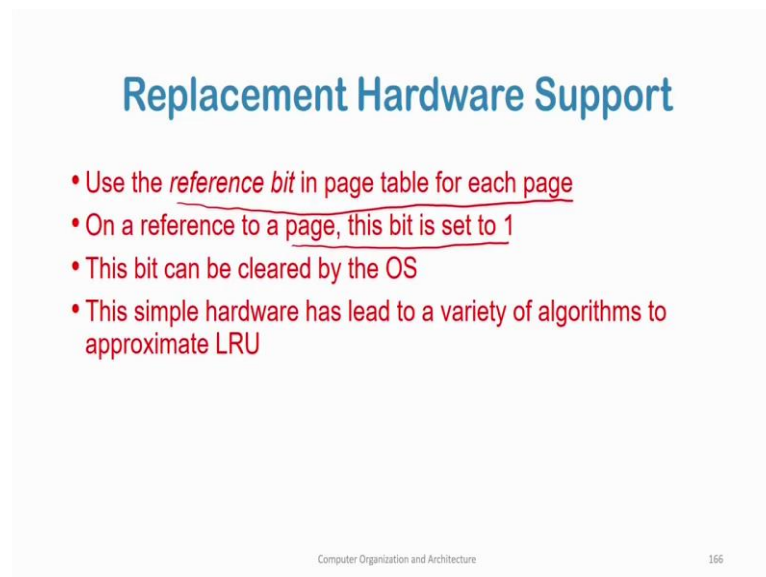


So, if I do not implement this counter method or the stack method in hardware I have to implement that in software, meaning that whenever there is a memory reference, I have to go to the OS and update data structures, which is also impractical because memory references are a very frequent phenomena. So therefore, the exact version of ALU is not often used and instead approximate LRU is commonly used.

(Refer Slide Time: 32:29)



### Replacement Hardware Support

- Use the reference bit in page table for each page
- On a reference to a page, this bit is set to 1
- This bit can be cleared by the OS
- This simple hardware has lead to a variety of algorithms to approximate LRU

Computer Organization and Architecture 166

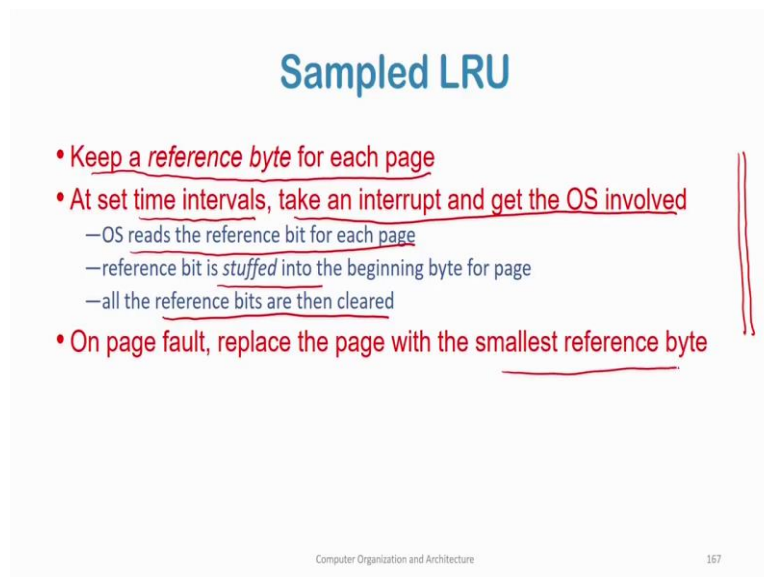
So, one of the most common implementations is to use an approximate version of ALU, which uses reference bit in the page table for each page. So, what how does this operate? On a reference to a page this bit is set to 1 ok. So, each page has a reference bit when I when I am referring to this page when I am accessing this page and this page is in memory I am setting this bit from 0 to 1, if it is not already 1 ok.

Now, at and I have time periods or frames or intervals. So, fixed size intervals after which I check for the after which I set the bits of all pages, I said the reference bits of all pages to 0. So, I when at the start of a time interval, I set the reference bits of all pages to 0 and then within that particular interval every page that is referenced their reference bits are set to 1. And then when the when a particular page has to be replaced, I will try to use a page for which the reference bit is 0. So, a reference bit is 0 means that in the in the current time interval it has not been accessed; that means, it has it is it is of higher probability that it is of higher probability that this page will also not be used in the recent future because, it has not been used in the current timestamp.

This anyway does not mean that this is the least recently used, but it is the it is an approximate LRU to avoid the high hardware cost of LRU, I will just keep one reference bit on each memory reference if that reference bit is 0 I set it to 1 and during replacement I try to find a page for which the reference bit is 0; that means, which has not been used it is with the approximate assumption that this is not a very frequently used page because, this has not been accessed in the current time interval and therefore, this one could be a good page to replace ok.

Now if all bits are same for a for a given reference bit suppose I find out among all pages for which the reference bit is 0, I may like to select the one which has which came into memory at the earliest; that means, first in first out, I will use FIFO for a given value of the reference bit. So, if I get a set of pages with reference bit 0, I can choose the one which came into memory at the earliest ok. So, that will be that will be evicted; so among all pages which have the same reference bit, I use the FIFO strategy to find out the page which needs to be replaced.

(Refer Slide Time: 35:46)



### Sampled LRU

- Keep a reference byte for each page
- At set time intervals, take an interrupt and get the OS involved
  - OS reads the reference bit for each page
  - reference bit is stuffed into the beginning byte for page
  - all the reference bits are then cleared
- On page fault, replace the page with the smallest reference byte

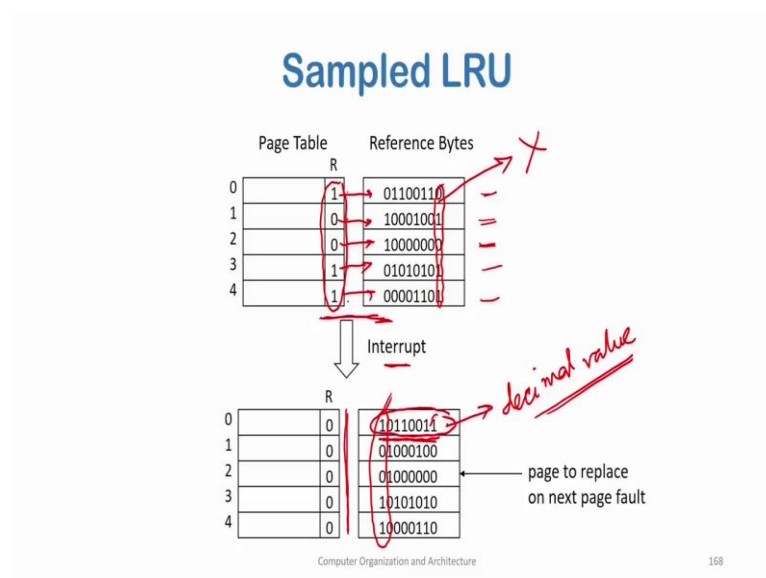
Computer Organization and Architecture 167

The next one is sampled LRU, sampled LRU who is an extension over the approximate LRU which we studied using just one reference bit. And it has slightly higher hardware cost in that, the first byte of each page is used by this strategy. Now what happens is this, that instead of in addition to the reference bit I have a reference byte for each page. So, the first byte of a page will be used as a reference byte reference byte for the page. Now at set time intervals I have similar time intervals as was for the simple approximate LRU I have similar time intervals.

And we take an interrupt and get the OS involved. So, at the end of each time interval there what I was doing, I was I was setting all the reference bits to 0. Here, additionally what I do is I get the OS involved and what does the OS do here? The OS reads the reference bit for each page; the OS reads the reference bit for each page and the reference which is stuffed, so, at the beginning byte for the page. So, in addition I already have the reference bits for each page.

So, at the beginning of the interval I read the reference bit of each page and, stuff it into the reference byte. And then I all reference bits are again clear; very similar to the previous scheme, this one is very similar to the previous scheme, the only difference being that before setting all the reference bits of each page to 0, I copied the reference bits of each page and stuff it into the reference byte. And then on a page fault I replace the page with the smallest reference byte.

(Refer Slide Time: 37:54)



So, how does this scheme work? Now let us say these are my reference bytes for each page. So, these are distinct pages; page 1, page 2, page 3, page 4, page 5 and this is the page table. So, this is the page table and the page table contains my reference bits; these are my reference bits for these pages.

Now, what happens is that now when at the end of so, what has happened is that here my time interval has come to an end; the current time interval. So, at the beginning of every time interval, the OS takes charge and let us say this one is the end of one time interval, there is an interrupt and the OS has taken charge. What does the OS do? It first throws these bits. So, these

the least bits are thrown away and what am I doing? I am stuffing all these bits to these places ok.

So, I am stuffing these bits so, when I stuff this bits, so, see these bits here are the same as the MSBs here, as the MSB here, then I am clearing all my reference bits ok, I am clearing all my reference bits. Now what happens? The values this byte; if I take the numerical value of this byte. So, I take this value of this each of these reference byte; this reference byte. So, I take the numerical decimal value say decimal value of this byte. Now what does this decimal value tell? This decimal value tells me in the last 8 intervals what was the access pattern of it.

And this MSB has the highest weight meaning that it could be that in the last few intervals this page was not used, but this page was used immediately here. So, even if these bits are 0, I should keep this page. So, and let us say so therefore, the numerical value of this byte tells me how good is this page for replacement; lower the value of this byte better is a better is this page or candidate for replacement. So, I should replace a page having least value for this byte. Why? Because this tells me this byte tells me two things. A very low value of this byte tells me that this page was not accessed recently and this page was not accessed possibly many times recently.

So, this page was not accessed very recently and this page was not accessed possibly not many times recently. So, this is what it tells me the numerical value of this byte. So, I will replace that page having low numerical lowest numerical value of this byte. Again for all pages having the same value of this numerical byte, so, having the same numerical value all pages for which the numerical value of this byte is same, I will choose that page which came into the memory at the earliest. So, among all pages having the same value of this numerical byte numerical value of this byte, I will use the FIFO strategy to choosing for choosing the page which has to be replaced.

(Refer Slide Time: 41:33)

## Clock Algorithm (Second Chance)

- On page fault, search through pages
- If a page's reference bit is set to 1
  - set its reference bit to zero and skip it (give it a second chance)
- If a page's reference bit is set to 0
  - select this page for replacement ✓
- Always start the search from where the last search left off

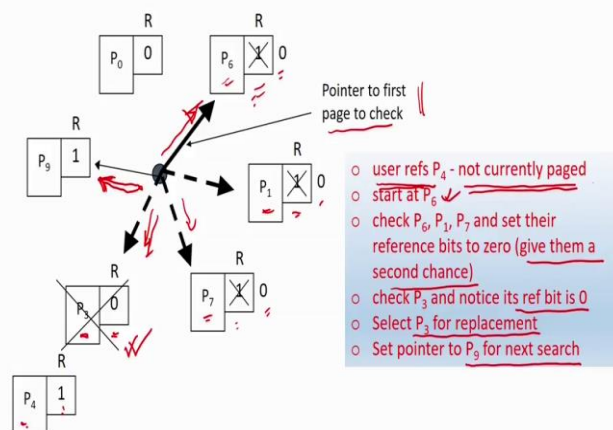
Computer Organization and Architecture

169

So, we will now look at the clock algorithm or the second chance page replacement algorithm. So, it is an extension, it is it uses the reference bits you uses the reference bits in a different way. So, how does this operate? So, on a page fault it searches through pages and then if the pages reference bit is set to 1, then it sets it to 0 and skips it. So, it gives this page as second chance. So, if it is 1 it does not it does not replace it, but it sets it to 0 ok. Now if a pages reference bit is 0 this is selected for replacement, if it is 0 then among all pages that is 0 I use the FIFO strategy to replace the one which has 0. Now it searches it starts the search from where the last search was left off.

(Refer Slide Time: 42:38)

## Clock Algorithm



Computer Organization and Architecture

170

So, how does this scheme work? So, let us say the previous search ended with the pointer; that means, the last page which was accessed was page 6 ok, which was accessed which was page number 6. Now this is the pointer to the first page to check. So, for all pages I have a circular linked list, for all pages I have a circular linked list. And for you suppose the user references page number 4 and that is not currently in the physical memory. So, I have to replace to bring in page 4, I have to replace some page. So, last time the search ended just before page 6, in the physical frame containing page 6. So, now, I will start from the frame containing page 6,  $P_6$ . So, this one becomes the first page to check.

Now, therefore so, first I find that the reference bit is 1 and not 0. So, I make the reference between 0 and I don't replace this page, then I come to the next frame; this frame contains  $P_1$ , the reference bit is 1 not 0. So, I only set it to 0 and go to the and check the next frame, I come to the next frame, it contains page number 7 and the reference bit is and the reference bit is 1. So, again I set it to 0 and I again proceed, when I come to page when I come to the next page frame I see that it contains page 3 and a reference bit for page 3 is 0; which means that in the current time interval page 3 was not accessed.

So, therefore, I select page 3 for replacement and after replacement I so, I select page 3 for replacement and it goes to  $P_4$  page number 4 comes in page of page 3 and the reference bit again is set to 1. And then for the next search it now points to this page frame. So, my current search ended at this page frame and then my next therefore, search will start from the next page frame. So, user references page 4 this is not currently paged, we start at page 6, we check  $P_6$ , then  $P_1$ , then  $P_7$  and set their reference bits to 0, give them a second chance. And then check page 3 and notice that the reference bit is 0 and then we select  $P_3$  for replacement and set the pointer to  $P_9$  for the next search. Now what is good about this algorithm is that if all pages are 1, then ultimately in the next round I will I will select this page.

This one has a low overhead why? Because I am not searching for all pages which has a reference bit of 0. So, possibly which has a reference bit of 0 and I am going on searching for the next page which has a reference bit of 0. So, if all pages have a reference bit of 1, it will the search will circle through all page frames in physical memory and then come back to the first page which was for which the memory bit was set to 0 from 1 and it will be the 1 to be evicted. So, although this page is referenced, but I have been able to give a second chance and because all pages have been recently used I could not find a page with the reference bits 0 and I choose I will choose that one for replacement.

(Refer Slide Time: 46:20)

## Dirty Pages

- If a page has been written to, it is dirty
- Before a dirty page can be replaced it must be written to disk
- A clean page does not need to be written to disk  
—the copy on disk is already up-to-date
- We would rather replace an old, clean page than an old, dirty page

Computer Organization and Architecture

171

Now, one aspect which the second chance algorithm does not take care, or ignores is that was is this page dirty or not is this has this page been modified has been written to or not, if a page has not been written to it is dirty. So, before a dirty page can be replaced it must be written to disk. Before a dirty page has been replaced it must be written to disk and this has higher overhead. A clean page does not need to be written to the disk and therefore, it has much lower overhead as replacement I can just discard this page because it is not written and therefore, it does not need to be written back to disk, it can just be discarded and in its place a new page can come to ok.

The page on disk is already up to date. So, we would rather replace an old clean page old and clean page will rather an old clean page than an old dirty page. So, if I if both pages are old I will choose one which is which is which is clean and which is not dirty, because it will I don't have to write that page back to disk.

(Refer Slide Time: 47:29)

### Modified Clock Algorithm

- Very similar to Clock Algorithm ✓
- Instead of 2 states (ref'd and not ref'd) we will have 4 states
  - (0, 0) - not referenced clean ✓
  - (0, 1) - not referenced dirty ✓
  - (1, 0) - referenced but clean
  - (1, 1) - referenced and dirty
- Order of preference for replacement goes in the order listed above

Computer Organization and Architecture 172

So, now, we will go and see an extension of the modified clock replacement algorithm, in which we use the 2 bits both the reference bits and the dirty bit. It is similar to the clock algorithm, but now each page instead of having 2 states whether it is referenced or not referenced will all have 4 states, whether it is referenced or not referenced and also whether it is dirty or non dirty. So, if a page has its reference bit 0 and its dirty bit also 0, it means that this page was not referenced in the current interval and is also clean; that means, it is while when replaced I don't need to write this page back on to disk.

If it is 01 it is not referenced and but it is dirty. So, it is not referenced in the current time interval, but in the last where ever it was last used whenever, it was last used it was written to therefore, if I choose this page for replacement I need to write this page first back into disk then bring a new page. Then the next one is 10, the page has been referenced in the current interval, but is clean that means, I don't I can discard this page ok. However, it was referenced so, it could be that it will again be referenced in the near future; however, because I have no one to replace I may need to replace this if the other options are not there.


So, and if it is 11 it is both referenced and then dirty, this one is the least preferred set of pages, if the if a page has the pages which have this 11 set they will be the least preferred set of pages to be replaced. So, the order of preference for replacement goes from the one above. So, how will the modified clock replacement algorithm work?



(Refer Slide Time: 49:38)

## Modified Clock Algorithm

- Add a second bit to page table - dirty bit
- Hardware sets this bit on write to a page
- OS can clear this bit
- Now just do clock algorithm and look for best page to replace
- This method may require multiple passes through the list



Computer Organization and Architecture

173

So, add a second bit to the page table the dirty bit, hardware sets this bit on write to a page fine, the OS can clear this bit ok. Now, just do clock algorithm and look for the best page to replace. So, what do you do? In one round of the clock you try to find 0 a page which is 00 and in this one if you have for all those pages which are 01 you set it to 00 ok. So, if you have the least significant bit 1 in the first round you set it to 0 and, you go on looking for a page which is which has a page which has both bits 0. If you find a page which has both bit 0, you use it for replacement. If you see that if both bits are not 0, then you find out whether the least significant bit is non-zero, if the least significant bit is non-zero you set it to 0 fine. Now, in the in the first round if now no page is found, then you find try to find a page for which is which is if you try to find a page which is then 10. So, you go on making multiple passes in the order of preference setting 1 bit to 0 at that time and hence you will you will you may require multiple passes to passes through the list to get to a desired page to get the desired page for replacement.

(Refer Slide Time: 51:17)

## Page Replacement - Example

- Consider a computer system with ten physical page frames. The system is provided with an access sequence  $(a_1, a_2, \dots, a_{20}, a_1, a_2, \dots, a_{20})$ , where each  $a_i$  is a distinct virtual page number. Determine the difference in the number of page faults between the last-in-first-out page replacement policy and the optimal page replacement policy.

Ans = 1

$a_1, a_2, \dots, a_{10} \rightarrow 10$  page faults.

$a_{11}, a_{12}, \dots, a_{20} \rightarrow 10$  page faults.

LIFO = 31 page faults

Opt = 30 page faults

11 page faults

$a_{10}, a_{11}, \dots, a_{19}, a_{20}$

Computer Organization and Architectur

174

So, now before proceeding we will take a small example. Consider a computer system with 10 physical page frames, so, I have 10 physical page frames. The system is provided with an access sequence  $a_1, a_2, \dots$  up to  $a_{20}$ ,  $a_1, a_2$ , so. Pages  $a_1$  up to  $a_{20}$  are accessed one after the other and again in sequence page  $a_1$  up to  $a_{20}$ ; where each  $a_i$  is a distinct virtual page number. So, each  $a_i$  is a distinct virtual page number. Determine the difference in the number of page faults between the last in first out page replacement policy and the optimal page replacement policy.

So, let us first see what will happen in the last in first out page replacement policy? I have 10 physical page frames. So,  $a_1, a_2, \dots$  up to  $a_{10}$  will result in compulsory misses. So, 10 page faults. So, compulsory page faults 10 page faults ok. Now  $a_{11}$  who so it is last in first out. So, last in is  $a_{10}$ ; so  $a_{11}$  will replace  $a_{10}$ ,  $a_{12}$  will replace who will who will it replace it will replace  $a_{11}$  ok. And then there will be again these 10 page faults up to  $a_{20}$  up to  $a_{20}$  so, 10 page faults ok. Now after this what do you have? You in the in the page frames you have  $a_1$  to  $a_9$ , so, after this sequence after this one is done, you have after the first set of 1 to 20 references you in the in the physical memory you have pages 1,  $a_1, a_2, a_9$  and  $a_{20}$ ; this is what you have. Now, therefore, the next set of 9 accesses do not result in a page fault in the LRU, in the least last in first out page replacement policy these will be hits. So, these will be page hits, this will not be page faults these will be hits.

Now,  $a_{10}$  will again result in a fault. Now who will  $a_{10}$ ,  $a_{10}$  will be who will it replace? So, this is a last in last in first out page replacement policy. So, therefore, so therefore,  $a_{10}$  will  $a_{10}$

when it is accessed it will be replaced by the one which came in last; who came in last  $a_{20}$  because,  $a_1$  to  $a_9$  resulted in page hits they were not brought in. So, last in, in this system of in this system that means, in the set of 10 page frames that I have currently this set of 10 page frames that I have currently  $a_{20}$  is the last in page.

So,  $a_{10}$  will be again replaced by this one. So,  $a_{10}$  will replace  $a_{20}$ ,  $a_{11}$  will again replace  $a_{10}$  will again replace  $a_{10}$  will again replace  $a_{10}$  and likewise,  $a_{20}$  will result in a page fault and it will replace  $a_{19}$  likewise. So, therefore, how many page faults will you have? You will you will have here, let us see these are 10 page faults again these are 10 page faults and, then from  $a_{20}$ ,  $a_{10}$  to  $a_{20}$  you will have 11 more page faults. So, this one will be 11 page faults.

So, for the last in for the LIFO strategy for the LIFO strategy, you will have 31 page faults; for the LIFO strategy, you will have 31 page faults. Now when you use the optimal page replacement, you see that this 20 page faults will still be there. In the optimal page replacement, these 20 faults will still be there because these are all compulsory page faults ok. Now in the in the optimal policy also when 11 comes it will replace page 10 because, it will see in future and find out whom to replace ok. It will find out that  $a_1$  and to  $a_9$  will be replaced will be needed later. So, it will go on replacing them ok. So, in a similar way as the as the LIFO strategy it will also do the same and at the end of the first 20 accesses it will also have the optimal policy will also have  $a_1, a_2, a_9$  and  $a_{20}$  in the in the memory ok.

Now, what will happen  $a_1$  to  $a_9$  will be page hits, when  $a_{20}$  comes in now which 1 will which page will the optimal page replacement policy replace? It will replace any one of the pages between  $a_1$  and  $a_9$ , but it will not replace  $a_{20}$  ok. So, why will that happen? Because  $a_{20}$  it knows that will be needed again. So,  $a_{10}, a_{11}, a_{10}, a_{11}, a_{12}$  these when these will be accessed it will go on accessing these one, it will it will keep  $a_{20}$  in the memory.

So, that when  $a_{20}$  is needed again later it will be found in the physical memory;  $a_{20}$  the access of  $a_{20}$  will not result in a page fault for the optimal page replacement algorithm because, it has a oracle to know which page will be used in future ok. So,  $a_{20}$  will not result in a page fault and therefore, the optimal policy will incur only 30 page faults. So, the question determine the difference in the number of page faults between the LIFO page replacement policy and the optimal page replacement policy, the answer will be 1, the difference is 1.

(Refer Slide Time: 58:05)

### Belady's Anomaly

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process):

	1	2	3
1	1	4	5
2	2	1	3
3	3	2	4

9 page faults
- 4 frames:

	1	2	3	4
1	1	5	4	
2	2	1	5	
3	3	2		
4	4	3		

10 page faults
- FIFO Replacement manifests Belady's Anomaly:  
—more frames  $\Rightarrow$  more page faults

Computer Organization and Architecture

Now we go on to understand one aspect of page replacement, which was important for which is important from a theoretical perspective and, also because this was quite a concern for early page replacement designer page replacement policy designers, who used a FIFO for their replacement algorithms. So, the people who used FIFO well encountered an anomaly which is which is popularly, or commonly known as Belady's anomaly.

Now, what is this? Let us say that you have 3 page frames, let us say you have 3 page frames in memory ok. The your whole memory consists of only 3 page frames, in one situation and 4 page frames in another situation. So, in one case you have a system containing 4 frames in memory and in one case you have 3 frames in memory. Now, sometimes it so, happens that for example, for this reference string it so happens that FIFO replacement for the FIFO replacement policy, when you have lower number of frames in memory, you will have lesser lower number of page faults, than when you have more number of frames in memory, you will have more page faults.

Now this should not happen right because, you have more space in physical memory when you have more space 4 frames in physical memory means that you have higher space the capacity of the physical memory is higher, than when you only have 3 frames in physical memory ok. So, the capacity of the physical memory is higher, but you still are incurring higher number of page faults. So, this is an anomaly which is referred to as Belady's anomaly.