Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

# 440: Garbage Collection

November 25, 2014

# Outline



1 Overview of Garbage Collection

2 Reference counting

3 Mark and Sweep

4 Copying

5 Generational GC

6 GC in parallel processing

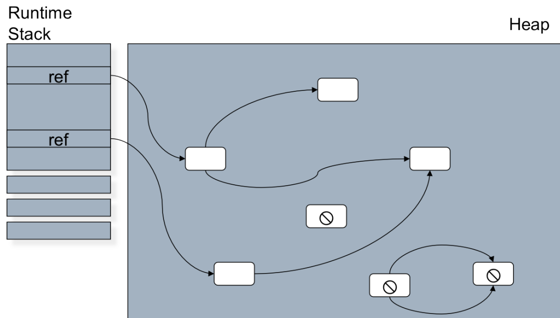
Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

# What is garbage collection

- Memory Management technique.
- Process of freeing objects.
- No longer referenced by the program.

## Fundamental Garbage Collection Property

"When an object becomes garbage, it stays garbage"



# Dynamic Memory Allocation


Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing


- In Java, “new" allocates an object for a given class.

```
1  Account account = new SavingsAccount("1234567");
```

- But there is no instruction for manually deleting the object.
- Objects reclaimed by a garbage collector when the program “does not need it" anymore.

```
1  account=null;
```

- “The Garbage Collection Handbook: The Art of Automatic Memory Management" by Richard Jones, Anthony Hosking, and Eliot Moss.
- Some slides are from Erez Petrank

# Manual vs automated memory management

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

## Manual allocation and de-allocation

```
1 ptr=malloc(256 bytes);
2 /* use ptr */
3 free(Ptr);
```

- manual memory management: let programmer decide when objects are deleted.
- automatic memory management: let a garbage collector delete objects.

## Problems of manual management

- Manual memory management creates severe debugging problems
  - Memory leaks,
  - Dangling pointers.
- In large projects where objects are shared between various components it i sometimes difficult to tell when an object is not needed anymore.
- Considered the big debugging problem of the 80's

# Solution: Automatic Memory Reclamation

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

## users allocate space for an object

```
1   Account account = new SavingsAccount("1234567");
```

## system collects the space when not used

- When the system “knows" the object will not be used anymore, it reclaims its space.
- Telling whether an object will be used after a given line of code is undecidable.
- Therefore, a conservative approximation is used.
- An object is reclaimed when the program has “no way of accessing it".
- Formally, when it is unreachable by a path of pointers from the “root" pointers, to which the program has direct access.
- Local variables, pointers on stack, global (class) pointers, JNI pointers, etc.
- It is also possible to use code analysis to be more accurate sometimes.

# What's good about automatic “garbage collection"?

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

## Software engineering

Relieves users of the book-keeping burden. Stronger reliability, fewer bugs, faster debugging. Code understandable and reliable. (Less interaction between modules.)

## Security (Java)

Program never gets a pointer to ”play with”.

# GC and languages

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

**Sometimes it's built in:**

- LISP, Java, C #.
- The user cannot free an object.

**Sometimes it's an added feature:**

- C, C++.
- User can choose to free objects or not. The collector frees all objects not freed by the user.

Most modern languages are supported by garbage collection.

# what is bad about automated GC

## It has a cost:

- Old Lisp systems 40%.
- Today's Java program (if the collection is done “right") 5-15%.
- Considered a major factor determining program efficiency.
- Techniques have evolved since the 60's
- We will only go over basic techniques.

## How to evaluate GC

- Overall collection overheads (program throughput).
- Pauses in program run.
- Space overhead.
- Cache Locality (efficiency and energy).

# Three classic GC algorithms

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

- Reference counting
- Mark and sweep (and mark-compact)
- Copying

The last two are also called tracing algorithms because they go over (trace) all reachable objects.

# Outline

1. Overview of Garbage Collection
2. Reference counting
3. Mark and Sweep
4. Copying
5. Generational GC
6. GC in parallel processing

# Reference counting (Collins 1960)


Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing


## reference counting algorithm

Each object has an RC field, new objects get o.RC:=1. ;
**if** *p that points to o1 is modified to point to o2* **then**
  o1.RC–;
  o2.RC++. ;
**end**
**if** *o1.RC==0* **then**
  Delete o1;
  Decrement o.RC for all children of o1 ;
  Recursively delete objects whose RC is decremented to 0 ;
**end**

```
1  Account x, y;
2  x = new SavingsAccount("1234567");
3  y = x;
4  x = null;
5  y = x;
```

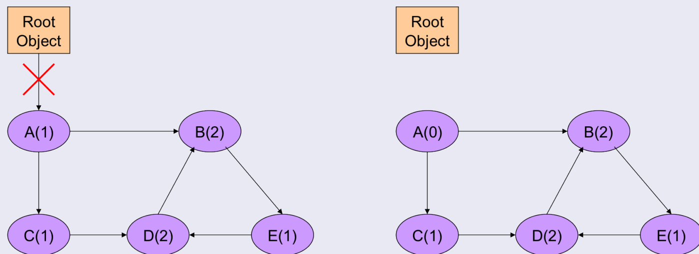
rc=1, 2, 1, 0

- Recall that we would like to know if an object is reachable from the roots.
- Associate a reference count field with each object: how many pointers reference this object.
- When nothing points to an object, it can be deleted.
- Very simple, used in many systems.

# A problem for cycles

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

- The Reference counting algorithm does not reclaim cycles.
- Solution 1: ignore cycles, they do not appear frequently in modern programs;
- Solution 2: run tracing algorithms (that can reclaim cycles) infrequently;
- Solution 3: designated algorithms for cycle collection;




- Other algorithms?
- Observation: rather than explicitly keep track of the number of references to each object we can traverse all reachable objects and discard unreachable objects

# Outline



1. Overview of Garbage Collection
2. Reference counting
3. Mark and Sweep
4. Copying
5. Generational GC
6. GC in parallel processing

# (Naive) Mark-and-Sweep Algorithm [McCarthy 1960]

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

## Mark phase:

- Start from roots and traverse all objects reachable by a path of pointers.
- Mark all traversed objects.

## Sweep phase:

- Go over all objects in the heap.
- Reclaim objects that are not marked.

(animated figure from Wikipedia)

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

# When GC starts

Garbage collection is triggered by allocation.

## Triggering

New(A)= **if** *freeList is empty* **then**
    markSweep() ;
    **if** *freeList is empty* **then**
        return (“out-of-memory")
    **end**
    pointer = allocate(A);
    return (pointer);
**end**

# Properties of Mark-and-Sweep

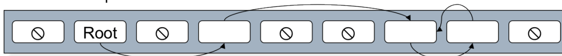
Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

## Complexity:

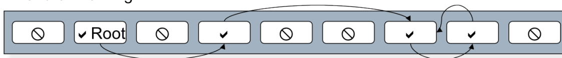
- Mark phase: live objects (dominant phase)
- Sweep phase: heap size.
- Termination: each pointer traversed once.

- Most popular method today (at a more advanced form). Simple.
- Does not move objects, and so heap may fragment.
- Various engineering tricks are used to improve performance.

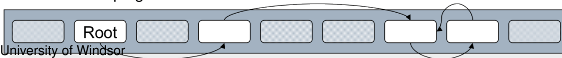
1. mark-sweep start




2. end of marking




3. end of sweeping



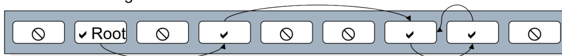
# mark and compact

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

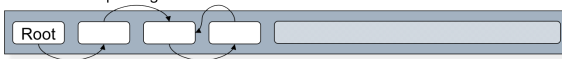
- During the run objects are allocated and reclaimed.
- Gradually, the heap gets fragmented.
- When space is too fragmented to allocate, a compaction algorithm is used.
- Move all live objects to the beginning of the heap and update all pointers to reference the new locations.
- Compaction is considered very costly and we usually attempt to run it infrequently, or only partially.

- some plots are from Faizan Ahmed

1. end of marking




2. end of compacting



# Outline

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

1 Overview of Garbage Collection

2 Reference counting

3 Mark and Sweep

4 Copying

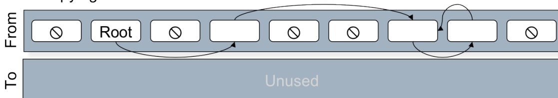
5 Generational GC

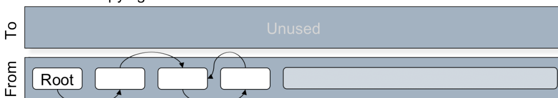
6 GC in parallel processing

# Copying

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

- Heap partitioned into two.
- Part 1 takes all allocations.
- Part 2 is reserved.
- During GC, the collector traces all reachable objects and copies them to the reserved part.
- After copying the parts roles are reversed:
  - Allocation activity goes to part 2, which was previously reserved.
  - Part 1, which was active, is reserved till next collection.



# Properties of Copying

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

- Compaction for free
- Major disadvantage: half of the heap is not used.
- “Touch" only the live objects
- Good when most objects are dead.
- Usually most new objects are dead, and so there are methods that use a small space for young objects and collect this space using copying garbage collection.

# Comparison

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

| | Reference Counting | Mark & Sweep | Copying |
|---|---|---|---|
| Complexity | Pointer updates + dead objects | Size of heap (Live objects) | Live objects |
| Space overhead | Count/object + stack for DFS | Bit/object + stack for DFS | Half heap wasted |
| Compaction | Additional work | Additional work | For free |
| Pause time | Mostly short | long | long |
| More issues | Cycle collection | | |

# Outline

1 Overview of Garbage Collection

2 Reference counting

3 Mark and Sweep

4 Copying

5 Generational GC

6 GC in parallel processing

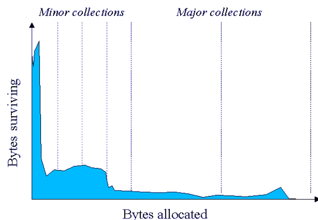
Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

# Generational Garbage Collection

**“The weak generational hypothesis":**

most objects die young.

- Using the hypothesis: separate objects according to their ages and collect the area of the young objects more frequently.
- The heap is divided into two or more areas (generations).
- Objects allocated in 1st (youngest) generation.
- The youngest generation is collected frequently.
- Objects that survive in the young generation “long enough" are promoted to the old generation.



# Advantages

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

- Short pauses: the young generation is kept small and so most pauses are short.
- Efficiency: collection efforts are concentrated where many dead objects exists.
- Less fragments:
  - young and small objects might cause fragments. They are cleaned out in the young generation area.
  - old space becomes more compact
- Locality:
  - Collector: mostly concentrated on a small part of the heap;
  - Program: allocates (and mostly uses) young objects in a small part of the memory.

# copying vs. mark-sweep

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

- Copying is good when live space is small (time) and heap is small (space).
- A popular choice: Copying for the (small) young generation. Mark-and-sweep for the full collection.
- A small waste in space, high efficiency.

# Outline

1 Overview of Garbage Collection

2 Reference counting

3 Mark and Sweep

4 Copying

5 Generational GC

6 GC in parallel processing

# Concurrent GC

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

**Problem: Long pauses disturb the user**

- An important measure for the collection: length of pauses.
- average pause time for compacting 1to 2GB: one second

- Can we just run the program while the collector runs on a different thread?
- Not so simple!
- The heap changes while we collect.
- For example, we look at an object B, but before we have a chance to mark its children, the program changes them.

# Memory Management with Parallel Processors

Overview of Garbage Collection
Reference counting
Mark and Sweep
Copying
Generational GC
GC in parallel processing

- Stop the world
- Parallel (stop-the-world) GC
- Concurrent GC.

- Trade-offs in pauses and throughput
- Difference in complexity
- Choose between parallel (longer pauses) and concurrent (lower throughput).

# recap



1 Overview of Garbage Collection

2 Reference counting

3 Mark and Sweep

4 Copying

5 Generational GC

6 GC in parallel processing