HEAP MANAGEMENT

- The heap is the portion of the store that is used for data that lives indefinitely, or until the program explicitly deletes it. The local variables may be needed by many programming languages even after the procedure in which was declared gets completed.
- Memory Manager is a subsystem that allocates and deallocates space within heap. It serves as an interface between application programs and the operating systems. It is also responsible for deallocation when operation like free or delete is used by languages C or C++ manually.
- Garbage Collection is an important subsystem of Memory Manager. It is the process of finding spaces within the heap that no longer need and therefore be allocated for other data item. Java uses Garbage collector for deallocating memory.

Memory Manager

The memory manager keeps track of all the free space in heap storage at all times. It performs two basic functions:

- Allocation: When a program requests memory for a variable or object, the memory manager produces a chunk of contiguous heap memory of the requested size.
- Deallocation: The memory manager returns deallocated space to the pool of free space, so it can reuse the space to satisfy other allocation requests.

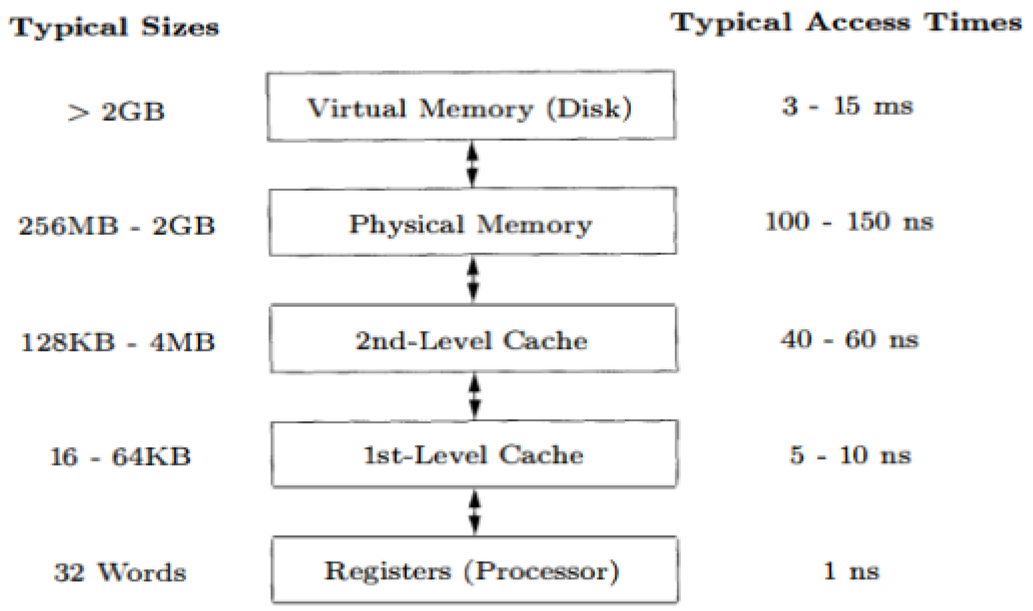
Properties of memory managers:

- Space Efficiency: A memory manager should minimize the total heap space needed by a program. Doing so allows larger programs to run in a fixed virtual address space. Space efficiency is achieved by minimizing "fragmentation,".
- Program Efficiency: A memory manager should make good use of the memory subsystem to allow programs to run faster. By attention to the placement of objects in memory, the memory manager can make better use of space and, hopefully, make the program run faster.
- Low Overhead: Memory allocations and deallocations must be as efficient as possible. Should minimize of the fraction of execution time spent performing allocation and deallocation.

The Memory Hierarchy of a Computer

- A processor has a small number of registers; whose contents are under software control.
- Next, it has one or more levels of cache, usually made out of static RAM, that are kilobytes to several megabytes in size.
- The next level of the hierarchy is the physical (main) memory, made out of hundreds of megabytes or gigabytes of dynamic RAM.
- The physical memory is then backed up by virtual memory, which is implemented by gigabytes of disks.

- Upon a memory access, the machine first looks for the data in the closest (lowest- level) storage and, if the data is not there, looks in the next higher level, and so on



- Registers are scarce, so register usage is managed by the code that a compiler generates. All the other levels of the hierarchy are managed automatically.
- Caches are managed exclusively in hardware, for relatively fast RAM access times. Because disks are relatively slow, the virtual memory is managed by the operating system.
- Data is transferred as blocks of contiguous storage. Between main memory and cache, data is transferred in blocks known as cache lines, which are typically from 32 to 256 bytes long. Between virtual memory (disk) and main memory, data is transferred in blocks known as pages, typically between 4K and 64K bytes in size.

Locality in Programs

- A program has temporal locality if the memory locations it accesses are likely to be accessed again within a short period of time.
- A program has spatial locality if memory locations close to the location accessed are likely also to be accessed within a short period of time.

Reducing Fragmentation

- At the beginning of program execution, the heap is one contiguous unit of free space. As the program allocates and deallocates memory, this space is broken up into free and used chunks of memory.
- The free chunks of memory as holes. With each allocation request, the memory manager must place the requested chunk of memory into a large-enough hole.
- Unless a hole of exactly the right size is found, we need to split some hole, creating a yet smaller hole.

Best-Fit and Next-Fit Object Placement:

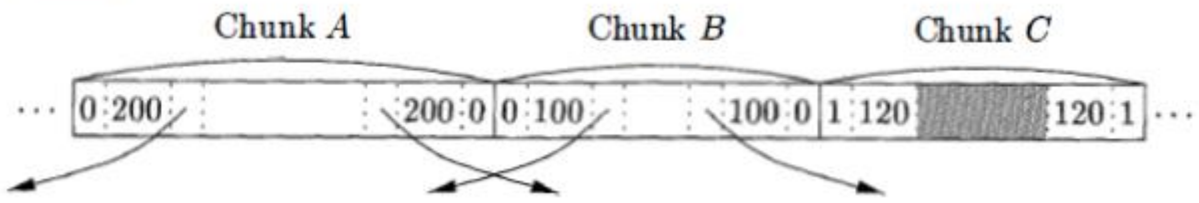
- Fragmentation can be reduced by controlling the placement of new objects by the memory manager.
- The best-fit algorithm can be used to satisfy the larger requests.
- An alternative called the first-fit algorithm, places in the first hole which fits in and takes less time compared to best fit.
- To implement best-fit placement more efficiently, we can separate free space into bins, according to their sizes. There is a bin for every multiple of 8-byte chunks from 16 bytes to 512 bytes. There is always a chunk of free space that can be extended by requesting more pages from the operating system which is called the wilderness chunk.

Managing and Coalescing Free Space:

- When an object is deallocated manually, the chunk can be made free and if possible can combine(Coalesce) with the adjacent chunk to form a larger chunk.
- If a bin for chunks is of fixed size, then no need to combine.
- It is simpler to keep all chunks of one size in as many pages as needed. A simple allocation/deallocation scheme it to keep a bitmap, one bit for each chunk in the bin.
- A 1 indicates the chunk is occupied; 0 indicates it is free. When a chunk is deallocated, we change it from 1 to 0. When we need to allocate a chunk, we find any chunk with a 0 bit, change that bit to a 1, and use the corresponding chunk.

1. Boundary Tags: At both the low and high ends of each chunk, whether free or allocated, we keep vital information. At both ends, we keep a free/used bit that tells whether or not the block is currently allocated (used) or available (free). Adjacent to each free/used bit is a count of the total number of bytes in the chunk.

2. A Doubly Linked, Embedded Free List: The free chunks are also linked in a doubly linked list. The pointers for this list are adjacent to the boundary tags at either end. Thus, no additional space is needed for the free list; they must accommodate two boundary tags and two pointers, even if the object is a single byte.



A, B, and C. Chunk B, of size 100, has just been deallocated and returned to the free list. The left end of B, namely A is free which is known by the 0 value in right of A. So we can coalesce A and B into one chunk of 300 bytes.

The chunk C, right to B is not free, which is known by the value 1 in start of C. If chuck C is free, then A, B and C can be combined. If A and B has to be coalesced, need to remove one of them before and after each of A and B. Automatic garbage collection can eliminate fragmentation altogether if it moves all the allocated objects to contiguous storage.