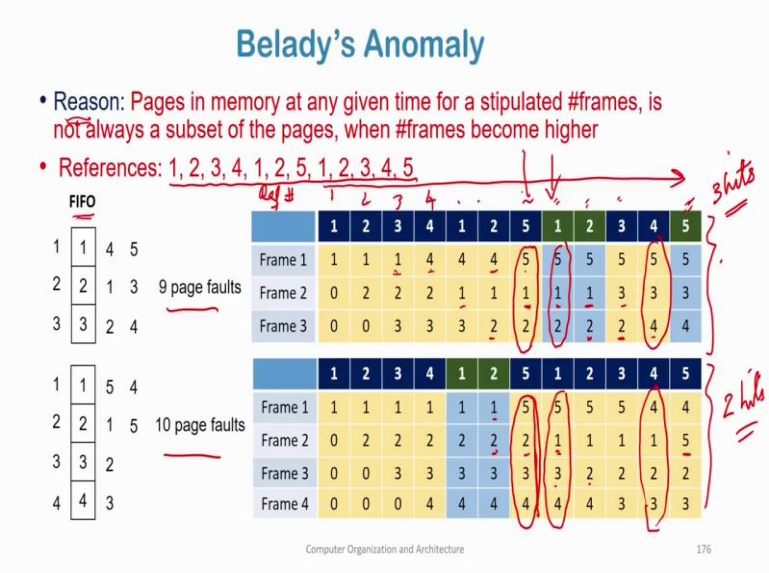
(Refer Slide Time: 60:08)



Now we will try to see in more detail why this happens? Now the reason for Belady's anomaly has been found to be this; pages in memory at any given time for the stipulated number of frames is not always a subset of the pages when the number of frames become higher. So, why does Belady's anomaly happen? Because, suppose I have 3 frames, at any given time the num, the frames the pages that are there in memory is not a subset of the of the pages that are there when my number of number of frames are higher.

So, when my number of frames are lower, if thus the pages that I am containing are not a subset when my number of pages are higher then Belady's anomaly occur. We will take an example we will take the example of this reference string and see why and how this happens. So, these are the set of memory references and these are the set of memory references shown here, 1 2 3 4 1 2 5 1 2 3 4 5. So, these are the references shown here. Now at time 0, so this 1 is time 0 time 1 time 2 first reference second reference third difference fourth reference like this.

So, ref #; this one is 1 2 3 4 like this it progresses so, at the first reference. So, I am bringing in page number page 1 to frame 1 and what happens is that it comes to frame 1 and frame 2, and frame 3 are empty. So, there is a page fault; there then 2 comes in and goes to the second empty frame, the frame 2 is empty and 2 goes here, for the 3rd reference 3 comes in frame number 3, for the 4th reference because I am using a FIFO policy first in first out. So, 1 was first in so, 4 replaces 1 and it again incurs a miss. Now, 1 comes in; now, 1 has already been replaced in the previous access. So, 1 again encounters a miss and it replaces 2.

Now, 2 is accessed again, 2 again incurs a miss because, 1 has replaced 2 and therefore, it replaces 3 then 5 is accessed now, 5 incurs a miss. So, the first in among these 3 pages is 4. So, therefore 5 replaces 4 in frame number 1. So, then what happens? Then we find that 1 comes in and 1 is already there. So, therefore, this is green and this is not a miss this is a hit then 2 comes in 2 is already there in frame number 3. So, this is again a hit, then 3 comes in; however, 3 has been replaced and because and because 1 is currently the one with the which is which is earliest which came into the memory earliest. So, 3 is replaces 1 and then 4 comes in and 2 is now the one which is the earliest one which has come into memory. So, 4 replaces 2 and then 5 comes in 5 is already there in memory. So, this one is a hit.

Now, when I have 4 frames in memory these are the set of accesses. Now, what has happened? See so, 1 comes in miss, 2 comes in miss, 3 comes in miss, 4 comes in miss, then 1 comes in 1 is a hit because it is already there I don't need to replace, 2 comes in 2 is already there it is a hit is not a miss, then 5 comes in 5 replaces 1 because 1 is the earliest one which came in it is in the FIFO order 1 is replaced, then 1 comes in; 1 is then 2 is the one in the FIFO order which needs to be replaced.

So, 1 replaces 2 then 2 comes in 2 replaces 3 and then 3 comes in then 3 replaces 4 and all other accesses are miss. Now, we see that when you have when you have 3 frames in memory, when you have 3 frames in memory the number of page faults are 9; you have 3 hits ok. And when you have 4 frames in memory, you have you have only 2 hits. So, therefore, you have 10 page faults, here you have 9 faults and here you have 10 faults.

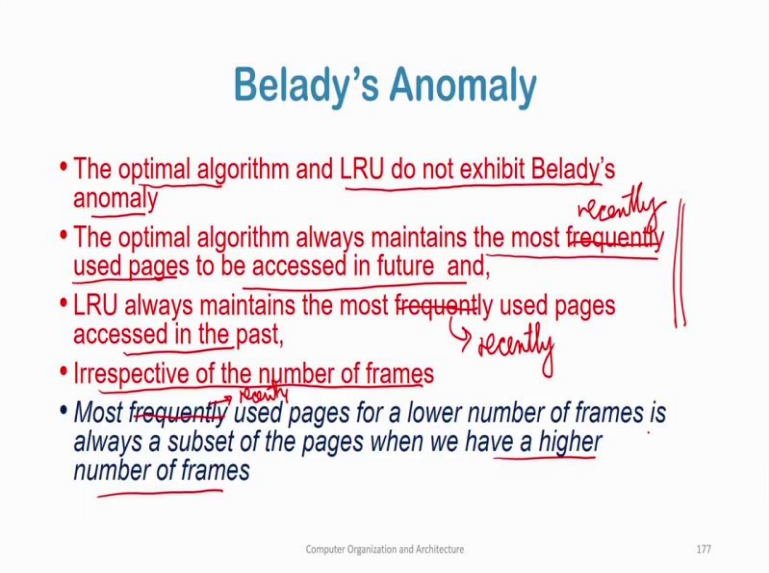
Now, let us see why this has happened? If you see what has happened. So, at this position I have 5 2 3 4 in the in the 4 frames and, here I have 5 1 2. Now, 5 1 2 is not a subset of 5 2 3 4 because, 5 1 2 because 5 1 2 is not a subset of 5 2 3 4 when 1 is accessed. So, when 1 is accessed 1 is there in memory here, but at this point in time 1 is not there in the physical memory when I have 4 frames. So, I have a miss here when 1 is accessed, but I have a hit here when 1 is accessed, why because if we go back to the definition again the pages in physical memory at any given time.

So, these are each different times, at any given time the pages in physical memory for a stipulated number of frames. So, pages are so for 3 frames for a stipulated number of frames 3, it's not always a subset of the pages that are there the pages that are there, when your number of frames become higher, when your number of frames become higher the pages are not a

subset. For example, here 5 2 3 4 is not a superset of 5 1 2 and therefore, for this case 1 is a hit and for this case it is not.

For the same reason again 2 is a hit so this 5 1 3 4 is so, 5 1 2 is not a subset of 5 1 3 4 and therefore is this 1 was a hit here, but this resulted in a miss. Again in this case 5 3 4 is not a subset of 4 1 2, 4 1 2 3 and therefore, when 5 is accessed here 5 results in a miss here however, 5 is a hit here ok. So, Belady's anomaly occurs because, pages in memory at any given time for the stipulated number of frames is not always a subset of the pages when number of frames become higher.

(Refer Slide Time: 67:41)



Now, the optimal algorithm and LRU both the optimal algorithm and the LRU do not exhibit Belady's anomaly. The optimal algorithm always maintains the most frequently used pages to be accessed in future. So, optimal algorithm what does it maintain? At any point in time the optimal algorithm maintains the most frequently used pages sorry, the most recently. The most recently used the most recently used to be used pages to be accessed in future and this one LRU keeps the most recently used pages most recently most recently used pages accessed in the past.

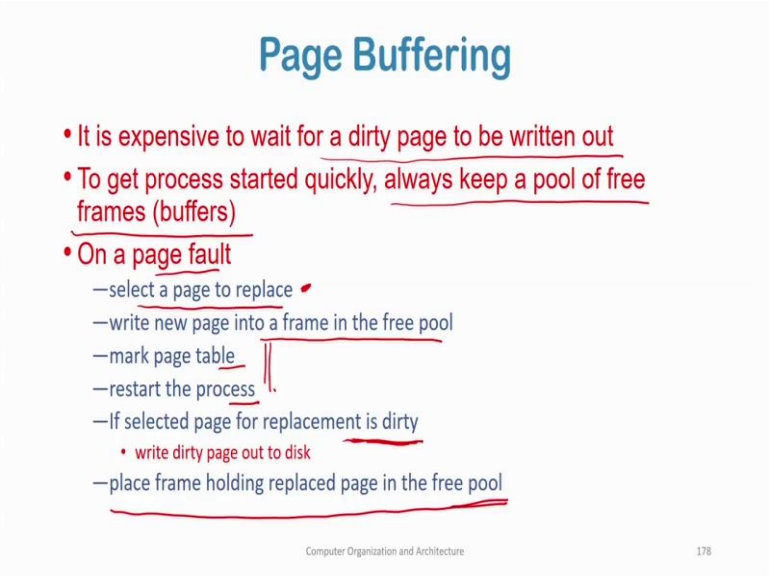
So, because LRU at any given point in time at any given point in time, whatever be the number of frames that are available. What does an LRU do? It replaces the least recently used page. So, therefore, what are the pages that are there in the memory at any given time the most recently used pages are there ok and what does optimal algorithm keep the most recently used pages that will be accessed in future. Whatever, I have written here may not be exactly correct,

but I am telling it that optimal algorithm will keep at any point in time the most recently to be accessed pages in future and LRU keeps the most recently used pages accessed in the past.

Irrespective of the number of frames: So, irrespective of the number of frames both these algorithms keep the most recently accessed pages. Now most recently used pages, most recently used pages for a lower number of frames is always the subset of the pages we have for a higher number of frames. Suppose I have a certain number of frames and, I and at a given time I have a certain number of pages in memory ok. Now the most recently you so, let us say I am using the LRU algorithm. So, the most recently used pages will be there in memory for 3 frames.

Now the most recently used pages at that point in time for 4 frames will always be a subset, will always be a superset of the most recently used pages if I have 3 frames. The most recently used pages for 4 frames will always be a superset of the most recently used pages for 3 frames. This is why Belady's anomaly is not there in the least recently used algorithm neither is it there for the optimal algorithm.

(Refer Slide Time: 70:38)



Now, we will go into another concept called page buffering. Now, page buffering it is expensive to wait for a dirty page to be written out. Now, let us say during a replacement I need to replace a dirty I have to replace a dirty page. Now, it is expensive to wait for the dirty page to be written out. Instead what we can do, to get this process quickly started we can always keep a pool of free frames. Now, these frames are free, how do we make it free on a page how

do we do this business? On a page fault, we select a page to replace ok and then write a new page into a frame in the free pool. So, at any given time I have a pool of free frames.

So, there I can just close my eyes and bring a page into this free pool without looking into anything else. But how do I maintain this free pool? I will first choose a page for replacement, mark the page table restart the process after replacement. So, I have brought a page from the disk and put it into this free frame, but then I take this page which I have used for replacement and put it into the free frame pool. So, at any point in time, I have a kitty of free frames that are ready for replacement. And for the current access I am using a free frame from my free frame pool, but I am using a free frame. So, I will replenish that free frame pool by one frame after replacement.

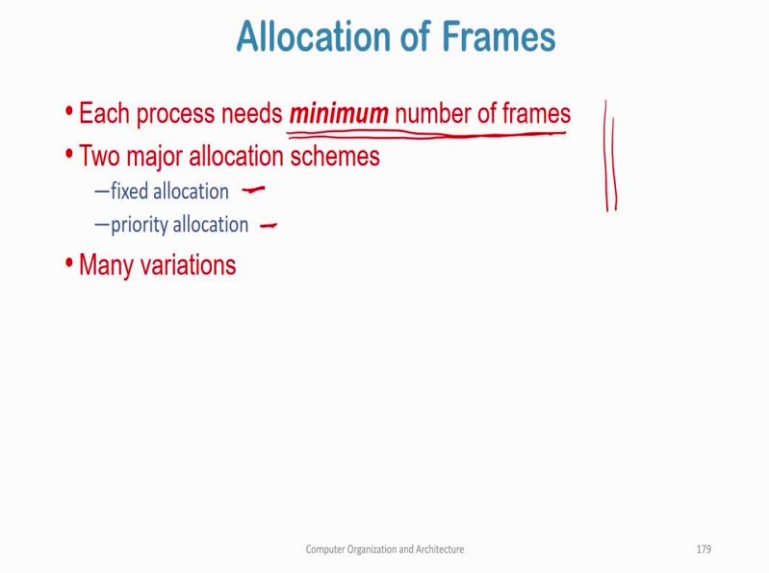
So, I will restart the process after replacement and then I will quietly do my business of creating one more free frame. So, if the selected page frame is dirty, then after replacement I will I am I will I will I will put that page into disk and then put this page into the free frame pool and because this dirty page was not put into the memory during replacement, this overhead is not visible to the page replacement. And this dirty page by putting this dirty page into the disk happens after the replacement and after the process has started and when this process is working in the CPU, I am replacing this dirty page into the disk I am putting this dirty page into the disk and creating the free frame pool. So, place frame holding the replaced bit in the free pool ok. So, this is how buffering will work.

Now, one more important aspect is that one more enhancement that we can do with this is that, when I am replacing when I am replacing I may not just I may not destroy the contents of the page, I have put it in the free frame, I have done the replacement is if required, but in the free frame let us keep the page intact don't destroy the contents of the page. If that page is required to be replaced in any case this page is clean and can be used because, it is there in the free frame pool, but by chance if this free frame which I have kept it in the free frame pool but by chance let us say this page is accessed by one process.

Now I can keep a pointer that this page is still there although it is there in the free frame pool, it is it is actually there in physical memory still now, it has not been replaced. So, I can I can access it from the free frame pool itself. So, I will get the page in main memory I have to I don't have to go to the disk to get this page. So, even if I put pages in the free frame I will just I will just keep a mark at as to what pages are there in the free frame pool at any given time

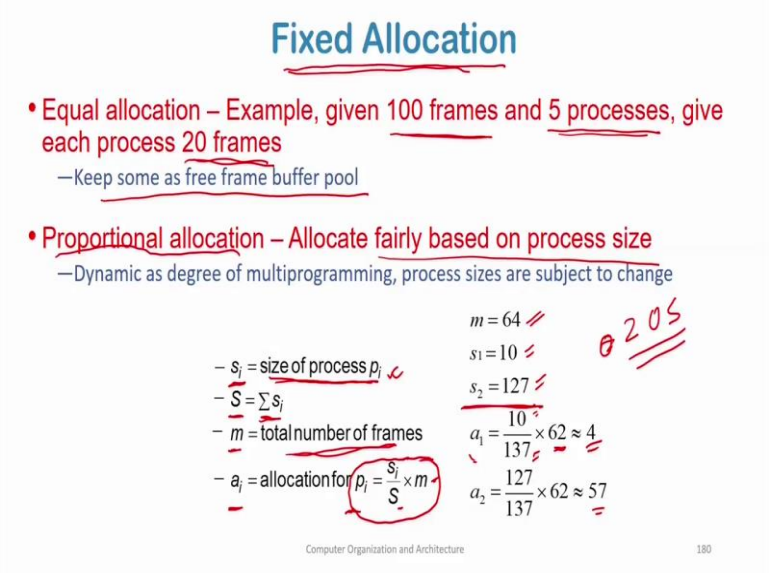
and if there are accessed I will be able to put that use that page from the free frame and use that instead of going to the disk.

(Refer Slide Time: 74:58)



Now, allocation of frames; now still now we are saying that a page does not a page frames have no connection with the processes. Now, this is entirely not true. So, each process requires a minimum number of frames. So, typically what do we do is that we allocate a certain number of physical page frames to each process. However, this is done using many schemes, there can be a fixed allocation scheme, there can be there can be a prioritized allocation scheme and it has many variations.

(Refer Slide Time: 75:39)



For example, the first strategy is to is the fixed allocation scheme. In the fixed allocation scheme let us say I have 100 frames in physical memory after these 100 frames are available after allocating frames to the OS and I have 5 processes. So, the degree of multi programming is 5, I give each process 20 frames, I divide the frames into this into this 5 processes. So, this one will be called a fixed allocation, fixed allocation. Otherwise we can do a proportional allocation, I allocate fairly based on the process size. So, the for example, let us say let s_i be the size of process P_i and S be the total so, size of the process P_i in say number of pages.

And S is the summation of the pages summation of the number of pages over all processes that are currently there that are currently active. And let m be the total number of frames that are there in physical memory. So, then the number of frames that will be allocated to a certain process to process P_i; a_i, the number of frames a_i that will be allocated to process the process P_i will be given by $\frac{s_i}{S} \times m$. So, this will this is so I allocate a number of frames proportional to the size of the process ok. For example, let us say I have 64, I have 64 frames in physical memory and s_1 has a size of 10 virtual pages, s_2 has a size of 27 virtual pages that it requires.

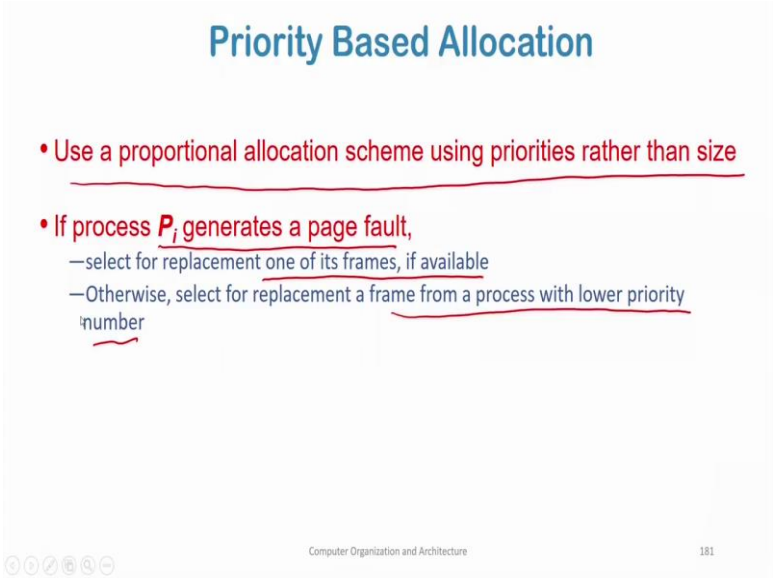
So, therefore $s_1 + s_2$ is 137 so, total number of pages summation si is 137 and out of which s1 has a size of only 10. So, out of so I allocate and let us say 2 pages out after out of this 2 this 64 - 1 is 64 - 2, so 2 pages I am using for OS. So, out of this remaining 62 pages out of this remaining 62 pages I will allocate 4 pages to process a1 and I will allocate the remaining 57

pages to process a_2. Why? Because P_2, the process P_2 has a much larger size s_2 equals to 127, 127 pages.

So that the number of frames that is expected to that this page process 2 is expected to require is much more than P_1 because, P_1 has a much smaller size with respect to in terms of the number of pages P_1 has a much smaller size. So, I will allocate when allocating frames, I will allocate P_1 only 4 page frames and, I will allocate P_2 57 page frames because, their sizes are also very skewed.

Now, if I allocate the same number of frames we understand that the memory the memory utilization will not be that good.

(Refer Slide Time: 79:25)

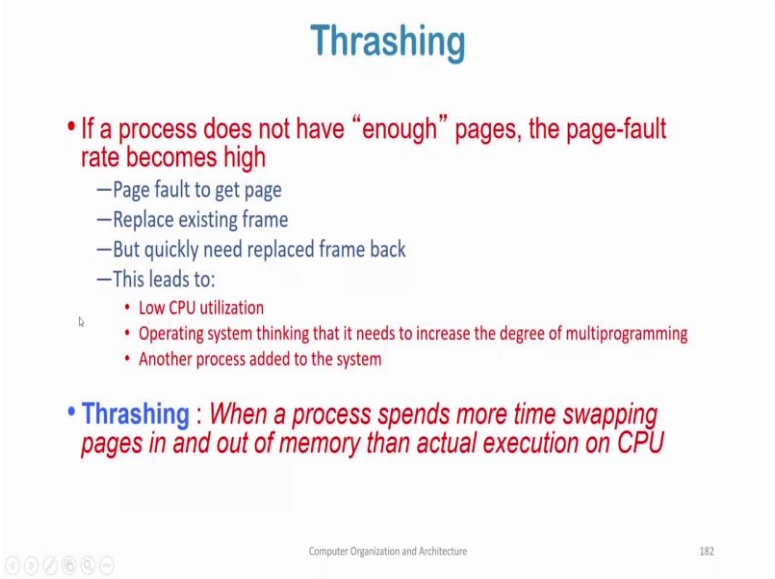


Now the next is priority based allocation. Now, priority based allocation is proportional allocation using priorities, now instead of instead of sizes I use priorities. So, if process P_i generates a page fault select for replacement 1 of it is frames is available, otherwise select for replacement of frame from a process with lower priority number.

So, now, what it does? So, I have allocated let us say based on proportion I have first allocated the frames to different processes. Now, during replacement, if I have free frames allocated to this process I do a local replacement, I will choose a frame from the frames I will choose a page for replacement from the frames which are allocated to this process. However, if none of the frames are free. So, if all frames allocated to this processes are busy; that means, are

allocated and no frame is free, then I will choose a frame the free frame from a lower priority process. So, I will take a frame from a lower priority process ok.

(Refer Slide Time: 80:45)



This is the priority based allocation scheme. Now, after this we come to the concept we understand the concept of thrashing. So, this we will continue in the next lecture.