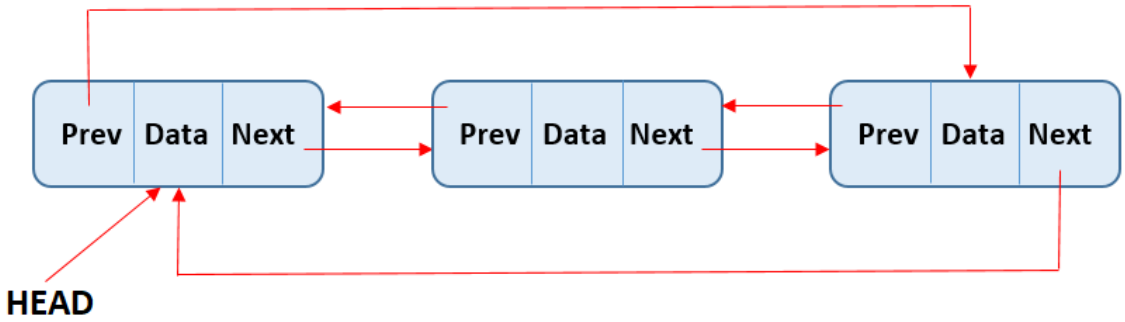
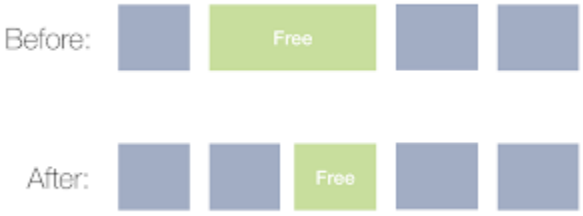
The General Dynamic Storage Allocation Problem

Data Structure: We used a doubly linked list to store different sized chunks of memory with block nodes. Each node holds the size of the block, ptr to the start of the block, and a valid field to determine if the block is being used or is currently free.



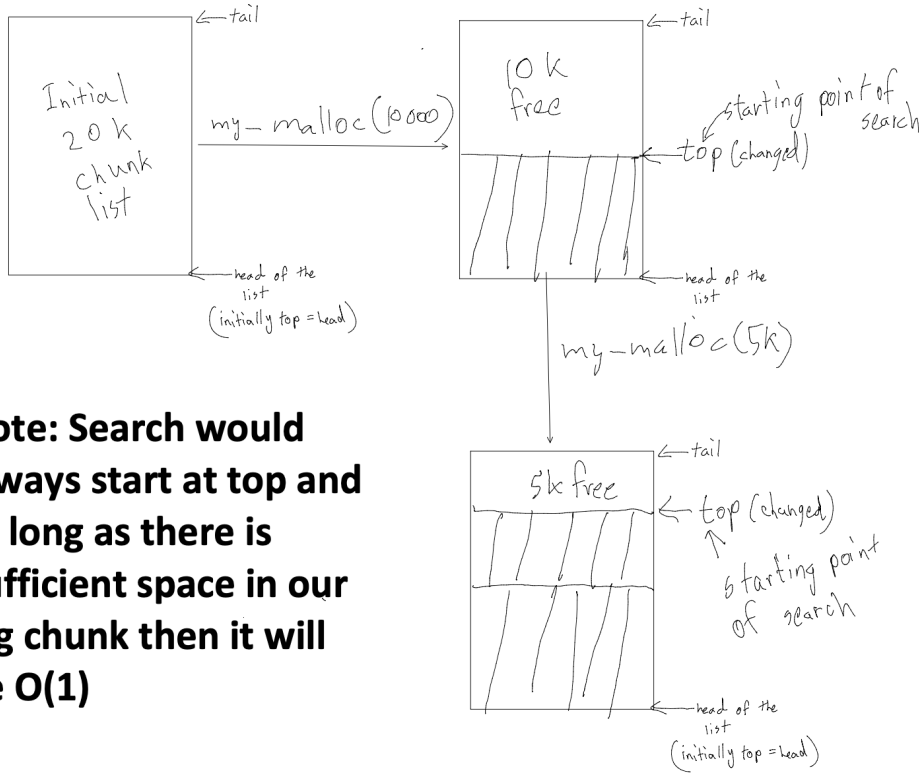
Internal Fragmentation: We used MVT structuring to allocate a new smaller chunk from an existing bigger chunk of memory. By splitting bigger blocks based on size, we reduced any possibilities of internal fragmentation from occurring.



External Fragmentation: To help reduce external fragmentation, the my_free() method frees the block and also does some coalescing. It tries merging adjacent blocks of memory together into one bigger chunk of memory. This makes it so the program will not end up with numerous small chunks of memory.

Main Algorithm: Our implementation of malloc is a modified version of the first-fit algorithm. In general, first fit allocations are efficient but aren't as fast as O(1) since it always starts from the head and iterates until it finds the first free block. This leads to searching and going through both already allocated blocks and freed blocks. Our tweak in the first fit algorithm made it so that whenever an allocation occurs, our head moves to the next biggest available memory chunk so the next malloc call would be pointing to the memory that is ready to be used without any iteration. This is somewhat like a stack version of malloc where the chunks are getting pulled and the top is also shifting down towards the tail. Malloc would automatically start allocating from the next free chunk until we reach the end of the list(tail) or if the block is too small.

However, we do know the downside of this and that is when the top reaches the end of our initially allocated memory list, and there is no way for the top pointer to go beyond the end of the list. To address that problem we made a switch functionality that would switch our implementation to our first-fit algorithm when the top reaches the tail. The switch takes the top pointer and points back to the head of the list and also signals that now onwards start doing basic first fit i.e start every time from the head of the list and look for the first available chunk and if the size requested fits, then that chunk of memory is sent back to the user.



Note: Search would always start at top and as long as there is Sufficient space in our big chunk then it will be O(1)

Conclusion: In conclusion our malloc isn't perfect in both speed and fragmentation. Although the speed is O(1) for malloc initially, however the speed decreases after some repetition of malloc calls as it starts transitioning into our first-fit algorithm. Finally, our implementation does take care of fragmentations in many ways but for the long run there will still be some fragmentation.