So, now before proceeding further we will we will now look at page replacement algorithms, but just before that we will take a look back will recapitulate with 2 small examples into the performance factor of paging and caching and together how CPU instruction time is determined we will just take a 2 small examples.

So, CPU time taken by taken per instruction; let us say CPU time is given by the CPU execution clock cycles. So, the number of times the CPU is doing work and the number of time slots in which the CPU is doing work (number of cycles in which the CPU is doing work + the memory stall clock cycles × clock cycle time). So, number of clock cycles in which the CPU is working plus the number of memory stall cycles in which it is waiting for data or instructions. So, the CPU time for a program will be given by the number of clock cycles in which the program code is getting executed and the along with the number of clock cycles for which the code is a program is waiting for instructions or data multiplied by the clock cycle time.

And now the memory stall cycles; again the memory stall cycles again will be given by memory accesses into miss rate into miss penalty; what will be the memory stall cycles number of memory accesses for instructions or data among these accesses; what is the rate at which I miss the miss rate; what is the amount of which let us say in the cache how many are missed in the memory how many I miss so, that I have to go to the lower level of the memory at a particular level of memory what is the miss rate and if I have a miss what is the amount of penalty that I have to incur and with this we will take an example suppose a processor executes at a clock rate of 200 megahertz. So, my cycle time is 5 nanoseconds, fine.

And I have a base CPI of 1.1. So, if everything goes well and if everything goes well, I have a hit in the in the memory and I am executing everything fine, my pipeline is working I have no stalls then I have a CPI cycles per instruction is 1.1 and in this computer in the in this program that we are considering 50 percent are arithmetic logic instructions in this program, 30 percent are load store; that means, data access and 20 percent are control instruction.

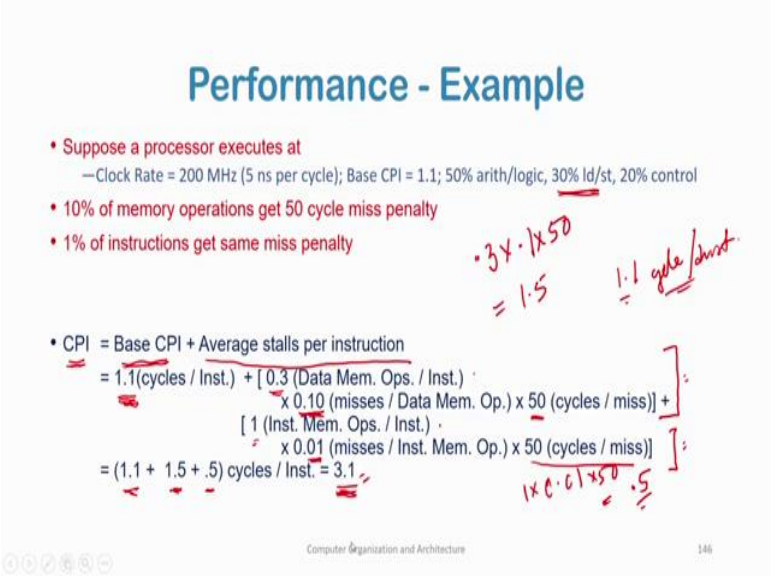
So, therefore, 70 percent instructions do not require do not require memory do not require memory access, memory access for data do not require memory access for data only for instructions all instructions everybody required to access memory for instructions of which some may be missed. So, I don't have the instruction at a certain level of memory, I have to go to the next level of memory to get the instruction and for the 30 percent load store instructions,

I have to access memory for data for all instructions, I have to access memory for the instruction itself and for 30 percent of the instructions, I have to access memory for data.

Now, let us say suppose I have a 10 percent memory operations get 50 cycle miss penalty. So, 10 percent of memory operations gets 50 cycles of miss penalty. So, the miss rate is 10 percent miss rate is 10 percent and miss penalty equals to 50 cycles ok.

Next one percent of instructions get the same miss penalty. So, this is the data miss rate and another is the instruction miss rate and that is equals to one percent and miss penalty is same penalty is same 50 cycles. So, both for instructions and data the penalty is 50 cycles, but the instruction miss rate is 1 percent the data miss rate at a given level of memory is 10 percent.

(Refer Slide Time: 44:03)



So, therefore, how do I get the average CPI, average cycles required per instruction is the Base CPI (base number of cycles required per instruction) + Average number of stalls per instruction how do I get that?

So, we knew that the base cycle base CPI is 1.1. So, if there are hits at the inns memory, if there is a hit in the memory then if there is always a hit in the memory then I require 1.1 cycle per instruction ok. So, this is what it said.

In addition I have stalls. So, what is the average stalls per instruction? That is we said that 30 percent or 0.3; 30 percent are data memory operations ok. So, 30 percent are load store; that means, data accesses are 30 percent out of this 30 percent, there is the miss rate of 10 percent

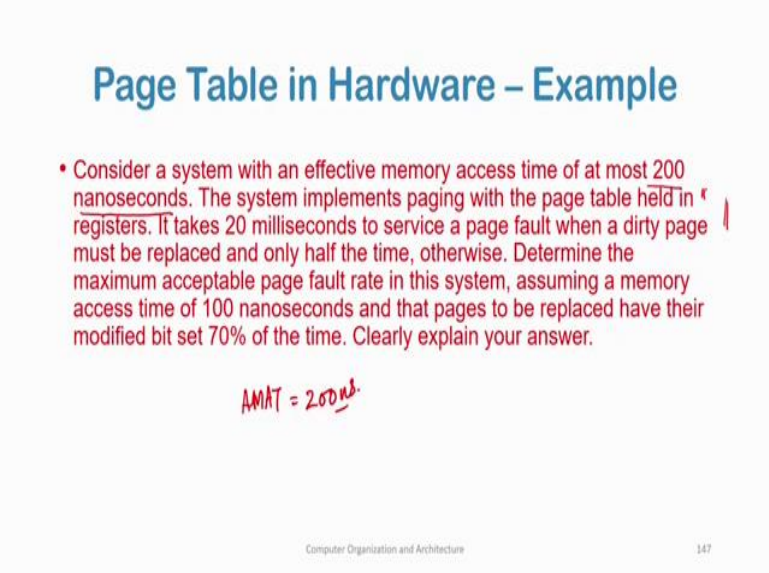
and each of them will incur. So, this out of this 30 percent I have a miss rate of 10 percent. So, 30 percent are load store, but out of this 30 percent; that means, 90 percent of the times, I get my data that I require at a certain level of memory let us say main memory we are considering now. And or cache memory let us say we are accessing now and otherwise I have to go in 10 percent of the times; however, when I am accessing data memory, I have to go to the next level of memory that is main memory and when I go to the main memory I incur 50 cycles ok. So, the overhead the effective overhead for going into of going accessing memory for data is this. So, 30 percent data instructions out of which 10 percent misses and each of that incurs 50 cycles for instructions all instructions. So, 100 percent all instructions must access the memory out of that 1 percent misses the memory and then when I have a miss in the memory I incur 50 cycles I incur 50 cycles.

So, what is the total CPI; what is the total CPI it is the addition over the base CPI plus the memory access overhead due to data, and data misses memory access overhead due to data misses plus memory access overhead due to instruction misses. So, if I calculate, how did I get this 1.5? So, I got 0.3 into 0.1 into 50 which is equals to 1.5 similarly for this case I have 1 into 0.01 into 50 which is equals to 0.5 ok. So, I got this in this means and therefore, the CPI is 3.1.

So, we can understand that although the base CPI that that everything is a hit if everything is a hit, then I require only 1.1 cycles per instruction, but due to the overheads caused by the misses in the memory in the cache misses in the cache, and because I have to go to the memory, sometimes 10 percent of the times in case of data and 1 percent of the times in case of instructions the effective cycle time for each instruction the effective amount of time required to execute each instruction becomes 3.1 cycles ok.

So, this was the amount of performance degradation.

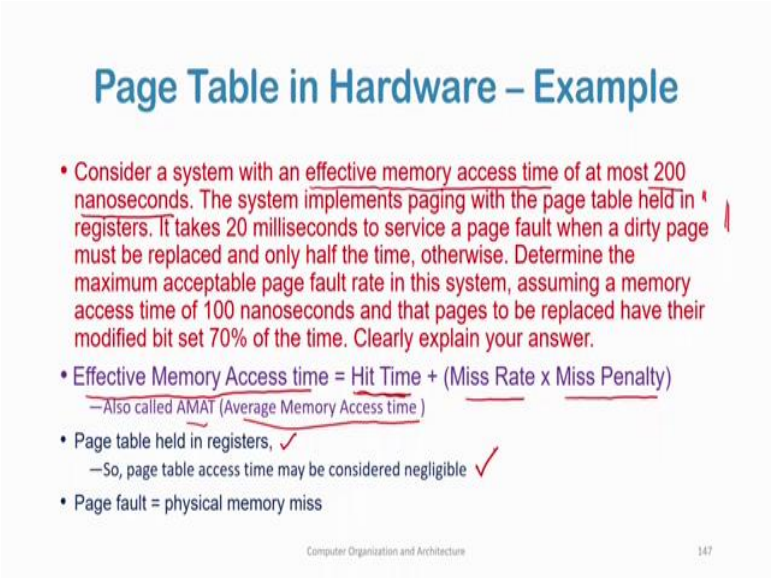
(Refer Slide Time: 48:04)



We will take another example before going into page replacement. So, consider a system with an effective memory access time of at most 200 nanoseconds. So, effective memory access time or average memory access time is equal to 200 nanoseconds the system implements paging with a page table held in registers. So, I have a hardware page table this means that I have a hardware page table and this means that I don't require to I don't have any overhead for accessing the page table, it is negligible the overhead for accessing the page table is negligible, because the page table is held in hardware registers.

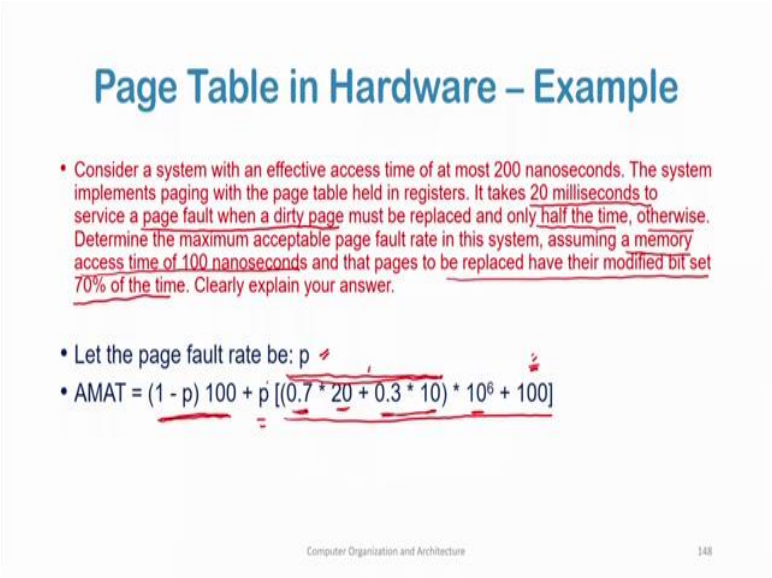
It takes 20 milliseconds to service a page fault when a dirty page must be replaced and only half the time otherwise. Determine the maximum acceptable page fault rate in the system assuming a memory access time of 100 nanoseconds and that pages to be replaced have their modified bit set 70 percent of the time.

(Refer Slide Time: 49:17)



So, what is effective memory access time? We said here effective memory access time. So, effective memory access time is the hit time. So, memory hit time + memory miss rate × memory miss penalty this is also called the AMAT or the average memory access time as we just told we said that the page table is held in registers. So, page table access time may be considered to be negligible and what do we mean by a page fault, it is a physical memory miss.

(Refer Slide Time: 49:52)



Now, how do I determine this effective memory access time? Now this effective memory access time will be given as this. So, let the page fault rate be p; that means, the rate at which

I miss the main memory no access I try to access the main memory and then the page is not there in the main memory this happens at the rate of p.

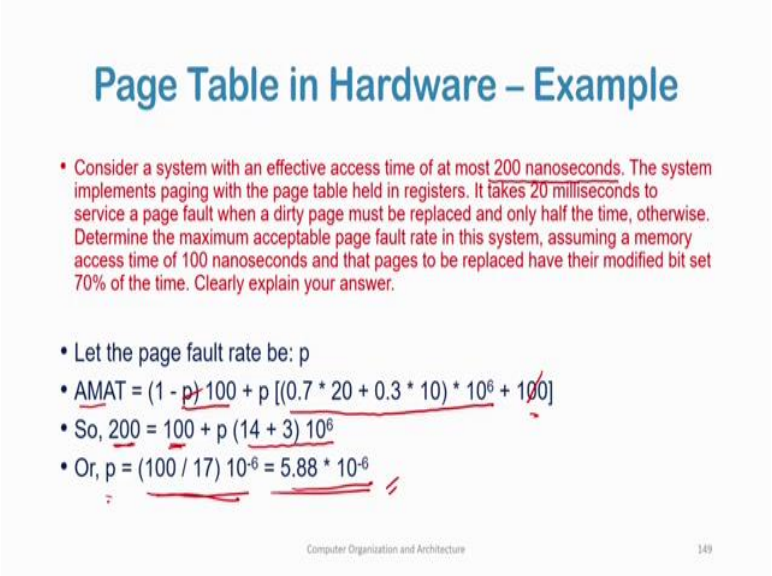
So, $1 - p$ is the hit time is the memory hit time and we said that the access time mem that we have a memory access time of 100 nanoseconds. So, for accessing the memory main memory, I require 100 nanoseconds, if I have a hit in the memory, I require 100 nanoseconds + miss rate × miss penalty. So, miss rate is p because the page fault rate is p and this is the miss penalty what do we have in this miss penalty part. So, in this miss penalty part I have to do what I have to bring this page from secondary memory into the main memory.

Now, in doing so, we said that it takes 20 milliseconds to service a page fault when the page is dirty and the page can replaced in half the time that is within 10 milliseconds otherwise. So, if the page is not dirty, then what then during page replacement I can just I can just find out the page to be replaced and that if that page is not dirty I will just discard this page I don't have to write the page back to the secondary memory. And then bring my required page frame I don't need to do this, I don't need to bring in my required page only after writing my previous page I don't need to do that why because the page is not dirty and if that is so, it takes only 10 milliseconds; however, if the page is dirty. That means, I have to write back the page, I find the page which must be replaced, I write back the page into physical memory and in the page frame for corresponding to this page I bring the new page in.

That takes 20 milliseconds and we said that that pages to be replaced have the modified bit; that means, the dirty bit set 70 percent of the time when I have a page fault 70 percent of the time I require 20 milliseconds and the remaining 30 percent of the time, I only require 10 milliseconds because in the this case the page is not dirty.

So, the effective time that is required to address to require to service the fault is this. So, to bring back whichever page I need back into memory, I need this and then and then what happens then, I need to again I will have a page hit, I have to access the memory again to get my data ok. So, I have a page fault I will go to the secondary memory bring my page and then access the memory again this time, I will have a page hit. So, I will only incur 100 nanosecond and get my data and here these are in milliseconds. So, I need to multiply this with 10^6 to convert it to nanoseconds.

(Refer Slide Time: 53:27)

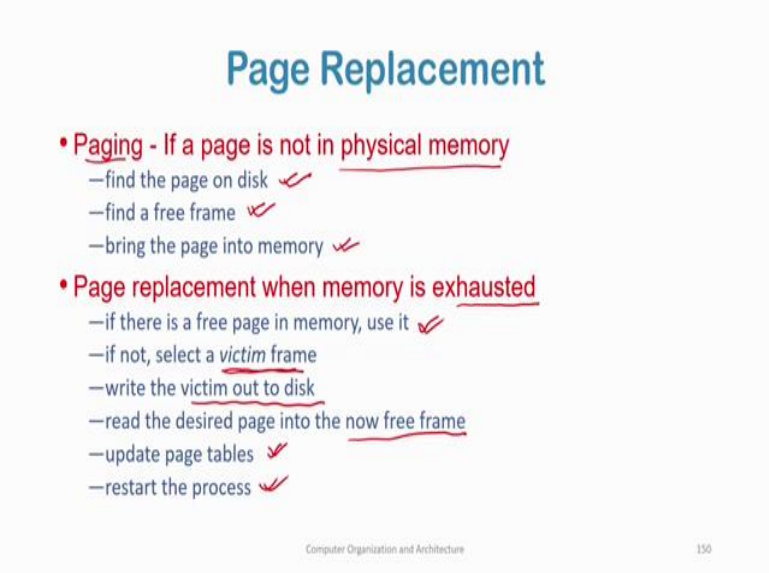


So, therefore, I we have first said that my affective memory access time is 200 nanoseconds. So, AMAT or average memory access time is 200 nanoseconds. So, $200 = 100 + p\,(\frac{7}{10} \times 20)$ which is which is basically 14. So, 14 + 3; so, 17×10^6; so, this part gives 17×10^6. So, $17 \times 10^6 + 100$; so, this way if you break up, if you break this up, this becomes $100 - 100 - p \times 100$ and this one again has a $p \times 100$. So, this one and this one cuts off and therefore, I have what do I have I have $200 = 100 + p \times 17 \times 10^6$. So, therefore, p is given by $(100/17) \times 10^{-6}$ or 5.88×10^{-6}.

So, therefore, what this means is that if the page fault rate is at most 5.88×10^{-6}. So, I have one page fault every 5.88×10^{-6}accesses if the page fault rate is as low as this. So, this means that if the page fault rate is at most 5.88×10^{-6}, then I will have a effective memory at most time which is at most 200.

So, the memory access time effective memory access time incurring page faults together if the effective memory access time will be less than or equal to 200, 200 nanoseconds if the page fault rate is as low as this.

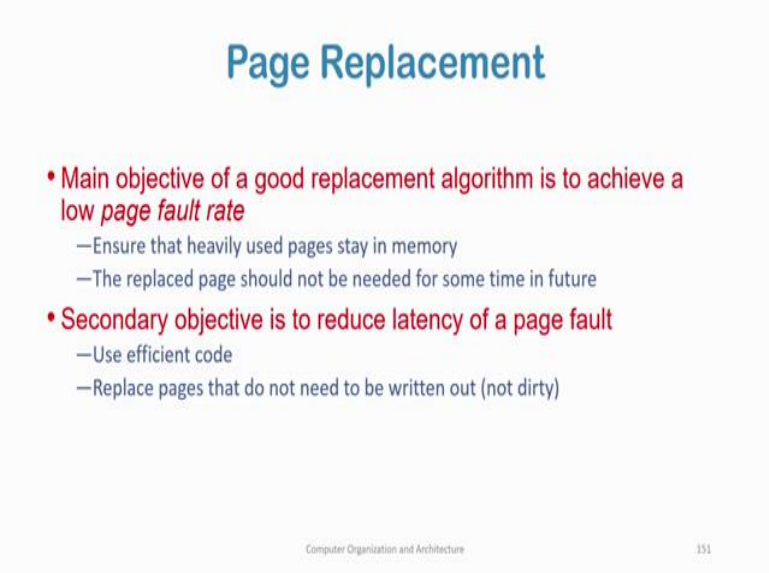
(Refer Slide Time: 55:29)



So, now we move to the next topic of discussion which is place replace page replacement. So, just to brush up, we discussed about the page replacement, but we will now discuss about algorithms just to brush up as to why we require page replacement. So, in paging if a page is not in physical memory, it has to be replaced though if in paging if the page is not in physical memory we have to find the page on the disk, find a free frame in the physical memory and then bring the page into memory and put it in that free frame.

So, I need page replacement when memory is exhausted if there is free page in memory then use it. So, during replacement if I have a I don't need any replacement I just need to bring the page into a free page, if a free page in memory exists free page frame exists, there is no need of replacement if not, I have to select a victim frame and herein comes replacement, write the victim out to the disk read the desired page into the now free frame update page tables and restart the process. So, I had a page fault, I service the page fault and restarted the process just as in the previous example, the question is what to which victim to replace and how to replace what is the replacement mechanism and whom to replace.

(Refer Slide Time: 56:55)



So, the main objective of a good replacement algorithm is to achieve a low page fault rate. So, I need to achieve a low page fault rate. So, page fault should be very low as I as we saw that if the page fault rates are not very low, the effective memory access times becomes will become very high. So, in that we could only in the previous example; we could only we could only control the page fault rate within 200 nanoseconds because the page fault rate was as low as 5.88×10^{-6}.

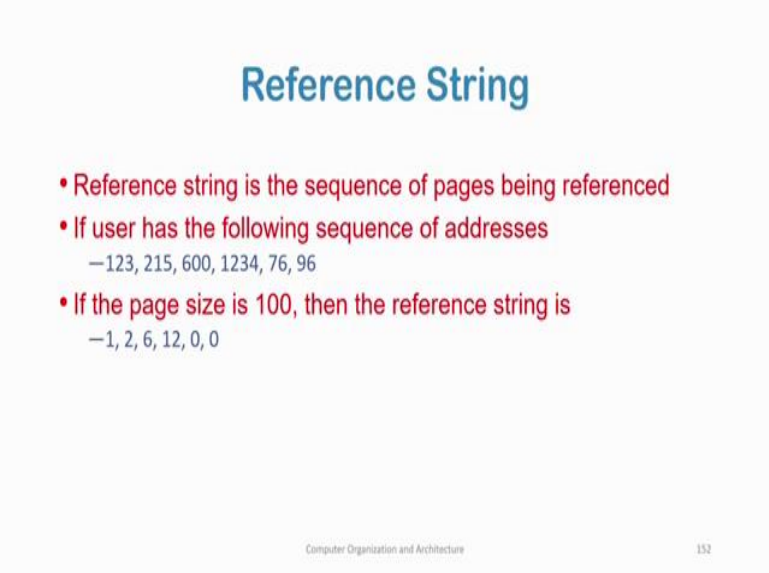
Now, a low page fault rate ensures that heavily used pages stay in memory how can we ensure a low page fault rate by ensuring that heavily used pages stay in memory I do not evict heavily used pages out of main memory and the why is this so, because of the locality of reference again because if a page is being heavily used it is very probable that this page will be used again in future.

The replaced page should not be needed for some time the replaced page whom I replace it is more difficult to determine this. The replaced page should not be needed for some time in future this is also based on the locality of reference meaning that if there is a page which is being rarely used in the recent past which have rarely been used in the recent past it is likely not to be used again in the recent future as well.

The secondary objective of page replacement is to reduce latency of a page fault. So, how do I reduce latency of a page fault by using efficient code for handling a page fault and by replacing pages that do not need to be written out? So, I will try to write a page which whose dirty bit is

not on if that is so, then I can just discard this page and the page fault can be quickly handled as we saw in the previous example although in there, it was not to exact numbers, but when it was it was not dirty when the when the page was not dirty I could I only had an overhead of 10 milliseconds; however, if the page was dirty then the replacement took an overhead of 20 milliseconds.

(Refer Slide Time: 59:28)

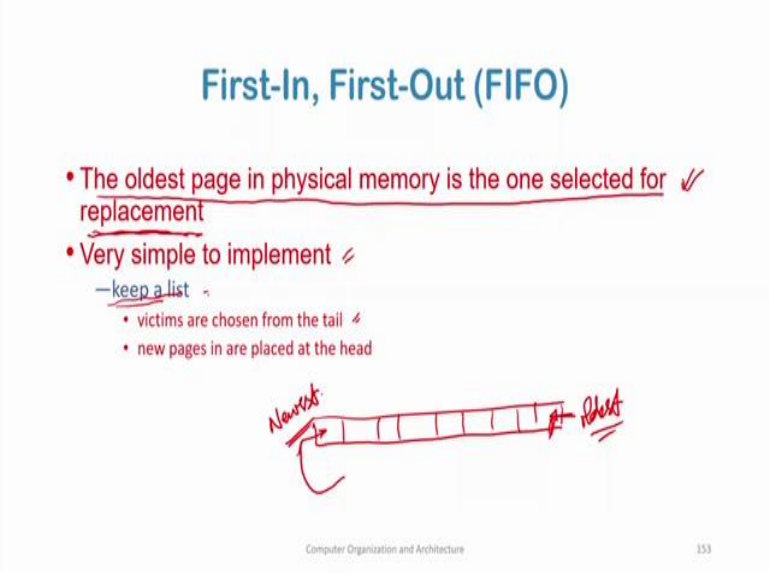


So, before going into specific algorithms, we will we will understand, what is called a reference string. A reference string is a sequence of pages that are being referenced. So, to understand the algorithms we will need to understand the algorithms based on a sequence of page accesses. And this sequence of page accesses of by a certain program say will be will be called its reference string.

So, for example, if user has the following sequence of if a user has the following sequence of memory accesses. So, he accesses memory 123, 215, 600, 1234, 76, 96; these are the memory references that he has done and if the page size is 100, then the reference string is 1, 2, 6, 12, 0, 0; that means, for 123; that means, if the page size is 100. So, what is the page number of this page address? it is modulo 100. So, basically it is it is not modulo 100, sorry, it is the it is page number 1 because 0 to 100 is in page number 0, 101 to 200 will be in page number 1. Similarly 201 to 300 will be in page number 2. So, therefore, 215 will be in page number 2, 600 will be in page number 6.

So, so, basically I did a small mistake nothing 0 to 99 will be in page number 1, 100 to 199 will be in page number sorry 0 to 99 will be in page number 0, 100 to 199 will be in page number 1, 200 to 299 will be in page number 2 likewise. So, that if we calculate in that way 600 comes in to page number 6. So, 1234 which means; it will come in to page number 12, 76 will come in to page number 0 and 96 will come in to page number 0.

(Refer Slide Time: 61:28)



So, now the first page replacement algorithm this is called the first in first out replacement algorithm this is the oldest page this what does this scheme say the oldest page in physical memory is the one selected for replacement. So, the oldest page in physical memory is the one selected for replacement very simple scheme whichever has first come into the physical memory should be the first one to go out of the physical memory as well.

So, I will choose that page for replacement which came which was brought into the memory physical memory the earliest among all physical pages that we have in memory whoever was brought in the earliest the oldest one will be replaced. It is very similar to implement simple to implement what do we do we keep a list of victims and chosen from the tail. So, we keep a list we keep a list and this list is the references and this one is the first one that that had come this is the newest one. So, because this is the oldest I will choose this one to be replaced.

So, if this is the oldest this is the newest then from the tail of the list I will choose the oldest and replace it, new pages are placed at the head new pages are placed at the head.