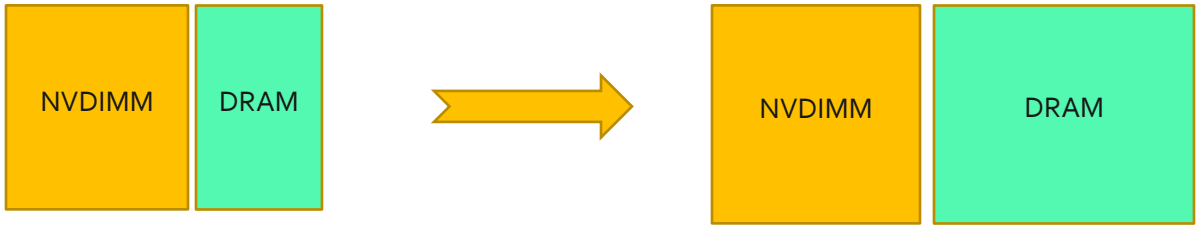
G1GC NVDIMM SUPPORT

HEAP LAYOUT

G1GC supports –XX:MaXGCPauseMillis parameter for automatic resizing of heap. G1 heap can only be expanded.

Without NVDIMM in picture, all parameters are supported as expected. With NVDIMM added, user only supplies –XX:AllocateOldGenAt to tell where file backed by NVDIMM is located so that JVM can use it.



User does not supply any sizing related option for NVDIMM. Maximum heap that can be used for given JVM (Xmx) is reserved and committed for NVDIMM based on **G1MaxNewSizePercent** parameter (under unlock experimental on JDK 11, use –XX:+UnlockExperimentalVMOptions). Based on this, heap is divided for Young on DRAM and Old gen on NVDIMM. Regions are also mapped and managed accordingly (i.e. expand and commit ALL NVDIMM regions in one shot based on Heap percentage remaining after assigning young on DRAM). For DRAM, JVM continues to start with Initial Heap Size and then continue to expand as needed.

JVM Max heap size and Initial heap size as well as all other ergonomic parameters remain unchanged whether NVDIMM is used or not.

JVM when recognizes the presence of NVDIMM, it reserves (Xmx - MaxYoungGenSize) memory at the top part of heap and commits it as we don't want to keep resizing NVDIMM and rest of the heap is laid out as it would have been laid out with out NVDIMM. JVM usually sets it up to Xmn size or InitialHeapSize if Xmn is not provided.

JVM expands DRAM memory as and when needed based on pause time constraints provided. Regions are laid out from NVDIMM to DRAM and are expanded as needed for DRAM only. NVDIMM region does not get resized as we are ONLY putting Old regions there and it is reserved and committed to Maximum Heap size provided from the start. This enables JVM to only do resizing in DRAM from tail end of regions.

All eden regions come from DRAM and old regions come from NVDIMM. JVM already implements taking OLD regions from top the the region free list and young regions from bottom of the list based on is_survivor check.

If regions are tagged as Old due to evacuation failure, we leave them on DRAM .. There is no effort applied to copy these to NVDIMM.

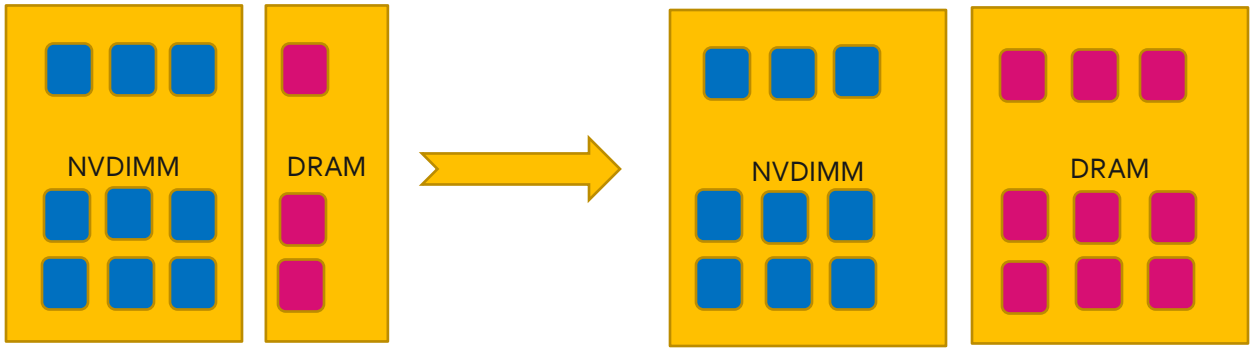


Figure above shows how regions are constructed and expanded as needed basis.

JVM commits NVDIMM based on maximum heap size know for that JVM instance and then expands DRAM regions (shown above in red) as and when needed to DRAM memory.

New region function passes is_survivor parameter indicating whether it is interested in Old or Young region. This ends up calling remove_region where based on is_survivor being true or not, regions are removed from free list; from head (if is_survior is true passing !is_survior making list to give up regions from head) or from tail (if Eden) based on is_survivor being false indicating !is_surviror during call to tell remove region to get regions from tail.

When regions are being removed from Tail, JVM checks if regions free_list length is greater than regions free_list length for NVDIMM first. As JVM does not want to give out regions from tail if region free list is already at the boundary of NVDIMM.

Regions from Head (old regions) can continue to come out as long as there are free regions and we only return NULL when region free list is empty.

This heap resizing happens at safepoint (see function **do_collection_pause_at_safepoint**) or whenever GC is done based on pause-time constraints are checked by calling expand which calls expand_at for given region index and size. Region index passed in always zero as internally during expansion, JVM tries to locate last index of regions that was committed and then add regions requested to the free list.

All changes are localized with in AllocateOldGenAt check. This is set when JVM can successfully mmap for Address that is below DRAM for the max heap size so that there is no change needed when –XX:-UseCompressedOops is used (i.e. JVM can continue to grow heap downwards).

Narrow_oop_base and heap base etc.. related checks are still honored even if NVDIMM is added. This means source does not disable any asserts for any reasons when NVDIMM is added. Asserts related to reserved space are modified to include NVDIMM.

POGC NVDIMM SUPPORT

This is done by taking over commit/uncommit functions which are implemented for file descriptors and offset as input (for commit). As these are called from OldGen (_old_rs), JVM directs all these to NVDIMM. Rest of the JVM continues to work with out making any other assumptions as how memory is mapped is not going to factor in as to how heap gets expanded (i.e. YoungGen heap is expanded and shrunk on DRAM where as OldGen is expanded and shrunk on NVDIMM if AllocateOldGenAt parameter is passed. This implementation is different from G1GC implementation where (1-G1MaxNewSizePercent) is mapped directly on NVDIMM in one go.

POGC + NVDIMM supports UseAdaptiveGCBoundary as well (off by default) as rest of the VM can expand_by and shrink as if NVDIMM is normally mapped memory (offset is maintained correctly as it was done for DRAM and is used to commit to NVDIMM).

On Windows, since there is no support for incremental mmaping, JVM fixes the size of OldGen to max allowed and maps the whole thing in one shot. This way OldGen is fully expanded from get go (as we did for G1GC) and YoungGen can expand/shrink.