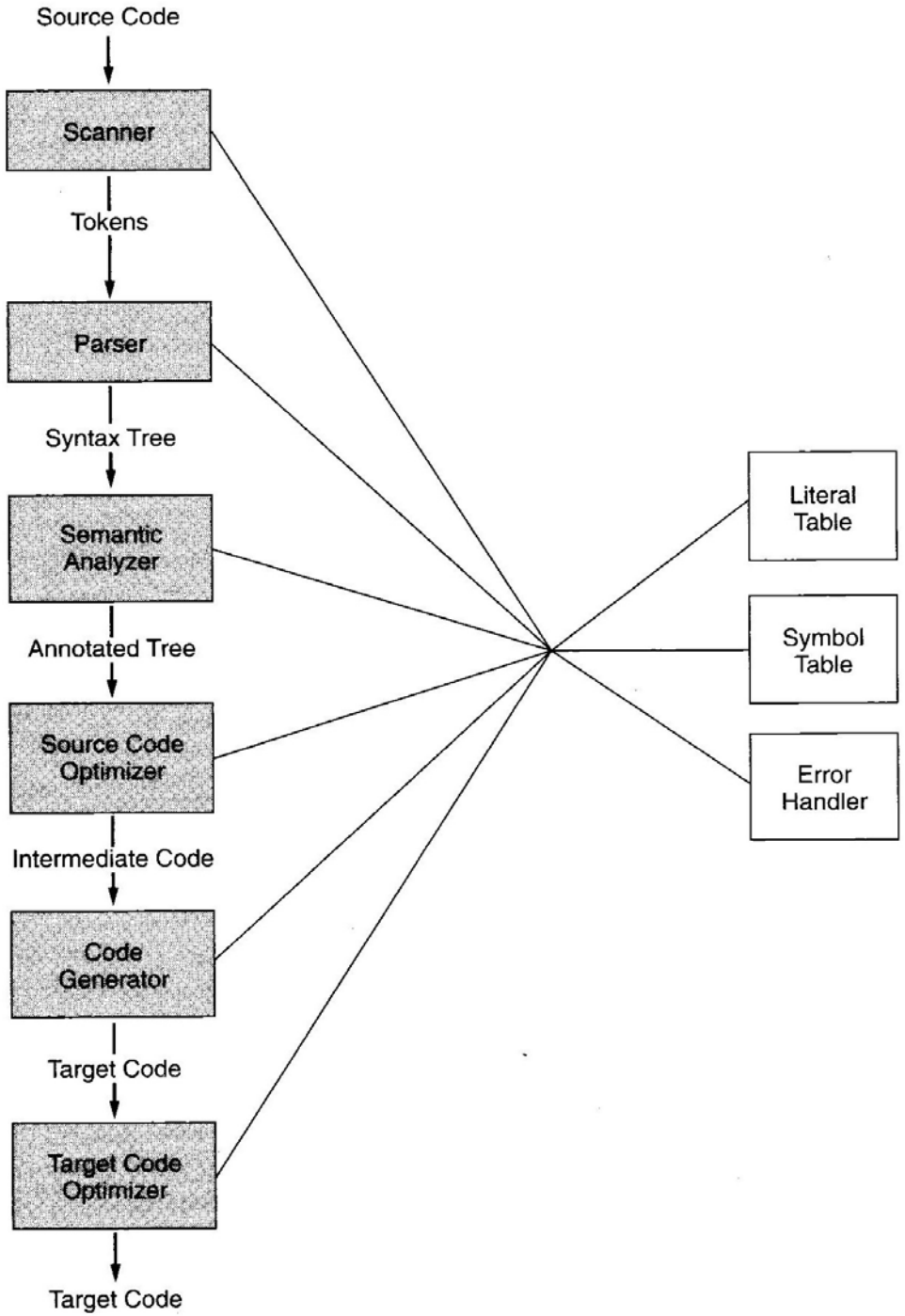
Runtime-systemer del III

Dynamisk minnehåndtering og objekt-orientering

Kapittel 7.4

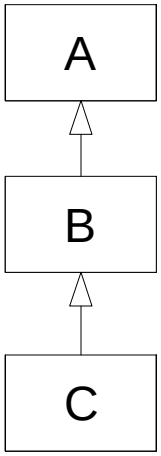


INF 5110 - 2013

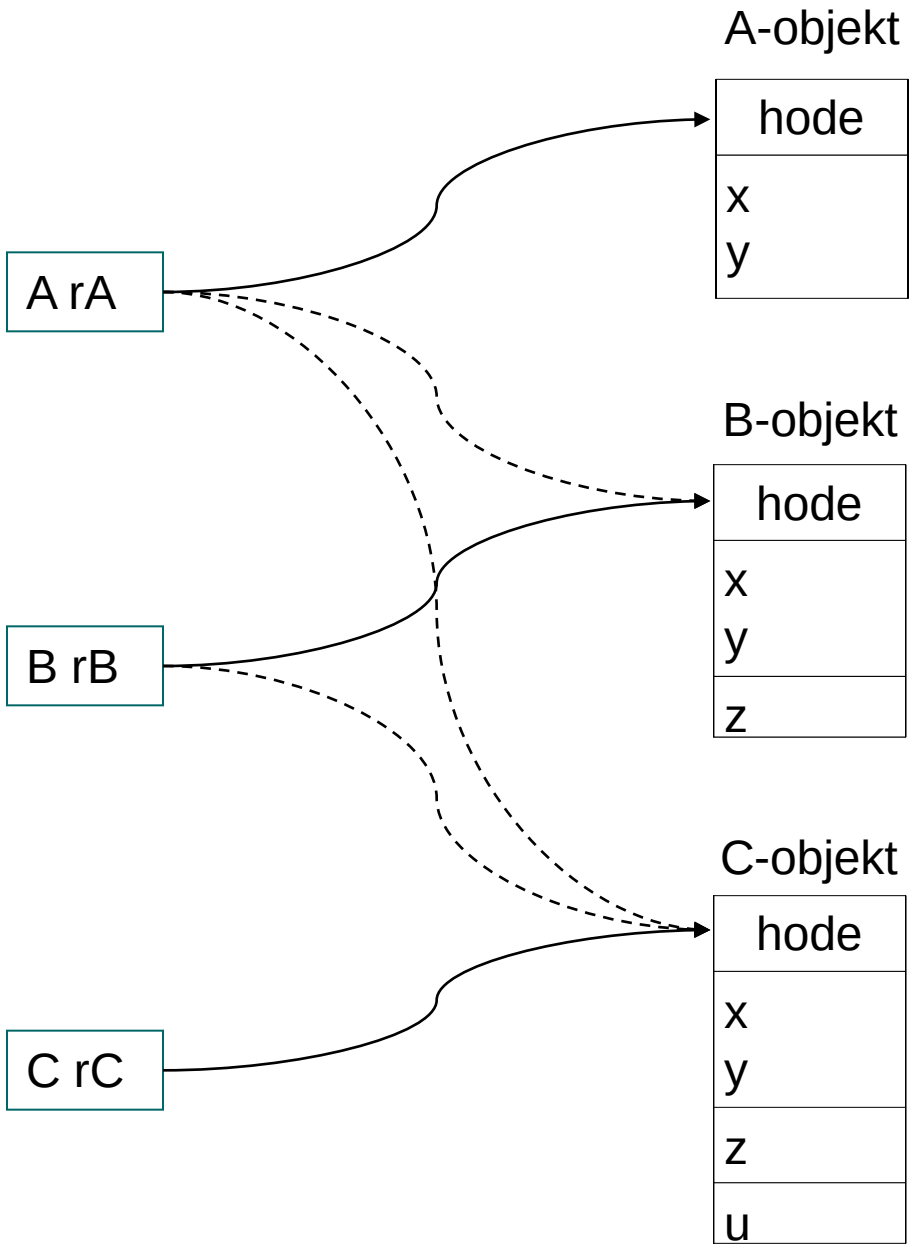
VIRTUELLE METODER

Objekt-orientering

- Klasser og subklasser
- Typede pekere
- Virtuelle og ikke-virtuelle metoder




```
class A {
  int x,y;
  void f(s,t) {...K...};
  virtual void g(p,q) {...L...}
}

class B extends A{
  int z;
  void f(s,t) {....Q...};
  redef void g(p,q) {...M...};
  virtual void h(r) {...N...}
}

class C extends B{
  int u;
  redef void h(r) {...P...}
}
```



Metode-kall – hvilken gjelder?

- Kall på ikke-virtuelle metoder (bruk pekerens type)
  - rA.f(1,2)
  - rB.f(1,2)
  - rC.f(1,2)
- Kall på virtuelle metoder (bruk objektets type)
  - rA.g(3,4)
  - rB.g(3,4)
  - rC.g(3,4)
  - rA.h(5)
  - rB.h(5)
  - rC.h(5)

```
class A {
  int x,y;
  void f(s,t) {...K...};
  virtual void g(p,q) {...L...}
}

class B extends A{
  int z;
  void f(s,t) {....Q...};
  redef void g(p,q) {...M...};
  virtual void h(r) {...N...}
}

class C extends B{
  int u;
  redef void h(r) {...P...}
}
```

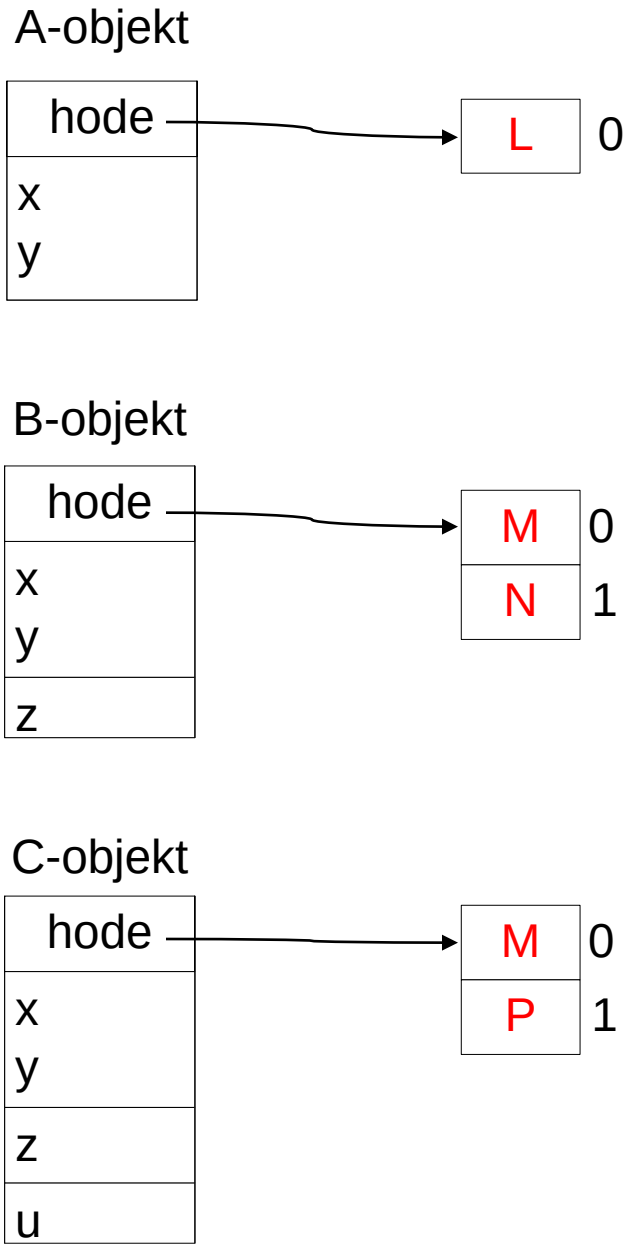
Implementasjon (typede pekere)

```
class A {
  int x,y;
  void f(s,t) {...K...};
  virtual void g(p,q) {...L...}
}

class B extends A{
  int z;
  void f(s,t) {....Q...};
  redef void g(p,q) {...M...};
  virtual void h(r) {...N...}
}

class C extends B{
  int u;
  redef void h(r) {...P...}
}
```

Kompilatoren vet:
g_offset = 0
h_offset = 1

rA.g(...) implementeres slik:

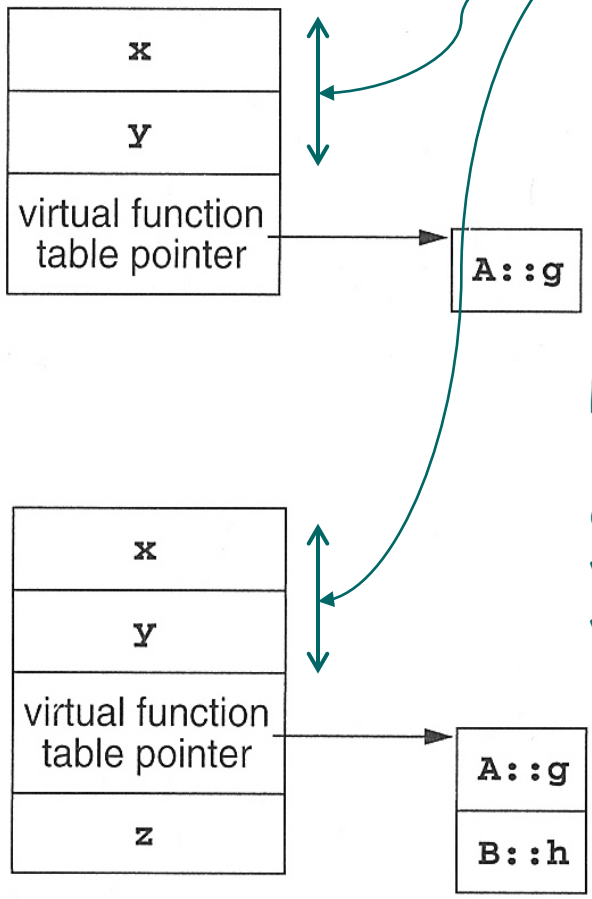
call(rA.virttab[g_offsett])



INF 5110 - 2013

Impl. av virtuelle metoder som i boken (C++)

```
class A
{ public:
  double x,y;
  void f();
  virtual void g();
};

class B: public A
{ public:
  double z;
  void f();
  virtual void h();
};
```



Virtuell-tabell pekeren har relativ-adresse som kan beregnes av kompilatoren

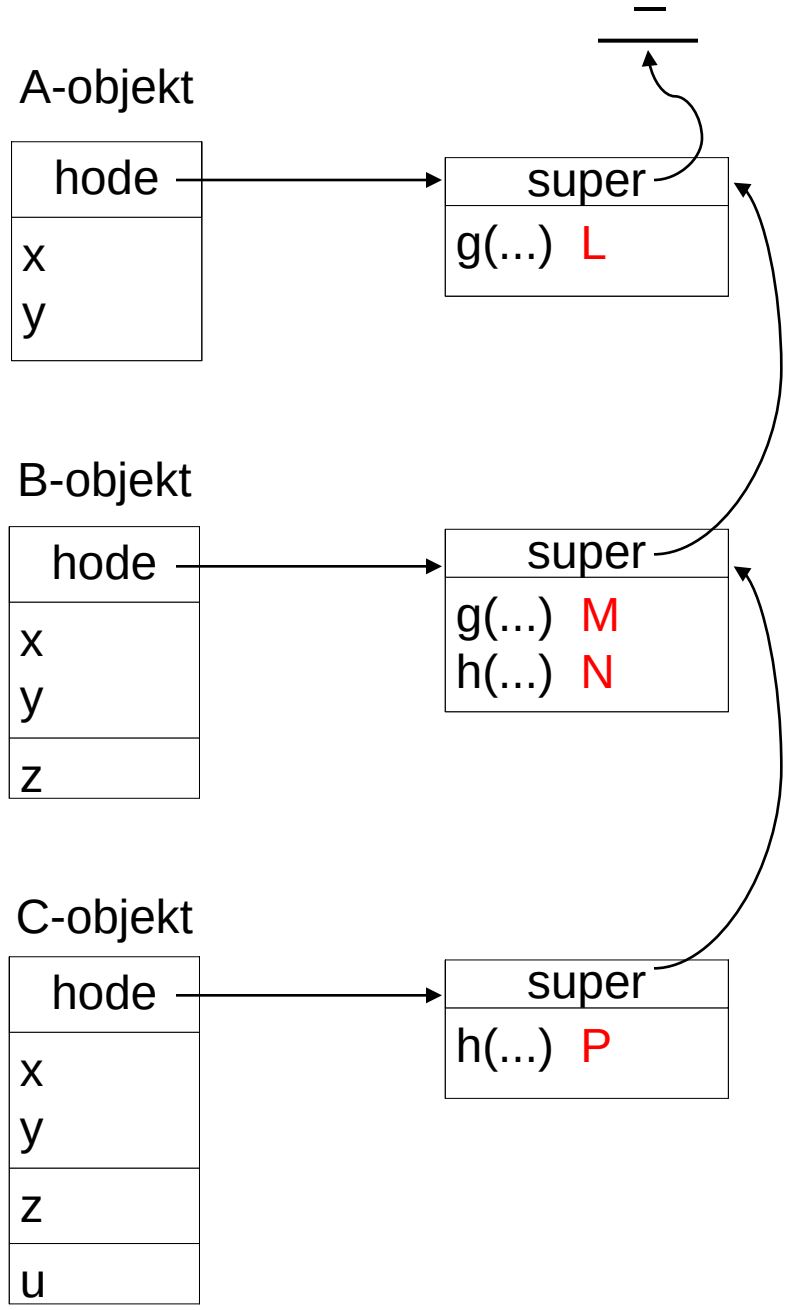
Pekeren til virtuell-tabellen ligger ikke først i objektet, da bare objekter med virtuelle funksjoner har en virtuell-tabell.

Utypede pekere (f.eks. Smalltalk)

- Ikke-virtuelle metoder finnes ikke
- Problem med virtuell-tabeller: Alle virtuell-tabeller måtte innholde alle metoder i alle klasser, altså for stor.
- I tillegg: I Smalltalk kan man legge til metoder underveis
- Derfor (antar at f er fjernet):

r.g(...) implementeres slik:

1. Gå til objektets klasse
2. Let etter 'g' ut gjennom superklassene



OPPGAVE 7.13 +
OPPGAVE 3B-E EKSAMEN 2005

LR-oppgave 1

Gitt følgende grammatikk:

L' $\rightarrow$ L
L $\rightarrow$ (LS)
LS $\rightarrow$ EL | LS EL
EL $\rightarrow$ atom | L

- Lag LR(0)-automaten.
- Avgjør om denne grammatikken er LR(0), eventuelt om den er SLR(1).

LR-oppgave 2

Gitt følgende grammatikk:

$S' \rightarrow S$

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

- Lag LR(0)-automaten.
- Avgjør om denne grammatikken er LR(0), eventuelt om den er SLR(1).

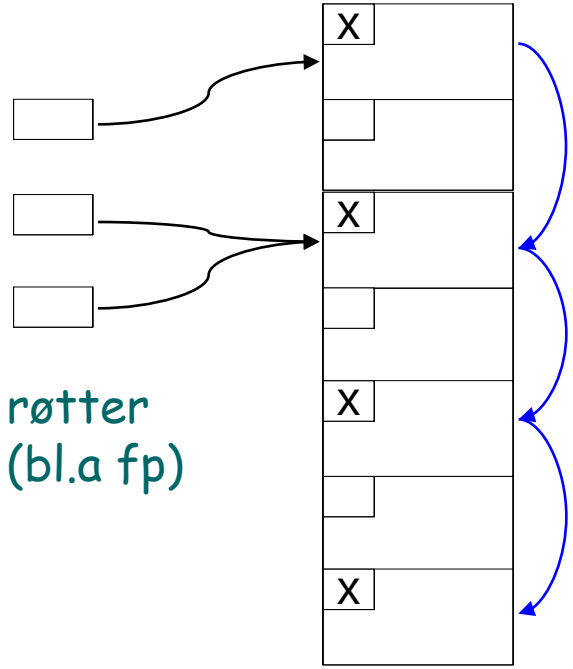
GARBAGE COLLECTION

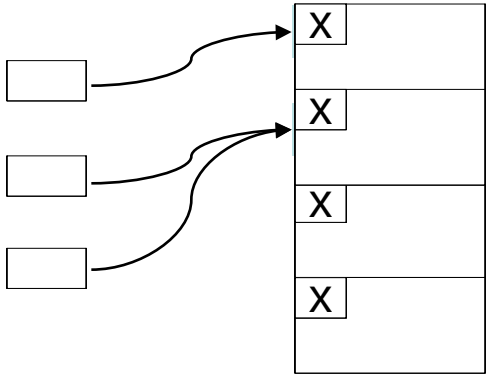
Noen alternativer

- Når blir plass ledig?
  1. Brukeren sier selv fra (alloc/free)
     - Kan lett bli feil og inkonsistenser
  2. Systemet finner automatisk hva som blir ledig
     - Krever ekstra administrasjon/informasjon
- Gjenbruk av frigjort plass
  1. Man flytter aldri objekter
     – Fører lett til fragmentering
  2. Man flytter sammen 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: mark (and sweep)

- 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

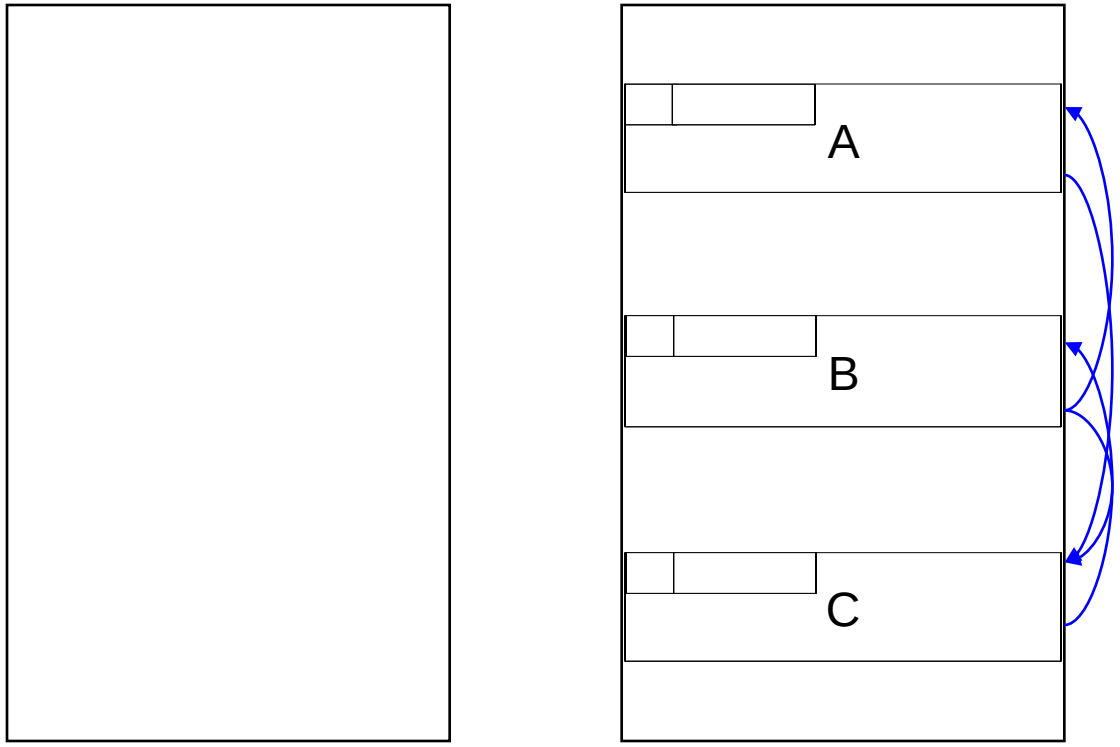
Garbage Collection: (mark and) sweep

- Etter merkingen gjøres et sekvensielt gjennomløp av lageret, der plassen til de umerkede objekter frigjøres.
  - 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: stop-and-copy (two-space)

- 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

Neste forelesning: Tirsdag 16. april (Sed?)

Start på kodegenerering (kap 8)