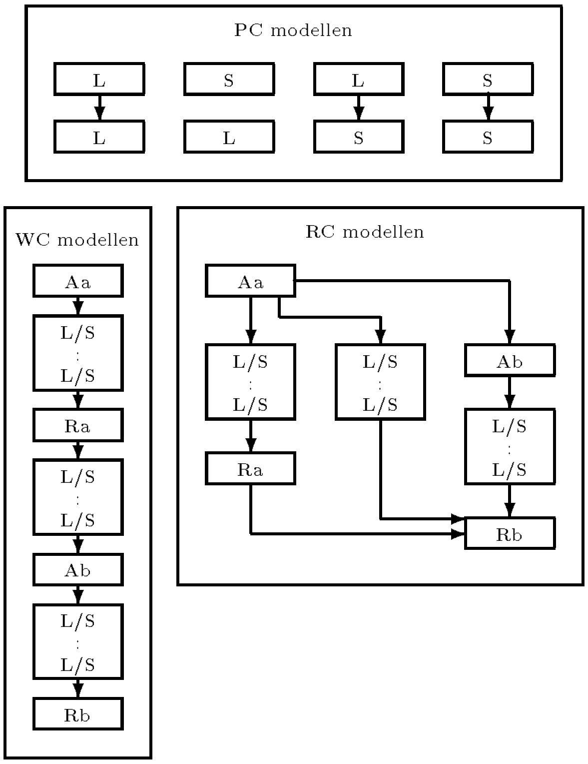
Løsningsforslag til Øving 9:

Oppgave 1:

a)
- For å implementere RC trengs to minne instruksjoner (load-lock og store-unlock) og en oppslags-fri hurtigbuffer, i tillegg til "scoreboarding" for å holde orden på gjenværende forespørseler.
- For å implementere PC trengs multiport minne for å tillate prosessorene å utføre "out-of-order" skriving.
- For å implementere WC trengs store buffere i hver prosessor med passende maskinvare for å omgå stor last.

b) Forskjellige konsistens-modeller medfører forskjellige krav til rekkefølgen på minne-aksesser for hver prosess. Det følgende diagrammet hentet fra [Gharachorloo91] illustrerer hendelses-rekkefølgen til PC-, WC- og RC-modellene. I figuren står L for hent (Eng: load), S for lagre (Eng: store), A for "acquire" og R for "release". En pil betyr at program-rekkefølgen må iakttas. Hent- og lagre-operasjoner i samme blokk kan utføres i hvilken som helst rekkefølge så lenge avhengighets-relasjonene blir respektert. Indekser på "acquire"- og "release"-operasjonene står for synkroniserings-variable eller minne-lokasjoner.



Noen fordeler og svakheter ved hver modell er listet opp under:

Fordel ved PC: Hente-operasjoner kan ta over lagre-operasjoner for samme prosessor hvis aksessene er til forskjellige lokasjoner. Hvis en lagre- og en hente-operasjon er til samme minne-lokasjon, kan hente-operasjonen tilfredsstilles av lagre-operasjonen, slik som i TSO og PSO modellene. Dermed venter aldri hente-operasjoner på uferdige lagre-operasjoner.

Svakhet ved PC: Lagre-operasjoner i hver prosessor må følge program-rekkefølgen, og dermed øker sjansen for at skrive-bufferen kan fylles, som igjen "bremser" prosessoren.

Fordel ved WC: WC sikrer sekvensiell konsistens bare ved synkroniserings-punkter. Hente/-lagre-operasjoner mellom disse synkroniserings-punktene kan derfor utføres i en tilfeldig rekkefølge så lenge kontroll- og data-avhengigheter ikke ødelegges i hver prosessor. Derfor kan lagre-operasjoner bufres (benytte samlebånds-prinsipper), som igjen øker ytelsen.

Svakhet ved WC: Prosessoren må vente ved "acquire"-operasjoner, på at tidligere lagre/-"release"-operasjoner skal fullføres. Den må også vente ved første hente-operasjon etter en "release"-operasjon. Derfor vil WC være dårlig sammenlignet med PC for fin-struktur (Eng: fine-grain) beregninger med hyppige synkroniseringer.

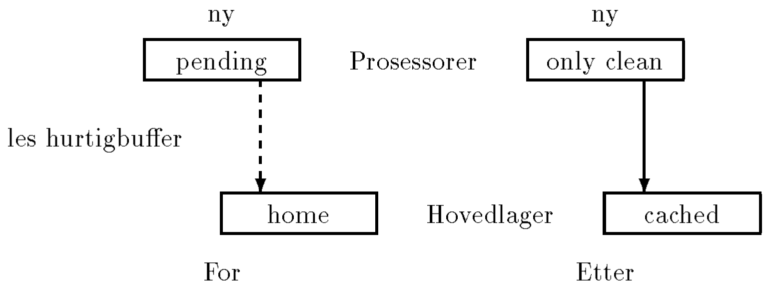
Fordel ved RC: Svakhetene ved WC ved fin-struktur beregninger fjernes med RC, siden RC ikke blokkerer prosessoren ved hente/"acquire"-operasjoner når en lagre/"release"-operasjon skal fullføres. Uavhengige synkroniseringer trenger ikke vente på at de andre skal fullføres. Slik kan man oppnå en høyere grad av parallellitet.

Svakhet ved RC: RC trenger mer kompleks maskinvare- og programvare-støtte. Kompilator-støtte og spesielle språk-konvensjoner er nødvendig for å generere koden som trengs for denne modellen.

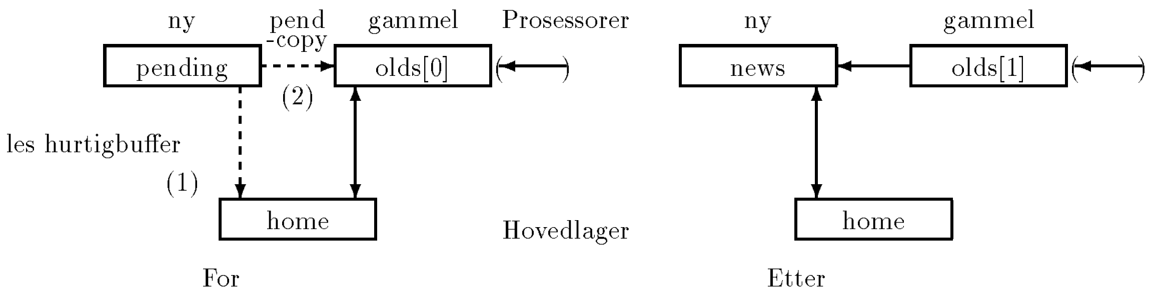
Oppgave 2:

a) • Alle hurtigbuffere er initielt ugyldige, og alle hovedlager-blokker er markert "uncached". Den delte listen lages i den hurtigbufferen hvor et innslag er endret fra ugyldig til en vente-tilstand ("pending"). Når en les-hurtigbuffer transaksjon blir startet av en prosessor og dirigert til minne-kontrolleren, blir minne-tilstanden endret fra "uncached" til "cached" og de ønskede data blir returnert.

Innslaget i hurtigbufferen til den prosessoren som ønsker data, blir deretter endret fra "pending"-tilstanden til en "only-clean"-tilstand. Oppbyggingen av den delte listen er vist i figuren under. Multipple forespørsler kan genereres samtidig, men blir prosessert sekvensielt av minne-kontrolleren.



• For senere minne-aksesser er hovedlager-tilstanden "cached", og hurtigbuffer-hodet i den delte listen har sannsynligvis "dirty" data. Som vist i figuren under, vil en ny



forespørsel først rette dens les-hurtigbuffer transaksjon til hovedlageret, men vil motta en peker til hurtigbufferen som har konsistente data.

En annen hurtigbuffer-til-hurtigbuffer transaksjon, kalt "prepend", blir flyttet fra hurtigbuffer A til B. Hurtigbuffer B setter sin bakoverpekende peker til å peke på hurtigbuffer A og returnerer de ønskede data. De halv-trukne linjene viser transaksjoner mellom en prosessor og hovedlager eller en annen prosessor. De hel-trukne linjene viser delt liste-pekere. Etter transaksjonen blir den innsatte hurtigbufferen A det nye "hodet", og det gamle hodet (B) vil følge A i listen.

b) Sammenlignet med en bakplan-buss, vil en lenket katalog gi større båndbredde og bedre mulighet for skalering. Kostnaden reduseres siden man ikke trenger buss-snusings-kontrollere. Den tillater et invaliderings-signal å bli sendt til spesielle prosessorer i stedet for å kringkaste signalet til alle prosessorene. Det kan imidlertid ta lengre tid før signalet når fram til de riktige prosessorene.

Fordelen med en lenket katalog sammenlignet med en "full-map" katalog er at man sparer plass når katalog informasjonen skal lagres. Vi antar at det er P prosessorer i systemet og at det er M minne-blokker. M vil typisk være proporsjonal med P. Hvis en "full-map" katalog brukes trengs en bit for å angi om en prosessor har en spesiell blokk i hurtigbufferen. Det totale antall slike bit blir $O(MP) = O(P^2)$. På den annen side hvis en lenket katalog blir brukt, trenger hver blokk bare å opprettholde en peker til den første prosessoren som har den respektive blokken. Hver peker trenger $O(logP)$ bit, slik at det totalt trengs $O(MlogP) = O(PlogP)$ bit. Dette gjør SCI mer skalerbar enn en "full-map" katalog.

En lenket katalog har også to ulemper sammenlignet med en "full-map" katalog. For det første vil tiden det tar for å sende et ugyldiggjør-signal til alle prosessorene som har en kopi av en minne-blokk som ikke lenger skal være gyldig, være større når antall prosessorer blir stort. Grunnen til det, er at i en "full-map" katalog kan ugyldiggjør-signalet sendes i parallell til alle prosessorene, mens i en lenket katalog må signalet propageres gjennom lenken, noe som tar tid. For det andre vil det være vanskeligere å konstruere protokollen. På grunn av større forsinkelse, kan "race" tilstander lettere oppstå. Dette må man ta hensyn til under konstruksjonen av protokollen.

Oppgave 3:

a) En fin-struktur prosessor har typisk lite minne. Når stor-skala datamaskiner skal konstrueres, vil fin-struktur prosessorer passe bedre med fin-struktur programvare for parallellitet og har også lavere kostnader sammenlignet med middels-kornete prosessorer.

b) I en-prosessor systemer er det bare ett adresserings-rom. Mange programmer har blitt utviklet med tanke på dette. Ett enkelt, globalt adresserings-rom tilbyr en kontinuitet og kan forenkle system-utviklings-prosessen ettersom programmererne ikke trenger å bry seg så veldig mye med meldings-overførings-mekanismer på individuelle maskiner. Det forenkler også oppdeling av data og dynamisk last-balansering og øker portabilitet til programmer mellom maskiner med forskjellig arkitektur.

c) På grunn av høy synkroniserings-kostnad, trenger grovkornet parallellitet store mengder beregninger, slik som mange iterasjoner, til hver prosessor. Dermed blir ikke lav-nivå parallellitet fullt utnyttet for individuelle iterasjoner og instruksjoner. Med tanke på skalerbarhet, når tallet på prosessorer øker, vil det være viktig å ta hensyn til slik lav-nivå parallellitet for å redusere beregnings-tiden og øke ytelsen. Dette taler for fin-struktur parallellitet i stedet for middels- og grovkornet parallellitet.