Garbage Collection

- Garbage collection is the automatic reclamation of computer storage.
  - Program does not need to invoke `free()` or `delete()` methods to explicitly reclaim heap memory.

Garbage Collection

- Garbage collection is the automatic reclamation of computer storage.
  - Program does not need to invoke `free()` or `delete()` methods to explicitly reclaim heap memory.
- The garbage collector's role is to identify data objects that are no longer in use and make their space available for reuse by the mutator.

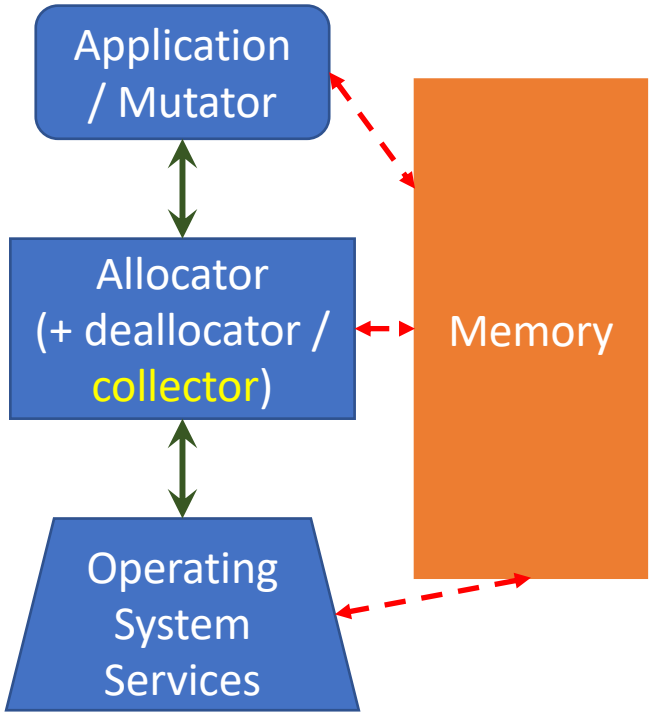
Garbage Collection

- Garbage collection is the automatic reclamation of computer storage.
  - Program does not need to invoke `free()` or `delete()` methods to explicitly reclaim heap memory.
- The garbage collector's role is to identify data objects that are no longer in use and make their space available for reuse by the mutator.
- An object is considered to be garbage and subject to reclamation if it is not reachable by the mutator via any path of pointer traversals.
  - Objects that are potentially reachable through such paths of pointer traversals are said to be live.
  - Liveness is a global property of an executing process.
  - Note that this run-time notion of liveness is very different from the compile-time notion of liveness that we discussed in the context of register allocation.

Motivation for Garbage Collection

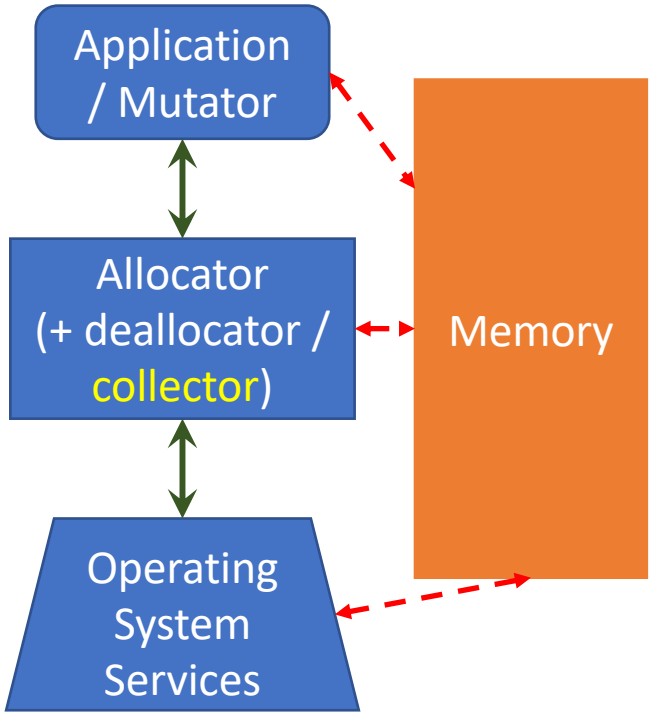
- Necessary for fully modular programming.
  - A method operating on a data structure should by default not need to know what other methods may be operating on the same data structure.
  - If explicit deallocation is the norm, then some module must be responsible for knowing when other modules are no longer interested in the object – thereby introducing unnecessary cross-module dependencies.
- Unnecessary complications of explicit memory management can:
  - Break the basic abstractions of the programming language.
  - Lead to slow memory leaks.
  - Create “heisenbugs”.
  - Make code brittle.

Garbage Collection In The Block Diagram



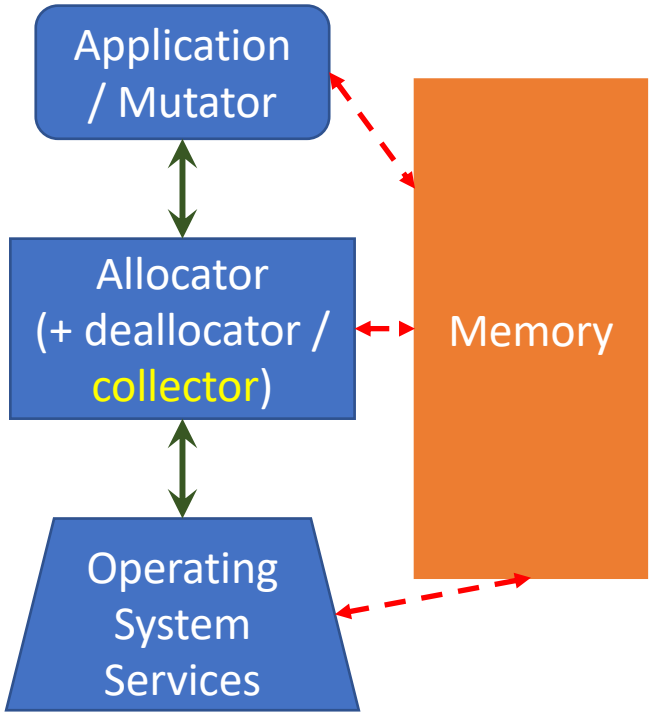
Garbage Collection In The Block Diagram

- The run-time allocation routines perform special actions to reclaim space, as necessary, when an allocation request is not easily satisfied.
  - Calls to the “deallocator” are implicit in calls to the allocator.



Garbage Collection In The Block Diagram

- The run-time allocation routines perform special actions to reclaim space, as necessary, when an allocation request is not easily satisfied.
  - Calls to the “deallocator” are implicit in calls to the allocator.
- Most collectors require some cooperation from the compiler or interpreter.
  - Object formats must be recognizable by the garbage collector.
  - Certain invariants must be preserved by the running code.
  - These requirements do not prevent the use of compiler optimizations.



The Two-Phase Abstraction

- Conceptually, the functioning of a garbage collector consists of two distinct phases (i.e., sets of activities).
  - Garbage detection, i.e., distinguishing live objects from garbage objects through some means.
  - Storage reclamation, i.e., reclaiming the storage of the garbage objects so that it is available for use by the mutator.

The Two-Phase Abstraction

- Conceptually, the functioning of a garbage collector consists of two distinct phases (i.e., sets of activities).
  - Garbage detection, i.e., distinguishing live objects from garbage objects through some means.
  - Storage reclamation, i.e., reclaiming the storage of the garbage objects so that it is available for use by the mutator.

- In practice, these two phases may be functionally or temporally interleaved.

Conservatism in Garbage Collectors

- In order to declare an object as being garbage, the collector needs to be able to prove that it will never be used again.
  - An ideal garbage collector would be able to reclaim every object's space just after the last use of the object.

Conservatism in Garbage Collectors

- In order to declare an object as being garbage, the collector needs to be able to prove that it will never be used again.
  - An ideal garbage collector would be able to reclaim every object's space just after the last use of the object.
- In general, the problem of determining exactly which objects are live is undecidable.

Conservatism in Garbage Collectors

- In order to declare an object as being garbage, the collector needs to be able to prove that it will never be used again.
  - An ideal garbage collector would be able to reclaim every object's space just after the last use of the object.
- In general, the problem of determining exactly which objects are live is undecidable.
- All garbage collectors use *some efficient but conservative approximation to liveness*.
  - In reference counting, the approximation is that an object can't be live unless it is referenced.
  - In tracing garbage collection, the approximation is that an object can't be live unless it is reachable.

Conservatism in Garbage Collectors

- In order to declare an object as being garbage, the collector needs to be able to prove that it will never be used again.
  - An ideal garbage collector would be able to reclaim every object's space just after the last use of the object.
- In general, the problem of determining exactly which objects are live is undecidable.
- All garbage collectors use *some efficient but conservative approximation to liveness*.
  - In reference counting, the approximation is that an object can't be live unless it is referenced.
  - In tracing garbage collection, the approximation is that an object can't be live unless it is reachable.
- Two forms of conservatism.
  - Temporal conservatism occurs when garbage goes uncollected between collection cycles.
  - Topological conservatism occurs when different paths that share an edge in the graph of pointer relationships are not distinguished.

Run-Time Criterion for Liveness

- Defined in terms of a root set of variables and reachability from these roots.

Run-Time Criterion for Liveness

- Defined in terms of a root set of variables and reachability from these roots.
- At a garbage collection point, all globally visible variables and local variables of active methods are considered live.
  - This is a conservative notion.

Run-Time Criterion for Liveness

- Defined in terms of a root set of variables and reachability from these roots.
- At a garbage collection point, all globally visible variables and local variables of active methods are considered live.
  - This is a conservative notion.
- The root set consists of the global variables, local variables on the activation stack, and any registers used by active methods.

Run-Time Criterion for Liveness

- Defined in terms of a root set of variables and reachability from these roots.
- At a garbage collection point, all globally visible variables and local variables of active methods are considered live.
  - This is a conservative notion.
- The root set consists of the global variables, local variables on the activation stack, and any registers used by active methods.
- The set of live objects at a collection point is defined inductively as follows.
  - Heap objects directly reachable from any of the variables in the root set are live.
  - Any object directly reachable from a live object is live.
  - No other object is live.

Object Representations

- Heap objects must be self-identifying.
  - It must easy to determine the type of an object at run-time.
  - Statically typed languages use an extra field containing type information in the hidden headers of allocated objects, and use this information to decode the format of the object.
  - Dynamically typed languages typically use tagged pointers.

Object Representations

- Heap objects must be self-identifying.
  - It must easy to determine the type of an object at run-time.
  - Statically typed languages use an extra field containing type information in the hidden headers of allocated objects, and use this information to decode the format of the object.
  - Dynamically typed languages typically use tagged pointers.
- Actually, we can get by with less for statically typed languages.
  - All we need are the types of the root set variables.
  - From that, we can figure out the types of their referents, and decode the fields in these types, and continue this process transitively.

Object Representations

- Heap objects must be self-identifying.
  - It must easy to determine the type of an object at run-time.
  - Statically typed languages use an extra field containing type information in the hidden headers of allocated objects, and use this information to decode the format of the object.
  - Dynamically typed languages typically use tagged pointers.
- Actually, we can get by with less for statically typed languages.
  - All we need are the types of the root set variables.
  - From that, we can figure out the types of their referents, and decode the fields in these types, and continue this process transitively.
- A class of conservative garbage collectors can work even without this minimal information, at the cost of a more conservative approximation to true garbage.