Disposal and Garbage Collection in C#

...

Gunnar Jackson, Toufic Majdalani, Aayush Bajoria

What is garbage collection

- An automatic memory management process.
- Reclaims memory occupied by objects no longer in use.
- Compacts remaining objects to optimize memory usage.
- For C# we utilize Generational Garbage Collection.

Automatic memory management can eliminate common problems such as forgetting to free an object and causing a memory leak or attempting to access freed memory for an object that's already been freed.

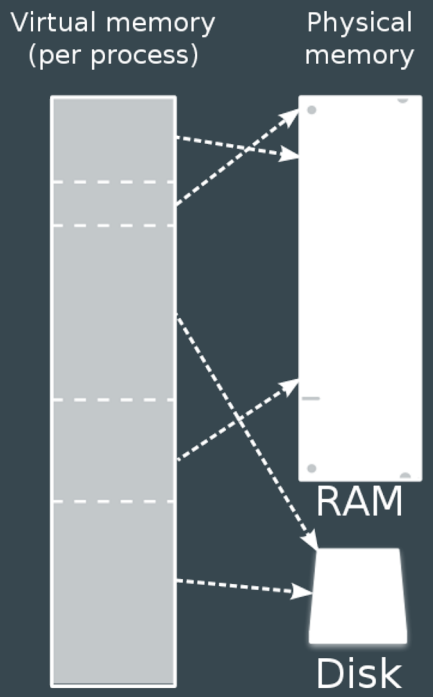
Source: Fundamentals of garbage collection | Microsoft Learn

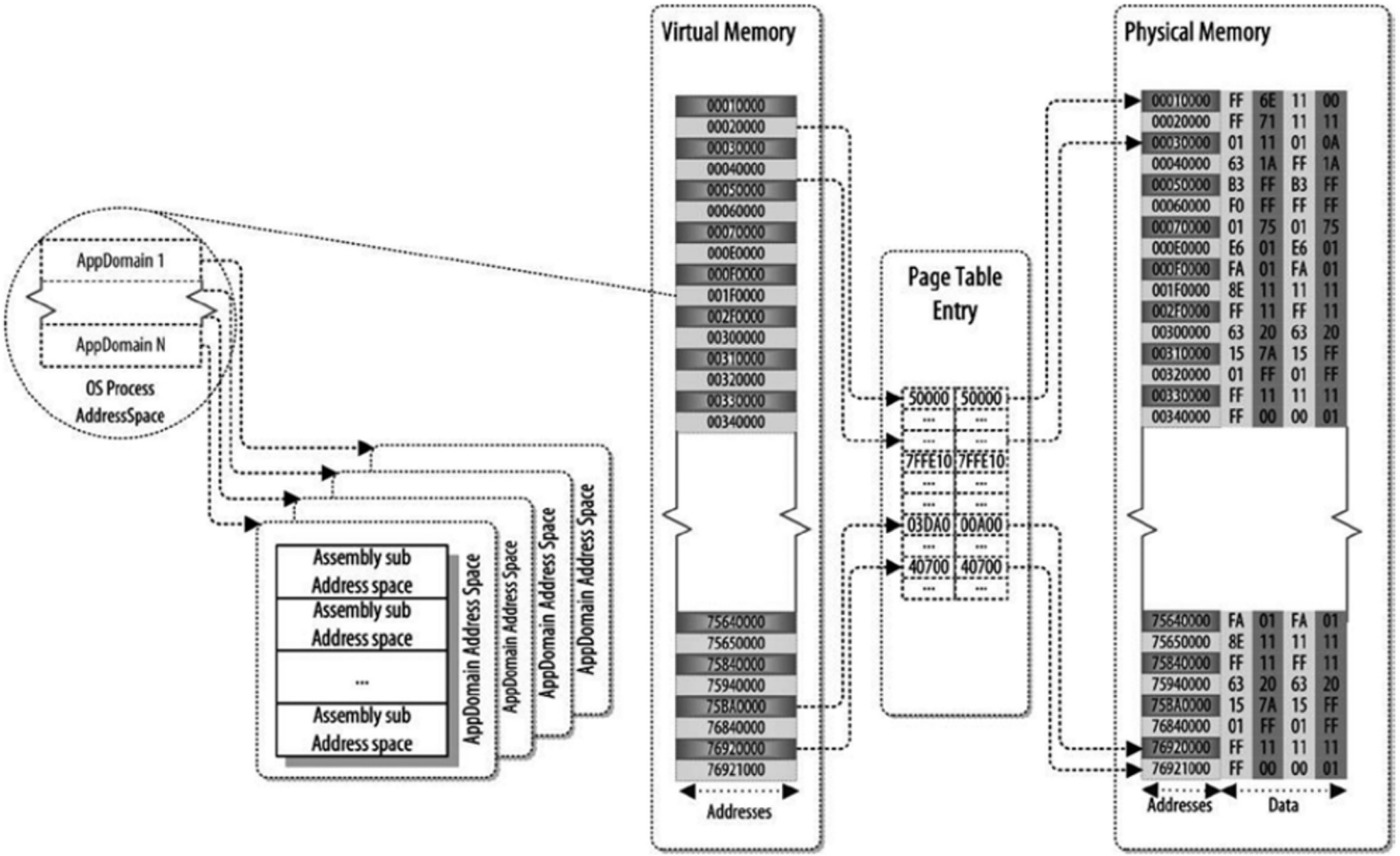
Memory Allocation

- Each process has its own, separate virtual address space.
- During the initialization of a new process, the runtime reserves a contiguous region of address space.
- This reserved address space is called the **managed heap**.
- Garbage Collection does these allocations.

What is virtual memory?

- Abstract view of the physical memory, managed by the OS.
- Is a series of virtual addresses.
- These are translated by the CPU into the physical address when needed.
- Memory management implemented by OS using virtual memory.
- OS uses virtual memory to allocate memory for applications (processes)




Virtual Memory
Physical Memory
Page Table Entry
AppDomain 1
AppDomain N
OS Process AddressSpace
Assembly sub Address space
Assembly sub Address space
...
Assembly sub Address space
AppDomain Address Space
AppDomain Address Space
AppDomain Address Space
AppDomain Address Space
00010000
00020000
00030000
00040000
00050000
00060000
00070000
000E0000
000F0000
001F0000
002F0000
00300000
00310000
00320000
00330000
00340000
75640000
75650000
75840000
75940000
75BA0000
76840000
76920000
76921000
Addresses
50000 50000
7FFE10 7FFE10
03DA0 00A00
40700 40700
00010000 FF 6E 11 00
00020000 FF 71 11 11
00030000 01 11 01 0A
00040000 63 1A FF 1A
00050000 B3 FF B3 FF
00060000 F0 FF FF FF
00070000 01 75 01 75
000E0000 E6 01 E6 01
000F0000 FA 01 FA 01
001F0000 8E 11 11 11
002F0000 FF 11 FF 11
00300000 63 20 63 20
00310000 15 7A 15 FF
00320000 01 FF 01 FF
00330000 FF 11 11 11
00340000 FF 00 00 01
75640000 FA 01 FA 01
75650000 8E 11 11 11
75840000 FF 11 FF 11
75940000 63 20 63 20
758A0000 15 7A 15 FF
76840000 01 FF 01 FF
76920000 FF 11 11 11
76921000 FF 00 00 01
Addresses
Data

Benefits of Garbage Collection

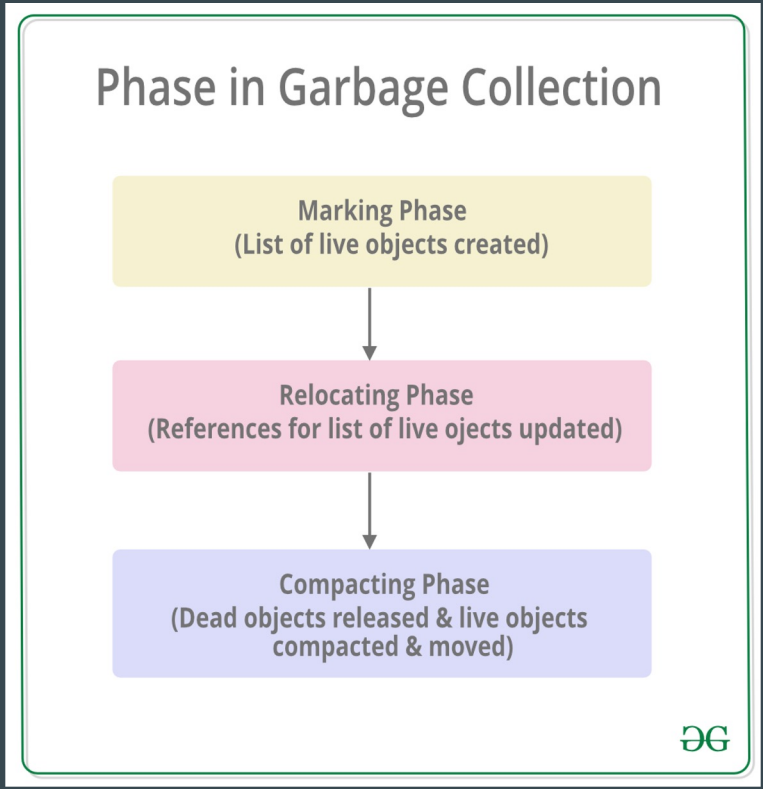
- Free Developers from having to manually release memory
- Allocates objects on the managed heap efficiently
- Managed objects automatically get clean content to start with
    - Constructors don’t have to initialize every data field
- Provides memory safely

Conditions for Garbage Collection

The Garbage collection occurs when one of the following conditions is true:

- The system has low physical memory.
- The memory usage on the managed heap surpasses an acceptable threshold.
- The GC.Collect method is called.

3 Steps of Garbage Collection



Marking Phase

- Creates a list of all the live objects by following the reference to all the root objects
- Objects that are not on the list of live objects can potentially deleted from the heap memory

Relocating Phase

- References of live objects point to a new place in memory.
- Now point to where the objects will be relocated to in the compacting phase.

Compacting Phase

- References of live objects point to a new place in memory.
- Now point to where the objects will be relocated to in the compacting phase.

Heap Generation in Garbage Collection

- Organized into three generations
  - Gen 0, 1, 2
- Memory size of each generation determined at runtime by the CLR
  - Common Language Runtime
- Optimization Engine determines which objects go into which generation.

Generation 0

- All the short-lived objects such as temporary variables are contained in the generation 0 of the heap memory.
- All the newly allocated objects are also generation 0 objects implicitly unless they are large objects.
- In general, the frequency of garbage collection is the highest in generation 0.

Generation 1

- If space occupied by some generation 0 objects that are not released in a garbage collection run, then these objects get moved to generation 1.

- The objects in this generation are a sort of buffer between the short-lived objects in generation 0 and the long-lived objects in generation 2.

Generation 2

- If space occupied by some generation 1 objects that are not released in the next garbage collection run, then these objects get moved to generation 2.

- The objects in generation 2 are long lived such as static objects as they remain in the heap memory for the whole process duration.

Garbage collection in CLR (managed heap)

- Common Language Runtime (CLR):
    - A component of the .NET framework that provides a runtime environment for executing C# and other .NET languages.
- An application's roots include static fields, local variables on a thread's stack, CPU registers, GC handles, and the finalize queue.
- The garbage collector uses this list to create a graph that contains all the objects that are reachable from the roots.
- The garbage collector considers unreachable objects garbage and releases the memory allocated for them.
- Organized into generations (0, 1, and 2) to optimize garbage collection (relative to frequency of use).
- As it discovers each unreachable object, it uses a memory-copying function to compact the reachable objects.
- Since memory compaction is expensive, the runtime allocates memory for large objects in a separate heap.

Source: Fundamentals of garbage collection | Microsoft Learn

List of garbage collection coverage we will showcase

- Static variables
- WPF bindings
- Caching functionality
- Captured members

Static keyword

- **Static** variables, collections, and static events in-particular are **GC Roots**, so they are never collected by the GC.
- Static keyword represents data that is to be shared across all instances of method, and for the lifetime of the program.
- In C#, static variables are not garbage collected because they are stored in a different heap, which remains in memory for the entire lifetime of the application, separate from the managed heap.

WPF Bindings

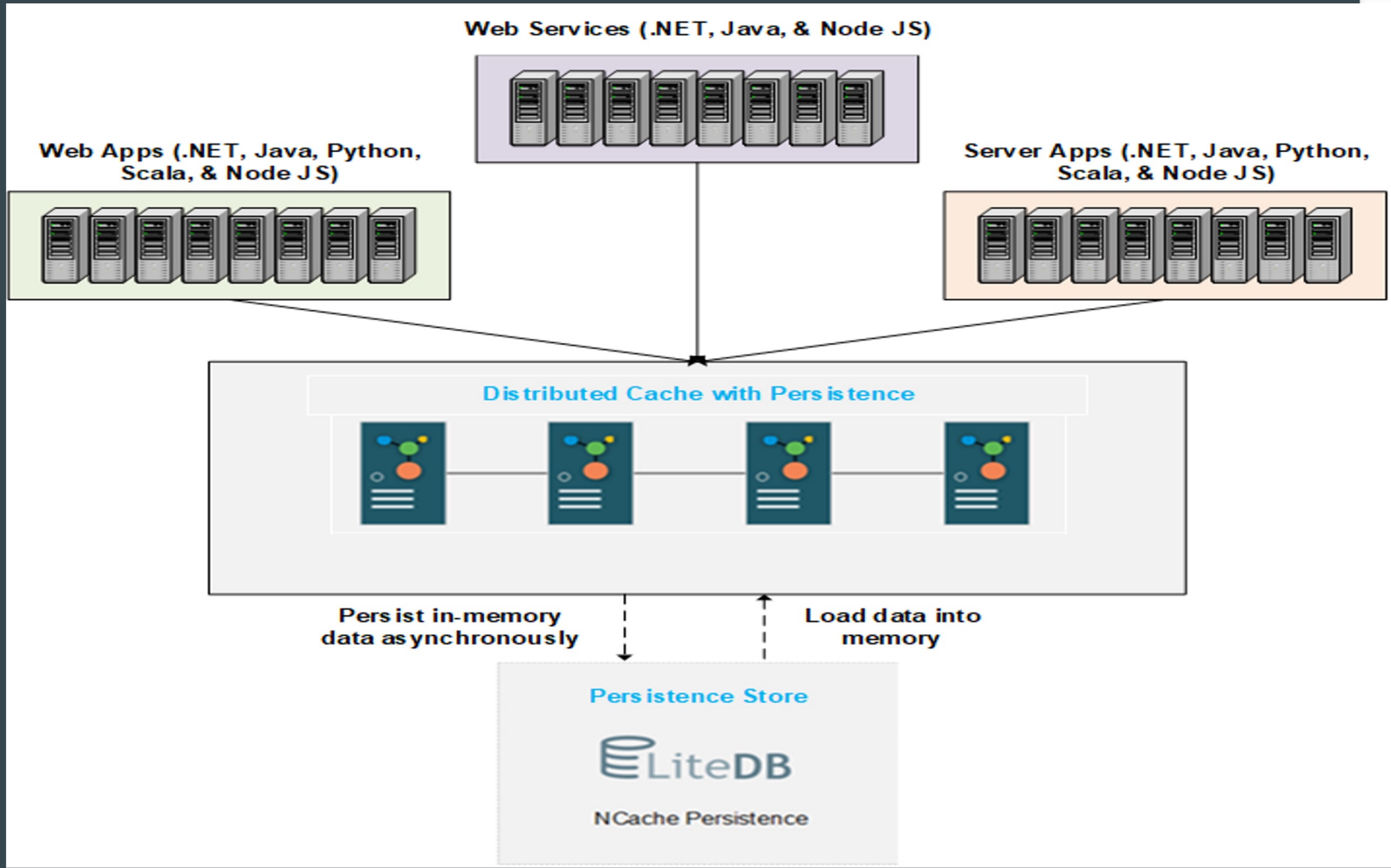
- “In applications, it's possible that handlers attached to event sources won't be destroyed in coordination with the listener object that attached the handler to the source”
- “For example, the C# statement source.SomeEvent += new SomeEventHandler(MyEventHandler)”, creates a strong reference to the event from the source to the listener.
- “Unless the event handler is explicitly unregistered, the object lifetime of the listener will be influenced by the object lifetime of the source.”
- This can happen when a long-lived object (e.g., a ViewModel) subscribes to events from a short-lived object (e.g., a View).

Caching functionality

- Caching functionality works in cached-size memory containers to store & reuse any form of data
- Three instances of caching functionality:
  - In-memory cache:
    - Implements when a single process is executed.
    - Same process can be implemented in multiple instance having separate cache containers
  - Persistent in-process cache
    - Cache memory is stored in a process memory using either a file or data storage
    - When set, a process can reset without deleting the cache that is set in the program
  - Distributed cache
    - The ability to store cache on several machines or servers
    - Can be implemented using external libraries or programs

Disposable

- An object that can be discarded into garbage collection after its use
- In C#, the garbage collector automatically deallocates memory for objects that are no longer being used
- Helps prevent memory leaks by releasing any unmanaged resources in memory to the garbage when the Disposable object
- The Disposable Object is implemented in C# using the IDisposable interface


Web Services (.NET, Java, & Node JS)
Web Apps (.NET, Java, Python, Scala, & Node JS)
Server Apps (.NET, Java, Python, Scala, & Node JS)
Distributed Cache with Persistence
Persist in-memory data asynchronously
Load data into memory
Persistence Store
LiteDB
NCache Persistence

Captured members

- Members such as objects or variables that are referenced in anonymous delegates/classes and lambda expressions
- When a variable is declared in a method, and then it is used in a lambda expression or anonymous class in the same method creating a form of closure blocking the garbage collector from dumping it from the memory.
- This helps with being able to save sets of data points that want to be tested over and over again without the garbage collector removing them from memory & creating new instances of them.

```
private void RefreshButton_Click(object sender, RoutedEventArgs e)
{
    Person selectedPerson = PersonListBox.SelectedItem as Person;

    var repository = new PeopleRepository();
    repository.GetPeopleCompleted +=
        (repoSender, repoArgs) =>
        {
            PersonListBox.ItemsSource = repoArgs.Result;
            if(selectedPerson != null)
            {
                foreach(Person person in PersonListBox.Items)
                    if(person.FirstName == selectedPerson.FirstName &&
                        person.LastName == selectedPerson.LastName)
                    {
                        PersonListBox.SelectedItem = person;
                    }
            }
        };
    repository.GetPeopleAsync();
    selectedPerson = null;
}
```

GC Demo Overview

- The following program calculates the amount of memory stored in cache & outputs the average amount of memory stored in the Garbage Collector along with the average percentage
- It implements the IDisposable interface which disposes of any unmanaged variables & attributes within memory which accumulates to the percentage in the garbage collector
- This demonstrates an overview of caching functionality & captured members in many different programs

Bibliography

- https://michaelscodingspot.com/find-fix-and-avoid-memory-leaks-in-c-net-8-best-practices/
- https://michaelscodingspot.com/cache-implementations-in-csharp-net/
- https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals
- https://stackoverflow.com/questions/16740741/does-static-members-prevent-the-instance-class-from-being-gced
- https://www.youtube.com/watch?v=MqxRpXUVZ8E
- Weak event patterns - WPF .NET | Microsoft Learn

ANY
QUESTIONS