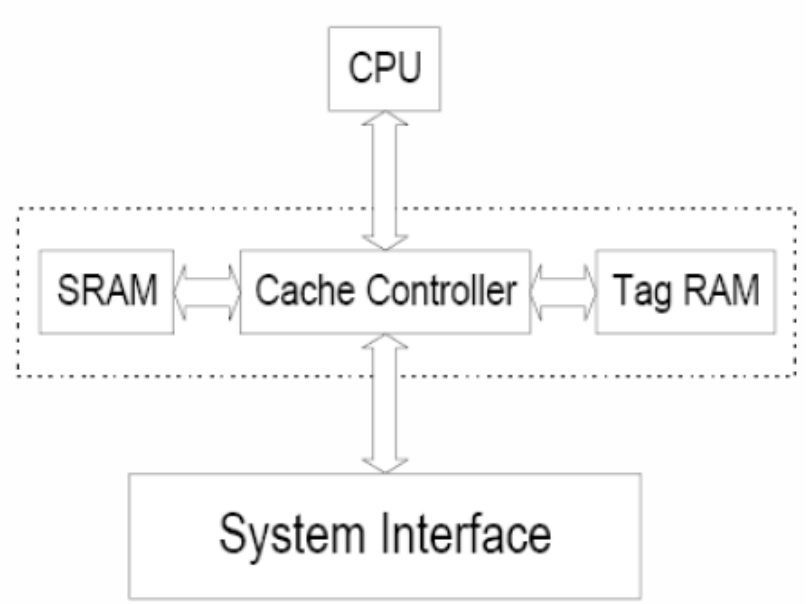
- Hvis ”read miss”:
  - Hovedminnet (RAM) vil besvare henvendelsen fra prosessoren. Cache ”snarfer” data slik at de er i cache neste gang prosessoren ber om dem.

- 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 arkitektur (1)

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



Look-through arkitektur (2)

- Når prosessoren starter en minne-aksess vil cache-kontolleren sjekke om det er en ”read-hit”
- Hvis ”read hit”
  - Cache leverer data til prosessoren uten aksess til hovedminnet
- Hvis ”read miss”
  - Buss-syklusen sendes videre til hovedminnet
  - Hovedminnet (RAM) vil besvare henvendelsen fra prosessoren. Cache ”snarfer” data slik at de er i cache neste gang prosessoren ber om dem
- Fordeler:
  - Prosessoren er isolert fra resten av systemet og kan jobbe uavhengig (viktig i fler-prosessor systemer)
- Ulemper
  - Mer komplisert siden cache-kontrolleren må håndtere all minneaksess
  - Tregere ved cache-miss siden cache og RAM sjekkes i sekvens