Runtimesystemer - III

- Dynamisk lager-allokering/når trenger vi en heap
  - For objekter/recorder som allokeres dynamisk (new) og som man kan ha pekere til
    - Gjelder ofte også array-objekter
  - Under visse forhold også aktiveringsblokker for prosedyrer
    - Simula, pga korutiner
    - Hvis man har
      - Prosedyre-variable
      - Nestede prosedyrer

Problemer med frie pekere

```
int * dangle(void)
{ int x;
  return &x;}
```

- Dette og lignende problemer (spesielt i funksjonelle språk, men også for korutiner i Simula) gjør det nødvendig å legge aktiveringsblokker for prosedyrer på heapen
- Altså: dynamisk prosedyre allokering/deallokering- ikke stakk-basert

```
typedef int (* proc)(void);

proc g(int x)
{ int f(void) /* illegal local function */
  { return x;}
  return f; }

main()
{ proc c;
  c = g(2);
  printf("%d\n",c()); /* should print 2 */
  return 0;
}
```

Her defineres navnet 'proc'

Leverer en prosedyre

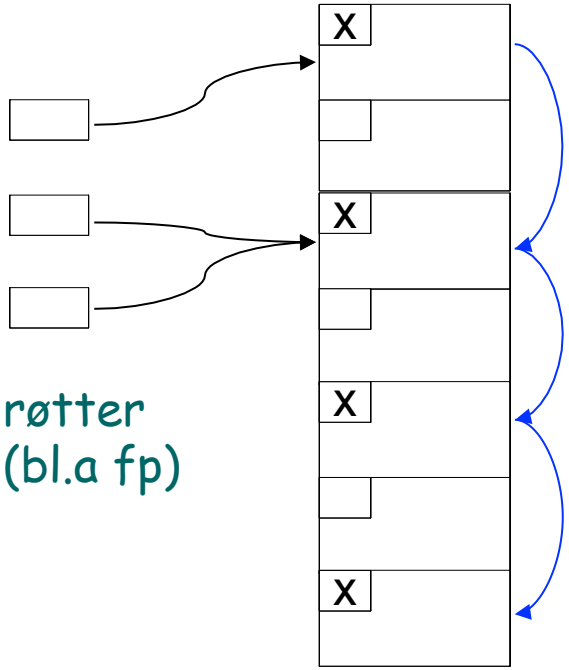
Prosedyren som leveres er lokal i g, og g() terminerer

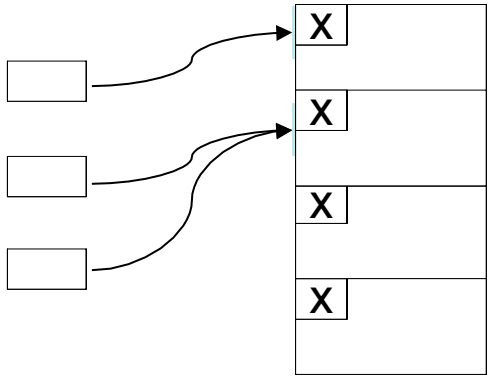
Noen alternativer

- Når blir plass ledig?
  1. Brukeren sier selv fra (alloc/free)
     - Kan lett bli feil og inkonsistenser
  2. Systemet finner automitisk hva som blir ledig
     - Krever ekstra administrasjon/informasjon
- Gjenbruk av frigjort plass
  1. Man flytter aldrig objekter
     – Fører lett til fragmentering
  2. Man flytter sammmen de objekter som skal bevares
     – Krever ekstra administrasjon/informasjon
     – Alle pekere til flyttede objekter må forandres
     – All ledig plass samlet i et område

Garbage Collection I

- Deler ut plass ukritisk så lenge det er plass. Når det ikke er plass mer tar vi en større opprydning
  - Starter altid med et fullt rekursivt gjennomløp, der vi finner alle objekter som kan nåes fra variable vi kan nå direkte (røtter)
  - Alle objekter som kan nåes merkes. Krever eget bit i hvert objekt.





  - NB (ikke nevnt i boken): Krever også at man kan finne ut hva som er pekere i et gitt objekt

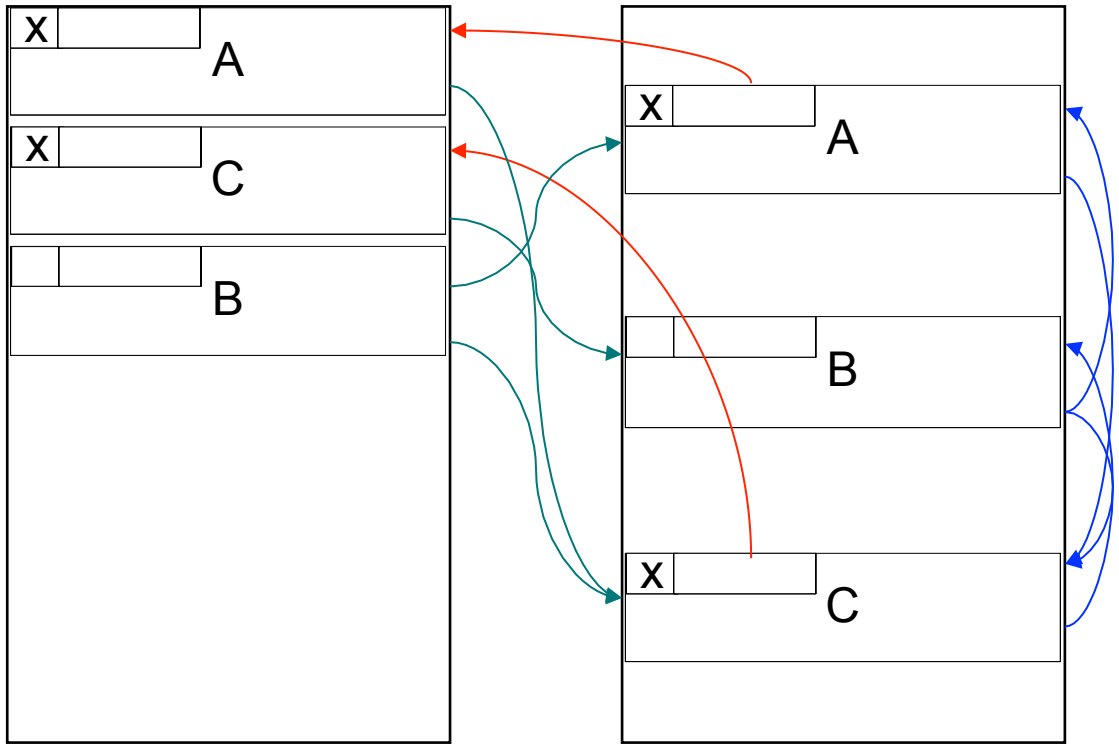
Garbage Collection II

- Etter merkingen (”mark”) gjøres et sekvensielt gjennomløp av lageret (”sweep”), der de umerkede objekter leveres tilbake
  - Må slå sammen ledig naboplass
  - Ledig plass holdes f.eks. i en eller fler frilister

- I stedet for ”sweep” kan man gjøre ”compaction”: flytte alle objektene tett sammen.
  - NB: Da må man også forandre alle pekere til det stedet objektet flyttes til

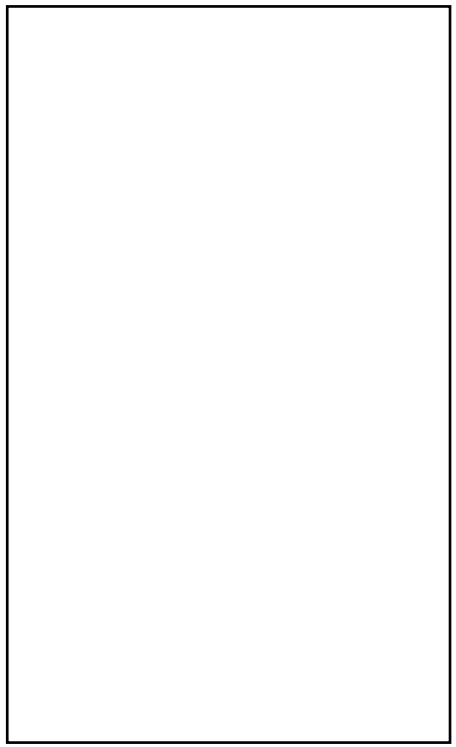
Garbage Collection III

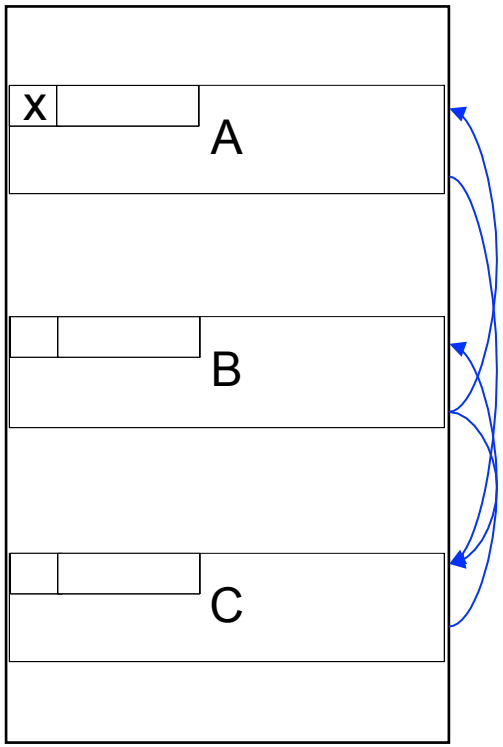
- To-delt lager
  - Deler plassen i to og bruker bare halvparten av gangen
  - ”mark” og ”compaction” kan da gjøres i ett rekursivt gjennomløp

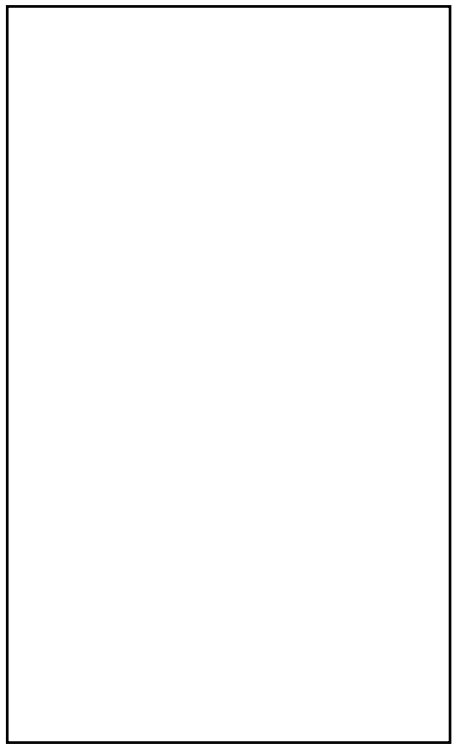


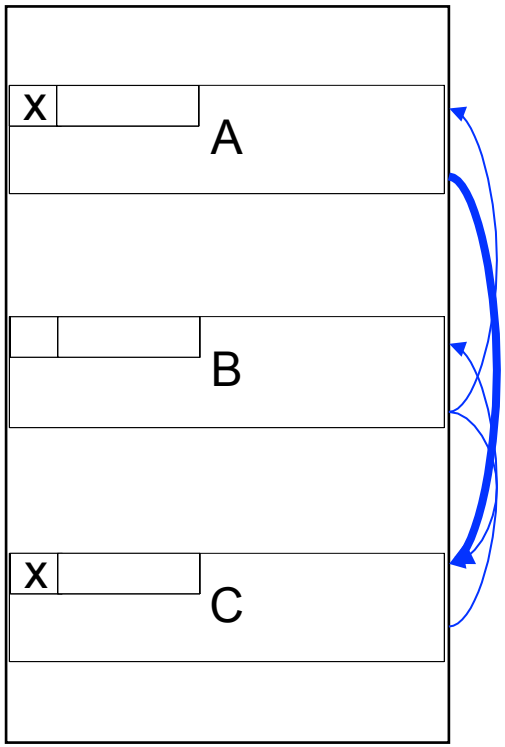
Hvert objekt må ha et ledig bit (”er flyttet”)

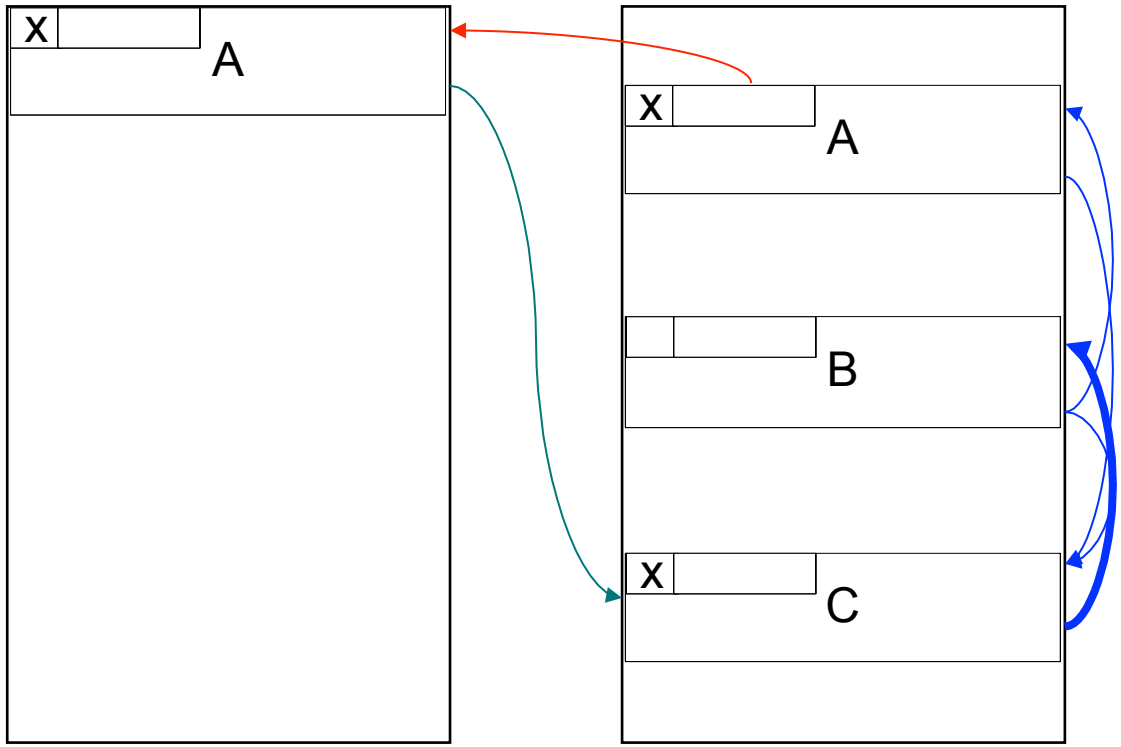
Da angir ”neste ordet” adressen det er flyttet til

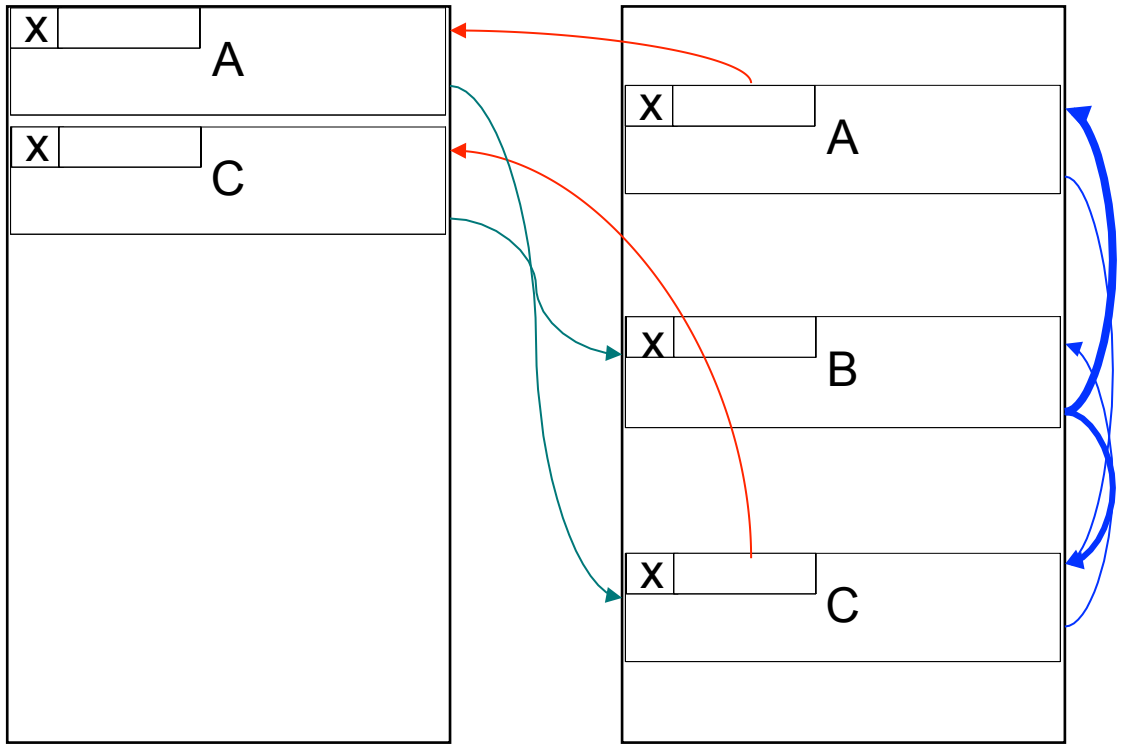


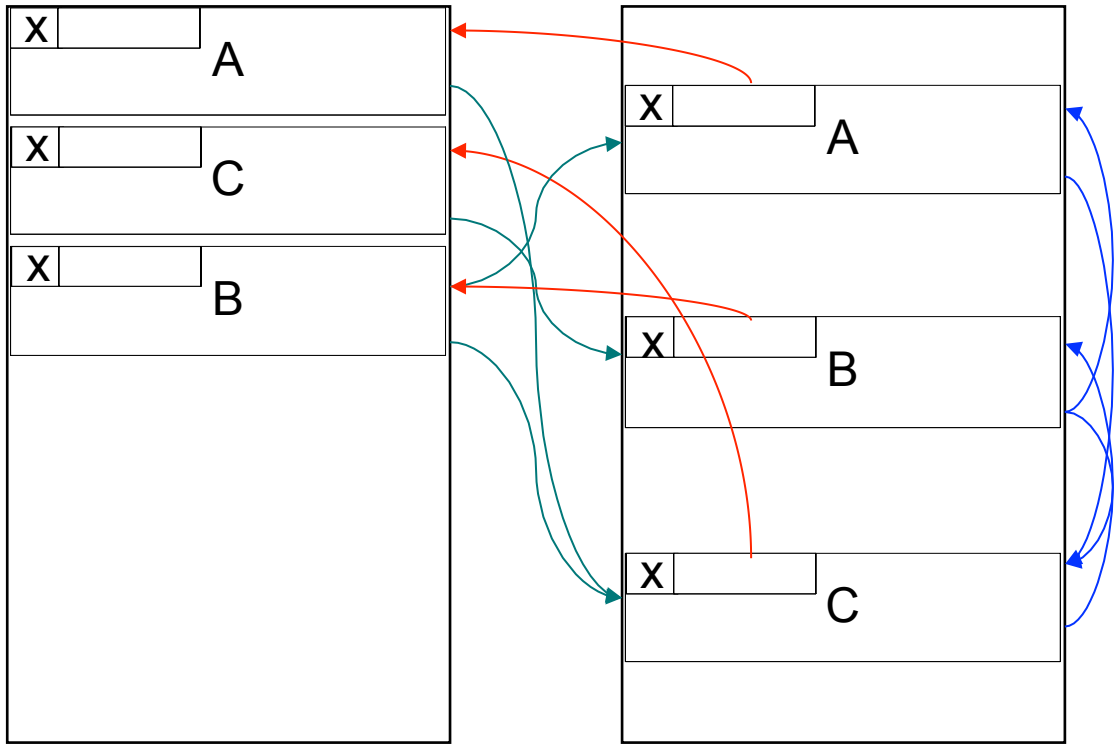

x
A
B
C




x
A
B
x
C


x
A
x
A
B
x
C


x
A
x
C
x
A
x
B
x
C


x
A
x
C
x
B
x
A
x
B
x
C