ARC Research Paper

Traditionally, most programming languages use a specific implementation of garbage collection in order to delete objects and free up more memory. In Swift however, it uses automatic reference counting in order to complete this task.

Automatic reference counting is Apple's method to efficiently keep track of class instances and decides on the appropriate time to deallocate the same class instances it has been monitoring. This method is widely efficient due to the fact that traditionally, class instances are passed by reference and strain compilers widely due to the fact that compilers struggle on knowing when it is safe to deallocate them. ARC typically operates within an applications heap as that is where RAM is dynamically allocated. It manages the memory by allocating a specific amount of memory for every class object created. Even further, ARC has strong , weak, and unowned references. Every class object created will have a reference count property that tracks everything with a strong reference. Basically every time a strong reference is created the reference count is increased by one and vice versa if the reference ever goes out of scope. Objects with weak references can be destroyed, and unowned references read by non-existent objects cause crashes.

Before the ARC process became known, there was and still is the implementation of the garbage handling approach to manage memory. As previously stated, garbage handling has the major weakness of not knowing when it is safe to deallocate objects. At other perspectives we see that objective-C used method manual retain release (MRR) in regard to memory management. Swift's ARC implementation is much faster and virtually takes away the worry of running out of memory if handled efficiently. This is fantastic but there are still risks and weaknesses that come with ARC. ARC cannot detect memory retention cycles so developers

have to worry about making strong variables so we do not end up with two objects which retain one another. The biggest weakness of ARC is the deallocation problem. For example if an object is in many UIKits and has a secondary thread that has a reference to tt, this can become an issue and even harder to debug.

As we've stated the grandeurs of ARC, how does it compare to the well known garbage collection implementation? Garbage collection is a dynamic method of automatic memory management and heap allocation. It identifies dead memory blocks and reallocates storage for use. ARC , as we've previously mentioned, provides reference counting for objects. In contrast to garbage collection, it is not a background process and runs asynchronously at runtime to remove objects. It is expensive to have a really good garbage collector as it requires more memory usage the better the GC. To put it simply, ARC is much faster and more cost effective when it comes to memory handling than the traditional garbage collection.