15
Garbage collection 2.

Inkrementel garbage collection.
"Trædemøllen".
Generations garbage collection.

Noter

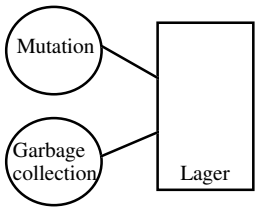
I dette kapitel fortsætter vi vores gennemgang af garbage collection teknikker.
Emnerne er inkrementel garbage collection og generations garbage collection.

Inkrementel garbage collection.

- **Problem**: Mark & sweep og mark & copy teknikkerne afstedkommer et afbrud i programudførelsen:
  - Irriterende i interaktive applikationer.
  - Utåleligt i realtidssystemer.
- **Løsning**: *Inkrementel garbage collection*:
  - Tillad programudførsel side om side med garbage collection.

Co-routine sekventiering (interleaving) eller

Parallel sekventiering.



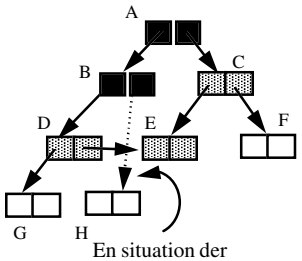
- **Udfordring**: Mutations processen kan forstyrre garbage collection processen, således at garbage collectoren ikke får opsamlet alt spild.

Noter

I ovenstående terminologi er mutationsprocessen applikationsprogrammet.

Inkrementel version af Baker's teknik.

- **Situation**:
  - Mutation og garbage collection forløber parallelt.
  - Garbage collection teknik: Baker's teknik med hvid, grå og sort liste.
- **Ønsket invariant**: Der må aldrig være referencer fra sorte data til hvide data.
  - Hvis mutatoren tilgår en reference fra et sort objekt til et hvidt objekt kræver det en koordinering med den fortløbende garbage collection:
    - Den hvide celle H skal gøres grå.
    - **Konsekvens**:
      - Mutationen forløber helt og holdent inden for sorte og grå objekter.
      - Mutationen vil aldrig kunne forstyrre spildopsamlingen, således at objekter ikke bliver taget i betragning for garbage collection.
    - Der opstilles en *læse barriere* mellem sorte og hvide objekter.
      - Kostbart at opretholde en læse barriere.



En situation der bryder invarianten

PS4 - Garbage collection 2

Noter

Bemærk, at teknikken nævnt ovenfor er en simpel udvidelse af Baker's teknik, som vi tidligere har set på. Vi har altså tre lister som er hhv. sort, grå og hvid.

Rollen og vigtigheden af den nævnte invariant: Sorte objekter er færdigbehandlede af garbage collectoren. Alle referencer fra sorte objekter er til grå objekter. Det er de grå objekter, som for nærværende behandles af garbage collectoren. Hvis mutatoren forstyrren 'denne orden' bliver der kuk i bogholderiet, og der vil blive objekter, der ikke vil blive forsøgt opsamlet i denne cykel.

I figuren vil objektet H (og hvad H måtte referere til) fejlagtigt blive garbage collectet. Situationen er angiveligt fremkommet derved, at mutationsprocessen bag ryggen af garbage collection processen har ændret på referencen i CDR-feltet af B objektet. Dette vil vi ikke acceptere, og vi må derfor gennemføre en eller anden form for *minimal koordination* mellem mutationsprocessen og garbage collection processen.

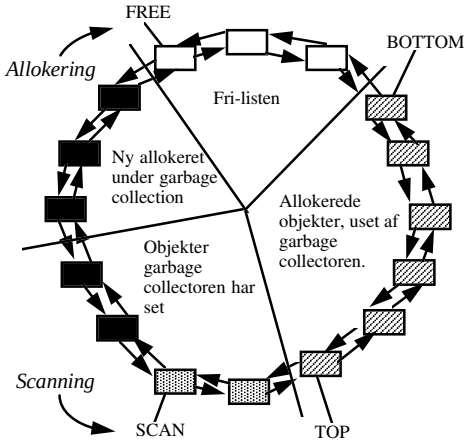
Læse barrieren sikrer, at hvis vi ser (læser, tilgår) en reference til et hvidt objekt, gør vi dette objekt gråt. Det er dog drastisk slet ikke at tillade inspektion af sådanne referencer. Det som virkeligt tæller er, når vi reelt laver (skriver, etablerer) en reference fra sort til hvid. Sådanne skrivninger kan fanges, og vi kan registrere (kopiere) sådanne referencer til en tabel, der tjener som "ekstra grå objekter" til garbage collectoren. Vi taler i så fald om en *skrive barriere*.

Hvornår og hvordan skifter vi mellem mutation og garbage collection? For hver allokeringsoperation, driver vi scan operationen én frem i den grå liste. Dermed vil garbage collection skride frem i takt med at lageret bruges op. Når garbage collection skrider frem, vil vi langsomt, men sikkert, nærme os den situation, hvor vi har identificeret de objekter (til rest på den hvide liste), som er garbage (og som så kan bruges til yderligere allokering).

På de næste sider ser vi på en snedig, cyklisk organisering af hvid, grå og sort liste.

"Trædemøllen" (1).

Optimalisering: De dobbeltkædede lister slåes sammen til én stor cyklisk liste.



Allokering: FREE flyttes med uret.

Scanning: Hvis objektet under SCAN referer en beige (skraveret) celle, hægtes denne ind i den grå liste ved TOP. SCAN og TOP bevæges mod uret.

Afslutning af garbage collection: Når SCAN møder TOP.

Behov for "flip": Når FREE møder BOTTOM:

Hvis mutatoren ser et skraveret objekt, skal de hægtes ind i den grå liste.

Noter

På denne figur vises kun de administrative referencer (dobbeltkædningen) mellem objekterne. De egentlige "sproglige" datareferencer vises altså ikke.

Objekter allokeres sorte, men kan blive til garbage inden garbage collection processen afsluttes. De opsamles først i den næste garbage collection runde.

Under en garbage collection runde kan mutatoren reelt gøre sorte eller grå objekter til garbage. Dette ignorerer vi, og dette spild vil først bliver opsamlet i næste runde.

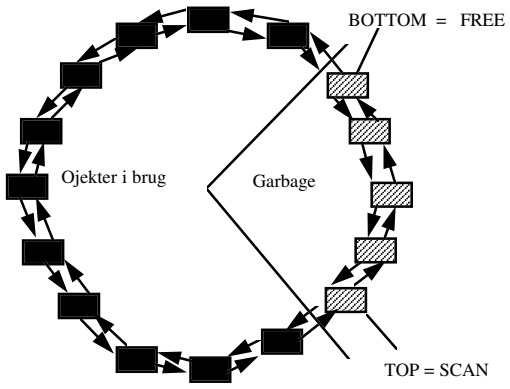
Invarianten udtrykker, at mutatoren udelukkende manipulerer sorte og grå objekter. Hvis ikke, er det et skraveret objekt, som øjeblikkelig "gøres grå" (hægtes ud af beige liste, og ind i grå).

Om farverne. Man taler om tri-coloring med farverne sort, hvid og grå. I det ovenstående har man dog brug for fire farver, som er:

- *Sort*: behandlet af garbage collectoren.
- *Grå*: under behandling af garbage collection (scannes).
- *Beige* (skraveret ovenfor): ikke set af garbage collectoren. Kan være spild. Dette er en afart af hvide objekter. I den engelske litteratur kaldes disse for "ecru".
- *Hvid*: ubrugt (og altså fri, og kandidater for ny-allokering).

“Trædemøllen” (2)

Garbage collection afsluttet og frilisten er udtømt.

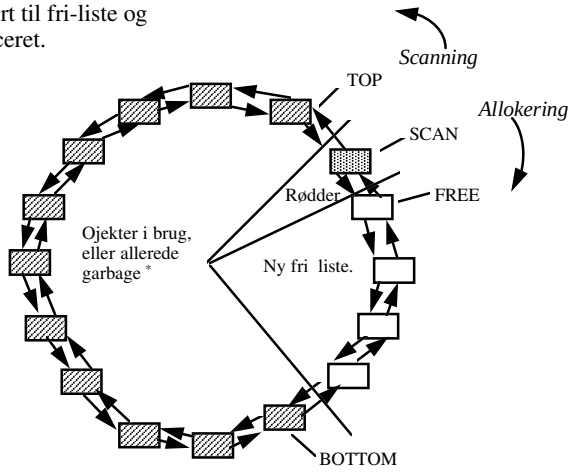


PS4 - Garbage collection 2

Noter

“Trædemøllen” (3)

“Flip”: Garbage er gjort til fri-liste og rod-mængden identificeret.



*: mutatoren kan have skabt spild i de skraverede objekter samtidig med, at den netop fuldførte garbage collection er ført igennem.

PS4 - Garbage collection 2

Noter

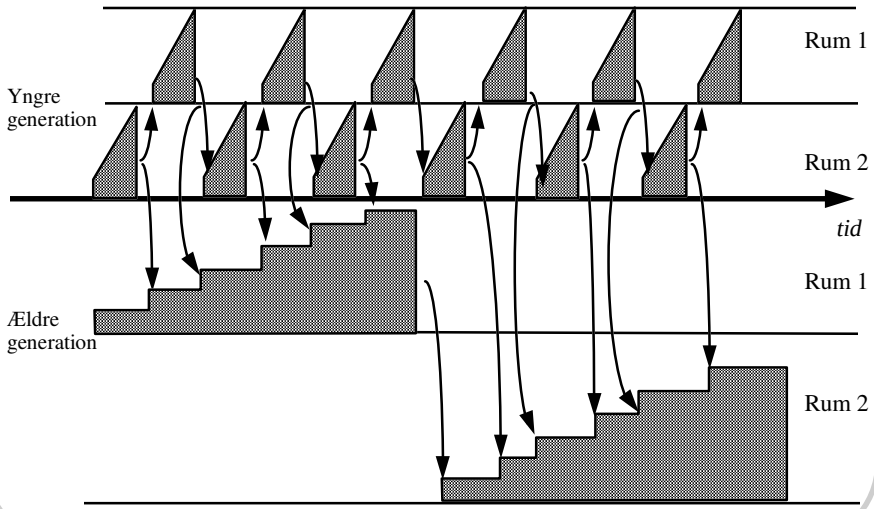
Man kan let overbevise sig om, at situationen på denne side er i god overensstemmelse med det oprindelige billede (udgangspunktet) af trædemøllen. Allokeringen foregår i samme retning. Scanning foregår i samme retning.

Generations garbage collection.

- **Iagttagelser**:
  - De fleste unge objekter dør hurtigt efter deres skabelse.
    - Der er meget spild at opsamle, hvis man udelukkende behandler unge objekter.
  - De fleste ældre objekter vedbliver med at eksistere.
    - Det er spild af tid at markere og/eller kopiere gamle og vedblivende data i gentagne garbage collection cykler.
- **Konsekvenser**:
  - Det er attraktivt at arrangere objekterne i to eller flere *generationer*, som tillader at man er bevidst om alderen af objekter.
  - Yngre generationer garbage collectes hyppiere end ældre generationer.
  - Generations-ideen er relativt ortogonal i forhold til spørgsmålet om 'mark-and-sweep'og kopierende garbage collection.
- **Problemer**, som skal håndteres:
  - Hvordan holder vi styr på objekternes alder?
    - Registrerer, hvormange gange de har været forsøgt garbage collectet.
  - Hvordan håndteres referencer mellem objekter i forskellige generationer?

Noter

Forløbet af kopierende garbage collection med generationer.

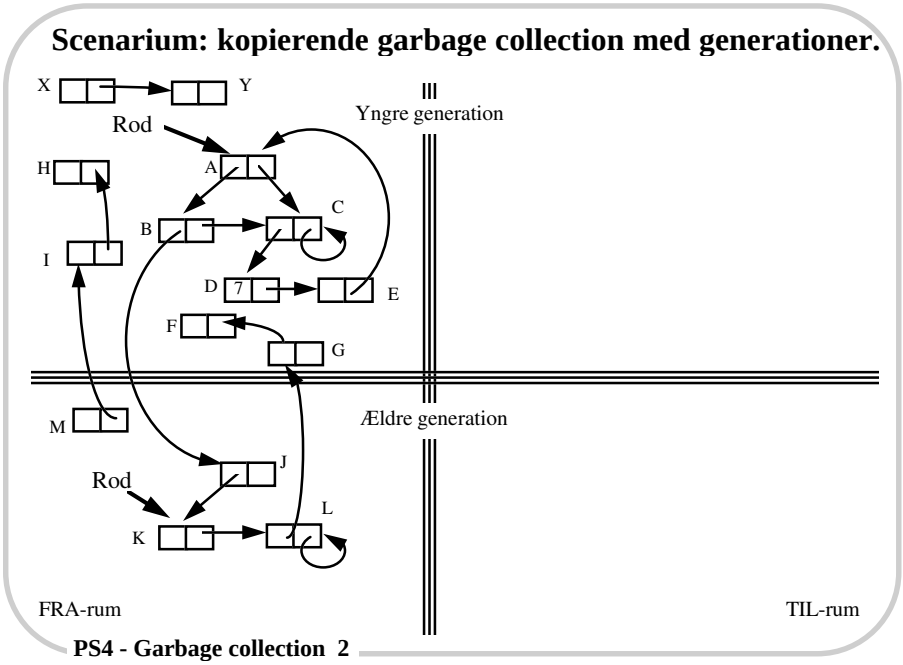


PS4 - Garbage collection 2

Noter

Denne figur er næsten identisk med figur 11 i artiklen. Man ser, som en funktion af tiden, hvordan generations garbage collection forløber. Den vandrette "x" dimension er tiden, og den lodrette "y" dimension betegner mængden af allokeret lager i fra- og til-rummet, og i de to generationer, som vi arbejder med.

Til en start er Rum 1 fra-rum, og Rum 2 er til-rum. Dette ændrer sig naturligvis undervejs i processen, idet de to rum jo skifter rolle efter hver garbage collection.



Noter

Vi vil her diskutere og illustrere *inter-generations referencer.*

Referencen fra objekt L til G er et eksempel på en **reference fra et gammelt til et nyt objekt** (gl->ny). Man vil forvente, at der er få sådanne referencer. Hvis man laver en spild-opsamling i den unge generation med den givne rod, og hvis man ikke tager den omtalte reference ind i billedet, vil man fejlagtigt identificere objekterne F og G som spild. Moralen er, at man må *holde styr på gl->ny referencer*, og opfatte sådanne som bidragende til rødderne i en garbage collection i den yngre generation. Da der forventes relativt få sådanne referencer, er det overkommeligt at registrere dem; men det tager naturligvis tid, at holde udkig efter etableringen af sådanne (*skrive barriere*).

Bemærk, at også referencen fra M til I er en gl->ny reference, som vi må registrere; men i modsætning til referencen fra L til G bevirker denne, at vi rent faktisk holder fast i noget spild, idet det jo fra en global betragtning let ses, at også M er spild.

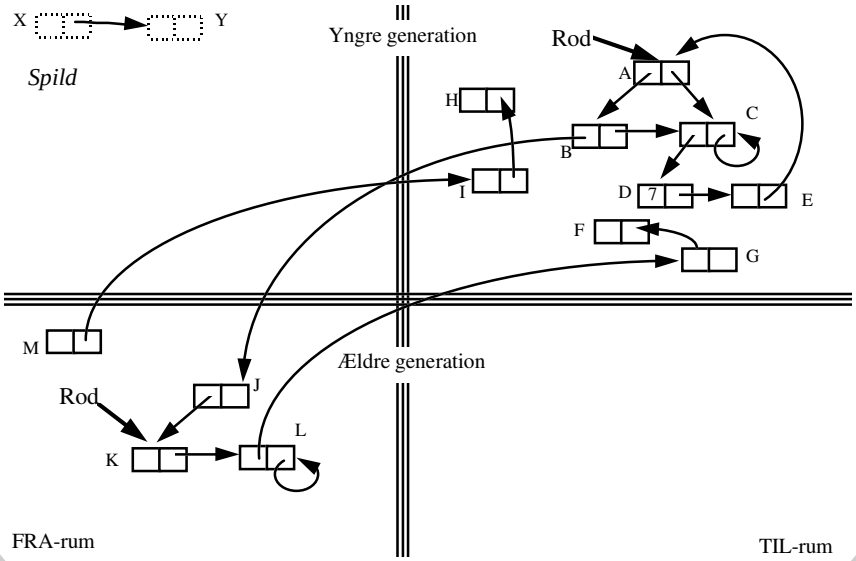
Omvendt er referencen fra objekt B til J en **reference fra et nyt objekt til et gammelt** (ny->gl). Vi vil forvente, at der er mange sådanne reference. Under garbage collection i den yngre generation generer denne ikke, idet det kun er objekter i den yngre generation vi da ønsker at opsamle spild iblandt. Når vi på et tidspunkt laver garbage collection i den ældre generation, må vi tage hensyn til en ny->gl reference. Men idet der forventes mange sådanne referencer kan det være meget dyrt at skulle administrere dem (holde udkig efter sådanne ved hver mutation af referencer).

Man kan undgå at dette ved enten

> **A**. At betragte alle objekter i den yngre generation som rødder i forhold til en garbage collection i den ældre generation, eller
>
> **B**. At inkludere den nye generation i en garbage collection i den ældre generation (opsamle begge under ét i det relativt sjældne tilfælde, hvor der opsamles spild i den ældre generation).

På de følgende sider, vil vi antage at vi følger strategi **A**. Vi viser på næste side situationen efter, at vi har lavet garbage collection i den yngre generation.

Efter garbage collection i yngre generation.

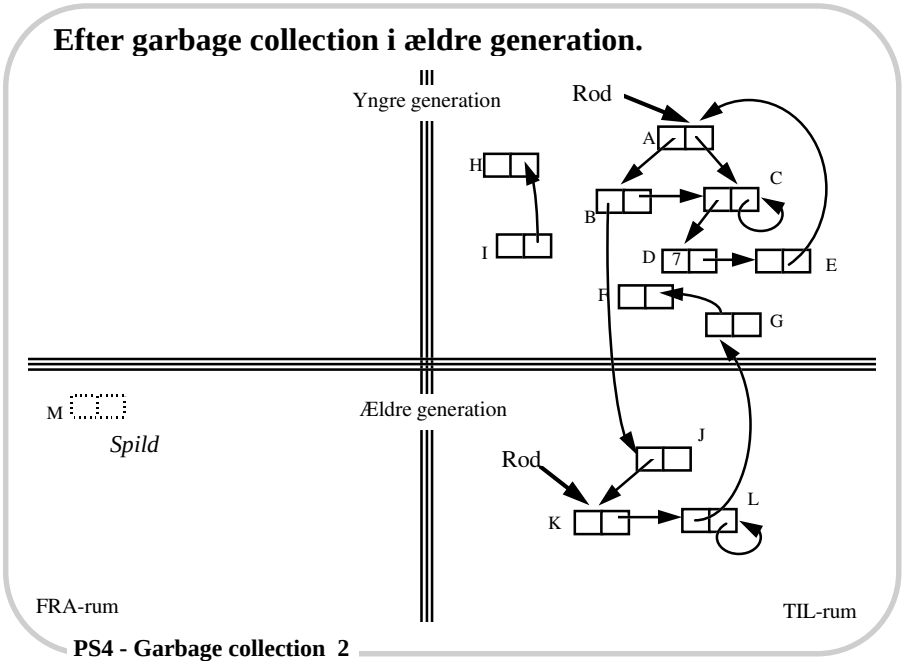


PS4 - Garbage collection 2

Noter

Globalt set er objekterne H og I spild, men det erkendes ikke med vores politik omkring inter-generations referencer.

Vi antager, at vi nu foretager en garbage collection i den ældre generation. Dette er vist på næste side.



Noter

Bemærk at referencen fra objekt B til objekt J er at regne for en rod i forhold til denne garbage collection. Dette er essentielt for at fastholde objekt J.

Bemærkt også, at objekt M nu bliver spild, og at vi derfor i næste garbage collection i den yngre generation får opsamlet objekterne H og I.