A Bit About Segments

Each managed process has its own heaps for both the SOH and the LOH. Initially, at the start of a process's execution, two memory segments are requested from the OS, one for the SOH and one for the LOH.

Segments are just units of memory allocation; when the GC needs to expand the heap, it requests an additional segment. Equally, when a heap has compacted, or realizes it has excess free space, it can release segments back to the OS.

Segment size is dynamic, is tuned by the runtime, and depends on the GC mode you are running: either **Workstation** or **Server** GC. For example, a typical Workstation segment would be 16 Mb. We'll cover the differences between Server and Workstation GC later.

At the start of a process's execution, the SOH is allocated a single segment, called the ephemeral segment, to be used for Gen 0 allocations and promotions to Gen 1 and Gen 2. When it's full, a GC takes place and a new segment is added. Any surviving Gen 0 objects are copied to the new segment (becoming Gen 1 objects), and the old segment is left with just the Gen 2 objects. Figure 3.2 illustrates the process.

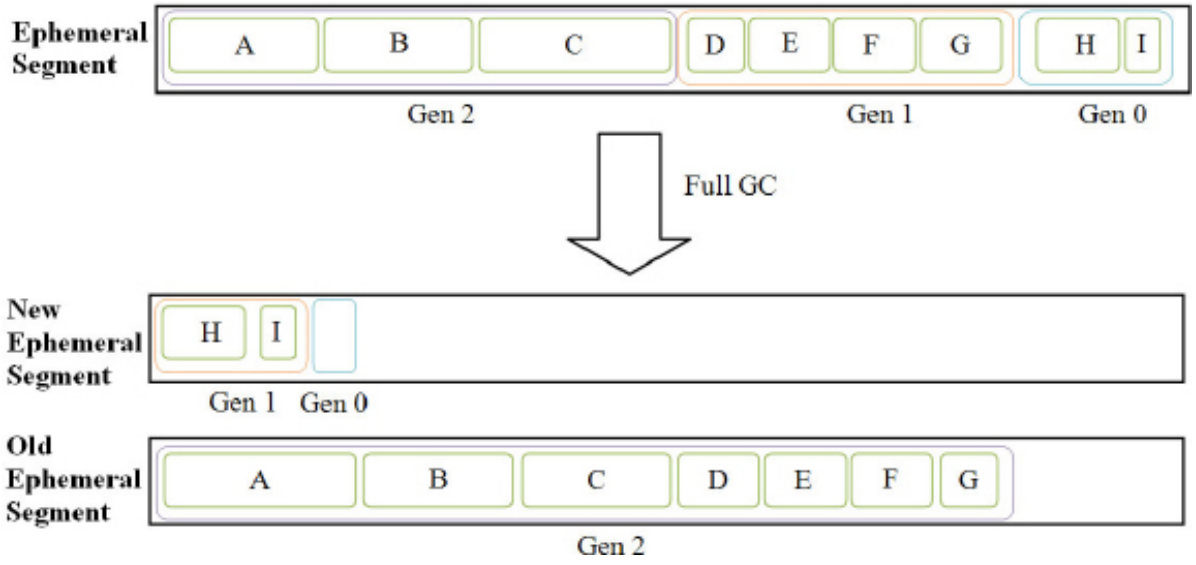

Figure 3.2: The ephemeral segment (assuming all objects are rooted).

In Figure 3.2, the ephemeral segment contains rooted objects from all three generations. To prevent the diagram from getting too messy, I've not included any root references on the diagram, so let's assume all of the objects are rooted.

Over time, the ephemeral segment becomes so full that a full GC takes place, and a new segment is requested from the OS which becomes the new ephemeral segment. Objects I and H are copied to it (as they are rooted), joining Gen 1, and everything else is left behind on the old ephemeral segment. If you're thinking what I think you're thinking, you're right! Gen 2 *is* just a collection of old ephemeral segments.

Segments and Pages

Segments are made up from virtual memory pages requested from the Virtual Memory Manager (VMM) of the OS. The VMM maps the physical memory and the disk-based page file to a single virtual addressable space, which is subdivided into pages.

Pages can be in one of three states: **free**, **committed** or **reserved**.

- **Free pages** are available to be allocated to a requesting thread.
- **Committed pages** are allocated and, ultimately, translate to pages in physical memory. As such, they can be swapped to and from the page file when necessary (paging).
- **Reserved pages** are a low overhead means of reserving virtual memory for future use, but don't translate to physical memory

and don't incur paging.

Applications can either reserve pages and then commit them later when needed, or reserve and commit all at once using specific API calls.

When the GC builds a segment, it reserves the required number of pages and then commits them when needed. Redundant segments can ultimately be released back to the OS.