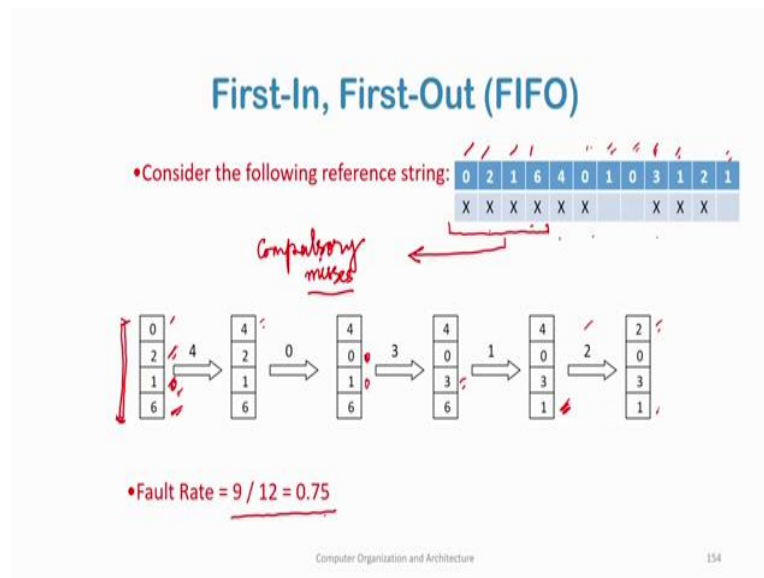


(Refer Slide Time: 63:00)



So, consider the reference string 2 0 1 6 4 2 0 1 6 4 0 1 0 3 1 2 1. Now here I have 4 page frames, let us say my physical memory only has 4 page frames available and these are the set of memory accesses. So, initially there is nothing in the physical memory. So, the first 4 misses are compulsory the first 4 misses are compulsory misses and. So, I bring in 0 2 1 6, 0 2 1 6 missed and then what happens 4 comes in whom will 4 replace 0 was the first one to go; 0 was the first one to come into memory. So, the 0 is the oldest. So, 0 is replaced and with 4.

Now, 0 is accessed again, now 0 is not there in the physical memory. So, 0 will again lead to another page fault and. So, whom will it replace 2 is now the oldest one who which came into memory. So, 0 replaces 2 then what happens one comes again one is referenced again this one does not result in a miss this one is already there in main memory. And this is not a miss, then 0 this also does not incur a miss 0 is already there in memory, it does not incur a miss then 3 is accessed when 3 is accessed 3 is not there in physical memory whom will it replace?

It will replace the oldest one because 0 2 1 6 was the order 0 and 2 has already been replaced. So, now, the oldest is one now the oldest is to 1. So, 3 replaces 1, 3 replaces 1.

Now, 1 is accessed again 1 is currently not there in physical memory 3 just replaced 1, 1 accessed again and then a one incurs a miss again whom does 1 replace will see 1 replaces 6? So, 1 replaces 6, now 2 is accessed 2 is accessed 2 is currently not there in physical memory. So, 2 replaces 4, 2 replaces 4 and because 4 is now the oldest and then 1 is now there again in physical memory. So, what is the fault rate? So, I had 12 the different string is of length 12.

So, I have 12 mem references out of 12 references nine resulted in a fault 1 2 3 4 5 6 7 8 9; 9 resulted in a fault. So, the fault rate is 9 by 12 or 0.75.

(Refer Slide Time: 66:05)

### FIFO Issues

- Poor replacement policy
- Evicts the oldest page in the system
  - usually a heavily used variable should be around for a long time
  - FIFO replaces the oldest page - perhaps the one with the heavily used variable
- FIFO does not consider page usage

Computer Organization and Architecture 155

So, FIFO although is very simple, I find out the oldest page in memory and replace it, this is the algorithm, it's a poor replacement policy why it evicts the oldest page in the system and it does not take care whether this page is being heavily used currently or not, it does sees who has been brought at the earliest and it will replace that, it does not care as to which if this page is being frequently used.

So, usually of a heavily used variable in a page should be around for a long time. So, we should try to keep a heavily used page for a long time, but FIFO is unaware as to how heavily a page is being used and may easily evicted. So, FIFO replaces the oldest page perhaps the one with the heavily used variable.

So, FIFO basically does not consider page usage.

(Refer Slide Time: 67:04)

**Optimal Page Replacement**

- **Basic idea** “  
—replace the page that will not be referenced for the longest time in future.”
- **This gives the lowest possible fault rate**
- **Impossible to implement** =
- **Does provide a good measure for other techniques**

Computer Organization and Architecture 156

Before proceeding to the next actual algorithm we will we will go into a hypothetical actual algorithm which actually can cannot exist in practice, but is used as a measure of comparison for all other algorithms.

So, this one is called the optimal page replacement algorithm what is the basic idea of the algorithm, I replace the page that will not be referenced for the longest time in future this is where is the why it is impractical, I want to replace that page which will not be referenced for the longest time in future. Now I cannot see the future and therefore, this algorithm cannot be implemented in practice because I am saying the future this algorithm will give the lowest possible fault rate page fault rate, this will give the lowest possible page fault rate. However, it is impossible to implement, it does provide a good measure for other techniques.

(Refer Slide Time: 68:00)

### Optimal Page Replacement

• Consider the following reference string: 0 2 1 6 4 0 1 0 3 1 2 1

0	2	1	6	4	0	1	0	3	1	2	1
X	X	X	X	X	✓	✓	✓	X	✓	✓	✓

*Compulsory* ←

0

2

1

6

4 →

0

2

1

4

3 →

3

2

1

4

• Fault Rate =  $6 / 12 = 0.50$  ✓  
• With the above reference string, this is the best we can hope to do

Computer Organization and Architecture 157

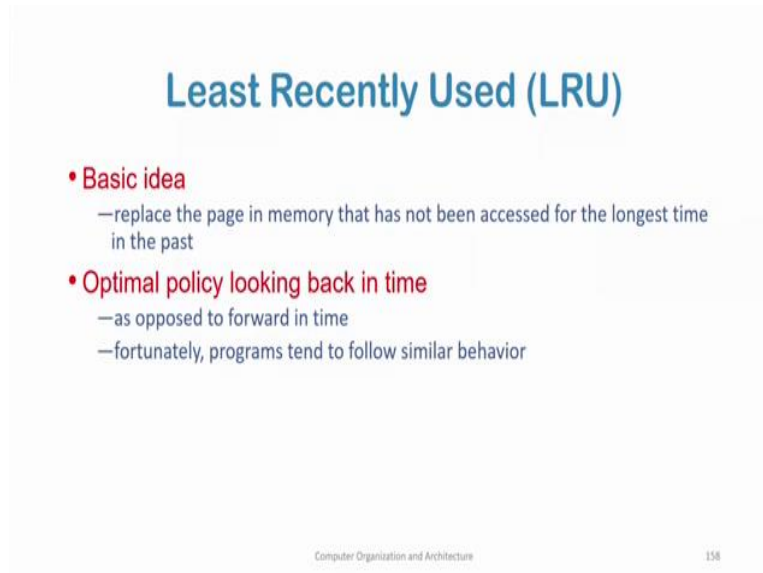
So, if we take the same set of 12 reference strings what happens is that this first 4 will still incur a miss because these are compulsory this is compulsory this is compulsory. Now the 4 is 4 is not there in memory. So, it will find out whom to replace it will find out that one which will not be used for the in for the longest time in future now out of this 12 reference 6 is never used in future.

So, therefore, this 4; this 4 will replace 6 here previously it replaced 0 in the in the FIFO algorithm 4 replace 0, but here 4 replaces 6 because it can see the future and it sees that 6 will not be used for the longest time in future. So, it replace 4 replaces 6 then 0 is there in memory. So, it is a hit one is a hit 0 is again a hit because these are there in the page frames in physical memory 3 is again a miss whom will 3 replace? 3 will replace 0 because out of these 12, it is never being used in the future. So, 3 replaces 0 and this one is a miss again 1 2 and 1. So, 1 2 and 1 are again hits.

So, in this algorithm we see that we have in addition to the 4 compulsory misses we only have 2 more um. So, we only have 2 more page faults I possibly have been talking of page faults as cache misses please bear with me then that was a mistake. So, these are all page faults not cache misses. So, I in addition to these 4 page faults I have I this one is a page fault and this one is a page fault. So, I have 6 page faults. So, the page fault rate is 6 out of 12 which is 0.50.

So, so, previously in FIFO it was 0 point in FIFO the page fault it was 0.75 and then optimal page replacement give me a fault rate of 0.5. So, with the above reference string this is the best we can hope to do.

(Refer Slide Time: 70:29)



## Least Recently Used (LRU)

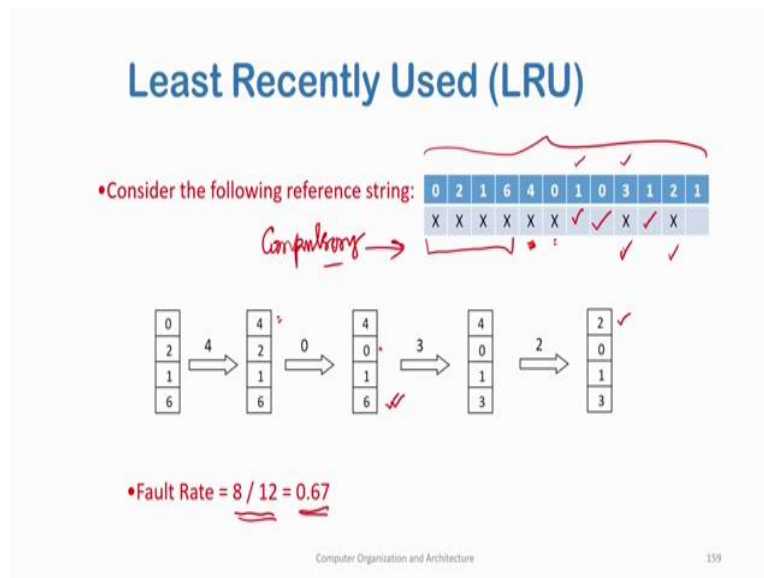
- **Basic idea**
  - replace the page in memory that has not been accessed for the longest time in the past
- **Optimal policy looking back in time**
  - as opposed to forward in time
  - fortunately, programs tend to follow similar behavior

Computer Organization and Architecture 138

Then we come to the next algorithm as its, it's a very popular algorithm; however, due to the problems with this implementation various approximations of the algorithm is used. So, we will first learn what the algorithm is. The basic idea is that you replace that page in memory that has not been accessed for the longest time in the past. So, this is practical because you can see the past you cannot see the future.

So, although it may it is algorithm is costly, it is possible to look into the past and therefore, this can give it is an optimal policy looking back in time. So, if I do not have the option of looking forward in time which I cannot have this is the best possible alternative that I have as opposed to forward in time, it looks back in time and finds, fortunately programs tend to follow a similar behavior. So, looking back in time is almost always very beneficial, because it allows you to utilize locality of reference.

(Refer Slide Time: 71:40)



Now, we take the same reference string again same set of 12 references on this the first 4 are again compulsory, these are compulsory, these are compulsory and then the fourth one replaces whom the fourth one replaces. So, all because it will replace 0 the 4 will replace 0 because all have been used once and the least recently used is 0 because 0 2 1 6 the least recently used is 0.


So, 0 is replaced and. So, by 4 and then 0 is accessed again. So, this will result in a miss. So, who we will replace; obviously, 2 we will replace because that is the least recently used. So, after this one here 2 replaces sorry 0 replaces 2 and then the subsequent one is a hit this one is a hit, again 0 is a hit and then 3, 3 is replaced by whom 3 is replaced by the least recently used. So, 3 is replaced by 6 because 6 is least recently used among all the page frames that we have currently in memory 6 is the least recently used and therefore, 3 replaces 6.

And then I have 1, 1 is a hit and then I have 2, 2 is a miss and who does 2 replace 2 replaces 4 because 4 is the least recently used. So, using a least recently used scheme the page fault rate is 8 out of 12 which is 0.67. Therefore, it is not as good as optimal page replacement algorithm because we cannot see the future, but it is not as bad as FIFO it is the best that we can do looking into the past.

(Refer Slide Time: 73:48)

### LRU Issues

- How to keep track of last page access?
  - requires special hardware support
- 2 major solutions =
  - counters
    - hardware clock “ticks” on every memory reference
    - the page referenced is marked with this “time” ✓
    - the page with the smallest “time” value is replaced ✓
  - stack
    - keep a stack of references ✓
    - on every reference to a page, move it to top of stack ✓
    - page at bottom of stack is next one to be replaced



The diagram shows a vertical stack of memory pages. The top page is labeled  $p_i$ . An arrow points from the top page to the bottom page, indicating the replacement policy. The stack is represented by a vertical column of boxes, with the top box being the most recently accessed page.

Computer Organization and Architecture 360

Now, as we told LRU is difficult to implement in practice because how do we keep track of the last page access, this is the problem requires special hardware support. So, I will I need to find out corresponding to each page; how recently, it was used in the past and for that we need to have we need to have special hardware support. So, there are 2 major solutions to this problem, one is a counter based solution.

So, it uses hardware clock ticks on every memory reference on every memory reference I have a hardware clock tick. So, reference one tick is one reference two; so, out of these 12 reference; tick 1, tick, 2 up to tick 12. So, 12th reference is a tick number 12. So, the page referenced is marked with time. So, each page when I have accessed because this tick is progressing globally over all memory references, when a particular page is accessed I will mark the time at which it was accessed.

So, in the future, I will be able to know when in the past was this page accessed because I am marking this page with the tick number when it is accessed. So, in future I will be able to access the tick number of this page to understand how back in the future this page was referenced. So, the page with the smallest time value is replaced the other way to implement this LRU is with the use of a stack we keep a stack of references, on every reference to a page we move we move it to the top of the stack. So, we keep a stack of references the. So, these are the memory references. So, pages  $p_1, p_2, p_3$  and  $p_1 p_j p_i p_j$  something these are the stack of references and

in every reference. So, if I reference this page in. So, this page is referenced then I take this page and move it to the top of the stack ok.

So, this will require possibly a linked implementation of the stack to handle it efficiently. So, on every whatever it is it is costly, but we will see these are the solutions then these solutions also these exact solutions for LRU also do not suffice. So, people have devised different approximation algorithms, but we need to see the solutions first. So, we saw the counter based solution the other one is the stack based solution, we keep a stack of references on every reference to a page we move this page to the top of the stack the page at the bottom therefore, because at every reference, if the page is referenced, I am moving to the top of the stack which one is the page at the bottom the page at the bottom is the least recently used page the page at the bottom of the stack is the next to be replaced, because this is the least recently used page and this page will be selected for replacement.

(Refer Slide Time: 77:06)

**LRU Issues**

- Both techniques require additional hardware
  - As memory reference are very frequent phenomena
  - Impractical to invoke software on every memory reference
- LRU is not used very often
- Instead, approximate LRU is more commonly used

Computer Organization and Architecture 161

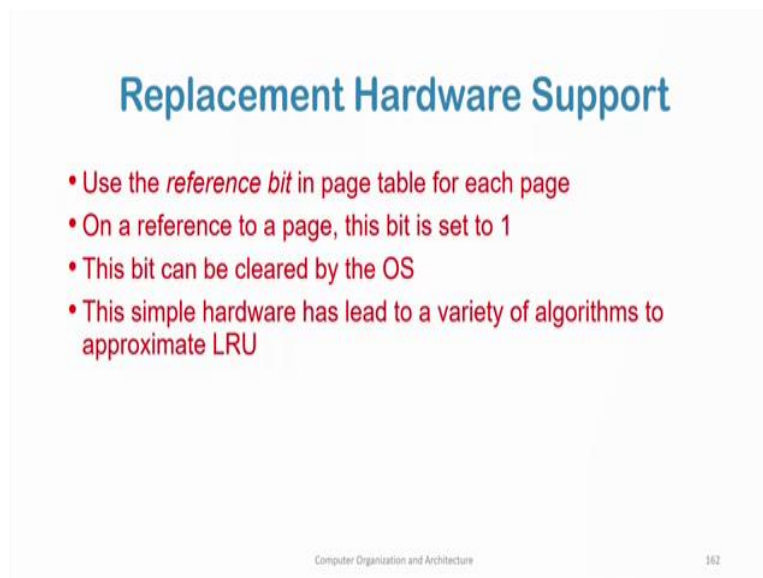
Now, both these techniques will require additional hardware support both the counter based technique which keeps count which for each page I need to keep the value of the tick when it was referenced. So, this one as well as the stack implementation will require additional hardware support a memory reference is very is a very frequent phenomenon. So, at every memory reference, I cannot go back to the OS I cannot interrupt the OS and update the value of the tick for a counter based implementation or I cannot update the stack when the stack is implemented as a software when the stack is implemented in software why because this will



be very costly because memory access is very common, and because it is very common, it is very frequent. So, it is impractical to invoke the software on every memory reference.

So, what software meaning here is the OS I have to invoke the OS to find out who referenced this to basically update the tick value corresponding to a page. So, I cannot do so in software because memory references are common and software will be costly the overhead will be very high. So, this is why LRU is not often used and instead, we approximate LRU we use an approximate LRU.

(Refer Slide Time: 78:37)



**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 362

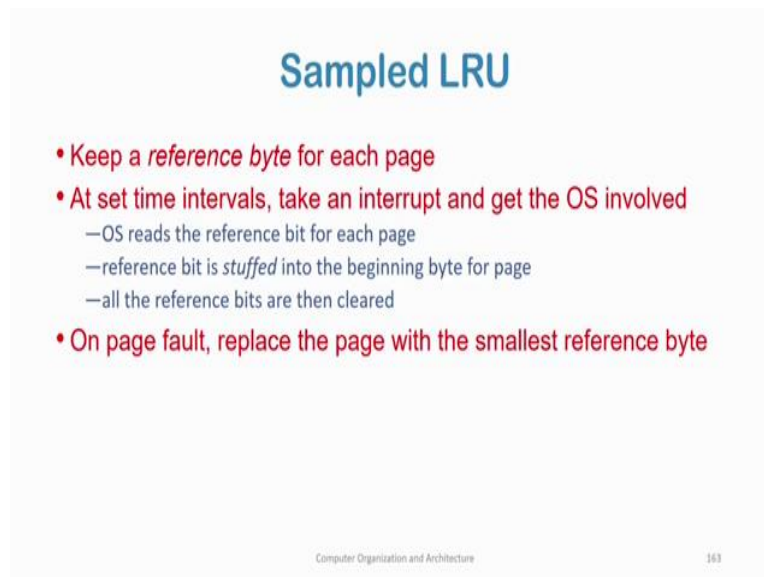
So, the first way to the way to approximate LRU is the use of the reference bit which we have which we have already studied earlier that reference bit tells me that within a given epoch of time whether I have referenced a page or not so corresponding to. So, I have in the page table in the page table corresponding to each page I have a reference bit for each page, right.

So, that reference bit tells me whether in the last epoch of time this page was replaced and at the beginning of each epoch a frame a stipulated interval of time at the beginning of the interval, what I do is I basically zero down all the differences and again find out, whether, it has it was referenced in the current epoch, I use a reference between the page table corresponding to each page and you know; what do I do this reference bit is set to 1, if this page is referenced in each epoch and at the end of the epoch I zero down all the reference bits I go to the OS I zero down all the reference bits and at a given time all the I try to find the page whose reference between is 0.

So, in the absence of the reference bit I take the FIFO order. So, in the FIFO order I try to find that page. So, I find out what I find out pages whose reference bit is 0 if the reference bit 0, it means that it was not referenced in the current epoch of time. So, it is not being heavily used it is not being heavily used I am approximating it is not exact, but I am approximating that it is not being heavily used because in the current epoch of time it was not referenced. And I replace that page ok.

This bit can then be cleared by the OS this simple hardware has led to a variety of algorithms. So, the first technique is that of approximate LRU is that I have a reference bit and that reference bit is there for each page in the page table and whenever a page is accessed within a given epoch of time, I set the reference bit at the end of the epoch I zero down all the reference bit corresponding to all pages. And then at any given time when I need a page to be replaced, I try to find out a page whose reference bit is 0 this will mean that in the current epoch it was not referenced and possibly this is not this page is not being heavily used and therefore, it is a good candidate to be replaced.

(Refer Slide Time: 81:24)



**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

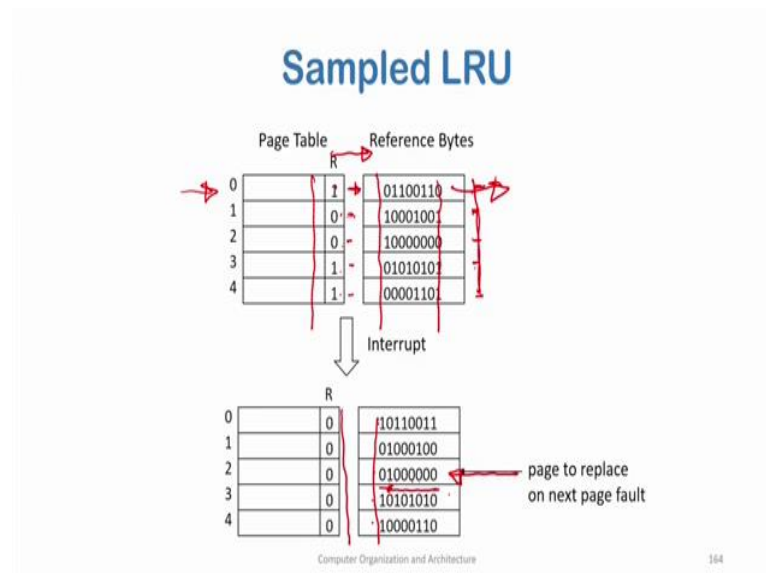
Computer Organization and Architecture 363

Then we come to sampled LRU. So, using the reference bit we generate a reference byte for each page in this in this technique. So, how do we get this reference byte at set time intervals which is this epoch I was talking about I take an interrupt and get the OS involved what do I do the OS reads the reference bit of each page and the reference bit is stuffed into the beginning

of the byte for page. So, at the beginning byte of the page I stuff this I stuff this bit all reference bits are then cleared.

So, on a page fault I replace the page with the smallest reference byte.

(Refer Slide Time: 82:12)



So, how do I take we will take an example? So, this is let us say the first byte of each page this one these ones are the this one is page one, page one, page 2, page 3, page 4, page 5. So, these are the first bytes and this is kept for reference bits. So, at a given point in time, let us say this is the value of the bytes and in this epoch the reference bit values are this; that means, in page for page 0, sorry, this is page 0 for page 0 in this particular epoch this page was accessed in this current time interval this page was accessed page 1 was not accessed, page 2 was not accessed, page 3 was accessed and page 4 was accessed.

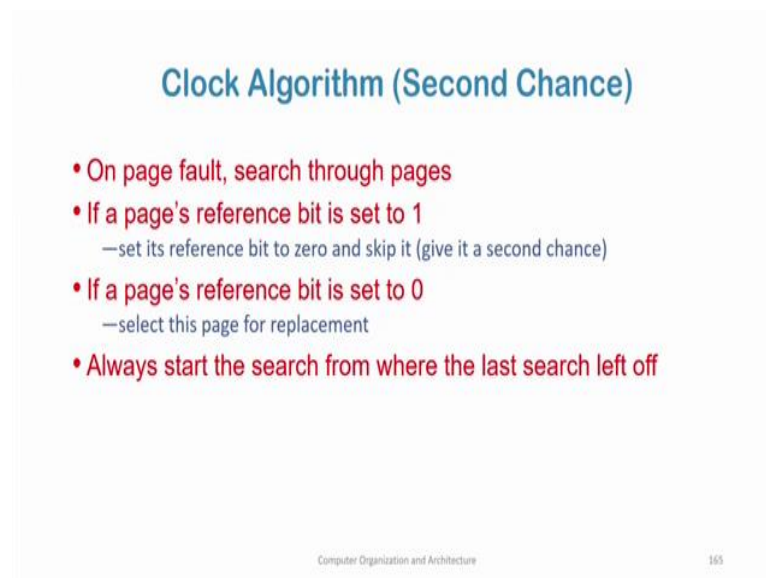
Then what happens I shift this bit sorry I shift this bit from here to the MSB this one at this place and I discard the I discard these I discard these ones I throw them off and I bring these bits into the MSB. So, after this I zero down all the references. So, what I have done I have taken the reference bit see that these reference bits these reference bits have now become the MSBs here have now become the MSBs here 1 1 0, 1 0 0 1 1, 1 0 0 1 1. So, these have now become the reference and these have come into the MSBs.

Now, which one will be replaced the one with the smallest reference byte will then be replaced. So, this one is the one with the smallest reference byte; what does this tell me that this value

basically the numerical value of this byte the numerical value of this byte. This is the smallest numerical value of this byte it tells me that it was replaced it was used this page was referenced in the previous memory.

This is the least recently this is the least frequently this is the least frequently being used it was only used among the last 1, 2, 3, 4, 5, 6, 7, 8 epochs; among the last 8 epochs, it was only used once and how recently, it was used in the in the current, but one. That means, the previous frame in the previous to previous epoch, it was used once, but all others have a higher value and therefore, this one is the least recently used it becomes the least recently used and is therefore, replaced.

(Refer Slide Time: 84:48)



**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 165

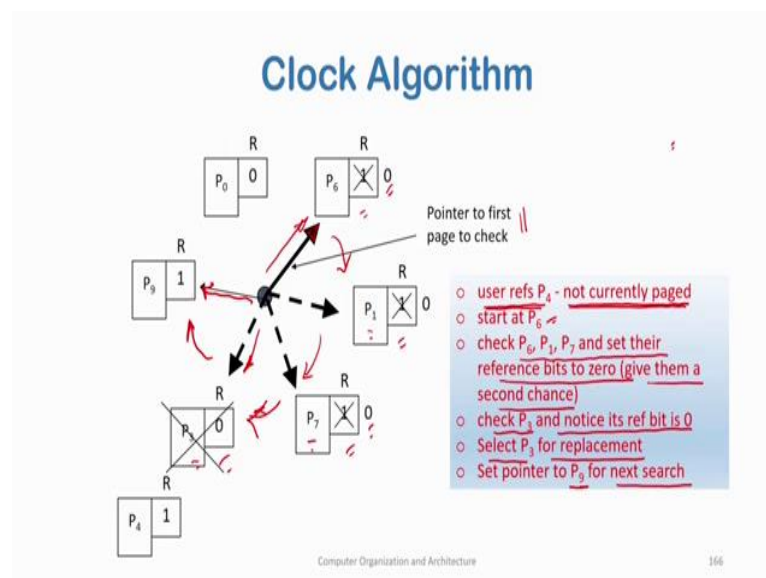
Then we come to the clock replacement algorithm now we come to the clock algorithm or the second chance page replacement algorithm this is another approximation of the LRU technique. So, here what happens on a page fault I search through pages if the pages reference bit is one if the reference bit is one; that means, it was referenced in the current epoch I set its reference bit to 0 and skip it. So, if the if the reference bit is one, I don't replace this page, one thing to note here is that in the absence of this reference bit I use I am using FIFO replacement algorithm.

Now, after seeing this among all these bits suppose I get a 0, I get a 0 reference bit. So, among all these bits which have 0 reference bit, I will use the FIFO technique whoever has among all

those pages for which the reference bit is 0, I will replace that page which came at the earliest ok.

Now, in the second chance page replacement algorithm if the next page that I am I am searching that the next page that I am looking for has a reference bit of 1, I don't replace that page I give it a second chance, I set its reference bit to 0 and skip it, give it a second chance, if the pages reference bit is set to 0 if the pages reference bit is 0 if it is already 0 then I replace the page. So, I always start the search from where I left off in the second chance page replacement algorithm as like other algorithms I am searching for pages to be replaced.

(Refer Slide Time: 86:47)



Now, let us say; the last time I replace the page frame was at page number six. So, according to the last slide as we saw my current search will start from page number 6, here my current search will start here. So, let us say the pointer to the first page to check. So, this gives me the pointer to the first page to be checked. So, I user references  $P_4$  user references p4 page number 4 which is currently not in the in the main memory.

So, I start at page 6  $P_6$  has the reference bit. So, I set to 0 and go I give it a second chance because if the reference bit was 0 I would immediately use it for replacement because the reference bit is 1 I give it a second chance. So, what do I set this reference bit 1 to 0 and go to the next one I come here, I get this reference bit 1. So, again I go to  $P_1$ , then I go to  $P_1$   $P_6$ , then  $P_1$  and  $P_7$  I find the reference bit to be one for all these 3 I change it to 0 and I move on when

I come to when I come to  $P_3$ , I see that the reference bit is already 0; so I therefore, choose  $P_3$  for replacement.

Now, let us say if all page frames had the reference bit of 1 if this would have a reference bit of 1 then it would go to  $P_3$   $P_9$   $P_0$  and then come back to  $P_6$ . So, when it comes back to  $P_6$  it is surely will get a reference bit of 0 because it it says because the algorithm has set the reference bit to 0. So, only when all reference bits of all pages are one, will it choose um a possibly frequently used page otherwise it will go and choose a page whose reference bit is 0 ok.

So, now so, what happens we start at page 6, we check page 6; page 1, page 7 and set their reference bit to 0 that is give them a second chance and then check page 3 and notice that its reference bit is 0, I select page 3 for replacement and set pointer to  $P_9$  for the next search, for the next search I will start from where I left off in the previous search. So, I replace this one in the in the previous search. So, I go to the next page frame which is  $P_9$  in the current in the next search, I will point to start mine next search with  $P_9$  for the next replacement.

(Refer Slide Time: 89:24)

**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 367

Now, in the last algorithm that we will study we use dirty pages. So, I all I along with the reference bit I also use the dirty bit that that we had studied earlier. So, if a page has been written to it is dirty before a dirty page can be replaced, it must be written to the disk. So, this is this is what I we don't want. So, reference a page can be reference, but a reference can be a write or a read, if it is only read if the page was referenced only for a read still the replacement

is less costly than if the reference bit is 1 and then the page is also dirty. So, in a replacement I have to write this page first back into the disk and then bring in a new page.

So, a clean page does not need to be written back. So, this is the advantage if the dirty bit is not set. So, the copy on disk is already up to date. So, I don't need to write back you would rather replace an old clean page; that means, yes than an old dirty page. So, I want to choose an old clean page than an old dirty page.

(Refer Slide Time: 90:36)

**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

*Handwritten notes:* "Refer bit", "Dirty bit", "for clock"

*Diagram:* A circular queue with a pointer and a small diagram showing the order of preference: 0, 1

Computer Organization and Architecture 168

So, this is the basis of the modified clock algorithm. So, the modified second chance algorithm.

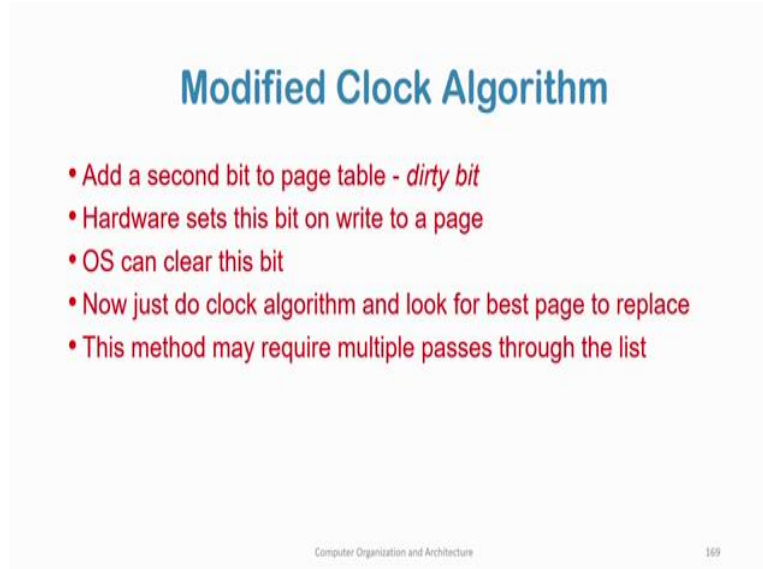
So, here it is very similar to the clock algorithm I similarly use a circular queue and keep the pointer to the last search and then start the search from there. So, you know; however, instead of 2 states. So, previously I already I only had 2 states for each page either the reference bit is 0 or 1, these are the 2 states of each page.

Now, I have 4 states and the 4 state is 00; that means, it is the reference bit is 0 as well as the dirty bit is 0 the ref not reference. That means, it is not referenced, it is an old page, but it is dirty; that means, that even if it is not replaced even if it is not referenced in the recent past if I replace if I choose this page I will have to replace this page if it is 10 it is referenced, but it is clean if; that means, it is used in the in the recent times in recent times it is being used this page is you being used heavily possibly, but it is being it is being read and it I don't have to write it and if both are one; that means, it is both being heavily used and the page is also dirty.



So, so, the order of preference for replacement goes down the order. So, this one is the highest preference highest preference and this is the lowest preference.

(Refer Slide Time: 92:12)



**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 169

So, add a second bits; how do I implement it add a second bit to the page table. So, we have the reference bit as well as the dirty bit the hardware sets this bit on write to a page the OS can clear this bit at the end of each epoch let us say. Now, just do clock algorithm and look for the best page to replace it is the same, I want to do the best page to be replaced, but now this method will require may require multiple passes through the list.

So, what will happen I will now first find try to find out none. Firstly, I will go through one round, I will setting the 0 reference bit to 0s, then I if all the ref, I do not find any if I do not find, I will try to find a page whose both bits are clean in the first round, if I do not find suppose for all of whom for which the reference bit is one I will set that to 0, then again in the next page, I will try to find someone with 0, if I don't get I will go on searching. So, I may require multiple passes over this over a particular set of page frames to find the page frames that I want to replace.

With this, we come to the end of this lecture.