Prügikoristus

Prügikoristus

- Prügikoristus (garbage collection) vabastab automaatselt kuhjaobjektid, mida programm ei saa enam kasutada.
- Koosneb kahest osast:
  - elusate objektide ja prügi eristamisest (garbage detection);
  - prügi vabastamisest (garbage reclamation).
- Elusus (liveness) on globaalne semantiline omadus, mida üldjuhul pole võimalik täpselt kindlaks määrata.
- Prügikoristus kasutab ligikaudset kriteeriumit: objekt on elus, kui ta on "juurtest" (root set) lähtudes kättesaadav.

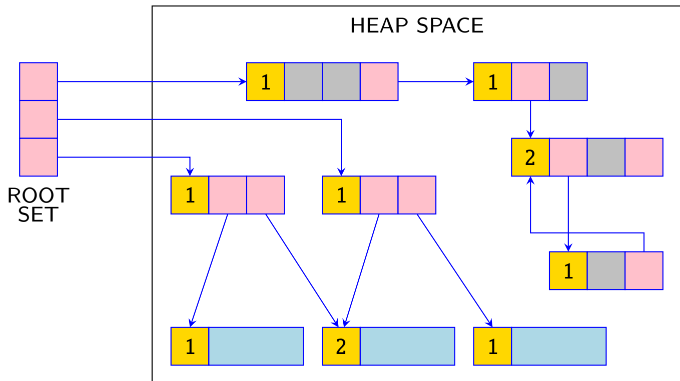
"Reference-counting" prügikoristus

"Reference-Counting"

- Iga objektiga seotakse loendur, mis näitab antud objektile viitavate viitade arvu.
- Loenduri muutmine toimub objektile viitade lisamisel/kustutamisel:
  - uue viida lisandumisel loendurit suurendatakse;
  - viida kustutamisel loendurit vähendatakse.
- Kui loendur on null, siis objekt vabastatakse:
  - vabastatud objekt lisatakse vabade objektide listi;
  - kõik viidad, kuhu see objekt viitas, kustutatakse.

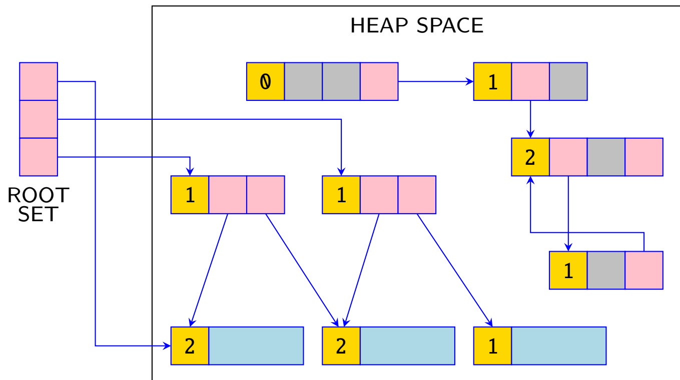
"Reference-counting" prügikoristus

Näide:



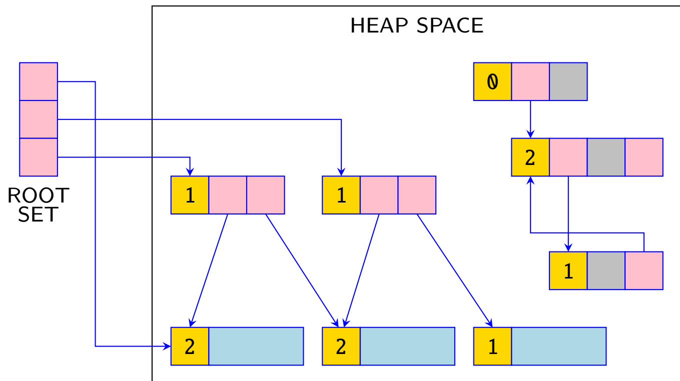
"Reference-counting" prügikoristus

Näide:



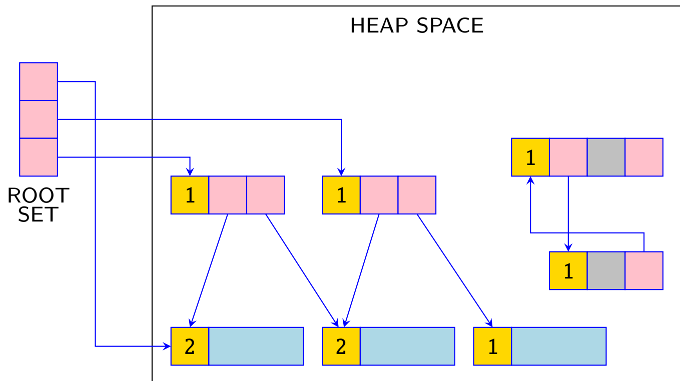
"Reference-counting" prügikoristus

Näide:



"Reference-counting" prügikoristus

Näide:



"Reference-counting" prügikoristus

Eelised

- ✔ on lihtne realiseerida;
- ✔ prügikoristusega seotud toimingud on hajutatud:
  - suhteliselt lihtsalt modifitseeritav inkrementaalseks;
- ✔ hea viitade lokaalsus:
  - muudetakse ainult lähte- ja sihtviitade loendureid;
- ✔ aeg objekti muutumisel prügiks ning tema vabastamise vahel (zombie time) on minimaalne;
- ✔ võimaldab lihtsasti realiseerida objektide "finaliseerimist".

"Reference-counting" prügikoristus

Puudused

- ✗ suhteliselt ebaefektiivne:
  - peab haldama loendureid isegi, kui prügi ei koristata;
- ✗ mälu fragmenteerumine:
  - analoogne teiste vabade objektide listil baseeruvate skeemidega;
- ✗ paljude väikeste objektide korral võib loendurite peale kuluda suhteliselt palju mälu;
- ✗ rekursiivne vabastamine on halvimal juhul tõkestatud kuhja suurusega;
- ✗ ei suuda vabastada kogu prügi:
  - tsüklilised andmestruktuurid.

"Mark-sweep" prügikoristus

"Mark-Sweep"

Toimub kahes faasis:

1. lähtudes juurtest, märgendatakse kõik kättesaadavad objektid;
2. teostatakse kuhja täisläbivaatus ning märgendamata objektid vabastatakse.

```
void gc () {
  foreach x ∈ Roots do
    mark (x);
  end;
  collect ();
}
```

"Mark-sweep" prügikoristus

Protseduur mark()

- Märgendab etteantud tipu ning seejärel rekursiivselt kõik kättesaadavad tipud.
- Rekursioon lõpeb, kui tipp on juba märgendatud või kui antud tipp sisaldab ainult baasväärtusi.

```
void mark (ref x) {
  if (x→mark == 0) {
    x→mark = 1;
    foreach y ∈ sons(x) do
      mark (y);
    end;
  }
}
```

"Mark-sweep" prügikoristus

Protseduur collect()

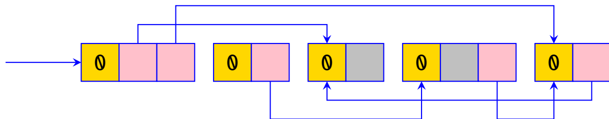
- Vaatab läbi kõik kuhjaobjektid ning lisab märgendamata objektid vabade objektide listi.

```
void collect () {
  freelist = NIL;
  foreach x ∈ objects() do
    if (x→mark == 0) {
      x→next = freelist;
      freelist = x;
    }
    else x→mark = 0;
  end;
}
```

"Mark-sweep" prügikoristus

Näide:

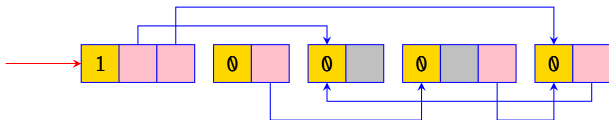
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

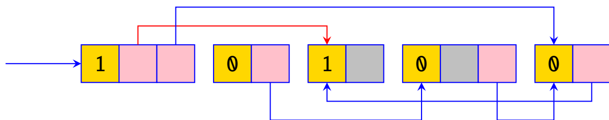
1. Rekursiivne märkimine:
2. Objektide vabastamine:



”Mark-sweep” prügikoristus

Näide:

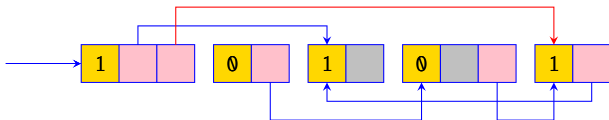
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

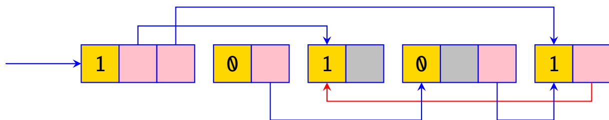
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

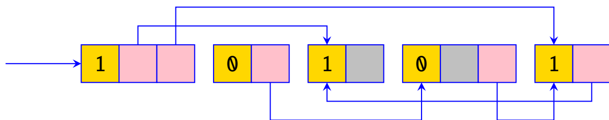
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

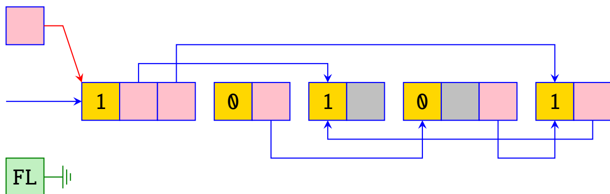
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

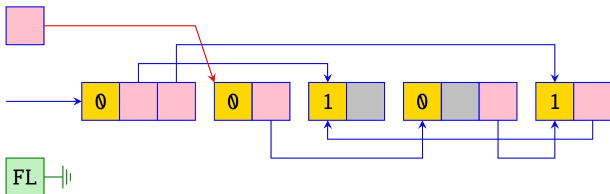
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

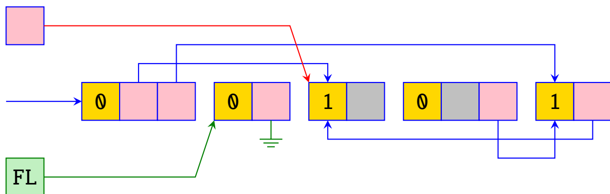
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

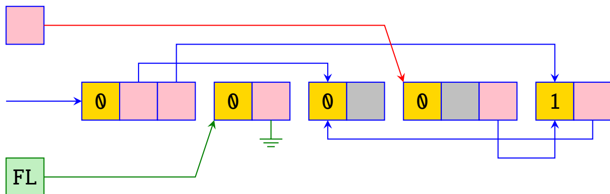
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

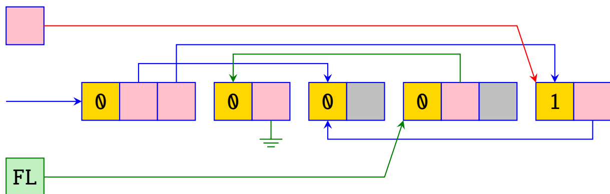
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

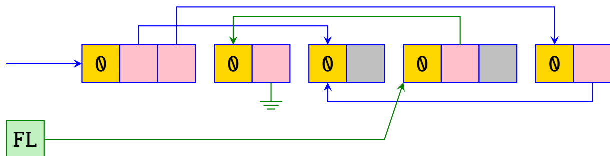
1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Näide:

1. Rekursiivne märkimine:
2. Objektide vabastamine:



"Mark-sweep" prügikoristus

Puudused

- ✘ Objektide märkimine toimub rekursiivselt.
  - Rekursioonimagasin kasvab halvimal juhul linearselt kuhja suurusega!!
  - Võimalik lahendus: Deutsch-Schorr-Waite'i "viitade pööramise" (pointer reversal) algoritm.
- ✘ Elusad objektid on kuhjas vabade mälupiirkondadega segamini.
  - Mälu fragmenteerumine.
  - Võimalik lahendus: "mark-compact" prügikoristus.

"Mark-compact" prügikoristus

"Mark-Compact"

Toimub kolmes faasis:

1. lähtudes juurtest, märgendatakse kõik kättesaadavad objektid (analoogselt "mark-sweep" prügikoristusega);
2. teostatakse kuhja täisläbivaatus ning arvutatakse märgendatud objektide uued aadressid;
3. nihutatakse märgendatud objektide uude asukohta ning viitadele antakse uued väärtused.

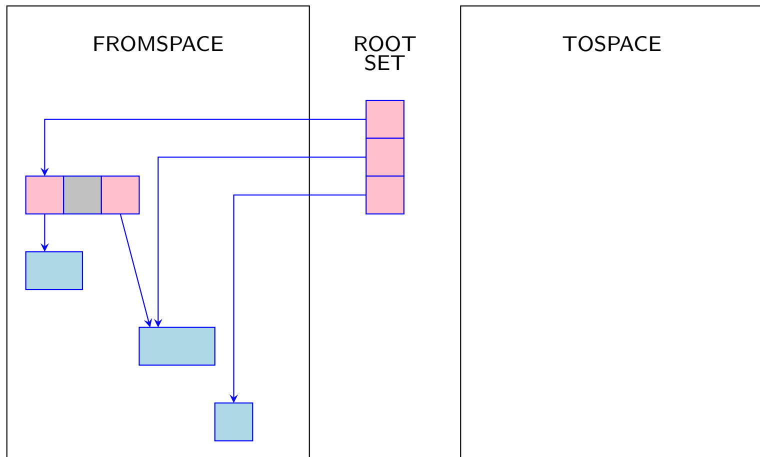
- ✔ Prügikoristuse tulemusena paikneb kogu vaba mälu kompaktselt kuhja lõpus.
- ✘ On suhteliselt aeglane, kuna kuhjaobjekte tuleb läbida palju kordi.

"Copying" prügikoristus

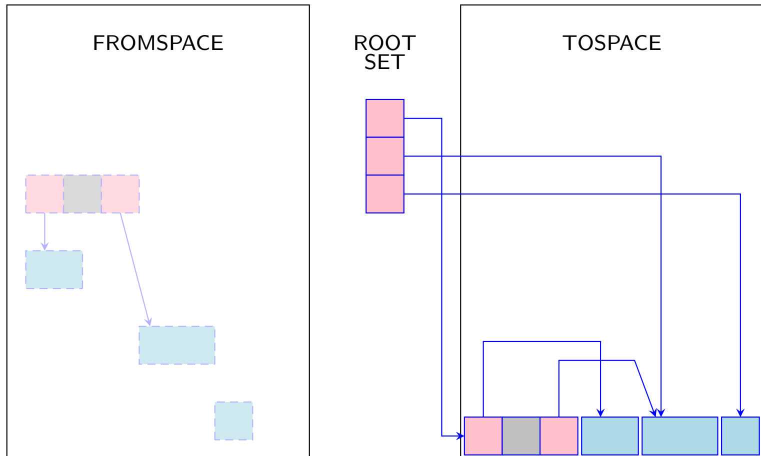
"Copying"

- Kuhi jagatud kaheks võrdseks alampiirkonnaks: FromSpace ja ToSpace.
- FromSpace on aktiivselt kasutatav mälupiirkond, kuhu salvestatakse uued objektid.
- Kui FromSpace saab täis, siis teostatakse prügikoristus:
  - elusad objektid kopeeritakse FromSpace'ist ToSpace'i;
  - FromSpace ja ToSpace vahetavad rollid (so. endine ToSpace muutub FromSpace'iks ja vastupidi).

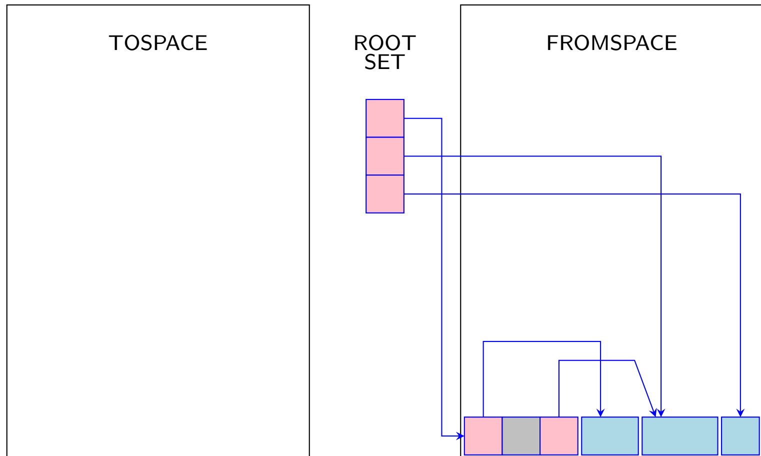
"Copying" prügikoristus



"Copying" prügikoristus



"Copying" prügikoristus



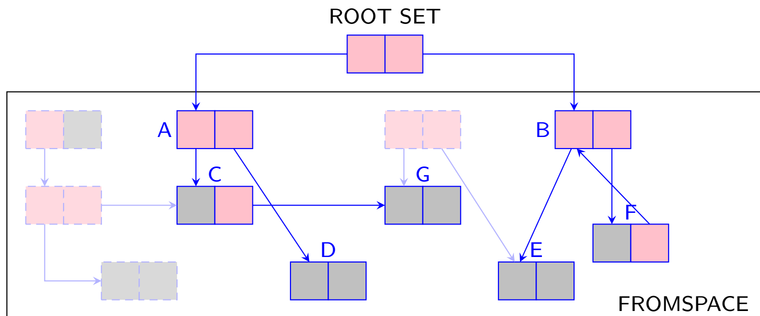
"Copying" prügikoristus

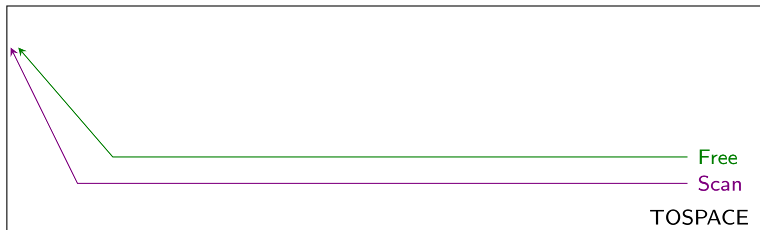
Cheney algoritm

Toimub kahes (üksteisega vahelduvas) faasis:

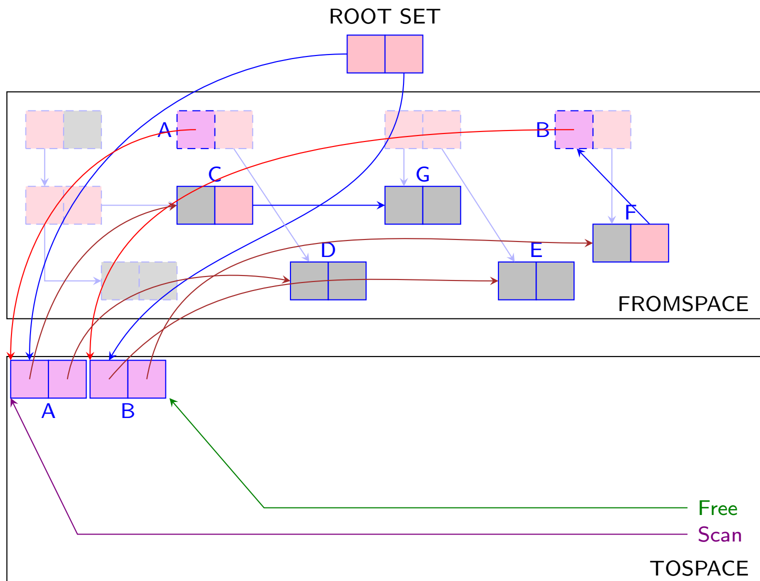
- esimese faasis (evacuate) kopeeritakse vahetult kättesaadavad objektid FromSpace'ist ToSpace'i, kasutatud viidad asendatakse viitadega vastavatele uutele objektidele ning vanade objektide asemele installeeritakse *"edasi toimetamise" viidad* (forwarding pointers);
- teises faasis (scavenge) skaneeritakse ToSpace'i kopeeritud objektid lineaarselt läbi ning kõik FromSpace'ist vahetult kättesaadavad objektid evakueeritakse ToSpace'i; kui evakueeritav objekt on juba varem kopeeritud, siis objekti uuesti ei kopeerita, vaid asendatatakse järgitav viit selle "edasi toimetamise" viidaga;
- protsess lõpeb, kui skaneerimisviit jõuab järele kuhja täitumisviidale.

"Copying" prügikoristus

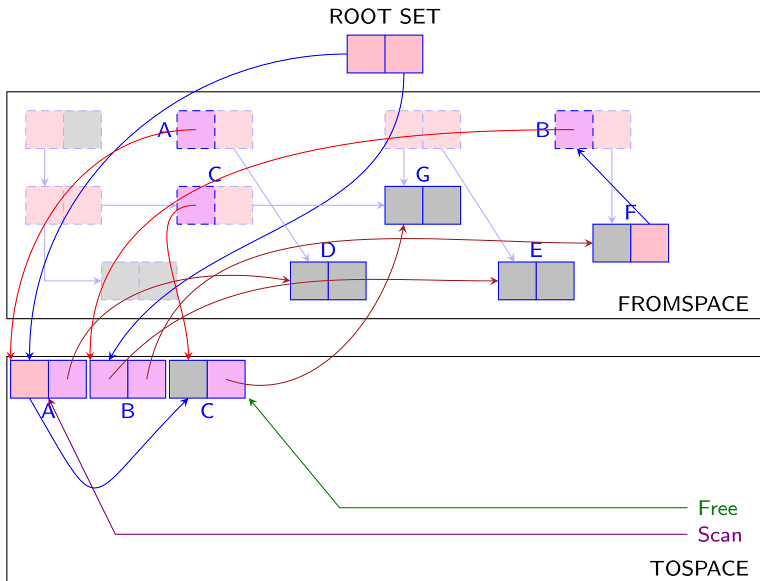




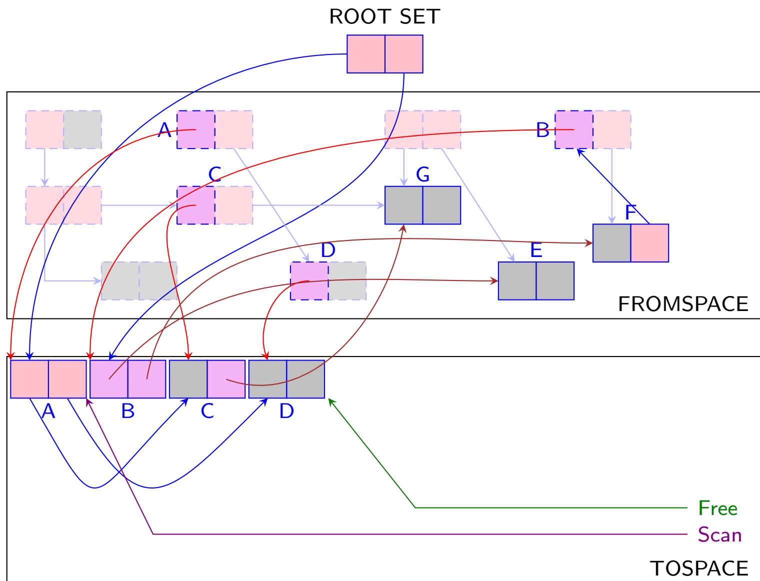
”Copying” prügikoristus



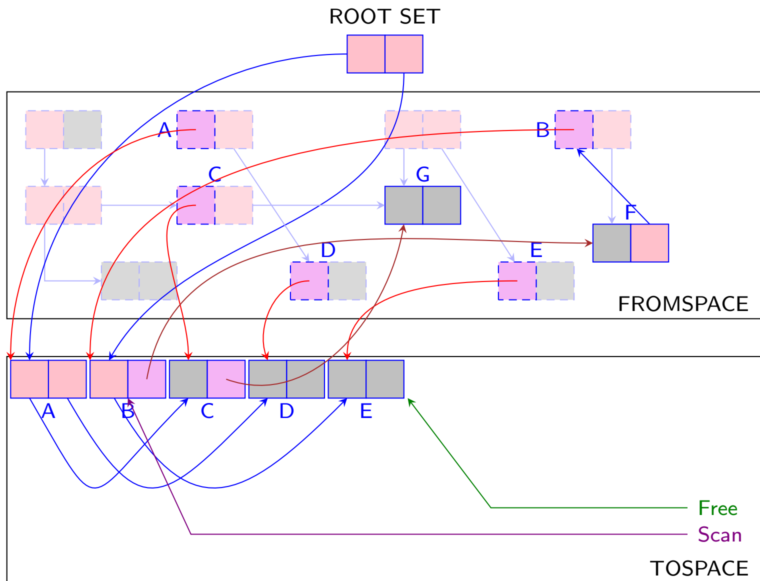
”Copying” prügikoristus



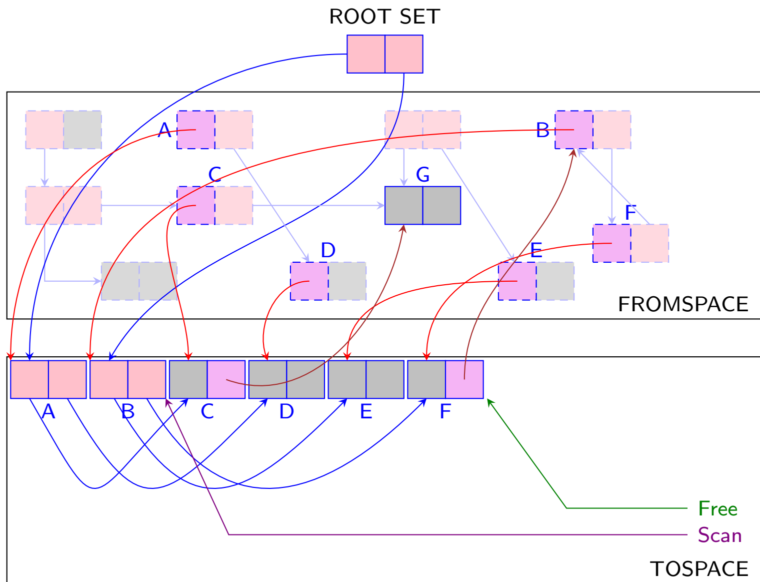
"Copying" prügikoristus



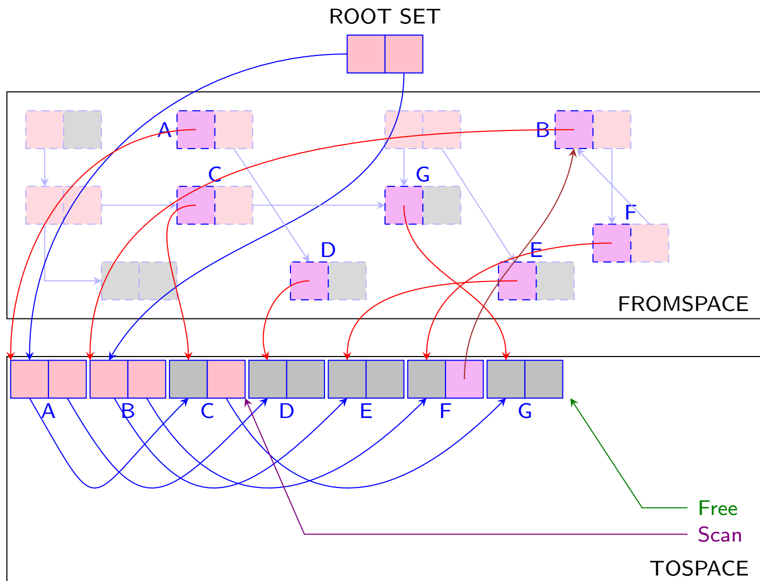
"Copying" prügikoristus



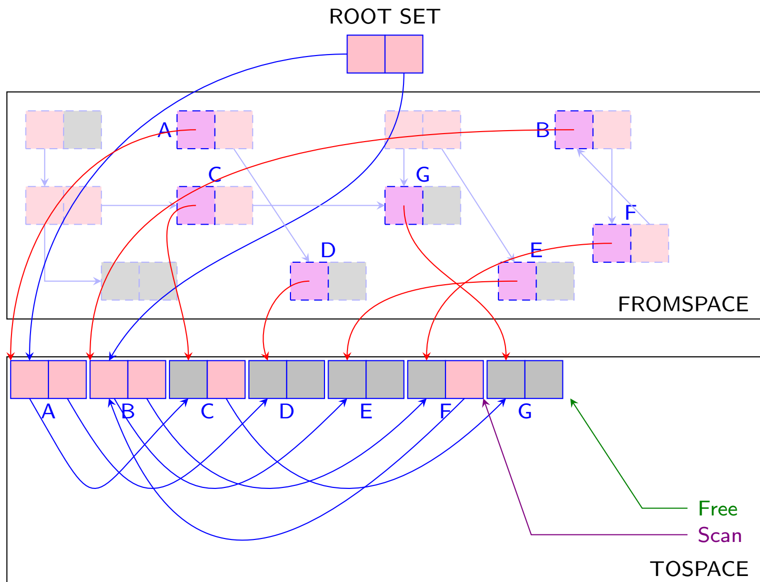
"Copying" prügikoristus



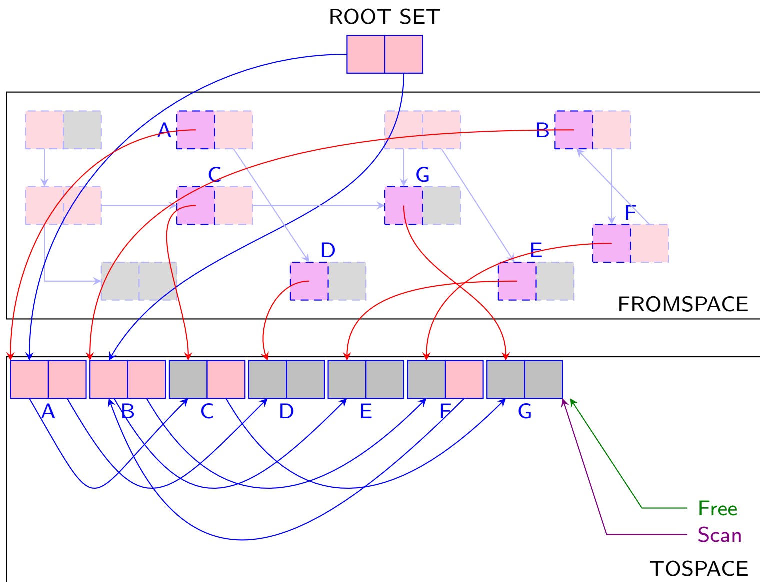
"Copying" prügikoristus



"Copying" prügikoristus



"Copying" prügikoristus



"Copying" prügikoristus

Eelised

- ✔ kogu vaba mälu on kompaktselt koos;
- ✔ suvalise suurusega uute objektide loomine on väga odav:
  - mälu reserveerimine on kuhja täitumisviida suurendamine;
  - kuhja täitumise kontroll on kahe viida võrdlemine;
- ✔ inspekteeritakse ainult elusaid objekte:
  - enamus objekte on reeglina suhteliselt lühikese elueaga;
  - elusaid objekte on seetõttu tavaliselt tunduvalt vähem kui prügi;
- ✔ teoreetiline amortiseeritud efektiivsus väga hea:
  - kuhja suuruse kasvades lähenevad kopeerimiskulud nullile!

"Copying" prügikoristus

Puudused

- ✘ kogu töö on kontsentreeritud prügikoristuse ajale:
  - võib tekitada häirivaid pause;
- ✘ laiuti läbivaatus võib segamini lüüa lokaalsusmustrid;
- ✘ kõik viidad tõstetakse ringi:
  - võib rikkuda mõnd invarianti, mida programm eeldab;
- ✘ pool mälust on "kasutu";
- ✘ pika elueaga objekte kopeeritakse igal prügikorstusel üha uuesti:
  - võib suurte "veteranobjektide' korral osutuda küllaltki kulukaks.

Põlvkonniti prügikoristus

Empiirilisi tähelepanekuid

- Vastsündinute suremus (infant mortality) – enamik objekte sureb väga noorelt.
  Reeglina 80-90% objekte sureb enne järgmise megabaidi kasutamist:
  - 60-90% CL ja 75-95% Haskelli objektidest surevad enne saamist 10 kb vanaks.
  - SML/NJ vabastab 98% objektidest iga prügikoristuse järel.
  - 95% Java objektidest on "lühiealised".
- Mida vanem on objekt, seda tõenäolisemalt elab ta üle järgmise prügikoristuse.
- Viitade suunatus (directionality of reference) – nooremad objektid reeglina viitavad vanematele.

Põlvkonniti prügikoristus

Põlvkonniti prügikoristus

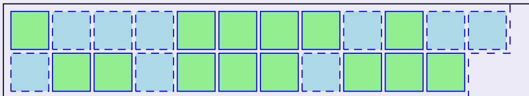
- Mälu on seal paiknevate objektid vanuse järgi jaotatud põlvkondadeks (generations).
- Põlvkondade koguarv ja suurus on reeglina eelnevalt fikseeritud.
- Uued objektid (infants) lisatakse noorimasse, "vastsündinute" (nursery), põlvkonda.
- Objekti vananedes (tenure) edutatakse ta järgmisse põlvkonda.
- Erinevates põlvkondades toimub prügikoristus erineva sagedusega
  - põhitähelepanu pööratakse noorimale põlvkonnale.

Põlvkonniti prügikoristus

Mälu jaotus põlvkondadeks

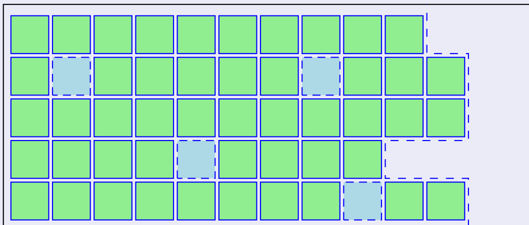
Generation 1 (youngest)

Generation 2



⋮

Generation n (oldest)



Live object

Dead object

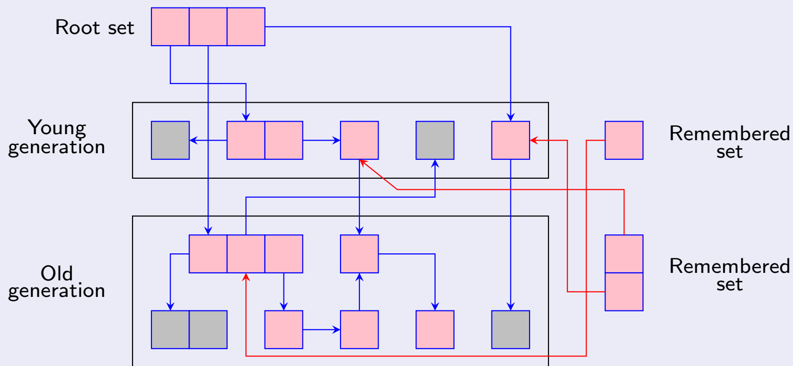
Põlvkonniti prügikoristus

Meelespead

- Lisaks "tavalistele" juurtele, on antud põlvkonna juurteks ka viidad teistest põlvkondadest temasse.
- Nende juurte asukoht pole staatiliselt kindlaksmääratav.
- Prügikoristuse aegne juurte otsimine teistest põlvkondadest on väga kulukas.
- Seetõttu seotakse iga põlvkonnaga meelespea (remembered set), mis sisaldab teistest põlvkondadest tulenevaid viitu
  - kui mingi viit on ühest põlvkonnast teise, siis lisatakse ta vastava sihtpõlvkonna meelespeasse.

Põlvkonniti prügikoristus

Meelespead



Põlvkonniti prügikoristus

Probleem

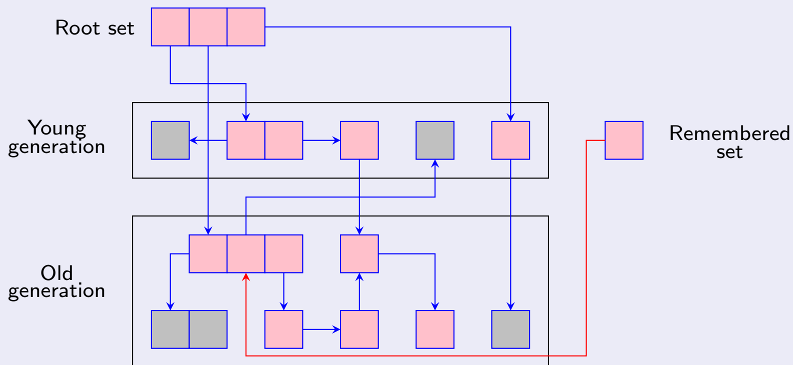
- Meelespeadele võib kuluda suhteliselt palju mälu
  - salvestada tuleb kõik põlvkondade vahelised sõltuvused.
- Meelespeasid peab haldama täitmisajal, mis võib olla väga kulukas
  - iga viitmuutuja omistamine võib potentsiaalselt olla erinevate põlvkondade vaheline.

Lahendus

- Salvestada meelespeades ainult viidad vanematest põlvkondadest noorematesse
  - kahe põlvkonna korral tarvis ainult üht meelespead (noorema põlvkonna jaoks).
- Kasutada ligikaudseid meelepeasid.

Põlvkonniti prügikoristus

Ühesuunalised meelespead



Põlvkonniti prügikoristus

Meelespead

- Viidad *vanemast põlvkonnast nooremasse* on noorema põlvkonna juurteks:
  - selliseid viitasid esineb suhteliselt harva;
  - nad tekivad vanas objektis viida destruktiivsel muutmisel;
  - selliseid omistamisi saab kindlaks teha kasutades kirjutustõkkeid (write barrier).
- Viidad *nooremast põlvkonnast vanemasse* on sagedased:
  - pole probleemiks, kui vanema põlvkonna prügikoristusel alati koristatakse ka noorem põlvkond.

Põlvkonniti prügikoristus

Põlvkonniti prügikoristus

- Tihti kasutatakse ainult kahte põlvkonda, kus noorem põlvkond on vanemast põlvkonnast väiksema suurusega.
- Reeglina toimub ainult pisikoristus (minor collection), kus:
  - eemaldatakse prügi ainult nooremast põlvkonnast;
  - piisavalt vanad objektid edutatakse vanemasse põlvkonda.
- Vanema põlvkonna täitumisel teostatakse suurpuhastus (major collection); so. eemaldatakse prügi mõlemast põlvkonnast.
- Pisikoristus ja suurpuhastus võivad kasutada erinevaid prügikoristuse skeeme (näit. pisikoristuse korral "copying" ning suurpuhastuse korral "mark-compact").

Põlvkonniti prügikoristus

Probleemid

- Pisikoristused ei eemalda vana põlvkonna prügi:
  - eakas prügi (tenured garbage) põhjustab ka nende noorte objektide säilimise, millele ta viitab (nepotism).
- Kui vana peab objekt olema enne järgmisse põlvkonda edutamist?
  - Üks pisikoristus pole piisav, kuna vahetult eelnevalt loodud objektidel pole olnud aega suremiseks.
  - Tavaliselt loetakse piisavaks kaks pisikoristust.
- Kui suur peab olema noorim põlvkond?
  - Peab ära mahtuma põhimällu.
  - Liiga suur teeb pisikoristuse pausid liiga pikaks.
  - Liiga väike ei anna noortele objektidele aega suremiseks.

Põlvkonniti prügikoristus

Põlvkonniti prügikoristuse eelised

- ✔ Väga edukas paljude rakenduste korral.
- ✔ Lühendab prügikoristuse pause interaktiivseteks rakendusteks talutavale tasemele.
- ✔ Heade lokaalsusomadustega.
- ✔ Reeglina vähendab prügikoristusele kuluvat koguaega.

Põlvkonniti prügikoristuse puudused

- ✘ Halvimal juhul on lihtsatest meetodidest kulukam.
- ✘ Objektid ei tarvitse surra piisavalt kiiresti.
- ✘ Rakendused võivad "takistuda"kirjutustõketesse.
- ✘ Liiga paljude viitade korral vanadest objektidest noortesse või väga sügava magasini korral võib pauside kestus pikeneda.