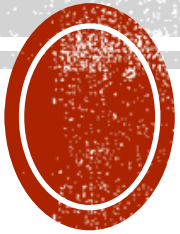
SWAPPING

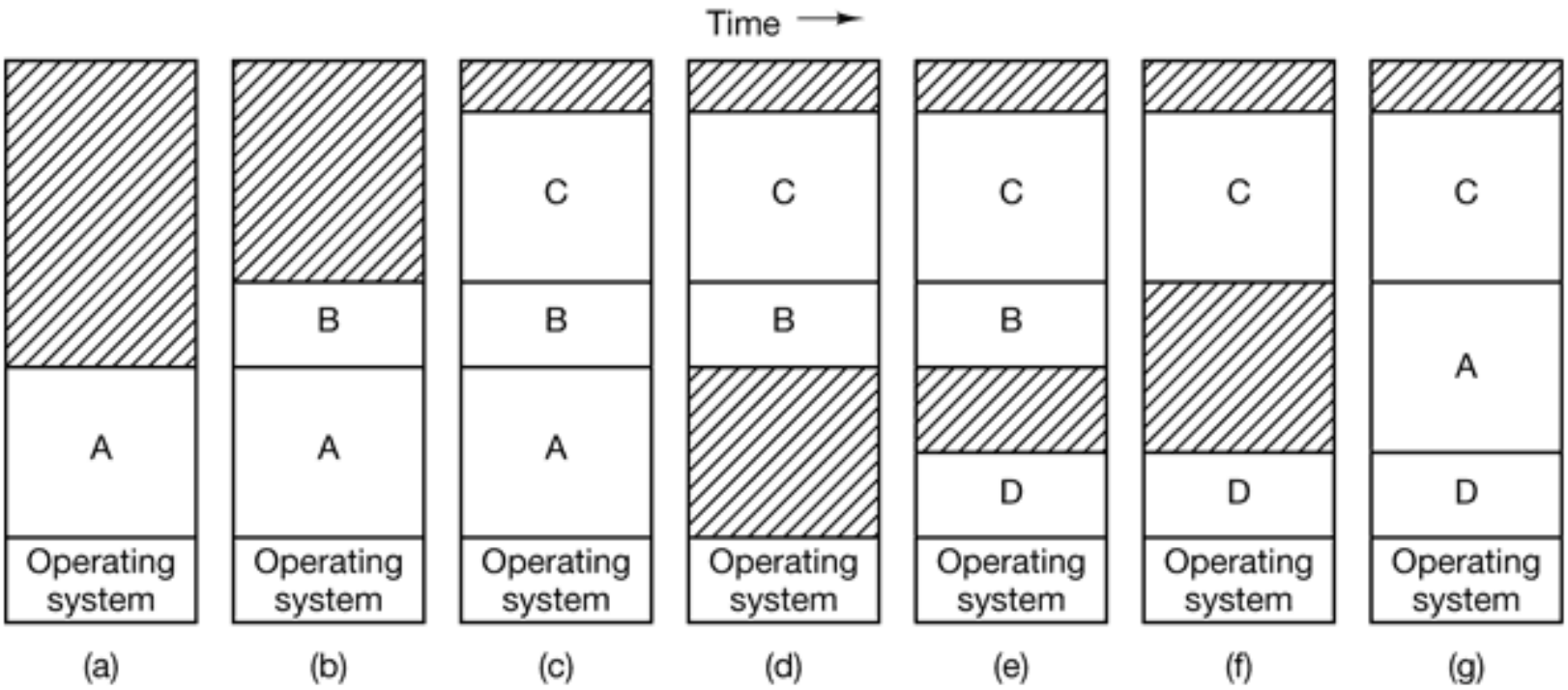
Chapter 4.2



SWAPPING

- When not enough memory to hold all the currently active processes,
- Excess processes must be kept on disk and brought in to run dynamically
- The simplest strategy, called **swapping**, consists of bringing in each process in its entirety, running it for a while, then putting it back on the disk.
- The other strategy, called **virtual memory**, allows programs to run even when they are only partially in main memory

SWAPPING

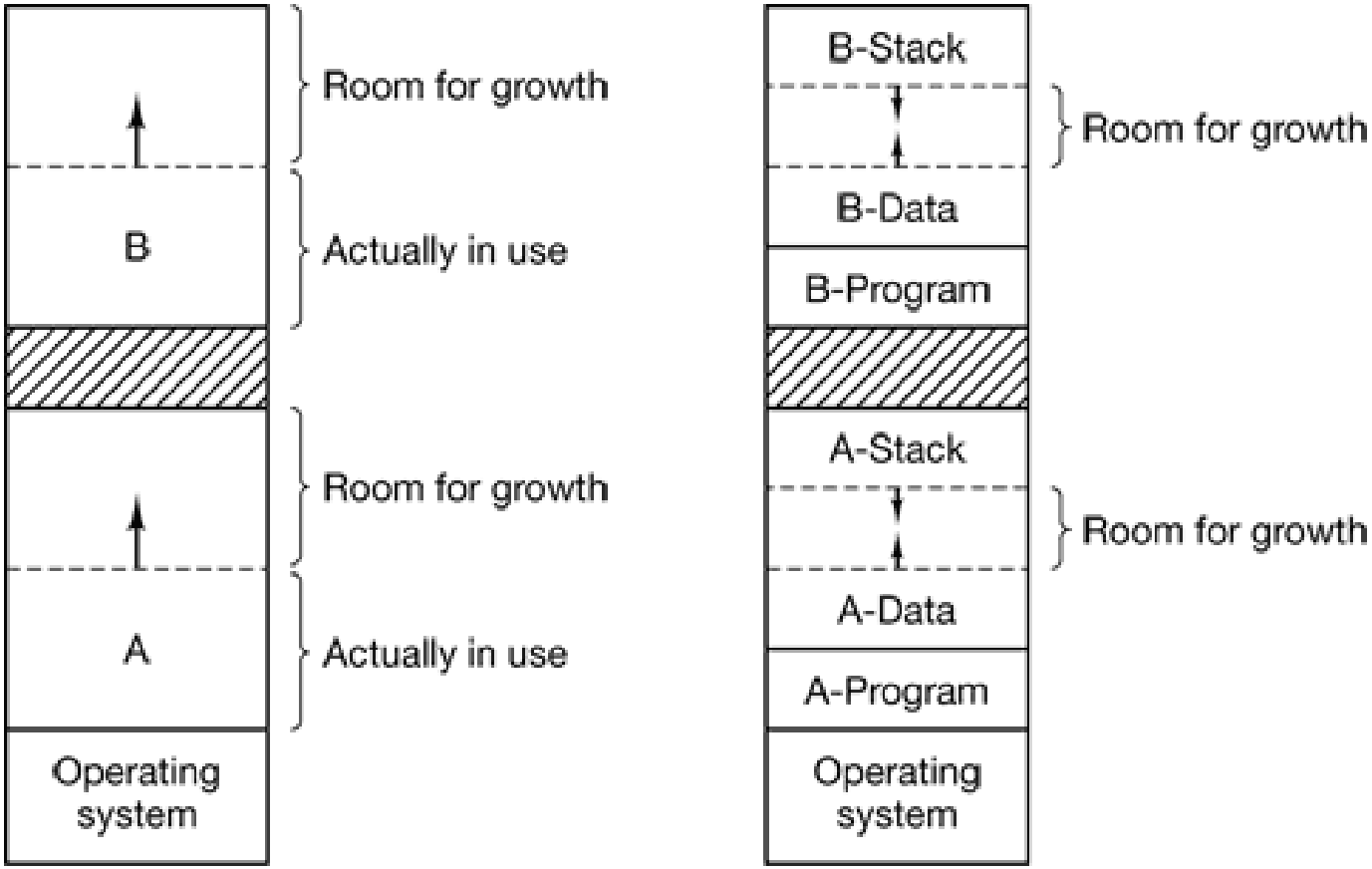


- When swapping creates multiple holes in memory, it is possible to combine them all into one big one by moving all the processes downward as far as possible. This technique is known as **memory compaction**.
  - Time consuming, expensive operation.

SWAPPING ISSUES

- Fixed size allocation
  - If process needs fixed size that never changes, then the allocation is simple: the operating system allocates exactly what is needed, no more and no less.
- Process grows dynamically
  - What if adjacent memory is allocated to another process
  - How much extra memory should you allocated
  - Disadvantage on extra memory:
    - Needs copying the empty memory while swapping

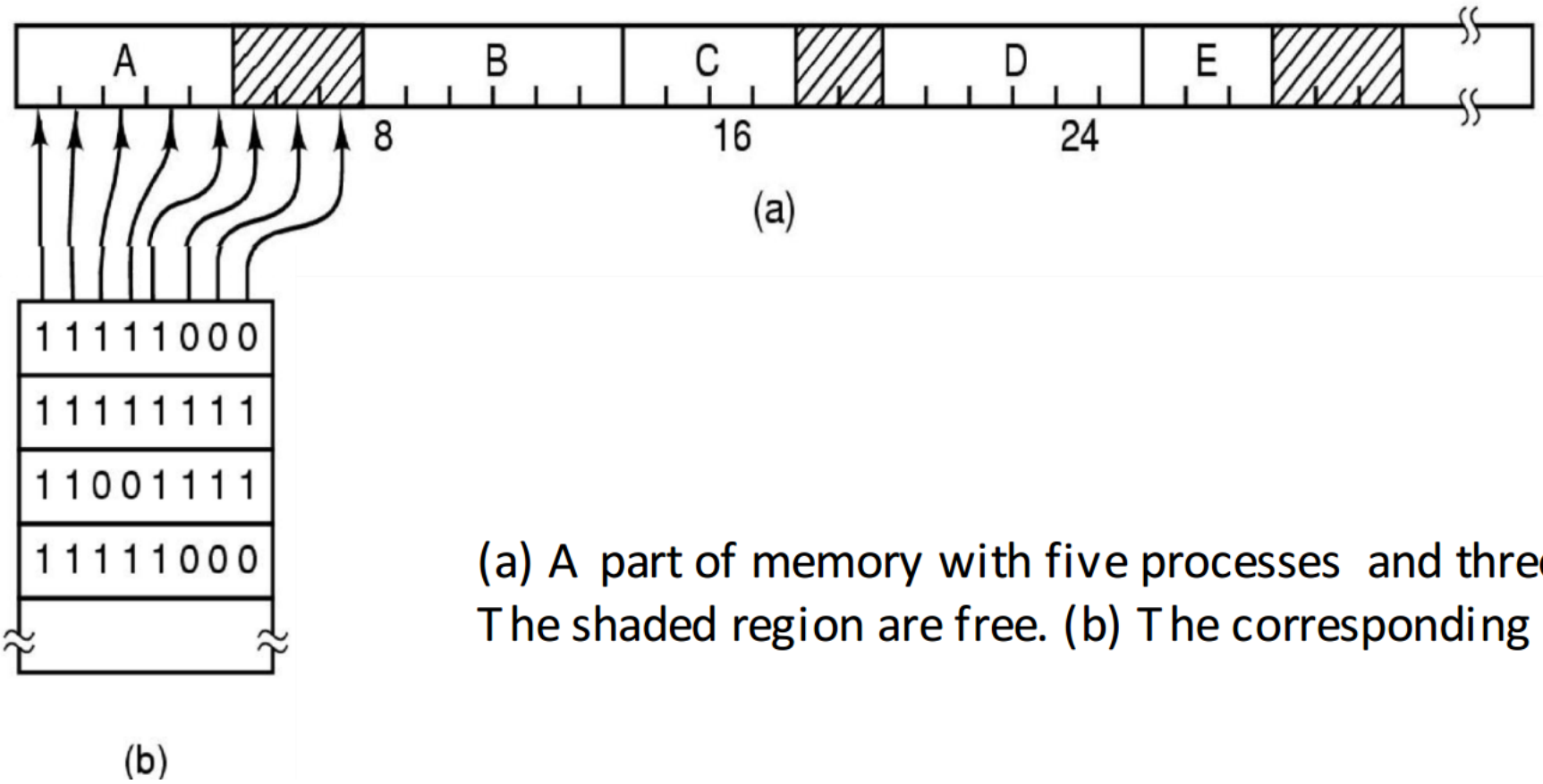
SWAPPING WITH EXTRA MEMORY



MEMORY MANAGEMENT WITH BITMAPS

- Each allocation unit(few words or several Kilobytes) is a bit in the bit map, 0 if the unit is free, and 1 if it is occupied (or vice versa).
- When a process come into the memory, it search required numbers of consecutive 0 bits in the map.
- The size of the allocation unit is an important design issue;
  - Increasing the size of allocation decrease the size of bitmap,
  - but when the process size is not a multiple of size of the allocation unit.
- Advantages:
  Provides a **simple** way to keep track of memory words.
- Problems:
  Searching a bitmap for a run of given length is a slow operation

MEMORY MANAGEMENT WITH BITMAPS

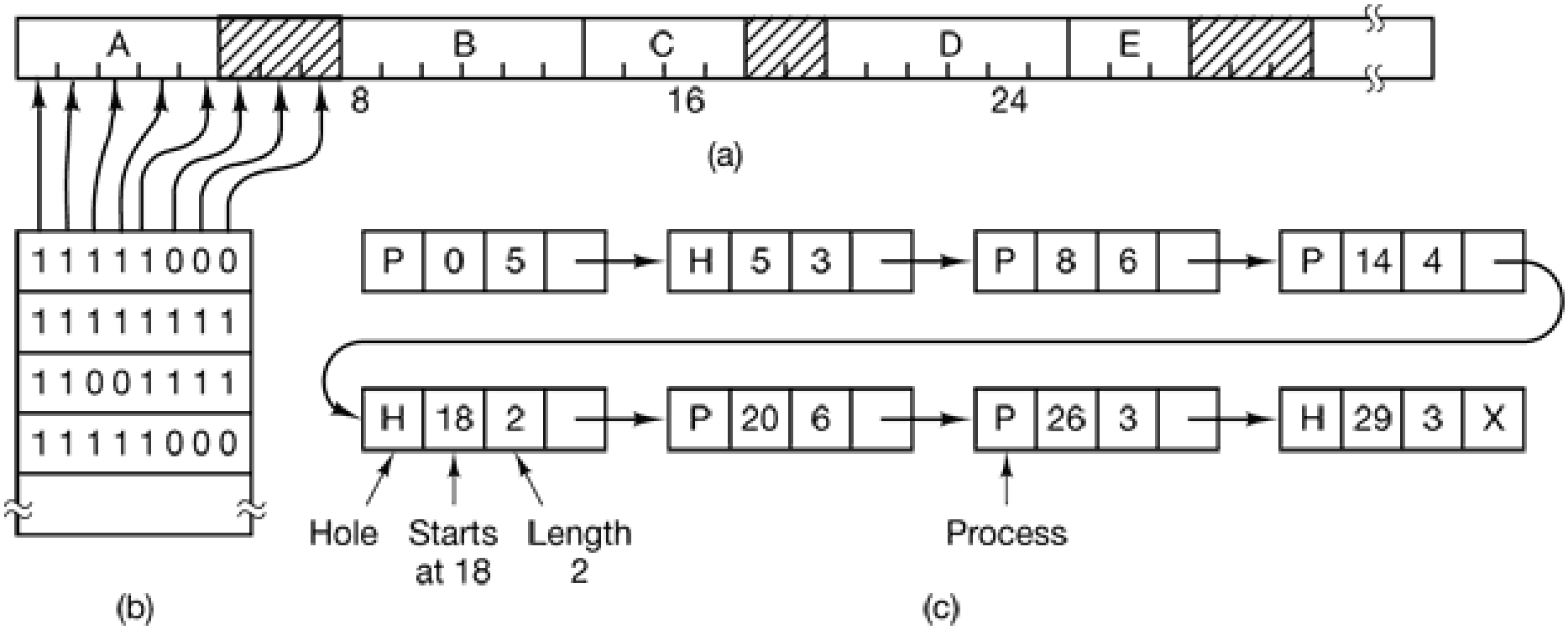


(a) A part of memory with five processes and three holes. The shaded region are free. (b) The corresponding bitmap.

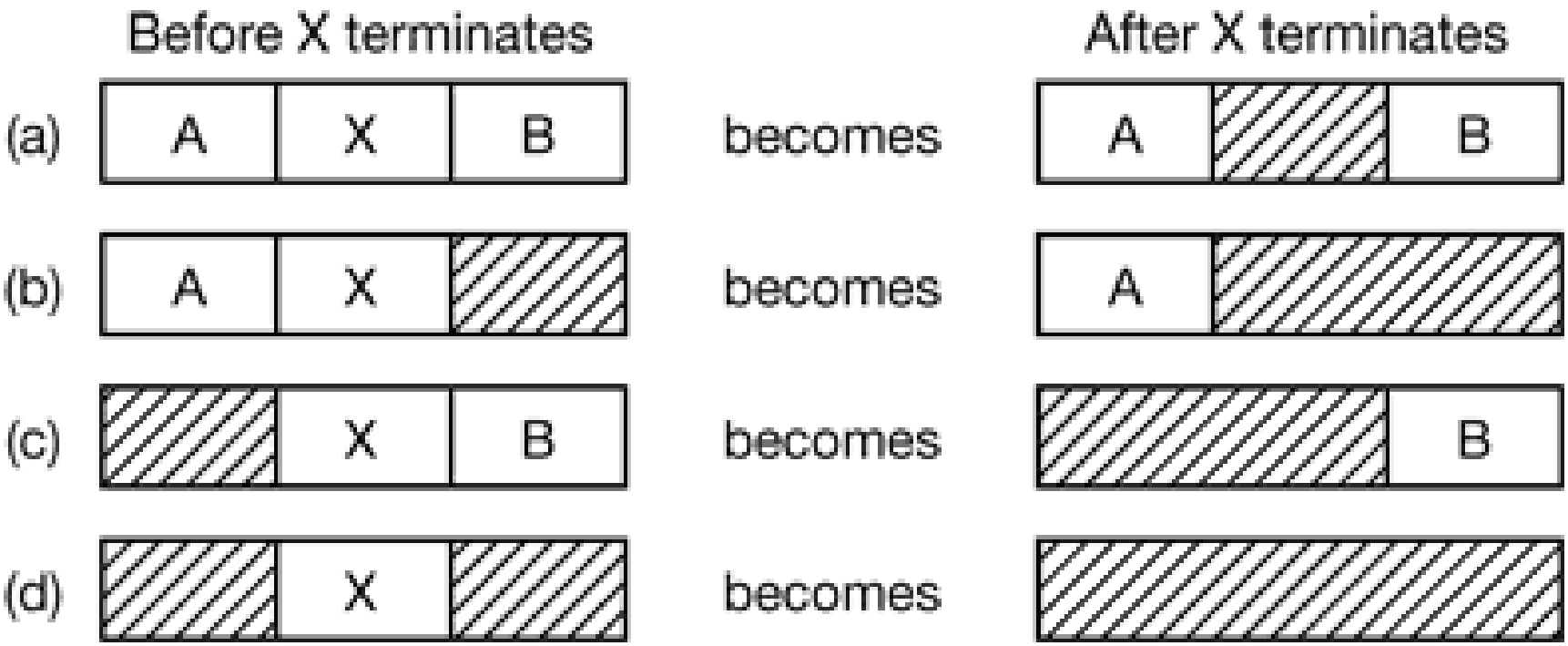
MEMORY MANAGEMENT WITH BITMAPS

- allocation unit = 4 bytes,
  - 32 bits of memory will require only 1 bit of the map.
  - A memory of 32*n* bits will use *n* map bits,

MEMORY MANAGEMENT WITH LINKED LISTS



MEMORY MANAGEMENT WITH LINKED LISTS



VARIABLE PARTITION SELECTION ALGORITHMS

- When the processes and holes are kept on a list sorted by address, several algorithms can be used to allocate memory for a newly created process (or an existing process being swapped in from disk).
- We assume that the memory manager knows how much memory to allocate.
- First Fit
- Next Fit
- Best Fit
- Worst Fit
- Quick Fit

FIRST FIT

- Allocate the first hole that is big enough.
- It stop the searching as soon as it find a free hole that is large enough.
- The hole is then broken up into two pieces, one for the process and one for unused memory.
- Advantage:
  - It is a **fast algorithm**.
  - because it search as little as possible.
- Problem:
  - not good in terms of **storage utilization**.

NEXT FIT

- It works the same way as first fit,
- except that it keeps track of where it is whenever it finds a suitable hole.
- The next time it is called to find a hole, it starts searching the list from the place where it left off last time, instead of always at the beginning, as first fit does.
- Simulations by Bays (1977) show that next fit gives slightly worse performance than first fit.

BEST FIT

- searches the entire list and takes the smallest hole that is adequate.
- Rather than breaking up a big hole that might be needed later
- As an example of first fit and best fit, consider Fig. 4-7 again. If a block of size 2 is needed, first fit will allocate the hole at 5, but best fit will allocate the hole at 18.
- Best fit is slower than first fit because it must search the entire list every time it is called.
- Somewhat surprisingly, it also results in **more wasted memory** than first fit or next fit because it tends to fill up memory with tiny, useless holes.

WORST FIT

- always take the largest available hole, so that the hole broken off will be big enough to be useful.
- Simulation has shown that worst fit is not a very good idea either.

ALGORITHM SPEED UP

- Manage separate list for Process and Holes.
- Advantage
  - All fours algorithms can be speed up
- Disadvantage
  - the additional complexity and slowdown when deallocating memory, since a freed segment has to be removed from the process list and inserted into the hole list.

QUICK FIT

- maintains separate lists for some of the more common sizes requested.
- For example, it might have a table with *n* entries, in which the first entry is a pointer to the head of a list of 4-KB holes, the second entry is a pointer to a list of 8-KB holes, the third entry a pointer to 12-KB holes, and so on.
- Holes of say, 21 KB, could either be put on the 20-KB list or on a special list of odd-sized holes.
- Advantage
  - extremely fast
- Disadvantage
  - schemes that sort by hole size, when a process terminates(or is swapped out) **finding its neighbors to see if a merge is possible is expensive.**
  - If merging is not done, memory will **quickly fragment** into a large number of small holes into which no processes fit.

OVERLAYS

- Many years ago when programs were too big to fit in the available memory
- programmers split the program into pieces, called **overlays**.
- Overlay 0 would start running first.
  - When it was done, it would call another overlay.
  - Some overlay systems were highly complex, allowing multiple overlays in memory at once.
  - The overlays were kept on the disk and swapped in and out of memory by the operating system, dynamically, as needed.
- the work of splitting the program into pieces had to be done by the programmer.
- Splitting up large programs into small, modular pieces was time consuming and boring.
- It did not take long before someone thought of a way to turn the whole job over to the computer.
- The method that was devised (Fotheringham, 1961) has come to be known as **virtual memory**