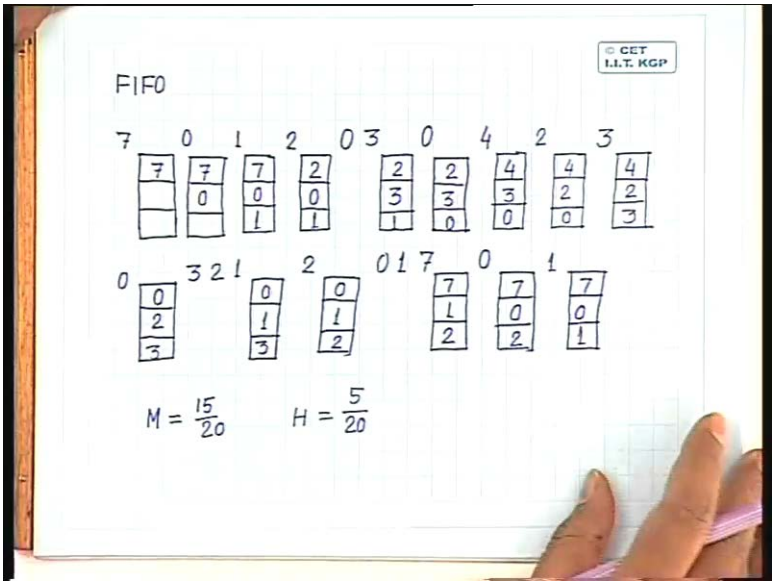
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture No. # 17
Memory Organization – V

So, we will discuss about the performance of these three different page replacement techniques with the help of this page trace. So, first let us see that how does the FIFO algorithm, the FIFO technique will perform. So in case of FIFO technique, as we said that the page which is the oldest one that will be replaced.

(Refer Slide Time: 00:01:29 min)



So first time when the page number 7 is referred, initially I don't have anything in these page frames. So obviously I will have a page fault and when I have this page fault, the page number 7 has to be brought from the secondary storage into the main memory. So initially out of these three frames, only the first frame will contain page number 7, the other frames will remain empty. Next time page number 0 is referred, again there is a page fault. This time I have 2 free frames, so one of the frames will be used to store page number 0 and I will have one more frame free.

Next time when page number 1 is referred then the situation will be like this. I will have frame numbers 7, 0 and 1 available in the main memory. So now all the 3 frames are occupied by 3 different pages. Next time when page number 2 is referred, there is a problem because I don't have any free frame in the main memory. And because this is FIFO algorithm, so out of these 3 pages whichever is the oldest which was brought in first that has to be replaced and in this case it is page number 7. So page 7 will be replaced by page number 2 and other 2 pages will remain in the main memory.

Next time you refer page number 0 which is there in the main memory, so there is no page fault. Next time page 3 is referred, 3 is not available in the main memory, so there is a page fault and now I have to select another page to be removed. So out of these 3 frames, these 3 pages page number 2, 0 and 1, page number 0 is the oldest one. So page number 0 is replaced by page number 3 and other 2 pages remains in the main memory. Now find the problem. Next page that is referred is again page number 0 and page number 0 has just been replaced. But I don't have any way, there is a page fault and so page number 0 has to be brought in and in this case the page which will be replaced is page number 1 because that is the page out of these 2, 3 and 1 which is the oldest.

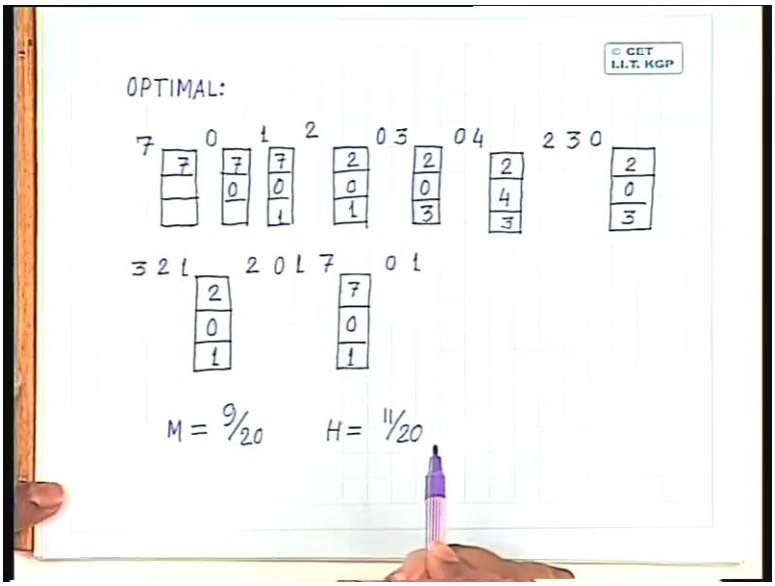
Next time page number 4 is referred, again there is a page fault and this time which page will be replaced? Page number 2, so 2 is replaced by 4, other 2 pages remain in the main memory. Next time 2 is referred; again I face the same problem. This time the page which will be replaced is page number 3. So I have page number 4, page number 2 and page number 0. Next time again page number 3 is referred. I have another page fault, so this time it is page number 0 which is to be replaced, so I have 4, 2, 3.

Next time which is the page that is referred? again page number 0. Again I have page fault and now I have to replace page number 4, page number 4 by page number 0 and other two pages page number 2 and page number 3 will remain in the main memory. What is the next page that is referred? Page number 0 then page number 3, it is there in the main memory so there is a no page fault. After that page number 2 is referred that is again in the main memory, there is no page fault. Then page number 1 is referred, again there is a page fault and you find that out of these 3 pages, page number 2 is the oldest. So I have to replace page number 2 by page number 1. So this is the situation that I have.

Again page number 2 is referred, so I have page fault again and this time page number 3 is to be replaced by page number 2 so I have 0, 1 and 2. Next time which page is referred? This is page number 0 I think, 3 2 1 2 0. Page number 0 is there in the main memory, so there is no page fault. Next time again page number 1 is referred which is again there in the main memory, so there is no page fault. Then page number 7 is referred. I have one more page fault and in this time which page I have to replace? It is page number 0, page number 0 is replaced by page number 7. So other pages remain in the main memory.

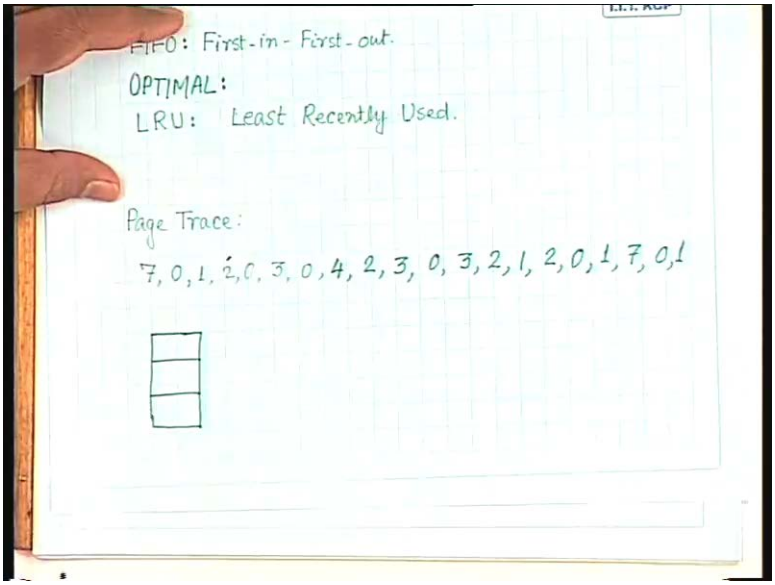
Next time again 0 is referred, so I have one more page fault and this time, the page which is to be replaced is page number 1. So I have 7, 0, 2. Next again 1 is referred, so I have one more page fault and this time page number 2 is to be replaced by page number 1, so I have this situation. So, you find that in this case and I think that is the last page in the example that I have taken. So total number of page references that we have is 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20. There are total 20 references out of which I have 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15; 15 page faults or 15 misses. So accordingly I will have missed issue to be 15 by 20 and I have only 5 hits. So hit ratio will be 5 by 20. Now let us see that what will be the situation, if we go for optimal page replacement?

(Refer Slide Time: 00:08:11 min)



Optimal page replacement is the one which replaces that particular page which will not be referred for longest duration of time. So that is the difference between the FIFO and the optimal. in case of FIFO, we have replaced the page which is the oldest in the main memory. In case of optimal page replacement, the page which will not be referred for the longest duration in future that will be replaced. So following that criteria, the initial three steps will remain as before because initially I don't have any page in the main memory, so initially these three page faults will come. Now next time when page number 2 is referred, I have to find out that which page out these three pages will not be referred for the longest period.

(Refer Slide Time: 00:09:31 min)



Now if you refer to this page trace, right now we are at this point, the page number 2 is being referred. the next page which will be referred is page number 0, then page number 3, then page number 0, then page number 4, then page number 2, 3, 0, 3, 2, 1. The page which will not be referred for the longest period is page number 7. So I replace page number 7 and incidentally in this case, it is same as in case of FIFO. So I replace page number 7 by page number 2 and the situation becomes like this. Sir how will you make sure that to, I will come to that later. For the time being let us assume that I know how the pages are going to be referred. So I replace page number 7 by page number 2 because it is page number 7 which will not be referred for the longest duration.

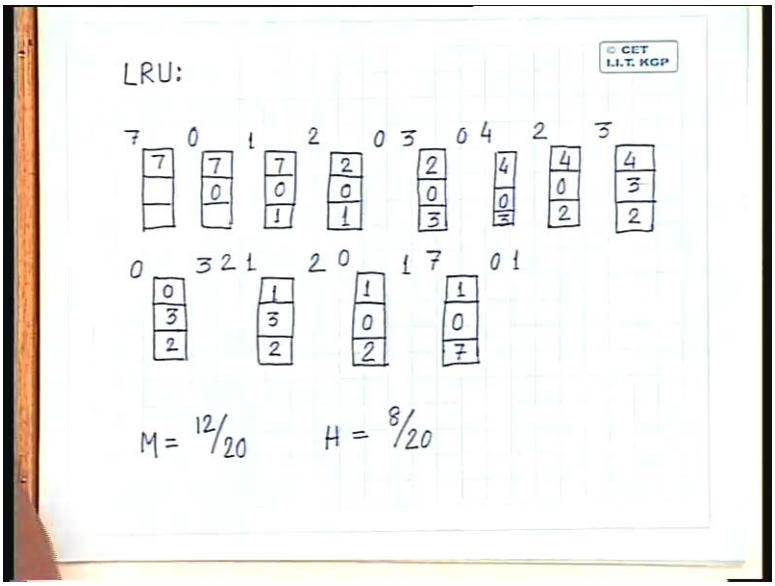
Next time 0 is referred, there is no page fault. Next time 3 is referred, again there is a page fault. So again when 3 is referred and I have a page fault then out of these three pages, I have to find out the page which will not be referred for longest duration. And you find that after these page number 0 is going to be referred, then page number 2 is going to be referred. Page number 1 will not be referred for longest duration, so I replace page number 1 by page number 3. Now we find the difference. In case of FIFO, we had replaced page number 0 and in case of optimal algorithm we are replacing page number 1, so it is 2 0 3. The next page which is referred is again page number 0, so I don't have any page fault. Next page which is referred is page number 4, again I have a page fault. Now again I have to find out that out of these three pages which is the one that will not be referred for longest duration. You find that just after this, now we are referring page number 4 so I am at this point. The next page that is going to be referred is page number 2 then page number 3, so both 2 and 3 will be referred in near future. So out of these three, it is page number 0 which will not be referred for longest duration, so I replace page number 0 by page number 4, so the situation is 2 4 3.

Next time page number 2 is referred there is no page fault. page number 3, no page fault then page number 0, again there is a page fault and at this point out of these 3 pages 2 4 and 3, you find right now I am at this point. The next page that is going to be referred is page number 3, after that page number 2 and it is page number 4 which will not be referred for longest duration. In fact in these trace page number 4 will not be referred any more. So I replace page number 4 by page number 0, it is 2 0 3. Next time 3 is referred, there is no page fault, 2 is referred no page fault, one is referred again there is a page fault. So right now I am at this point, out of this you find the next page that will be referred is page number 2 then page number 0. Three is the one which will not be referred for longest period and in fact here three will be not referred any more. so I replace three by page number 1. So it becomes 2 0 1. The next page that is referred is 2, there is no page fault.

Next page referred is 0, again there is no page fault. Next page referred is 1, no page fault. Next is 7, again I have a page fault. Now I have to find out that out of these three pages which one is to be replaced? You find that here right now I am at this point, page number 0 and 1 are going to be referred in future. It is page number 2 which is not going to be referred. So I simply replace page number 2 by page number 7. Next time 0 is referred, no page fault. 1 is referred, no page fault. So here you find that out of 20 total references I have miss in 1 2 3 4 5 6 7 8 9, 9 cases. So the miss ratio is 9 by 20 and obviously hit ratio will be 11 by 20 and which is a significant improvement over this FIFO technique.

In case of FIFO, I had 15 misses, in this case I have only 9 miss. Here I had only 5 hits, in this case I have 11 hits. So naturally the performance of this optimal replacement technique will be much much better than that in case of FIFO technique. However the problem is as just I pointed out that I am making an assumption, I am trying to find out that in future how the pages are going to be referred but which you may not know in advance. So that is why an approximation, I will tell you later why you call it an approximation. The approximation to this optimal technique is what is called LRU or least recently used technique. So in case of LRU technique, we will replace the page which is least recently used. I will never replace the page which is most recently used.

(Refer Slide Time: 00:15:54 min)



So again here the initial three situations will remain the same because initially I don't have any thing in the main memory, so initial page faults cannot be avoided. Next time the next page that is referred is page number 2. Now when I am referring page 2 in case of optimal algorithm, I try to find out that which page will not be referred for maximum duration of time. Now I had to find out that which page is least recently used. So you find that out of these three pages, page number 1 is most recent, before that page number 0, Least Recent is page number 7. So I replace page number 7 by page number 2 and incidentally in all these three cases, it has found that it has been page number 7 which is least recent but back may not be true always. So what I do is I simply replace page number 7 by page number 2, so I have 2 0 and 1.

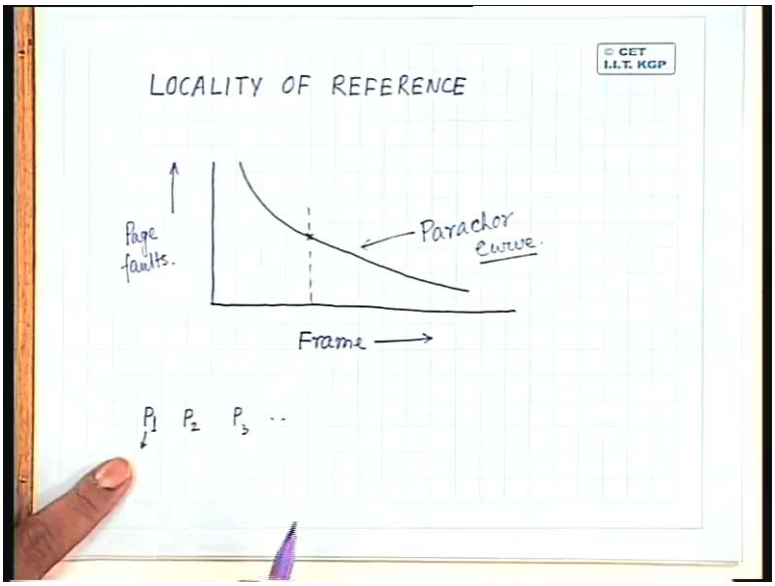
The next page that is referred is page number 0, I have no page fault. The next page that is referred is page number 3, again I have a page fault and I have to find out that out of these three pages in main memory which one is least recent. Zero is most recent, before that page number 2, 1 is least recent. So I replace page number 1 by page number 3, so situation is 2 0 3. Next time page number 0 is referred, no page fault. Page number 4 is referred, again there is a page fault and this time it is page number 2 which is least recent. Because 0 is most recent, before that 3, 2 is least recent. So I replace page number 2 by page number 4. Next time 2 is referred, again I have page fault.

So out of these pages, it is page number 3 which is least recent. I replace page number 3 by page number 2. Next time 3 is referred again I have a page fault. Now the prediction is not working properly and in this case which is the page that is least recent. That is not 4, it is page number 0. 2 is most recent, before that page number 4, page number 0 is least recent. So you replace page number 0 by page number 3. So the situation is 4 3 2 but again page number 0 is referred. So I have one more page fault and in this case which one is to be replaced? It is page number 4, so replace page number 4 by page number 0 and others will remain as it is. Next time 3 is referred, no page fault; 2 is referred, no page fault; 1 is referred again I have a page fault. And this time it is page number 0 that is to be replaced. So I replace 0 by 1 others remain as it is.

Next time 2 is referred, no page fault. 0 is referred again there is a page fault and this time which one is to be replaced. Page number write one it should be page number 3. So 3 is replaced by page number 0, others remain in main memory. Next time 1 is referred, no page fault, 7 is referred, I have a page fault and out of these pages, it is page number 2 which is least recent. So I replace page number 2 by page number 7, so it becomes 1 0 7. Next time 0 is referred, no page fault, one is referred no page fault. So what is the number of miss in this case? 12. So miss ratio is 12 by 20 and hit ratio obviously 1 minus that which is equal to 8 by 20; so I have 12 misses and 8 hits.

So now you compare these three. In case of FIFO, I had 15 misses, 5 hits. In case of optimal I had 9 misses, 11 hits. In case of LRU technique, I have 12 misses and 8 hits. So you find that these LRU technique comes in between FIFO and optimal technique where the performance is not as bad as FIFO and it is not as good as optimal. Of course optimal technique is the best beyond which we cannot go. So what is implemented is this LRU technique, though it is not as simple as FIFO but because of this improved performance it is better to implement LRU technique. Now why does LRU technique gives performance better than that in case of FIFO? What is said is LRU technique is an approximation of the FIFO technique. The reason is if you study the characteristics of any typical program, you will find that the moment you access an instruction or a data from a particular address L, next time the address will be generated around L. It will not be too far from L. So that is the property which is called locality of reference.

(Refer Slide Time: 00:22:08 min)



So the memory references that are made by a typical program over a short interval of time, over a short period of time tries to be a localized to a particular of region. So that is a property which is known as locality of reference. So it is because of this locality of reference, you find that once a particular page say page number 5 is referred, the probability that the same page will be referred in near future is very high. Whereas if a page is not referred for a longer period, the probability that same page will be referred next is very low, the same page will not be referred for a longer period in future. so because of this property of locality of reference, you find that LRU technique least recently used technique, it tries to approximate the optimal technique because it always replaces a page which has not been referred for the longest period that means the page will also not be referred in near future.

So you try to approximate the property of optimal technique by this LRU technique and because of this, the LRU technique performs better than in case of FIFO technique. Of course because we are going for a prediction, you are not taking the actual situation into consideration which is done in case of optimal technique because it is a prediction in some cases, prediction fails and that we have seen while expanding this LRU replacement. So that is why LRU will not be as good as the optimal technique but it will be better than FIFO technique and this is the one which is usually implemented.

Now this example we have taken with the help of three frames, we have assumed that I have three frames in the main memory which is to be used to accommodate this different pages as generated by an user process. If I increase the number of frames beyond three, suppose I make it 4 or I make it 5, you will find that the number of misses will go down considerably and the number of hits will increase obviously because I accommodate more number of pages in the main memories simultaneously. At the same time if I reduce the number of frames, the number of hits will decrease and the number of misses will increase. So if I plot for any particular program, the number of miss or the number page faults versus number of frames, so on this side I have frame numbers, on this side I have page faults.

Then I will get a curve something like this. As you increase the number of frames, the number of page faults will decrease, as you reduce the number of frames the number page faults will increase. So we will find that somewhere here I get optimal performance. If I increase the number of frames beyond this, though the number of page faults decreases but the relative improvement or the relative decrease in the number of page faults with respective to increase in the number of frames is not that profitable. Whereas if I reduce the number frames below this, in that case the number of page faults is going to increase tremendously. So for a particular application if I can maintain number of frames somewhere here, in that case I will get optimal performance and this region experimentally has been found to be half of the number of pages that you have in the logical address space.

If for a particular program, the number of pages is 100 then if I can maintain 50 frames to accommodate pages of that program, I will be in this region and this kind of curve is called a parachor curve. Now coming to this page replacement technique, again in addition to this LRU or optimal or FIFO technique, I can put one more consideration that is I can go for global replacement or I can go for local replacement. When I say local replacement it means that whenever a page fault is generated by a particular process and I find that no frames is available in the main memory for replacement, in that case I will take a page for removal which belongs to the same process. So if it is process P_1 which has generated a page fault then that means the next page for process P_1 has to be brought into the main memory or that one of the pages has to be removed from the main memory.

The page which will be removed that should belong to process P_1 only so that is what is called a local replacement policy. In case of global replacement policy, you don't put any such restriction. A page which is to be removed that may belong to any process in the system. So in that case replacement becomes global and in the other case, the replacement becomes local. Now find the advantage of the segmented and the paged memory management system. In case of local replacement what you said is suppose the page fault has been generated by some process say I have a number of processes running in the system process P_1 P_2 P_3 and so on. Now for all these processes there will be some pages deciding the main memory. Now while execution suppose process P_1 puts some memory reference or some page reference which is not existing in the main memory that means there will be a page fault.

Following this page fault, a new page has to be brought into the main memory from the secondary storage and at this point if there is no free frame available in the main memory then I have to replace one of the existing pages. To select one of the existing page, the policies that we have said, the algorithms can be FIFO algorithm, can be optimal algorithm. Optimal is not implementable, it can be error algorithm that means I am selecting one of the pages from the main memory following LRU to be replaced, to be removed. I can put an additional restriction that the page that will be removed should also belong to process P_1 because I am bringing in a new page for process P_1, so the page that is going to be replaced that should also be of process P_1 only. That is local to process P_1, so it is a local replacement.

In case of global replacement I don't put any such restriction. The page which is to be removed that may belong to process P_2 that may belong to process P_3, it may belong to process P_4 also but whichever is least recent considering all these pages in the main memory that is to be

replaced. I don't consider to which process that page belongs so that is a global replacement. And in the other case if I put a restriction that if I want to bring in a page for process P_1, a page of process P_1 should be removed. So that becomes local to process P_1 and it is local replacement policy. Now when you go for, when we had discussed about the segmented memory management, we have said that we wanted to go for segmented memory management simply because it maintains the logical structure of the program which is not there in simple paging technique. And maintaining that logical structure is very important when you talk about this virtual memory because in case of virtual memory to bring in a new page, I may have to replace an existing page. Now you find a case that I have for loop, for the for loop to be executed say for 1000 times.

If I simply go for paged memory organization in that case it is possible that the for loop is broken into a number of pages. So when you move from the first instruction in the for loop to the last instruction of the for loop, again I have to go back to the first instruction. Considering this demand paging or virtual memory management, if it so happens that ones; suppose first time I have loaded the first part of the for loop in the main memory. So I have executed a number of instructions. Next time when I want to execute an instruction for the second part of the for loop may be that page does not exist in the main memory. So I have to bring in this page in the main memory and while bringing it, I may have to replace another page. If it so happens that the page is to be replaced that is the first part of the for loop and this process will continue again and again. I have to bring in a page, swap it out, bring in another page, swap it out and that process will continue. Why not? It may happen.

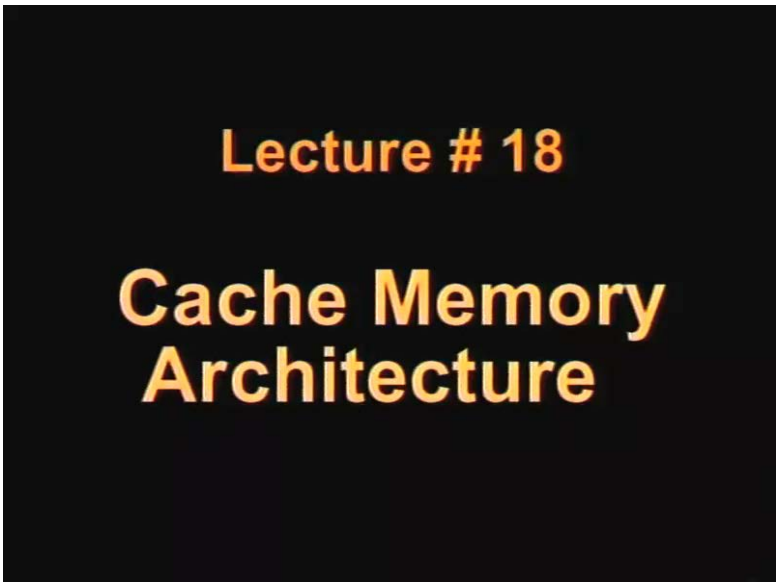
See once the process is running, it is competing with a number of other processes and suppose the replacement policy that you employ is global replacement. With respect to same process it may be most recent but in the global scenario, it may be least recent. So even in LRU technique also that may have, if I employ global replacement policy. So that will reduce the performance of the system, bring down the performance of the system to a great extent and that will not happen if I maintain the logical structure of the program. So if I simply employ segmented memory management and go for virtual technique that means whenever I bring in a new segment, the segment is brought in as a whole that means the entire for loop will be there in the main memory simultaneously. But what happens if I go for paged segmentation? So there one restriction I can put is that whenever a page office segment is needed, I will try to bring in all the pages belonging to same segment. So in that case may be that instead of replacing one page, I may have to replace multiple number of pages.

So if I ensure that all the pages belonging to the same segment should reside in the main memory at a time, then that kind of situation will not occur, that all instructions of a single for loop will reside in the main memory simultaneously. May be they are not available in contiguous locations but all of them are available. So the number of page faults can be reduced. No, what I am saying is whenever… but number of page faults will be less. If I don't do it, what will happen is number of page faults will be more but now the number of page faults will be less and because of less number of page faults, the swapping in and swapping out operation will also be less. When I am bringing in a new page, when I have a page fault? So if I can ensure that the entire structure is brought in simultaneously then while that structure is being executed, I will not have any page fault.

The page fault will occur only when you want to execute a new structure, you want go to a new structure which is not there in the main memory. Then only you will have page fault but because you are ensuring that entire module should be there in the main memory because the number of page faults will be less. and because the number of page faults is less, the number of swapping in swapping out operation will also be less. This is for a particular process. So with this, we close our memory management technique. Next we will try to see what is the physical memory organization.

(Refer Slide Time: 00:35:50 min)

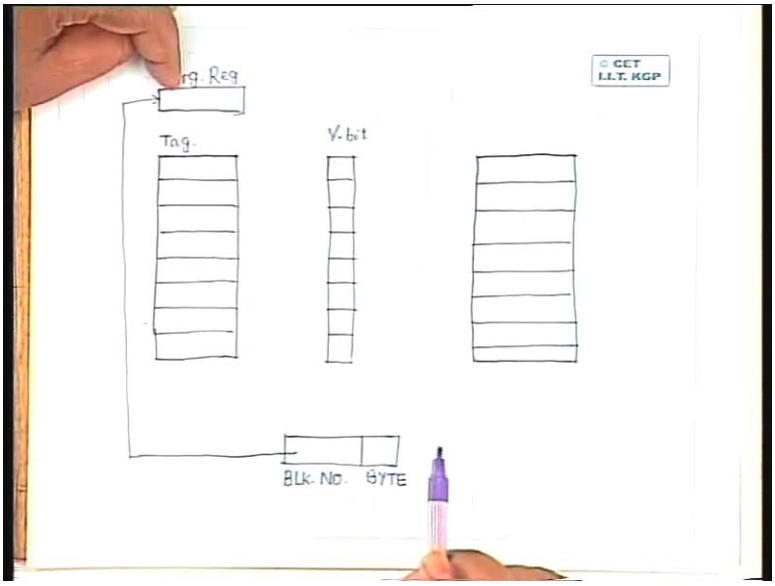
(Refer Slide Time: 00:35:52 min)



So today we will discuss about the cache memory organization. Till the last class we have seen that main memory is organized either in the form of pages or in the form of segments or if you want to take advantage of both, the segmentation as well as paging in that case the main memory organization that is used is paged segmented memory organization. And we have seen that what are the advantages of different schemes in case of paged memory management or segmented memory management or in case of paged segmented memory management. However if you think in depth, you will find that whichever memory architecture is used for accessing any particular memory location that means whenever the CPU generates the address of any memory location, firstly using that address you have to check either the segment table or the page table or the segment table and page table both in case of paged segmented memory organization before you actually reach the physical address in the main memory.

So naturally the question comes that if we store the page table or the segment table or both in the main memory itself in that case for every access to the main memory, we have to have one or more additional memory access before we get the actual physical address of the memory location, the data or the instruction from which is actually required by the CPU. So effectively the memory access in such cases will be very slow. So we have to think of an alternative where we can have faster access to memory and that is what is given by the cache memory. In case of cache memory, you get advantage in two ways. Firstly, the cache memory is implemented using the static RAM unlike the main memory which is implemented with the help of dynamic RAM.

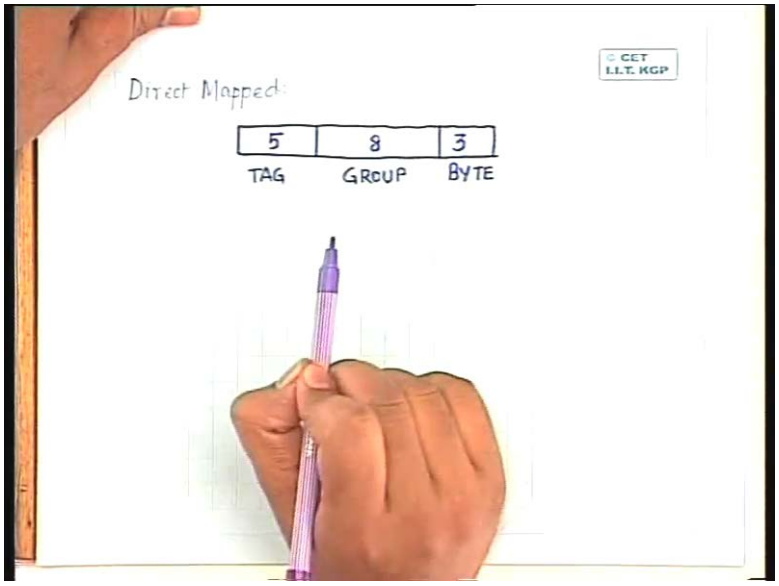
(Refer Slide Time: 00:37:56 min)



For every such cache line, I have to have a tag memory field. For every tag memory field or for every cache line, I have a corresponding valid invalid bit. So this is valid bit, this is tag field. Suppose the CPU generates an address and as I said that address will be broken into two components block number and byte identification within the block. Now in this architecture we have a register which is called an argument register. The block number which is obtained from the CPU generated address is placed in the argument register.

Then from the argument register, the argument register is hardware wise connected to each of these tag locations where they are compared parallelly. So the content of the argument register will be compared with the first tag location, it will be compared with the second tag location, it will be compared with the third tag location and so on and this comparison will be done in parallel. As I said that the main memory address which is generated by the CPU is divided into two fields in case of associative cache that is block number and byte within the block.

(Refer Slide Time: 00:40:16 min)



In case of direct mapped cache, the main memory address is divided into 3 fields. Let us take the same architecture that is 16 bit address lines, I will have 3 bits, the least significant 3 bits which will identify the byte within a given block. So for this I use 3 bits, the remaining 13 bits are used as block identification number in case of associative memory. In case of direct mapped cache, what we do is this remaining 13 bits is again divided into 2 fields. One field is used as group identification and the other field is used as tag identification. So let us assume that out of this 13 bits, say 8 bits are used as group identification and say 5 bits are used as tag identification. So unlike in the previous case where every block was identified by a 13 bit number which was the block number, now every block is having a 2 component address, its tag number and group number.