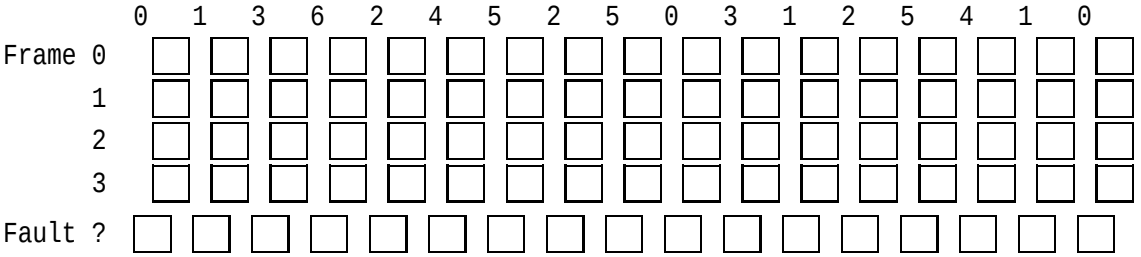
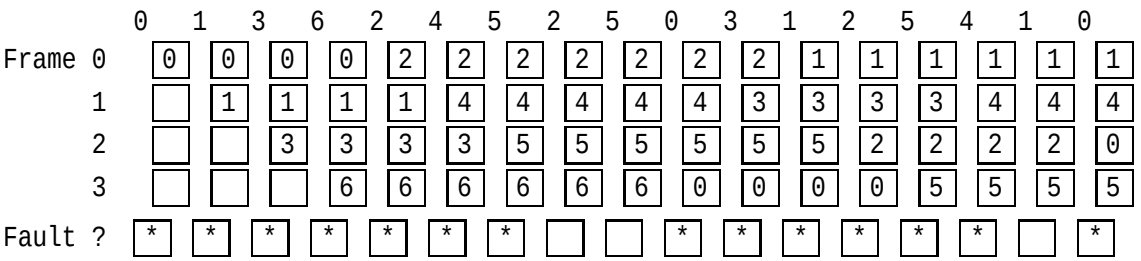
This is a set of examples of page replacement problems. Please keep a few things in mind:

- these are small problems and will not necessarily reflect all characteristics of the real world
- I have endeavored to be correct but errors are possible

The presentation is based upon a typical test problem. This is followed by a solution that we believe would lead to full credit. Of course, this solution was worked by hand and then converted to LaTeX which guarantees a nice presentation but not necessarily a correct mapping.

1. Consider the reference string shown along the top of the following graphical structure. The system has four frames. Use the LRU algorithm to select pages for replacement. Place the page number in the proper frame. Mark when page faults occur in the bottom line of boxes. State how many page faults occur. The numbers across the top indicate the reference string.

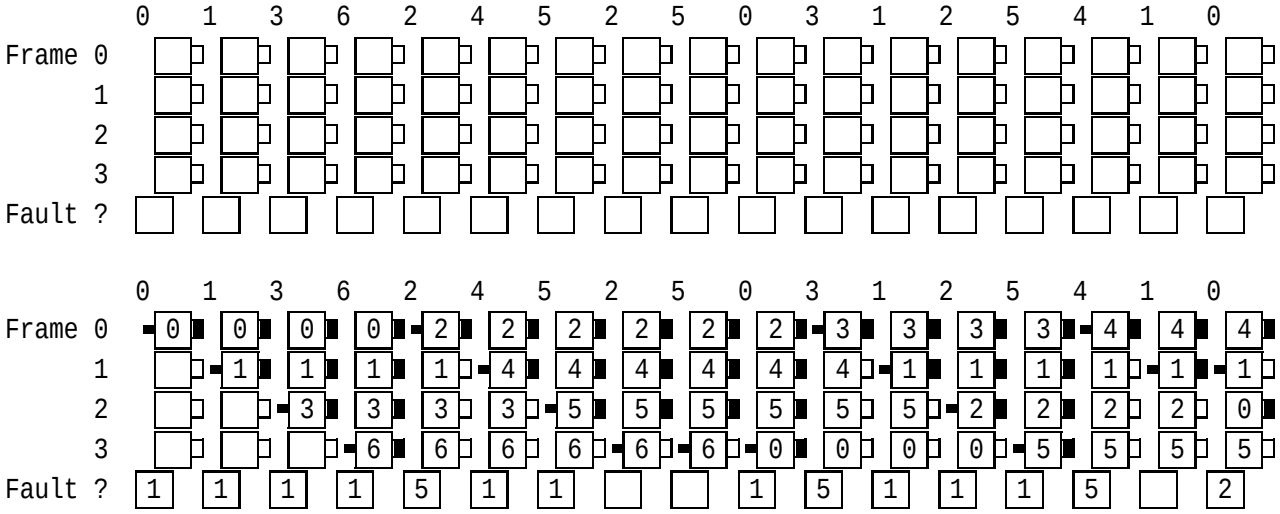




In the example above the faults are marked with an * There were fourteen page faults.

2. **Clock algorithm.** The second chance algorithm is an approximation of LRU based on using one use bit for each page. When a page is used its use bit is set to 1. We also use a pointer to the *next victim* which is initialized to the first page/frame. When a page is loaded, it is set to point to the next frame. The list of pages is considered as a circular queue. When a page is considered for replacement, if the use bit for the *next victim* page is examined. If it is zero [that page is replaced] otherwise [the use bit is set to zero, the *next victim* pointer is advanced, and the process repeated until a page is found with a zero use bit].

   Consider the reference string shown along the top of the following graphical structure. The system has four frames. Use the clock algorithm described in the previous paragraph. The narrow boxes to the right of the page number boxes can be used to keep up with use bits. Use the space between the page number boxes to **show** the *next victim* pointer. The initial one is shown, ▪ Place the page number in the proper frame. Mark when page faults occur in the bottom line of boxes. State how many page faults occur.



   Fourteen page faults.

   The number in a box indicates the page number in that frame after this operation
   ▮ indicates the page has been used
   The boxed numbers below and to the left of the frame columns indicate how many times the *next victim* pointer is advanced in this operation, including the one afterwards.
   Whenever the pointer is advanced more than one, it clears a use bit.
   Notice that the last operation cleared a use bit and advanced the *next victim* pointer twice although the pointer is off the page.

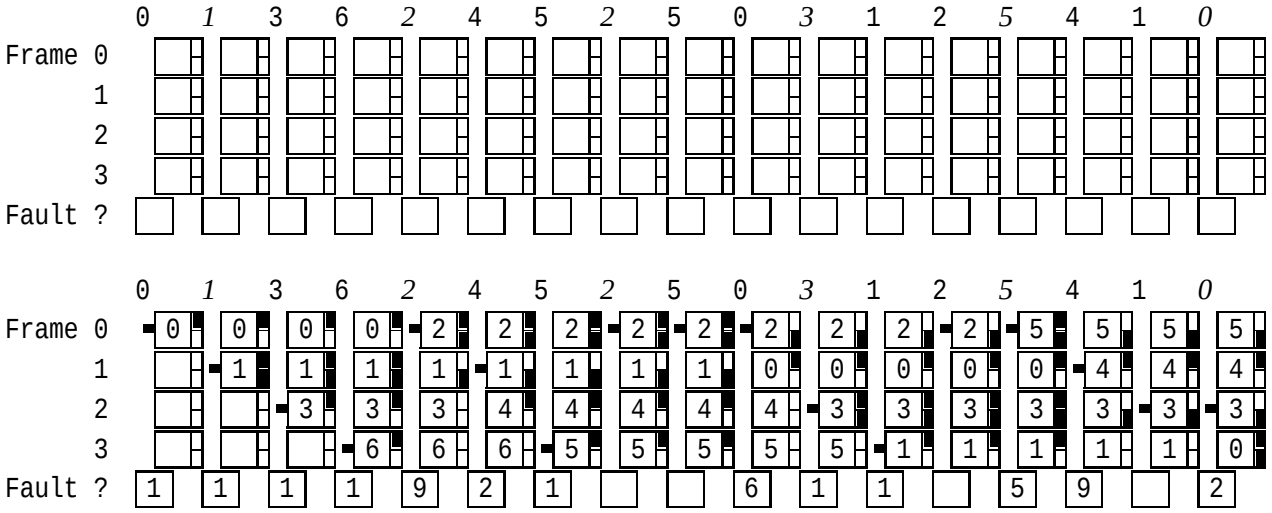
3. **Enhanced Second-Chance Algorithm.** The enhanced second chance algorithm uses one use bit and a modify bit for each page—many variations of this are possible. When a page is used its use bit is set to 1. We also use a pointer to the *next victim* which is initialized to the first page/frame. When a page is loaded, the *next victim* pointer will be set to the next frame after the one just loaded. The list of pages is considered as a circular queue. When a page is considered for replacement, the bit pairs for each page are considered `<ref,mod>`:

   `<0,0>` neither recently used nor modified, best!
   `<0,1>` modified but not recently used, will need to be written.
   `<1,0>` recently used but clean—likely to be used again.
   `<1,1>` both—likely to be used again and will need to be written.

   There are three loops through the circular buffer containing these bits that may be used. They are:

   (a) Cycle through the buffer looking for `<0,0>`. If one is found, use that page.
   (b) Cycle through the buffer looking for `<0,1>`. Set the use bit to zero for all frames bypassed.
   (c) If step 2 failed, all use bits will now be zero and repetition of steps 1 and 2 are guaranteed to find a frame for replacement.

   Consider the reference string shown along the top of the following graphical structure. The page numbers in the reference string that in *italics* are modified, set their **dirty bit**! The system has four frames. Use the enhanced second-chance algorithm. The narrow boxes to the right of the page number boxes can be used to keep up with use and modify/dirty bits. Use the space to the left of the page number boxes to **show** the *next victim* pointer. The initial one is shown, ▪ Place the page number in the proper frame. Mark when page faults occur in the bottom line of boxes. State how many page faults occur.



   Thirteen page faults.

   The number in a box indicates the page number in that frame after this operation
   ▮ indicates the page has been used
   ▮ indicates the page has been modified
   The boxed numbers below and to the left of the frame columns indicate how many times the *next victim* pointer is advanced in this operation, including the one afterwards. **The errors in the previous version** were that I did not incremement the *next victim* count for those that got to loop (b) in the `Fault ?` line. Whenever the pointer is advanced while in loop (b) use bits are cleared.

Comments based on reference string entries.

0 Load page 0 into frame 0, set the use bit.

1 Load page 1 into frame 1, set use and modify bits.

3 Load page 3 into frame 2, set use bit.

6 Load page 6 into frame 3, set use bit.
Now, all frames have been occupied so the enhanced second-chance algorithm in all its detail will be used.

2 Loop (a) fails as does loop (b).
During loop (b) the use bits are cleared.
When loop (c) directs the second time through loop (a), frame 0 satisfies the criteria.
Page 2 is loaded, use and modify bits are set, and the *next victim* pointer is set to 1.

4 The modify bit for dictates skipping one.

5 Straightforward.

2 This page is already in memory, so no page fault.

5 . . .