**Garbage Collection**

Objects on the heap that can no longer be reached (in one or more hops) from any pointers in the activation stack (i.e., in local variables of active functions) or from any pointers in the static storage area (variables declared in C++ as "static") are called garbage.

Garbage collection (GC) is a dynamic approach to automatic memory management and heap allocation that processes and identifies dead memory blocks and reallocates storage for reuse.The primary purpose of garbage collection is to reduce memory leaks. Garbage collection is a type of memory management. It automatically cleans up unused objects and pointers in memory, allowing the resources to be used again.

Some programming languages have built-in garbage collection, while others require custom functions to manage unused memory.

Some programming languages and platforms with built-in GC (e.g., Java, Lisp, C# and .Net) self-manage memory leaks, allowing for more efficient programming.

**Techniques:**

**1.Reference Counting**

It is based on the idea of counting the number of pointer references to each allocated object.

Under reference counting approach, each allocated object contains a reference count field.

The memory manager is responsible for maintaining the invariant that -- at all times -- the reference count of each object is equal to the number of direct pointer references to that object.

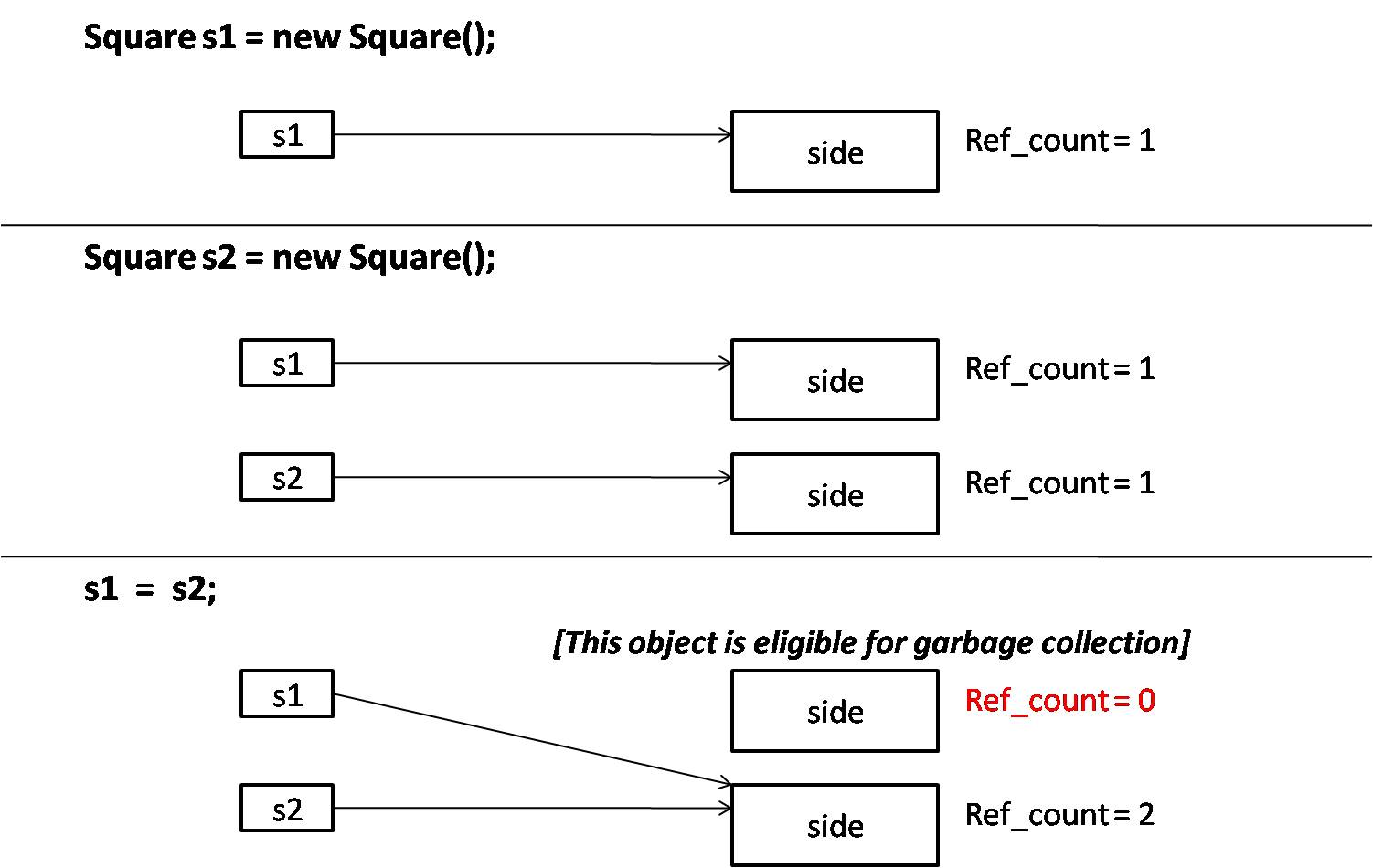
This garbage collection technique simply places a counter in each object that keeps track of the number of pointers that refer to the object. When the object is created, this reference count is set to one, to represent the pointer returned by the new operation. When one pointer is assigned into another, the run-time system decrements the reference count of the object (if any) formerly referred to by the assignment’s left-hand side , and increments the count of the object referred to by the right-hand side. When a reference count reaches zero, its object can be reclaimed. Recursively, the run-time system must decrement counts for any objects referred to by pointers within the object being reclaimed, and reclaim those objects if their counts reach zero.

Advantages:

The main **advantage** of the **reference counting** is that objects are reclaimed as soon as they can no longer be referenced. The garbage is easily identified.

Limitations:

The algorithm described above can't handle reference cycles, an object which refers directly or indirectly to itself. A mechanism relying purely on reference counts will never consider cyclic chains of objects for deletion, since their reference count is guaranteed to stay nonzero. So reference counting won't work if our data can form cycles of pointers. Also extra space as a count field is needed in each object.



For example given above,we have two objects their reference counts are set to one one respectively whenever they are allocated and memory and certain pointer is pointing to them but as first pointer,i.e.,s1 changes its pointing to location the reference count of both the objects change and as ref count for first obj has become 0,it can now be removed or deallocated through this technique.

**2.Mark and Sweep**

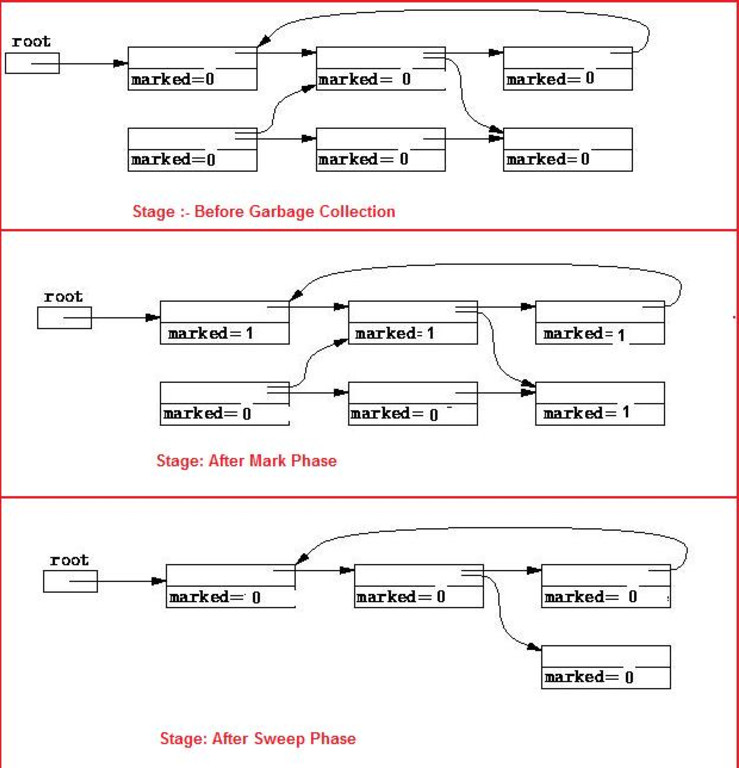
Mark and sweep algorithm has mainly two phases:

1) Mark phase  
2) Sweep phase

Mark phase-

In this phase, objects which are reachable from the program are marked as reachable. The garbage collector will start traversal from all references(on stack, registers, static variables) in the program and visit all the inner references in a Depth First Search (DFS) manner and mark objects as reachable. Every allocated object on the heap has a flag let’s call **marked** set to false when it is allocated. During the mark phase, this flag is turned to true if the object is reachable.

Sweep phase- This phase is used to clean up all the objects which weren’t marked in the Mark phase. The unreachable objects are deleted thereby allowing the program to allocate more objects subsequently.



For example given above first we mark the objects as 0,then in the mark phase we mark the reachable objects as 1 and then during the sweep phase all those who were marked as 0 during mark phase and are still 0 after sweep phase are removed considering them as garbage.

An alternative implementation of the exploration is Heap tracing with pointer reversal step uses a technique ﬁrst suggested by Schorr and Waite to embed the equivalent of the stack in already-existing ﬁelds in heap blocks. More speciﬁcally, as the collector explores the path to a given block, it reverses the pointers it follows, so that each points back to the previous block instead of forward to the next. As it explores, the collector keeps track of the current block and the block from whence it came.

The mark-and-sweep algorithm is called a tracing garbage collector because it traces out the entire collection of objects that are directly or indirectly accessible by the program.

**Advantages :**

a)No additional overheads incurred during the execution of algorithm

b)The algorithm handles cyclic references well and never results in an infinite loop

**Limitations:**

a)During the execution of GC algorithm, normal program execution is suspended

b)After multiple cycles of Mark & Sweep, reachable objects end up being separated by small unused segments of memory thus leading to fragmentation

**3. Conservative Collection**

Idea: if a word looks like a pointer (i. e. right address range, alignment, points into an allocated data page, ...) assume that it is a pointer. In Conservative Collection ,a garbage collector must scan all objects and invocations (execution stack) to identify all of the "live" addresses in the executing program and then "collect" objects that do not have "live" addresses. In some environments it's possible for the GC algorithm to be PRECISE and know exactly what is an object address and what is not. In other environments it must scan parts of storage (most notably the execution stack) where there are words of storage that MIGHT be an object address and make the CONSERVATIVE assumption that if it looks like a valid address, and there is an object that has that address, then the object should not be collected.

Conservative garbage collectors are garbage collectors that work without compiler cooperation. For this reason they must discover pointers on their own. Unfortunately pointers and integers look the same from the compiler’s point of view causing some integers to be treated as pointers. Conservative garbage collection also requires that the running program stop and wait for the garbage collector to ﬁnish before it may proceed. If there are multiple threads all threads are required to stop.

Advantages:

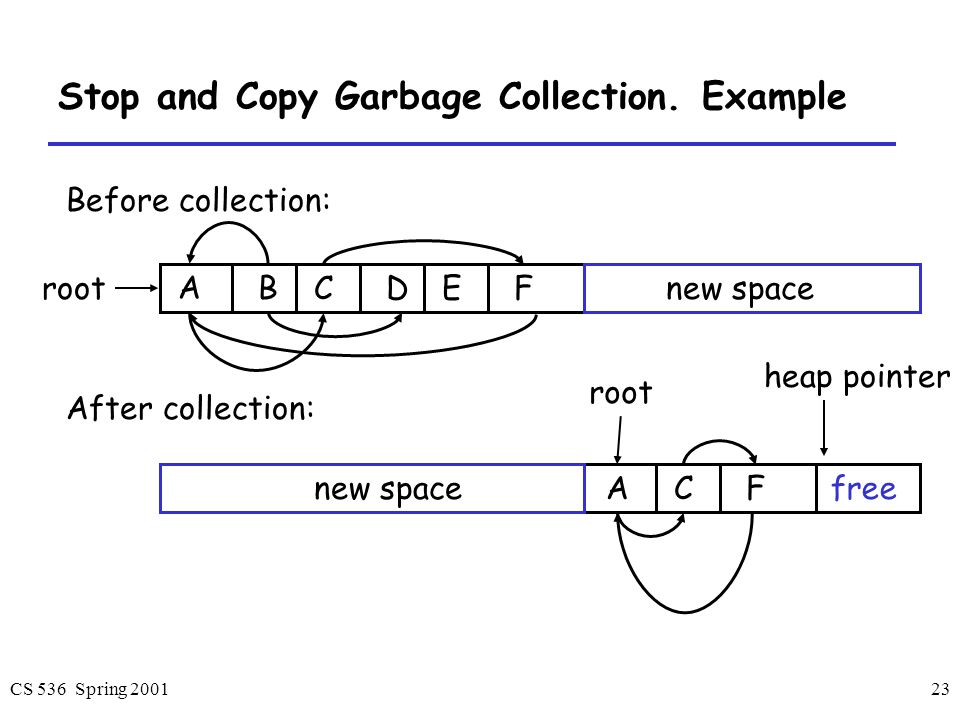
The algorithm is completely safe (in the sense that it never reclaims useful blocks) so long as the programmer never “hides” a pointer. It can collect unreachable values without having to work in harmony with the compiler.

Limitations:

Values that happen to look like pointers cause memory leaks by preventing parts of the heap from being recycled.

**4.Stop & Copy**

This algorithm gets rid of the sweep phase, and doesn’t fragment free memory. The idea is to split all of memory into two pieces, one called the From space(Active part) and the other called the To-space(Passive Part). Say we have twelve cells of memory:The ﬁrst six cells are ﬁlled with data: the objects A through F. At this point in the program, only A, C and F are reachable. When we run DFS to ﬁnd



live objects, instead of marking each object, we copy the object to the next free cell of the To-space. Once the DFS has completed, the To-space will consist of all the live objects, and they will be contiguous in memory. Now we swap the names of the two semi spaces.

This algorithm avoids doing a sweep phase as rather than trying to determine which objects are unreachable, which requires a sweep, the algorithm just copies out live objects to the To-space, and then everything in the From space can be treated as dead.

Advantages:

a)Only one pass over data.

b)Handles cycles.

Limitations:

a)Needs double the basic heap size.