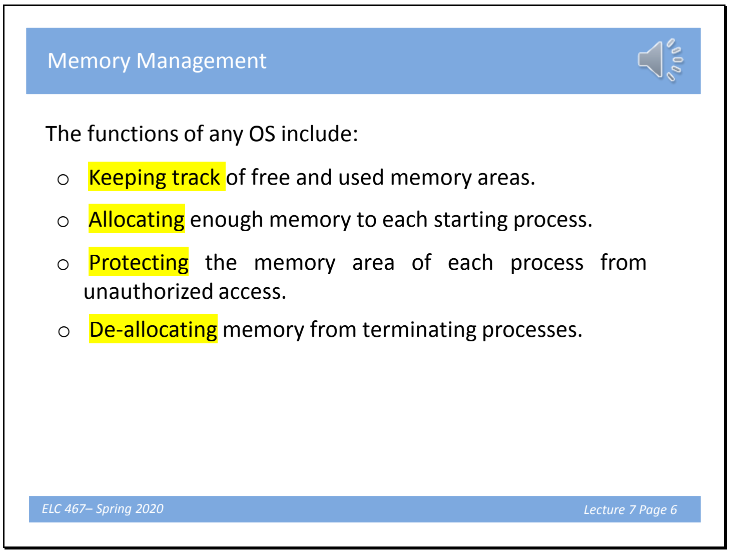
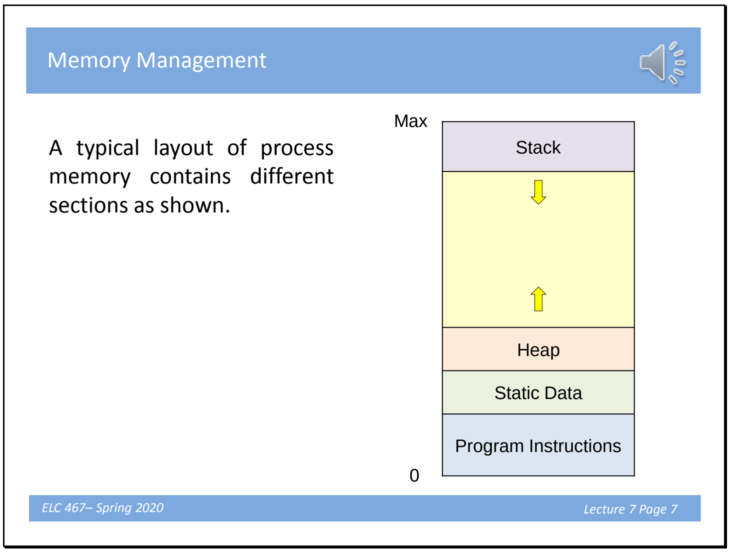


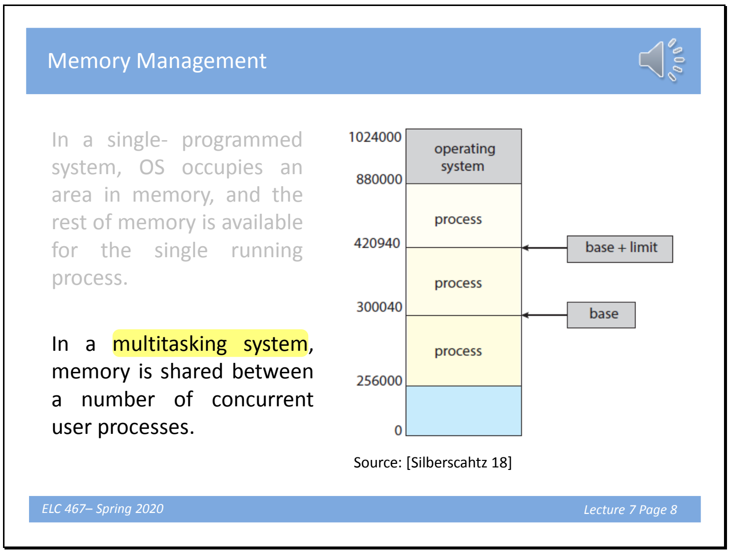
<u>Slide 1</u>

We now start the study of some operating system functions related to memory management. The operating system is responsible for assigning each process a portion in memory in which it can run, and protecting this portion from unauthorized access by other processes. System uses appropriate algorithms and data structures to perform these functions.



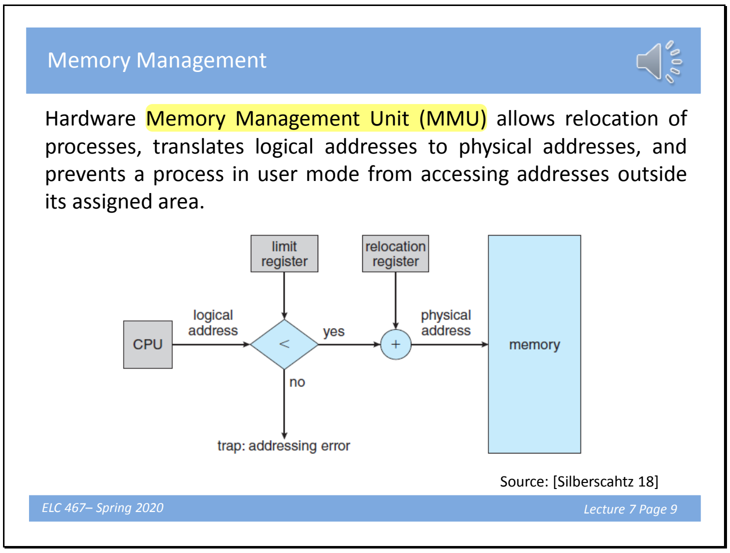
<u>Slide 2</u>

Typically, memory used by a process stores program instructions, static variables and data structures, and dynamic heap (growing into larger addresses) and stack (growing into smaller addresses). On contrast to the process execution time which is generally unknown to the system, we can assume that process memory requirements are declared and known to the system. Until otherwise stated, we assume that process must work from undivided (or contiguous) area in memory.



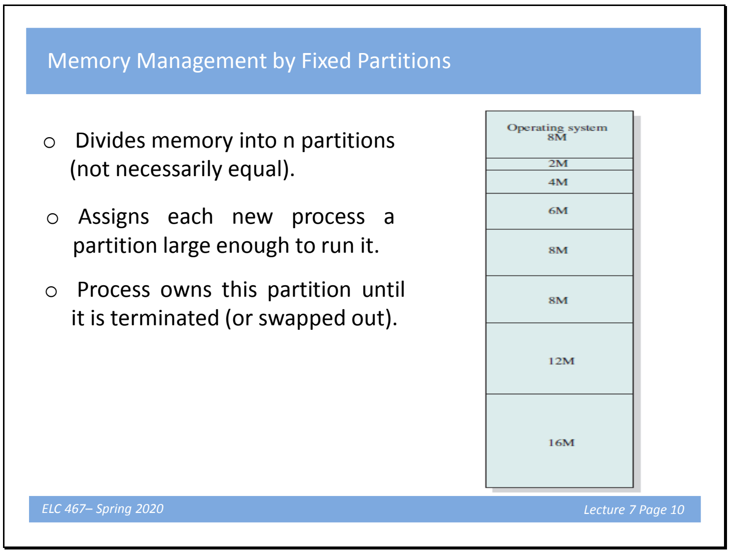
Slide 4

The idea of relocatable processes and base address was mentioned in first week lecture. Here, the limit is the threshold of memory addresses that process should not go across.



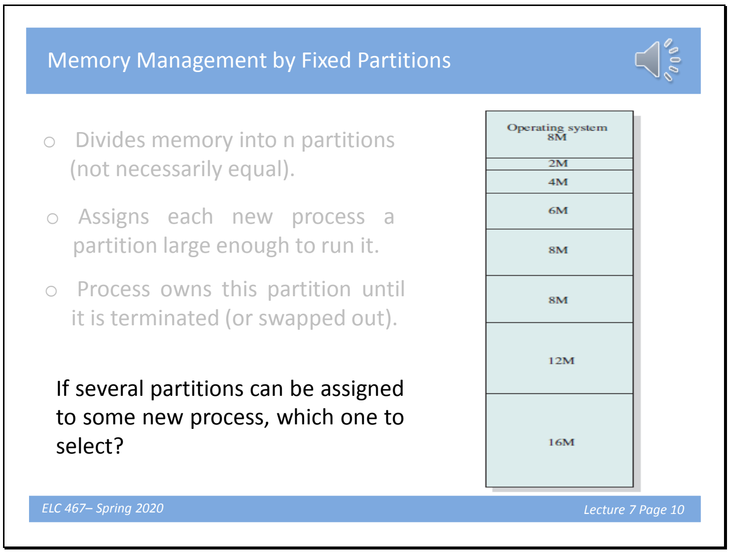
Slide 5

The logical address is the offset we mentioned before. The operation of checking if limit is passed, and adding base to offset will be needed for each memory access. For efficiency, these operations must be done in hardware MMU and not by software.



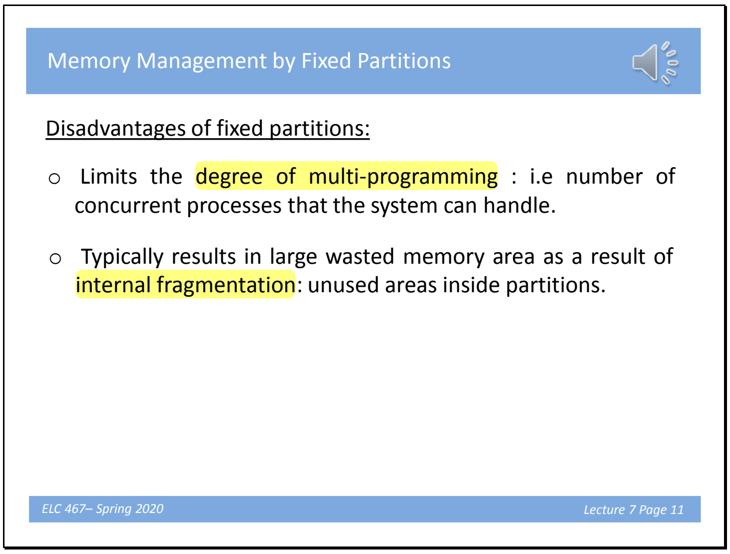
Slide 6

The first memory management technique we consider is the method of fixed partitions. Here, memory is divided into a fixed number of partitions with fixed sizes and limits. Each of these partitions can be assigned to a process. Giving different sizes to partitions allow running processes with different memory requirements.



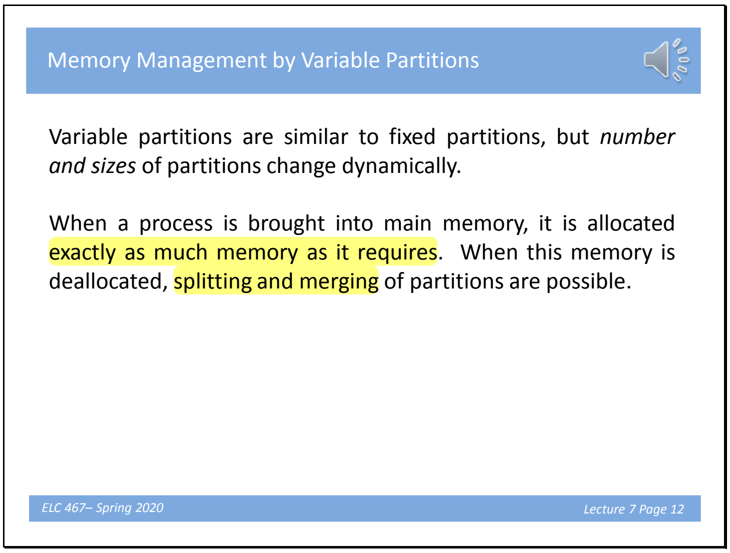
Slide 8

The obvious answer here is the smallest one. Size of partition assigned to a process must be equal to or larger than process requirements. If larger, the additional area cannot be used by any other process, and is hence wasted. This can be minimized if we select the smallest partition that can be used by the process.



Slide 9

If we have n partitions, system cannot run more than n processes concurrently, even if memory still have free space. In addition, internal fragmentation refers to the phenomenon that parts of partitions will be assigned to processes but not actually used. Fixed partitions are not used in current systems, except possibly for some simple embedded systems.

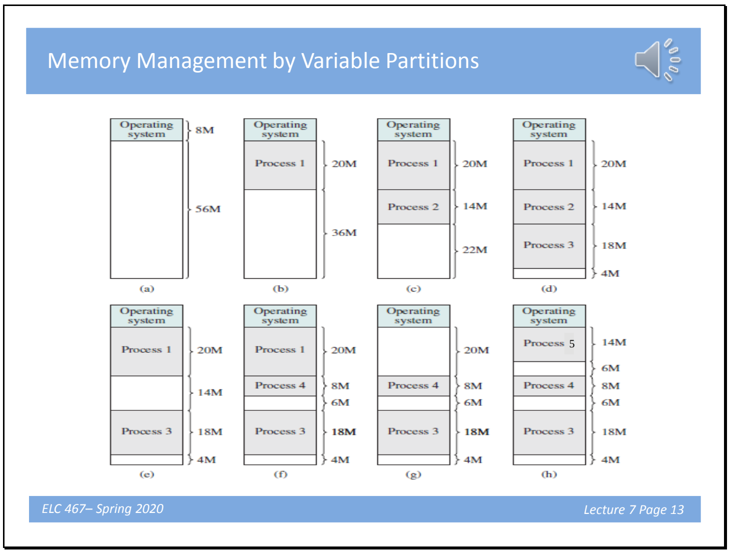


Slide 10

In this technique also, a memory partition of enough size is reserved for each process from its start to its termination. However, the partitions into which the memory is divided change dynamically.

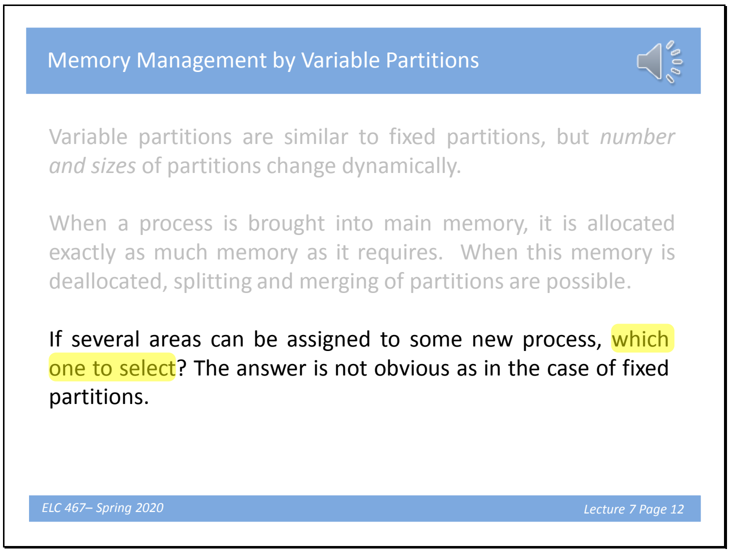
Slide 11

Process is assigned the exact memory size it needs. If this is taken from a previously larger partition, the remainder of this partition can be used by some other process. Adjacent partitions that become free can be merged into one larger partition. These variable partitions enable avoiding the disadvantages of fixed partitions.



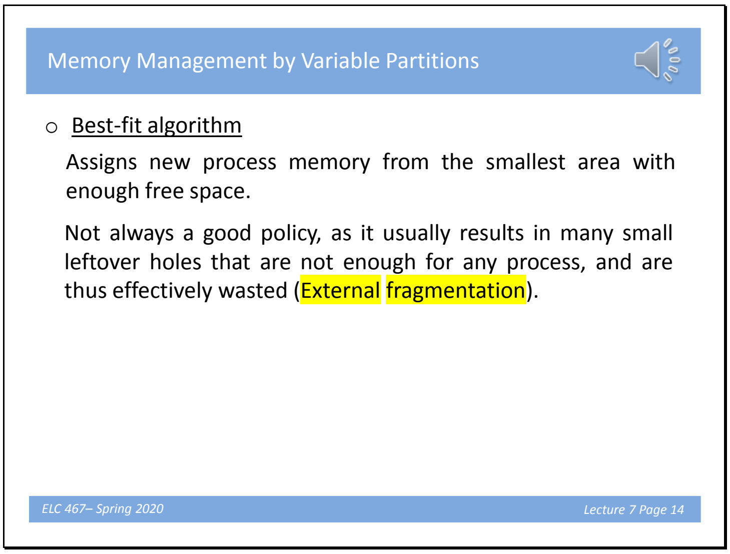
Slide 12

In figure (a), the operating system only is running, reserving a memory partition for itself. In figures (b),(c), and (d) processes 1,2, and 3 are assigned their memory requirements. In figure (e) process 2 terminates, thus releasing the partition it held. When process 4 starts in figure (f), it needs only 8M, thus the partition of size 14 M is split into 8M assigned to new process, and 6M that remains free. Same occurs when process 5 starts.



Slide 13

In fixed partitions, the answer to this question was to select the smallest, to minimize the waste in memory. Here, no waste occurs as a result of internal fragmentation. So what is the best selection?

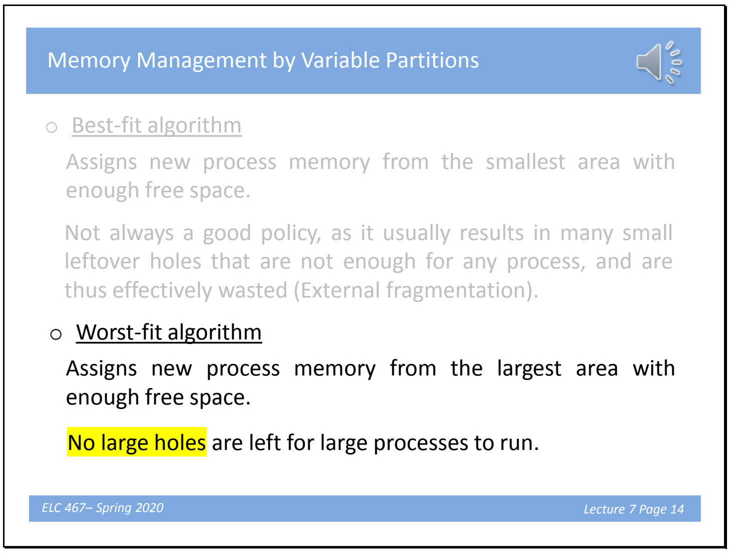


Slide 14

If we also take the memory requirements from the smallest partition, we call this the best-fit selection.

Slide 15

This tends to minimize the remaining free areas. These remaining areas may be not enough to run any process. Note that we still assume that a process needs to operate in one undivided partition. After using best fit for some time, the free area in memory will be scattered into small holes that are not enough to run processes. This phenomenon is called external fragmentation (i.e. fragments occur outside of partitions).

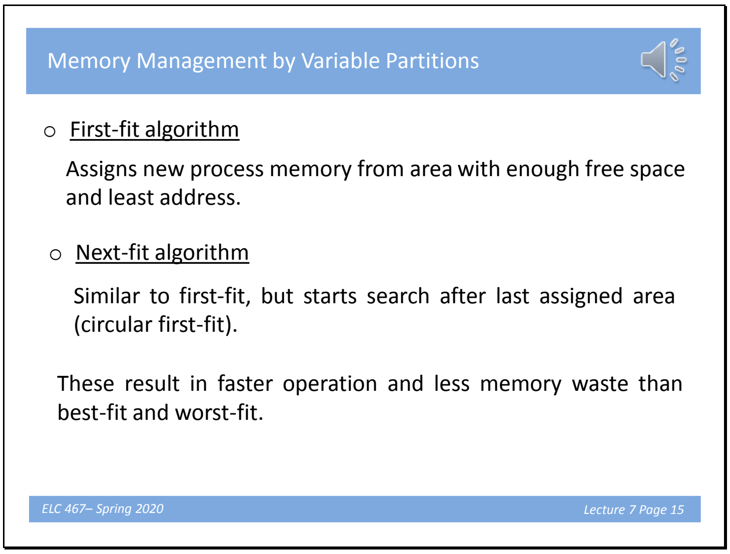


Slide 16

If we select instead the largest partition, we call this the worst-fit algorithm. It should avoid the disadvantage of best-fit.

Slide 17

However, a new problem arises. Since in worst-fit we systematically split large partitions, we expect that no large partitions will be left after a while. Processes with large memory requirements will not be able to run.

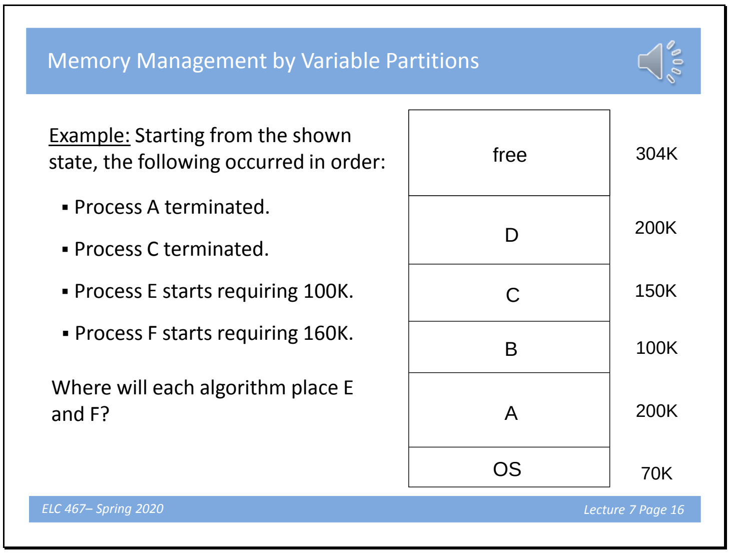


Slide 18

Since selection of smallest or largest partitions will both lead to problems, it is better not to base decision on size. The first-fit algorithm begins looking for a suitable partition from the start of memory, and assigns the first partition it finds. Sometimes this will be small, and sometimes it will be large, so disadvantages of best-fit and worst-fit will not persist.

Slide 19

A slightly different method is the next-fit algorithm. It also assigns the first partition it finds irrespective of size. However, it does not begin the search from the start of memory each time. Instead, it begins search after the last assigned area. This usually will speed up the search, as in first-fit the start of memory will typically be fragmented, and suitable partitions will be found at higher addresses. The name circular first-fit may be used since when search reaches the end of memory, it returns back to the start.



Slide 20

Note here that the order of events is important as it will change the results.

Slide 21

Now we have free areas of sizes 200 K, 150 K, and 304 K.