Memory Management: there are various memory management algorithms discussed this chapter including: swapping, contiguous allocation, paging, segmentation. They differ in many aspects:

- The binding of instructions and data to memory addresses can be done at different steps, 1) **compile time**, in which **absolute code** will be generated. If the location changes later, the code has to be recompiled; 2) **load time**, in which compiler must generate **relocatable code**. If starting address changes, we need to reload the user code; 3) **execution time**, in which the process can move during its execution from one memory segment to another. Special hardware must be used. Most general-purpose operating systems use this approach.
- **Logical** vs. **physical addresses:** logical address, only relevant to a process, is generated by CPU, while the physical address is the corresponding (actual) memory location seen by memory management unit or MMU. The logical address and physical address are different only if the *execution time address biding* is used, which is often the case in modern operating system. The translation is usually done by **hardware support**, which can be either simple *base register* and *limit register* or use complicated page/segmentation tables with TLBs.
- **Performance**: logical to physical address translation can be time-consuming since the translation tables (page or segmentation table) are usually in memory, in which each address translation can require multiple memory access. Fast registers or/and caches (such as TLB) helps to improve the access to translation tables.
- **Fragmentation**: fragmentation occurs when memory is broken up into small chunks, and some portions are too small to be used by memory requests. Fragmentation results in memory space waste. Fixed-size allocations such as paging suffer from **internal fragmentation** while variable-sized allocations such as contiguous allocation and segmentation suffer from **external fragmentation**.
- **Relocation**: *Compaction* as a solution to external fragmentation by moving programs to new memory locations, transparent to processes. This requires the logical address to be relocated dynamically at the execution time. If the address binding occurs at compile time or program load time, compaction is not feasible.
- **Swapping*:*** Swapping can be added to any algorithms. Processes are copied from main memory to a backing store and later copied back to main memory. This scheme allows more processes to fit into memory at one time to be running.
- **Sharing**: This allows certain pages or segments to be shared by different processes. Some *protection mechanism* must be in place to ensure proper access rights, read-only, execute-only, read-write. Both swapping and sharing can improve the degree of multiprogramming.

Contiguous Allocation

- Each process is contained in single contiguous section of memory
- *Base register* contains the smallest physical address. *Limit register* contains range of logical addresses – each logical address must be less than the limit register.

- This results in a **dynamic storage-allocation problem**, in which there are three methods to satisfy a memory request of size n (variable size) from a list of free memory partitions (holes) including **first-fit, best-fit** and **worst-fit**.

Segmentation

- The memory allocation to each process is a collection of segments, which are variable-sized blocks reflecting the users' logical views of the memory, e.g., main program, procedures, functions, methods, and variables.
- The logical address explicitly specifies both a *segment number* and an *offset* within the segment. This is in contract to a paging scheme in that user has one single address and the operating system partitions the address into a page number and an offset (invisible to users).
- Segmentation table is stored inside memory pointed by a *segmentation table base register* (STBR); a *segmentation-table length register* (STLR) specifies the number of segments of the process, i.e., number of entries in a segmentation table. A valid segment offset of a segment must be smaller than its corresponding segment length.
- Each entry in segmentation table specifies a base (the starting address where the segment resides in memory) and a limit specifying the length of the segment.
- Segmentation scheme suffers from external fragmentation problem, similar to contiguous memory allocation. But the problem is less severe since each segment is usually much smaller than the total memory required for each process in contiguous allocation.
- Sharing a segment among different processes makes more sense than sharing a page since each segment is a logical entity, for example, a subroutine.
- Each segment can be further handled by a paging scheme. In this case, suppose the logical address is s1/s2/d, s1 is used to locate one entry in the segmentation table, which points to corresponding page table. It then uses s2 to locate the entry to find out the corresponding frame number (f), so the physical address is f+d.

Paging Scheme

- The main memory is divided into fixed-size blocks called **frames**; logical address space is divided into the ***same*** fixed-size blocks called **pages**. The operating system keeps track of all free frames in the main memory.
- The address translation is done through a *page table*, which maps the page number of a process to a corresponding frame number. The actually starting physical address of a frame can be easily obtained, frame number x frame size.
- The page table is also stored inside main memory, pointed by a register called *page-table base register* (PTBR). Each memory access requires two memory accesses (one for page table and one for the data or code). The performance can be improved by storing part of the page table entries in a special fast-lookup hardware called *translation look-aside buffers* (**TLB**). The improvement depends on the *hit ratio* in the TLB (like caching scheme).
- Paging scheme suffers from internal fragmentation, in which on average half a

page can be wasted.

- Large logical address can result in excessively large page table, in which case a page table will be stored in multiple frames. Hierarchical paging scheme such as two-level paging can overcome this problem, in which it usually tries to keep each (smaller) page table to fit within one page (frame). However, this would result in more than two memory access for each address translation due to the multiple level address translation.