Placement of old generation on NV-DIMMs.

There are two levels at which we need enabling, first being the memory management level and second is at the GC level.
At the memory management level we need to define new abstractions for VirtualSpace to manage memory which is backed by NV-DIMMs. At GC level we need to use the correct VirtualSpace abstractions and do additional handling to manage young and old generation. The following text describes the design for the two GCs.

ParallelScavenge:
ParallelScavenge takes 'ReservedSpace' corresponding to the Heap and creates 'AdjoiningVirtualSpaces' which comprises of two 'PSVirtualSpace's corresponding to the two generations. These virtual spaces are assigned to PSOldGen and PSYoungGen.
'AdjoiningVirtualSpace' maintains an internal boundary within the reserved heap space, lower part of it belongs to PSOldGen->PSVirtualSpace and higher part belongs to PSYoungGen->PSVirtualSpace.

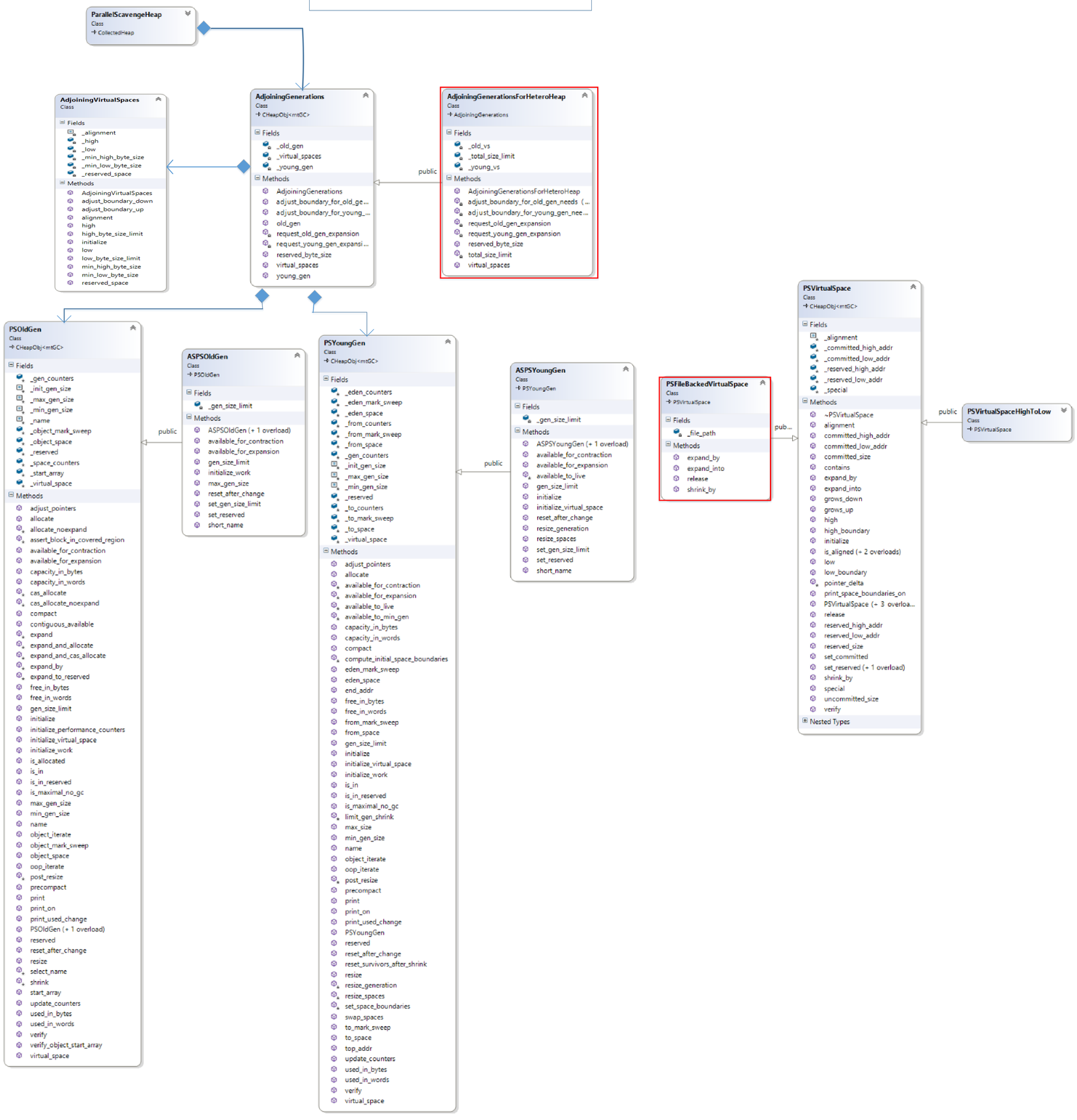
Depending on value of AdaptiveGCBoundary flag there are two cases:
1. AdaptiveGCBoundary = false (default):
    In this case, the boundary in AbjoiningGenerations is fixed and each generations expand/shrink within their respective reserved spaces.
    This case can be handled by creating a subclass of 'PSVirtualSpace' called 'PSFileBackedVirtualSpace' which manages memory exposed as filesystem (such as NV-DIMM). It has a file descriptor as its member and overrides calls to expand() and shrink() the virtual space.

2. AdaptiveGCBoundary = true:
    In this case, ParallelScavenge has more flexibility to size the generations. If the desired size of a generation is more than its reserved space, the boundary in 'AdjoiningVirtualSpaces' is moved to increase the reserved space for that generation.
    This case is a little tricky to implement, because adjusting the boundary would require changing physical memory mapping of the affected memory space (e.g. pages mapped to DRAM will be mapped to NV-DIMM). Such remapping is known to be very costly due to tlb miss penalties.
    To avoid this we need to have non-overlapping reserved memory space for old and young virtual spaces. We would need to reserve more than Xmx memory; reserved memory = (max_size_of_young + max_size_of_old).
    The young and old virtual spaces are assigned these non-overlapping reserved memories. To expand the committed memory of one virtual space, we need to shrink (uncommit) the other virtual space. In other words, we need an equivalent behaviour as adjust_boundary_up/down() calls in AdjoiningVirtualSpaces.
    To achieve this, we can implement a specialized implementation of 'AdjoiningGenerations' called 'AdjoiningGenerationsForHeteroHeap'. This implementation overrides calls such as adjust_boundary_*() and request_*_gen_expansion(). The overridden functionality maintains the invariant that total committed memory before and after expanding/shrinking of generations is same.

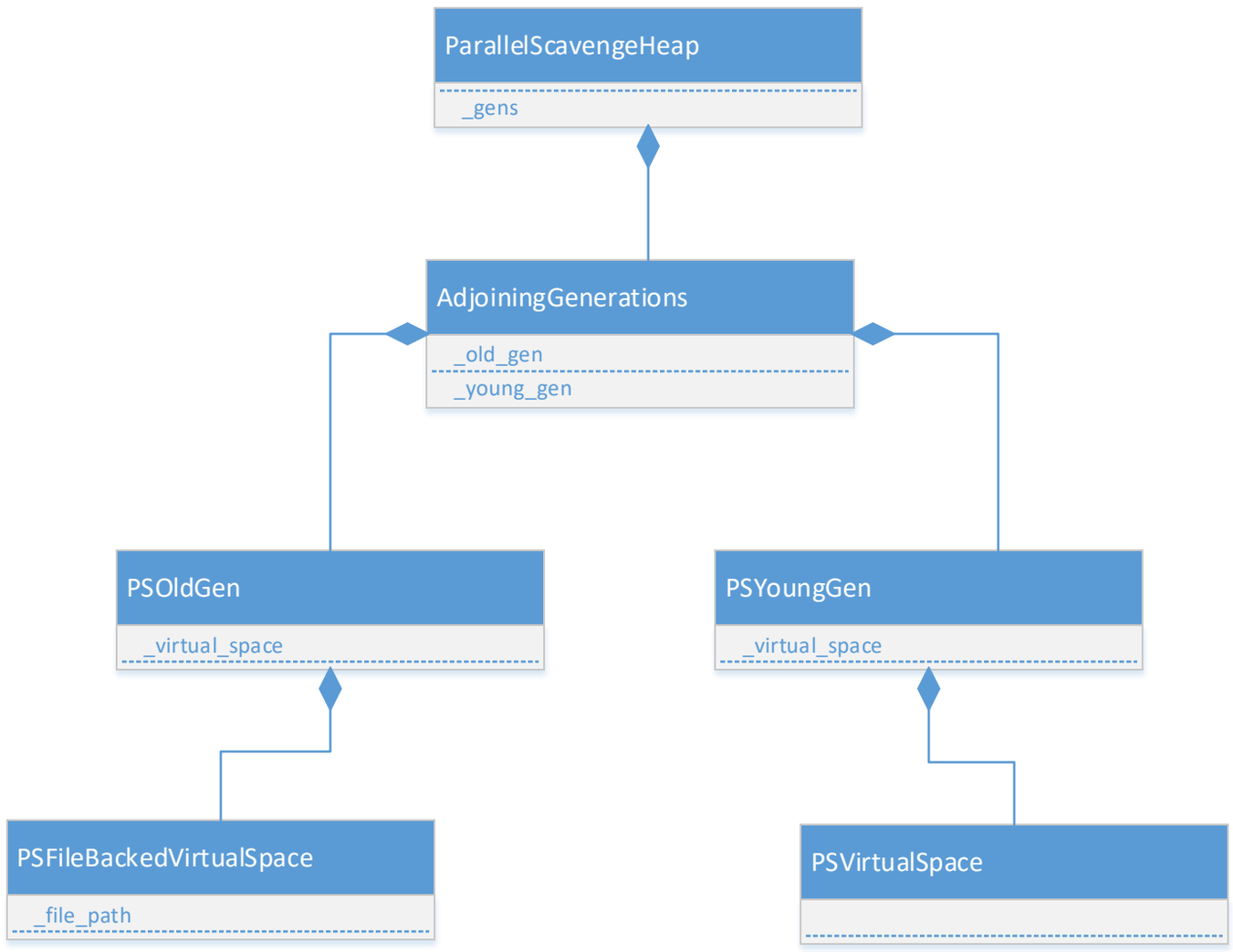
G1GC:
G1 divides the reserved heap space into regions; a region can end up as old, young or humongous. Thus old generation is not a range of memory address, its a collection of region which are spread throughout the heap. To avoid remapping of virtual pages to physical pages when a region gets reassigned to a different generation, we need non-overlapping reserved

memories for old and young regions. We reserve 2*Xmx memory; Xmx memory for young regions and remaining Xmx memory for old regions.
We can create a sub-class of HeapRegionManager called 'HeapRegionManagerForHeteroHeap' which manages regions which are physically backed by DRAM and regions backed by NV-DIMM. HeapRegionManagerForHeteroHeap can be internally composed of HeapRegionManager for DRAM regions and HeapRegionManager for NV-DIMM regions (this conceptual composition does not need to be implemented as separate classes)
This class overrides the api used by G1CollectedHeap to allocate new region, expand heap, allocate humongous regions, etc. This class maintains the invariant that total number of committed regions is less than current size of heap. E.g. if G1 needs more DRAM regions than available at a given point, unused regions from NV-DIMM have to be uncommitted so that new regions in DRAM can be committed.

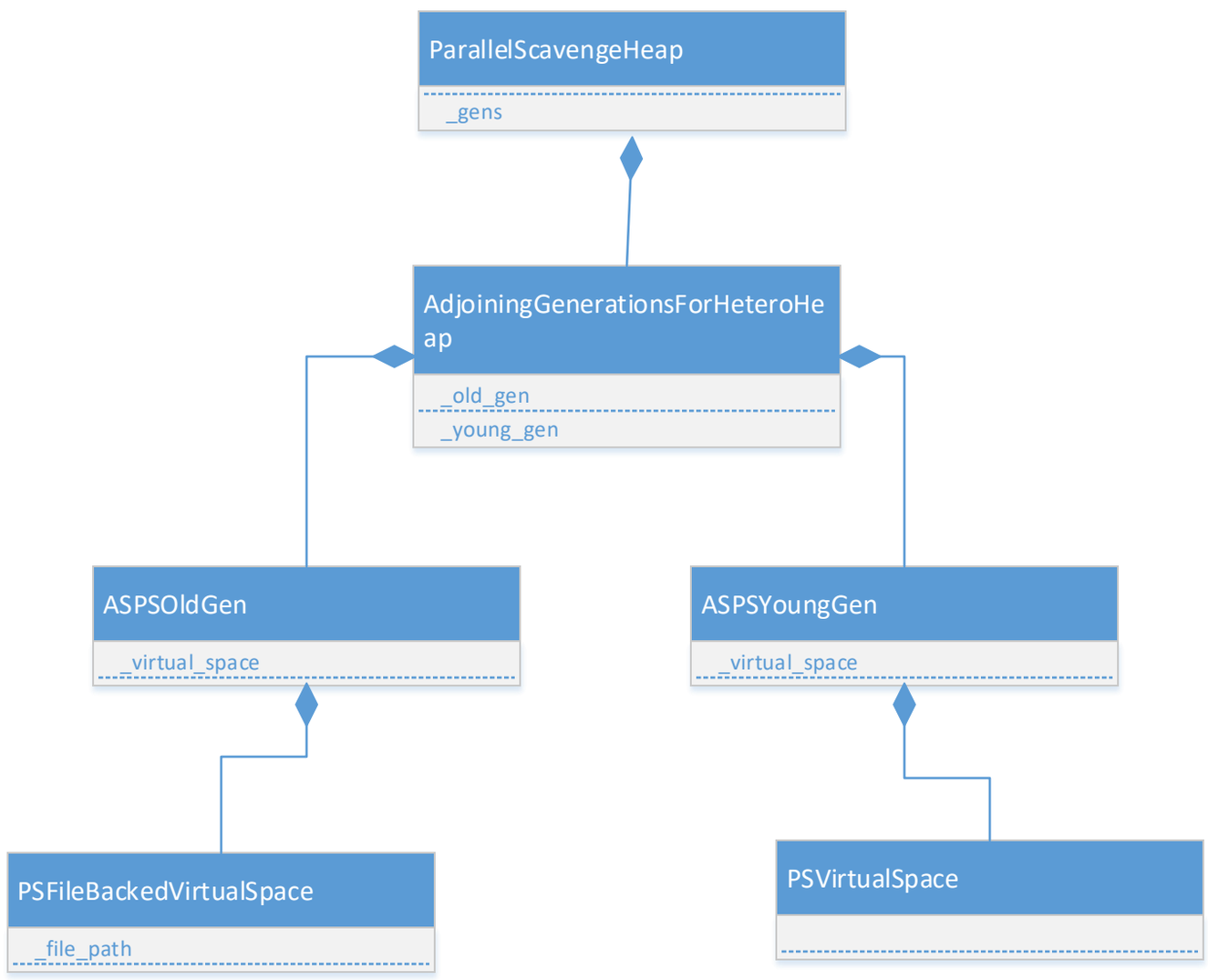
Class diagram for ParallelScavenge GC (new classes in red)



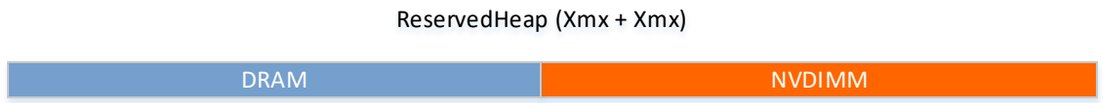
Object diagram when
UseAdaptiveGCBoundary == false

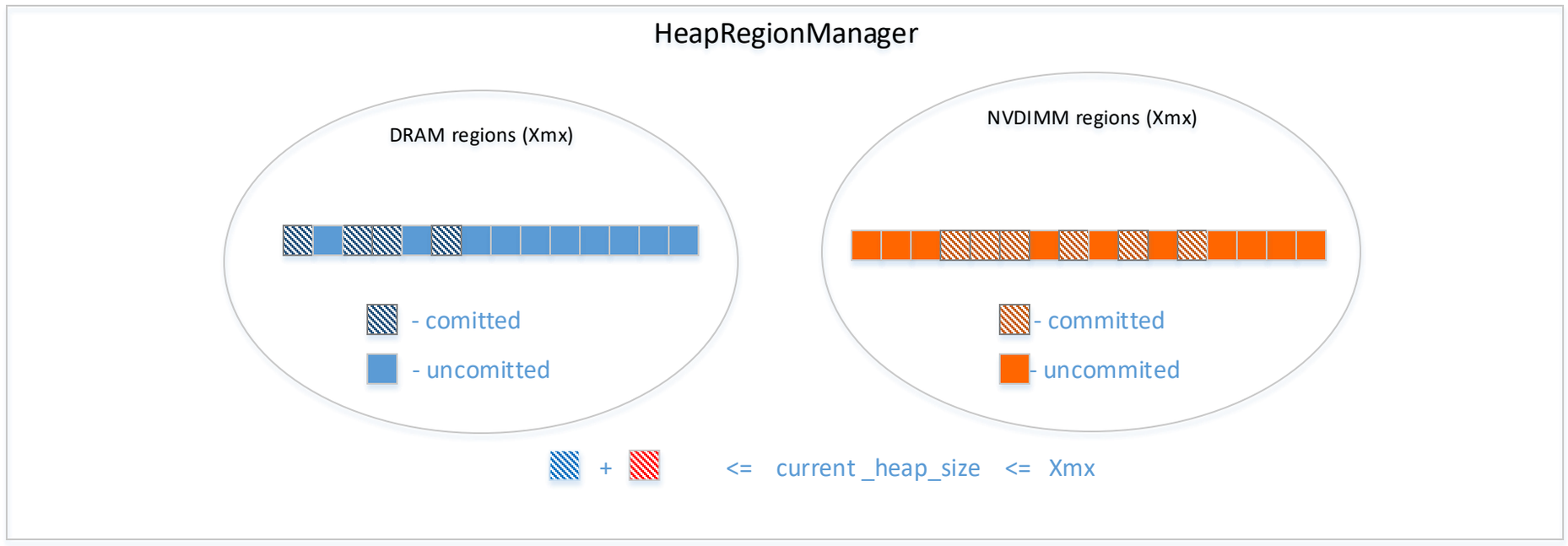


Object diagram when
UseAdaptiveGCBoundary == true

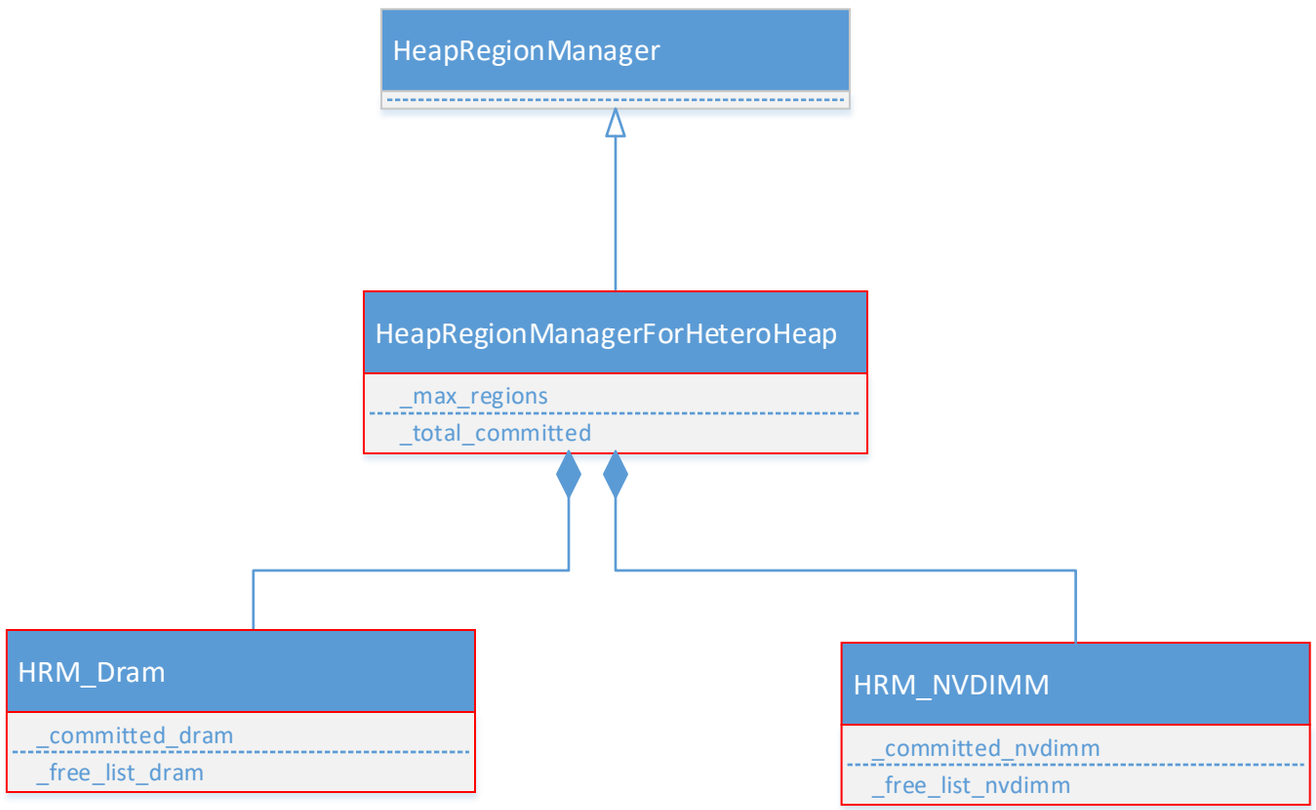


G1 GC design


ReservedHeap (Xmx + Xmx)
DRAM
NVDIMM


HeapRegionManager
DRAM regions (Xmx)
- comitted
- uncomitted
NVDIMM regions (Xmx)
- committed
- uncommited
+
<=
current _heap_size
<=
Xmx

G1 Class diagram



UML Sequence diagram showing interaction between objects