Minnehåndtering i operativsystemer

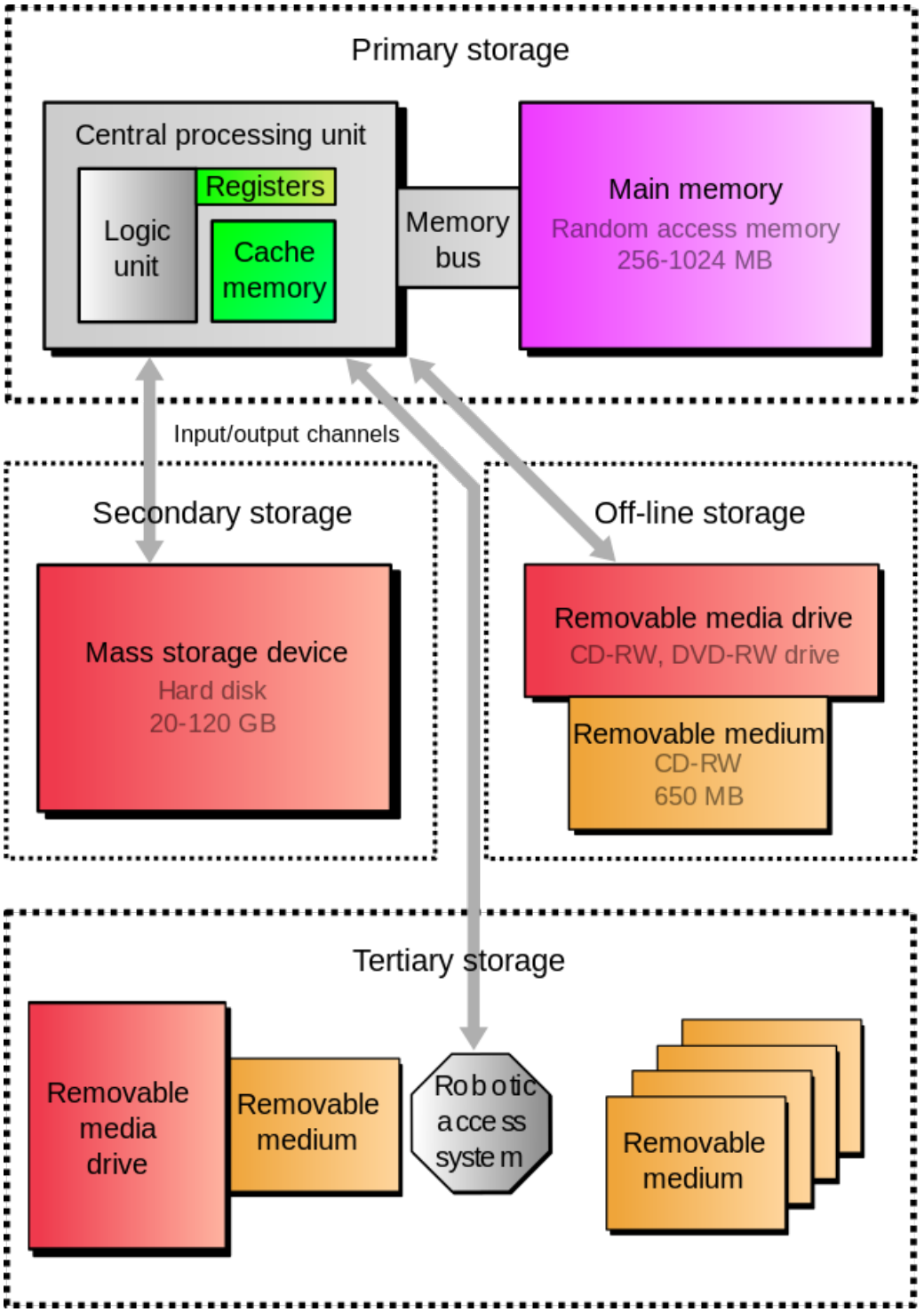
Minnehåndtering?

- Minne er en *begrenset* ressurs i datamaskinen
- Tilgjengelig minne må fordeles til *prosessene* som OS-et håndterer, på en korrekt og “rettferdig” måte
- Minnet må beskyttes, men samtidig kunne deles
- Minnehåndtering utføres av en egen del av OS som har ansvaret for maskinens primærminne:

  Memory Management System (MMS)

Hva er det som skal “håndteres”?

- MMS håndterer bare maskinens *primærminne*
- Aka RAM, (hoved)minne, (main) memory, internminne
- RAM er *direkte* tilgjengelig for CPU
- CPU leser *instruksjonene* som skal utføres fra RAM
- Alle *data* som CPU jobber med leses fra/lagres i RAM
- Primærminnet inkluderer *ikke* cache, ROM, grafikk-RAM, RAM-disker, cache-RAM i disk-kontrollere etc.


Primary storage
Central processing unit
Logic unit
Registers
Cache memory
Memory bus
Main memory
Random access memory
256-1024 MB
Input/output channels
Secondary storage
Mass storage device
Hard disk
20-120 GB
Off-line storage
Removable media drive
CD-RW, DVD-RW drive
Removable medium
CD-RW
650 MB
Tertiary storage
Removable media drive
Removable medium
Robotic access system
Removable medium

Minne: Adresserbare lagerceller

- Lagrer data i (tradisjonelt 8-bits) *minneceller*
- Hver celle har en (heltalls)adresse som kan brukes for å lese verdien eller lagre en verdi
- Lesing og skriving skjer ved å overføre data på databussen, mellom cellen og et register i CPU
- Cellene ligger oftest i “banker” med minnebrikker på hovedkortet


elixir
M2U51264DS8HC3G-5T
512MB DDR-400MHz-CL3
PC3200U-30331
Warranty Void If Removed
0542.MN05A1107A.5YM0410N08.XX.CN
elixir
M2U51264DS8HC
512MB DDR-40
PC3200U-303
Warranty Void
0542.MN05A11

Hvem gjør minnehåndteringen?

- **Operativsystemets programvare** (MMS) håndterer fordelingen av minne mellom prosesser
- **Maskinvare** for minneaddressering internt i CPU (memory management unit / MMU) sørger for riktig lesing/skriving i RAM
- **Run-time systemer** (f.eks. JVM) holder rede på variablene som brukes i et program som er skrevet i høynivå programmeringsspråk (f.eks. Java).

Hvorfor er minnehåndtering viktig?

- Uten god minnehåndtering “virker ikke” maskinen:
  - Feil i minnet gir bugs som er “umulige” å finne
  - Dårlig minnehåndtering gir en treg maskin
  - Usikre minnesystemer åpner for virus og malware
  - “Alt” som en *programmerer* gjør handler om minnehåndtering – både koden og dataene ligger i RAM under kjøring av programmet
- Moderne OS har derfor svært *komplekse* og *kompliserte* systemer for å håndtere minnet

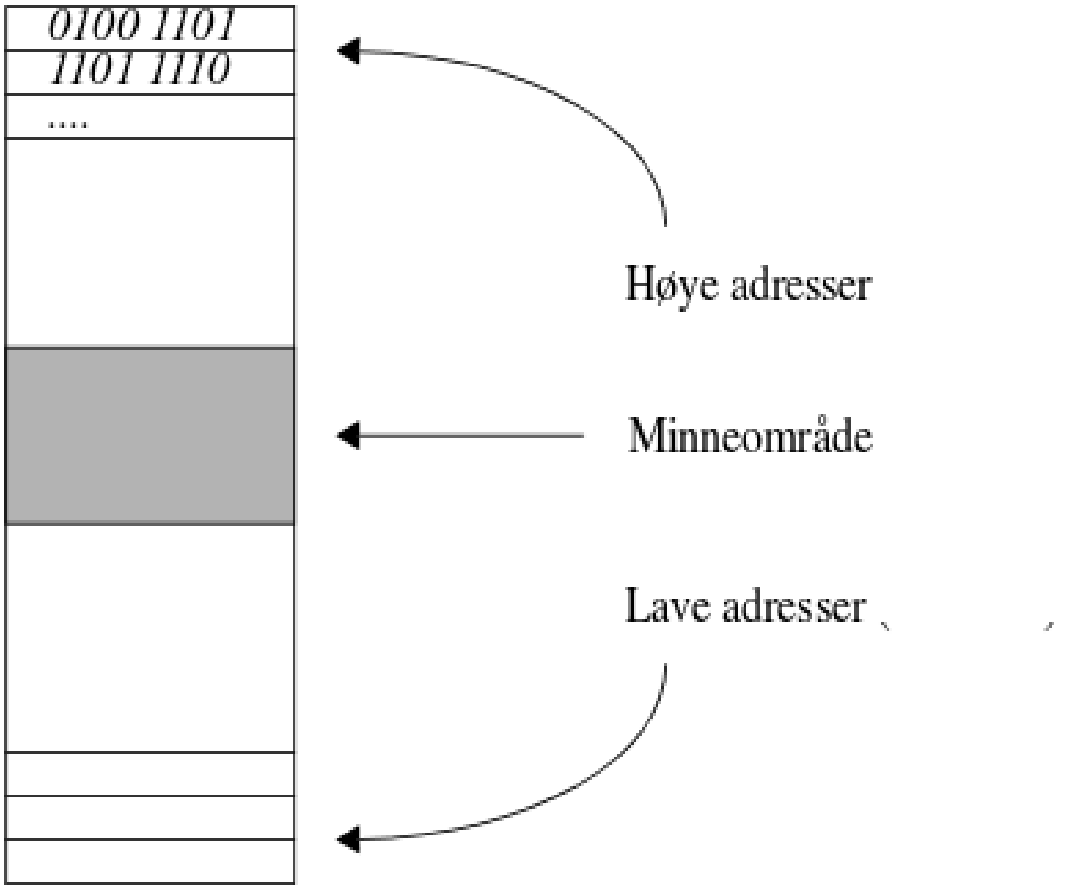
Hovedoppgaver i minnehåndtering

- Holde rede på status (tildelt/ledig) for minnecellene
- Fordele RAM mellom prosessene:
  - *Hvem* skal få tildelt minne?
  - *Når* skal en prosess få tildelt minne?
  - *Hvor mye* skal en prosess tillates å få?
- Finne “passende deler” av minnet ved tildeling
- Frigjøre minne som ikke lenger er i bruk
- Beskyttelse og deling av minne mellom prosesser

Forenkling av minnehåndtering *

- Bruker en *lineær modell* for å beskrive RAM
- Antar at hele primærhukommelsen ligger som en endimensjonal tabell – en “søyle” av minneceller
- Antar at vi kan adressere RAM omtrent som vi indekserer en tabell/array i f.eks. Java, fra lave til høye adresser
- Antar at prosesser får tildelt minneområder som er en *sammenhengende* del av denne “RAM-tabellen”

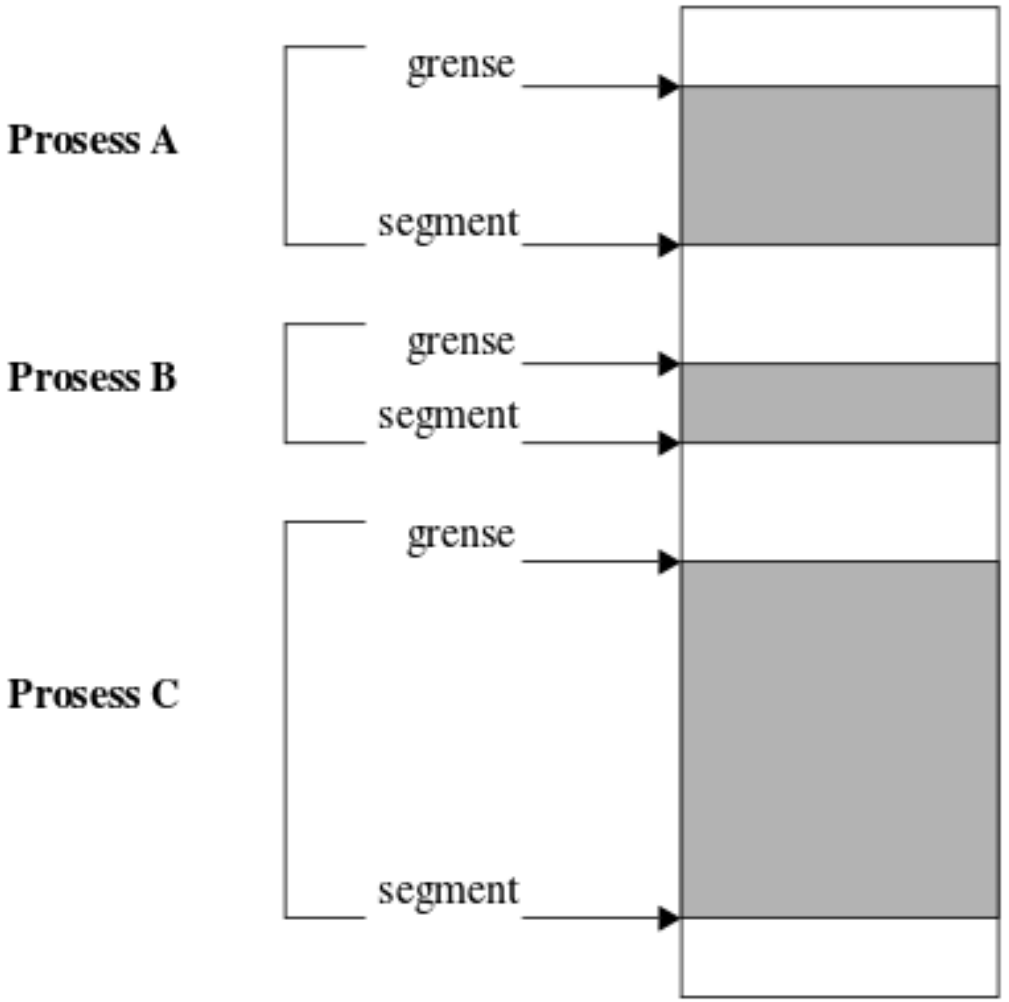
* Vi gjør dette for lettere å kunne forstå prinsippene. En godt fungerende MMS er *mye* mer komplisert både mht. algoritmer og håndteringen av maskinvare.



Minnet vist som en søyle av minneceller

Minneområdet til en prosess

- Anta at hver prosess får tildelt bare *ett* sammenhengende minneområde i RAM
- For hver prosess som kjører vil OS lagre informasjon om minneområdet i *registre* i CPU:
  - En *segmentindeks*, angir start på minneområdet
  - En *grense*, angir størrelse på minneområdet



Bruk av segment- og grenseregistre

Beskyttelse av minneområder

- OS bruker verdiene i segment- og grenseregistre for en prosess til å *beskytte* minnet:
  - Prosessen *nektes* tilgang til RAM utenfor eget område
  - Kan bare gis tilgang hvis *andre* prosesser tilllater det
- Registrene er *beskyttet* *
  - En prosess kan ikke endre data om eget minneområde
  - Kun OS har tilgang til segment- og grenseregistrene

* : Tidlige Microsoft-systemer hadde *ikke* beskyttelse av minneregistrene → *mye* systemfeil

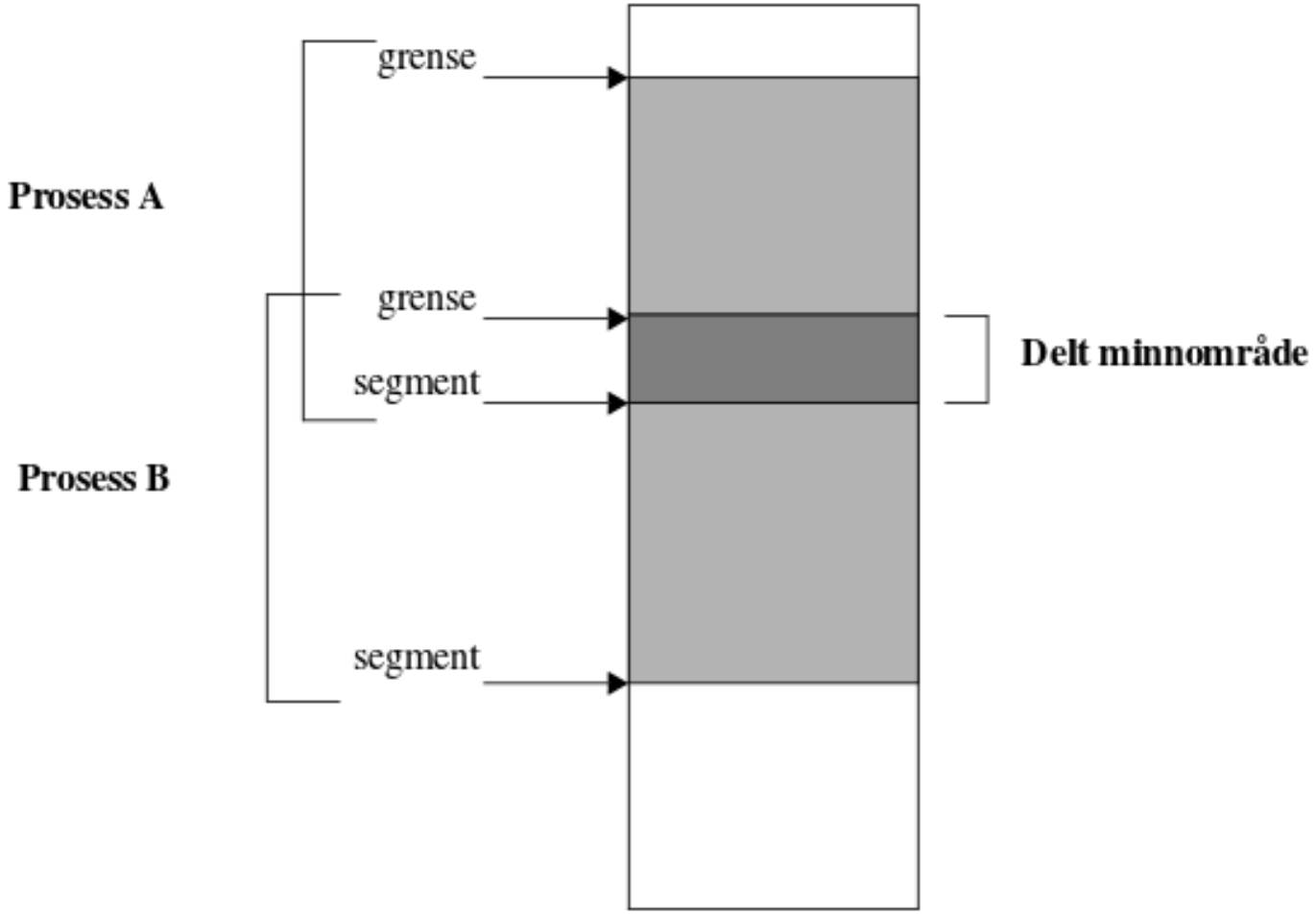
Deling av minne

To muligheter for deling av minne hvis hver prosess bare har ett sammenhengende minneområde:

1) Delvis overlappende minneområder:
   - Prosessene som skal dele minne har hver et *underområde* av RAM som er felles
2) Helt overlappende minneområder:
   - Prossessene som skal dele minne tildeles det samme minneområdet og deler hele dette området

Delvis overlappende minneområder

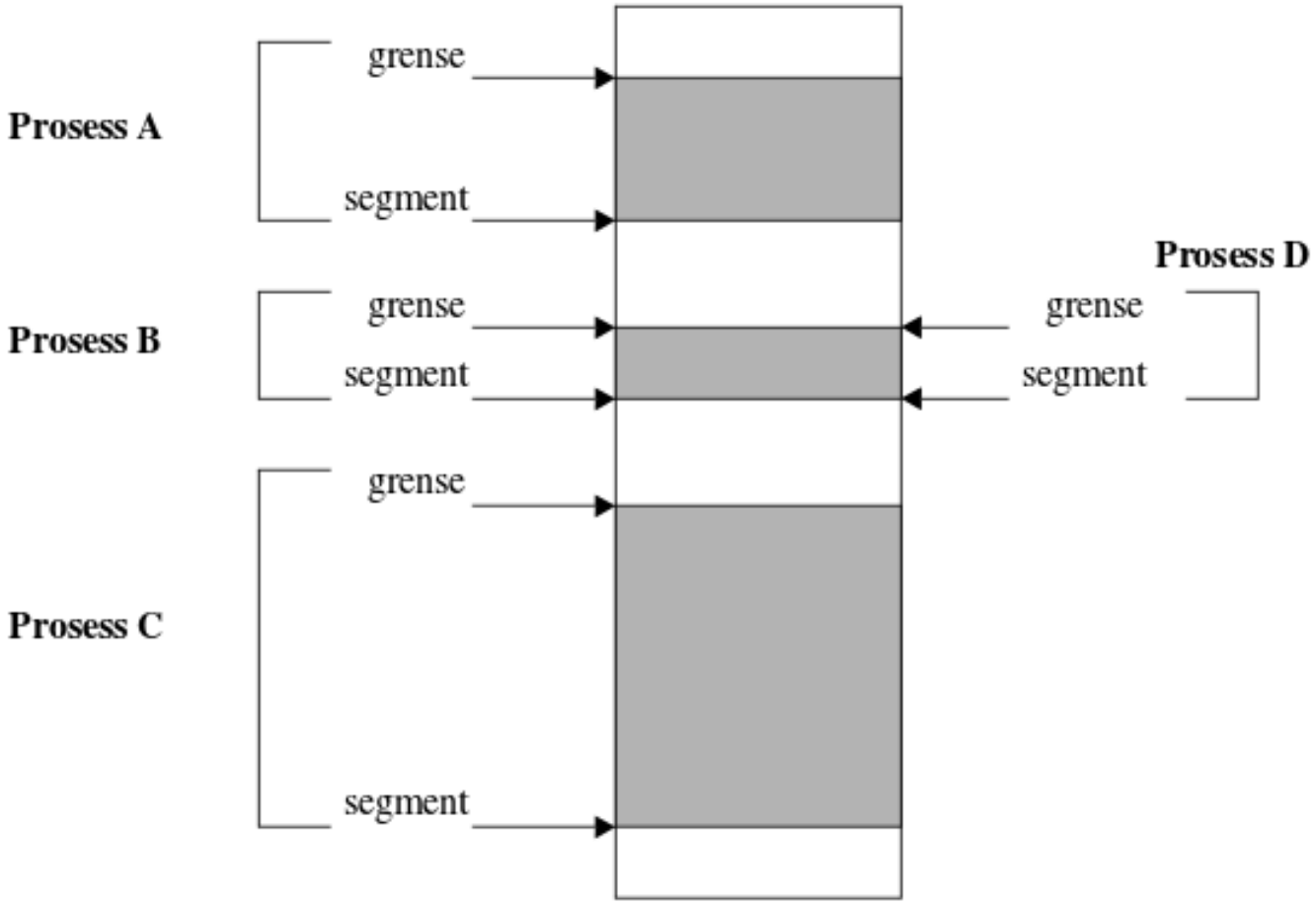
- Fungerer bare for *to* prosesser
- Ved *tre eller flere* prosesser må minst en av prosessene dele *hele* sitt minne-område
- Brukes ikke i praksis



Deling av minne gjennom delvis overlappende segmenter

Helt overlappende minneområder

- Prosessene deler hele minne-området
- Prosessene har identisk innhold i grense- og segmentregistre
- Brukes bl.a. for alle *trådene* i en prosess



Deling av minneområde gjennom felles segmenter. Prosess B og D har i dette tilfellet like verdier i segment- og grenseregisteret.

Deling av minne*segmenter*

- De fleste OS bruker *segmentert minnehåndtering*
- Hver prosess kan tildeles *flere* mindre minneområder
- Hvert minneområde kalles for et minne*segment*
- Segmentene er *sammenhengende*, mens *hele* minneområdet til prosessen er spredt i RAM
- Et eller flere av de tildelte *segmentene* kan *deles* med flere prosesser, mens andre segmenter kan *beskyttes* mot lesing og/eller skriving