

Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 56
Virtual Memory (Contd.)

(Refer Slide Time: 00:31)

The slide is titled "Working Sets and Page Fault Rates". It contains three bullet points:

- n Direct relationship between working set of a process and its page-fault rate
- n Working set changes over time
- n Peaks and valleys over time

Below the text is a graph with "page fault rate" on the vertical axis and "time" on the horizontal axis. The vertical axis has a mark for "1" and the origin is "0". The graph shows a line that starts at zero, rises to a peak, falls to a low level, rises to a higher peak, falls to a very low level, and then rises to another peak. A horizontal bracket labeled "working set" spans the duration of the second, highest peak. The slide also features a small video inset of a man in a purple checkered shirt in the bottom right corner and logos of IIT Kharagpur at the bottom left.

So in our last class we were looking into this strategies for improving this page fault rate, how to handle this page fault rate reduction. And, we have seen that there are two approaches; one is based on working set model which is for global page replacement, and another is for based on Page Fault Frequency or PFF so which works for the local page replacement policies.

And we have seen that there are direct relationship between this working set of a process and it is page fault rate. As long as the working set is not loaded in the memory page fault rate will be high. And, once the working set has been loaded then the page fault rate will become low. But, at the same time working set is a dynamic concept, like as it changes what times as the program is executing. So, it will refer to different regions of the program. As a result this number of pages or the set of pages that it refers to will change over time.

So there will be relationship between this working set and this page fault rate. So, in this diagram as we have discussed so, page fault rate if we plot if we normalize it in the range

of 0 to 1, then over the time this page fault rate increases for initially very few pages have been loaded. So this page fault rate increases. Then, when it reaches a peak after that it starts decreasing page fault rate starts decreasing means by this time, the working set has been loaded in the memory. For this region for this part of the execution time of the process, then again from this point again the page fault rate has started increasing significantly. So, in between there are some small changes in this page fault rate, but they are not that much significant, from this point again it has started increasing significantly.

So, perhaps we have come from one locality of a process to another locality of the process, so this marks the changeover ok. So, when this change over occurs again the local pages for that region has to be loaded. So, this will be increasing at this rate ok. So, this page fault rate increases. And, again when it reaches when all the pages of the working set at current working set have been loaded, then this page fault rate will start decreasing. So, that way again so, it go so, some small variations, till it comes to a point where this program has towards to another region of code where the previous locality is no more valid.

Now, this program is referring to a newer set of pages. So, then this page fault rate starts increasing again. And, there is a relationship because the each of these peaks. So, that will constituted that will come into one working set. So, whenever so, after if you plot this over time, then you can understand that as and when you have got these peaks. So, that actually talks about the change in the locality of reference. So, that way we can have and whenever you have got valley; that means, this local pages have already been loaded. So, they are in the same working set. So, you can mark this working set as a regions of the process by looking into this page fault rate.

And, this is particularly useful like, if the program is run repetitively like maybe we have run the program once and then we have found out this working set behavior. Next time the program is run, maybe we know we have got some idea about what is going to be the working set of the process over time and that way we can take appropriate action. So, that it only refers to the pages are already loaded. So, the reference comes to only those pages and the page fault rate is less.

(Refer Slide Time: 04:10)

Other Considerations

- Prepaging
- Page size
- TLB reach
- Inverted page tables
- Program structure
- I/O interlock and page locking

Now, other considerations for this demand paging and say pre paging, then the size of the page, then the TLB reach, inverted page table, program structure and this I O interlock and page locking. So, these are the other issues that we will discuss in our next few slides.

(Refer Slide Time: 04:29)

Other Considerations -- Prepaging

- Prepaging used to reduce the large number of page faults that occurs at process startup
- Prepage all or some of the pages a process will need, before they are referenced
- But if prepagged pages are unused, I/O and memory was wasted
- Assume that:
 - s pages are prepagged and
 - A fraction α of these s pages is actually used (α between 0 and 1)
 - Question -- is the cost of $s * \alpha$ saved pages faults greater or less than the cost of prepaging $s * (1 - \alpha)$ unnecessary pages?
 - If α near zero \Rightarrow prepaging loses
 - If α near one \Rightarrow prepaging wins

So, pre paging so, pre paging is used to reduce the large number of page fault that occurs at process startup. So, pre page all or some of the pages of a process will need before they are referenced. So, this is this was exactly what I was talking about so, if you have

some idea about the history of a process. So, if the process for example: this payroll program; so, payroll program this is done every month in an organization. So, naturally there are the it is reference behavior is known to the system, known to the system administrator.

So, it may decide that these are the pages that this payroll program will need at the beginning. So, if you look into this graph then you see that initially all the processes since number of frames are located is very low. So, it will show a sharp increase in the page fault rate. So, to avoid this so, if I initially if load the all the pages of the working set at this time itself, then naturally this peak will not occur. So, this region will come as a valley only that peak can be reduced.

So, that is what is said here. So, if you so, that is the pre-paging. So, if you know the pages that the process will need at startup then you can load those pages into the memory. So, pre page all or some of the pages a process will need before they are referenced. So, this is the concept of pre paging, but if pre paged pages are unused I O and memory was wasted. Now, it may so, happened that the program the last if the last time the program was run, it was with some data set. And, the next time the program is being done it is all for a different data sets, and the program execution sequence for the 2 sets of data 2 data sets are different.

So, as a result the pages which are relevant for the previous run may not be relevant for this run. So, this can also happen. And, in that case we have got wastage of this pre paging structure. So, the pages are the pages that have been loaded are unused and this I O operation and I O is as you know that I O is very significant, because it has to access the secondary storage and secondary storage access is a very slow. So, this I O and memory that was wasted; so, suppose there are s pages are pre paged and a fraction α of this s pages is actually used. So, α is between 0 and 1.

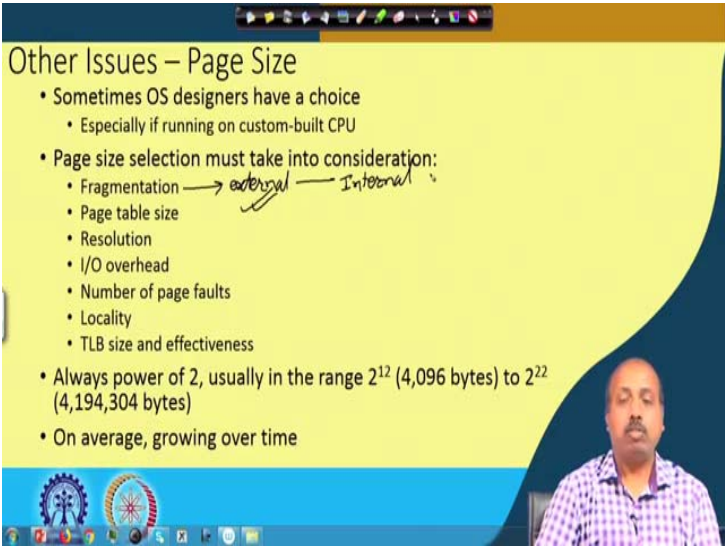
So, is the cost s into α saved pages saved pages faults greater or less than the cost of pre paging s into $1 - \alpha$ unnecessary pages. So, this if I load this s into α . So, this so, these many so, sorry so, this s into s into α saved pages. So, for them what is the fault? And, it is greater or if it is greater than this one ok. Then naturally we do not have we have got some saving, but if it is less then it is not. So, if α is near zero. So,

pre paging will be pre paging will lose and alpha is near one, then this preparing will win.

So, if alpha closed to zero; that means this cost was less than this one. So, unnecessarily we have wasted lot of time in pre paging those pages. And, the alpha is close to one, then we have gain because we have loaded the pages and they all are almost all of them are being used there. So, there is very little wastage.

So, this depends on the behavior of the program as I said that the execution sequence of the program maybe be different and then depending upon this outcome of alpha. So, you may gain in some cases you may lose in some cases. So, they are to be seen and if you can predict properly the value of alpha, then we can go for this pre paging type of policy.

(Refer Slide Time: 08:31)



The slide is titled "Other Issues – Page Size" and lists several factors to consider. A handwritten note next to "Fragmentation" points to "external" and "internal". The slide also includes a video inset of a man speaking in the bottom right corner.

- Sometimes OS designers have a choice
 - Especially if running on custom-built CPU
- Page size selection must take into consideration:
 - Fragmentation → external → internal
 - Page table size
 - Resolution
 - I/O overhead
 - Number of page faults
 - Locality
 - TLB size and effectiveness
- Always power of 2, usually in the range 2^{12} (4,096 bytes) to 2^{22} (4,194,304 bytes)
- On average, growing over time

Next important issue is the page size. What can be the page size? Sometimes the OS designers have a choice, especially if running on custom built CPU. So, many times as we have as you can see that the un organization who have his designing the CPU. So, they are in consultation with the waste development team. And, the waste development team can have a say on what should be the size of a page. Now, page size selection must take into consideration many things, first of all fragmentation. And, as you can as you remember that using this paging policy it can solve this external fragmentation.

So, this external fragmentation is solved, but what remains is the internal fragmentation. So, this is solved, but this internal fragmentation this problem remains. Now, if the page size is large then on an average. So, half of the last page will remain unutilized. We have discussed in a while discussing on partition memory management or this project memory management. So, this page size is guided by this fragmentation. So, if the page size is large then large amount of a space will be wasted by each of the processes in the last page. So, that way this fragmentation is a problem. Then the page table size.

So, to resolve this fragmentation problem so if I or this memory utilization problem, where you reduce the wastage corresponding to that. So, you may decide that we will use page size to be small. As a result half of the last page will be wasted, but that value will be small because the page size itself is a small, but that way it will increase that page table size. So, for every process we have got a page table size and the page table we has to be stored and if the page size is small, then the page table size will be large. So, there will be large number of entries in the page table. So, as a result this will be increasing the page table and again we will be wasting memory, because this page table has to be loaded into memory.

So, if there are large number of page table entries then this page table space will be required then the resolution. So, what is the size of each page? So, what is the number of bytes? So, you may be talking in terms of words memory words, but in terms of a bytes how many bytes it refers to? Then the I O overhead. So, for doing a block transfer how much time it takes. So, if you see that the block transfer takes lot of time, then it is advisable that we have to do larger sized page. So, that we can transfer mode amount of data in one I O access.

So, that is the I O overhead may be an important issue. So, I O overhead is low. So, maybe we may go for small page size, but if the I O overhead is high, then we do not have choice we have to go for large page size. So, that we do not do that I O very frequently, because if the page size is small then the number of page faults will also be low or also be high. Because, there will be more less amount of code available for the processes in the memory. So, it will give rise to more number of page faults and then more number of I O transfers will take place.

Then number of page faults as we have said that any system it will have a desired degree of a page fault. So, if the number of page faults is low then definitely I will have to have this page size to be large. Because, if the page size is large then more amount of a code or more amount of a program memory references they will be available in the memory. So, if and if number of pages are low. So, the page size is high number of pages are low.

So, in that case also it will be having enough memory references loaded in the memory for a process. So, if the number of page fault is desired value is low, then I have to increase the page size, then the locality. So, how big is the locality like, if you see if you talk in terms of bytes.

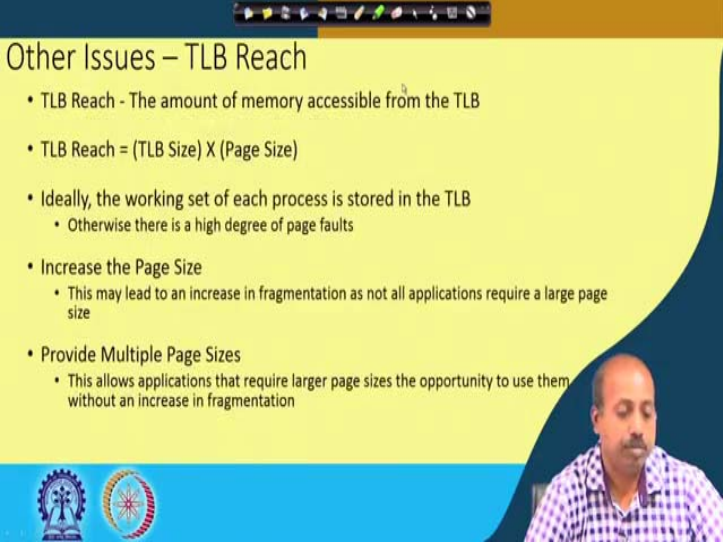
So, we have previously we have talked about this locality in terms of memory references the page references that it is doing. So, you now instead of that if you do in terms of absolute addresses then, what is the address range? So, how this address range reference changes over time. So, if you define that as the locality then I should have so, many locations in the memory. So, as a result that will determine the page size. Then the TLB size this translation look aside buffer so, this that is a case. So, I should have this TLB size if the TLB is large then I can have more number of more amount of memory pages loaded into the TLB. So, that is the one possibility.

So, all the if you are thinking in terms of loading the page table into TLB, then the number of TLB entries will determine like how many how be a page table you can have, because we want to load a significant amount of this page table into the TLB. So, if the TLB size is small then I have to make the number of page table entries less. So, as a result number of pages should be low. So, page size should be high. So, all these considerations will come.

And finally, this page size it should be a power of 2, because this page size this offset. So, they are derived from this address bits that are generated by CPU. So, we have to use some of the address bits so, as a result this page size should be power of 2. So, usually it is in the range of 2 to the power 12 to 2 to the power 22 bytes, to 2 to the power 12 is about 4 k. So, that is 4, 096 bytes and 2 to the power 12 be the huge number. So, naturally that will be making the page size to be very very large. So, it is almost like a sequential allocation.

So, on an average so, page size will be growing over time; so, that has happened like if you look into the history, the initial page size than that now the page size that are being used. So, that size is increasing, because more and more memory main memory space is becoming available and we can go for a larger page size.

(Refer Slide Time: 14:57)



The slide is titled "Other Issues – TLB Reach" and contains the following content:

- TLB Reach - The amount of memory accessible from the TLB
- $TLB\ Reach = (TLB\ Size) \times (Page\ Size)$
- Ideally, the working set of each process is stored in the TLB
 - Otherwise there is a high degree of page faults
- Increase the Page Size
 - This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes
 - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation

The slide also features a video inset of a man in a checkered shirt in the bottom right corner and logos of institutions at the bottom left.

So, next we will look into the TLB reach. So, TLB reach so, this is actually the amount of memory accessible from TLB. So, TLB reach equal to TLB size into page size. So, the size of TLB into page size so, that should be thus ideally the working set of each process is stored in the TLB, otherwise there is a high degree of page faults. So, if the page is available in the TLB then that is very good. So, if I can store it in the TLB then that is better. So, this TLB size into page size so, that actually tells how many pages. So, if I if I say that this, if I say that in the TLB itself I load the pages, then this page size for number of pages loaded in the TLB.

So, this so, many space so, so much space will be required for the TLB. So, the working set of each process is stored in the TLB, otherwise there will be a high degree of a page faults. So, if we increase the page size. So, this may lead to an increase in fragmentation as not all application requires the large page size. So, that is there. So, that were that problem we have already discussed. And, other possibilities that provide multiple page sizes. So, this so, for some applications they will require larger page size and they will

they will using they will be using this and some a processes, they may require smaller page size ok.

So, small size processes giving it larger page size is there be more of wastage. So, we can give it only a small page size and the process may change it is style like, once it starts maybe it is using small page size, then if you find that it is requiring large amount of memory, then will be changing over to larger page size for the process. So, multiple page size is may be available in the system and depending on the behavior of the process. So, it can go from one page size to another page size.

(Refer Slide Time: 16:52)

Other Issues – Program Structure

- Program structure
- `int[128,128] data;`
- Each row is stored in one page
- Program 1

```
for (j = 0; j < 128; j++)
  for (i = 0; i < 128; i++)
    data[i, j] = 0;
```

128 x 128 = 16,384 page faults

- Program 2

```
for (i = 0; i < 128; i++)
  for (j = 0; j < 128; j++)
    data[i, j] = 0;
```

128 page faults

Handwritten notes: (512) and a diagram showing a vertical stack of memory pages labeled `data[0]`, `data[1]`, ..., `data[127]`. The first page contains `data[0][0]` to `data[0][127]`, the second contains `data[1][0]` to `data[1][127]`, and so on.

Another very interesting phenomenon that occurs is the program structure. So, here we have an example suppose I have got an integer variable data whose which is an array of 128 by 128 entries. So, this is an integer array having 128 by 128 entries in it. And, each row is stored in one page. So, so, if I assume that each integer takes say 4 bytes so, it takes 4 bytes. So, one row is 128 into 4. So, that is 512 bytes. So, if a page size is 512, then one row will be stored in one page. So, these pages one will be storing say at the row one. So, data 1 then, after that page 2 will be storing this data 2 ok. So, that way it will it will go.

Now, if we see look into this program structure. So, it is it is a it is a loop so, it says that for j equal to 0 j less than 128 j plus plus, i equal to 0 i less than 128 i plus plus data i j equal to 0. So, what we are trying to do is that we want to initialize all the entries in the

in this data structure to 0. Now, when we were trying to do that the referencing pattern you see first it will be accessing 0 0 ok, then it will be accessing 1 0, then 2 0, this way it will go up to 127 0, that will be 1 iteration of the i loop. Then, this j value will become 1 and i value will again come to 0, then 1 1 1 2. So, it will go like this.

Now, given this referencing pattern; so if you look into the pages that it the program is referring to. So, first it is referring to this location 0 0, then it is referring to the location 1 0. So, 1 0 is in the next page. So, this page has got data 0 this page is having the entries data 0 0 to data 0 127, first page because it is a 1 row we are storing on 1 page. So, the first row first page is storing the first row, 0 0 to 0 127. The next page will be storing from data 1 0 to data 1 127, there will be a storing those values.

So, when we were referring to so, data 0 0 is in the first page, then data 1 0 is in the second page, data 2 0 will be the third page. So, that way every reference it will be generating 1 page faults. So, all the 127 references of this i loop, all the 128 references for the i loops. So, that will generate one page fault. Assuming that, I have got only one page loaded into the memory at a time. So, there initially no page is loaded. Now, we are allocating when the first reference comes it will generate a page fault.

So, a page 0 will be loaded after that it will be referring to page one. So, page 0 will be replaced and page one will be loaded so, again another page fault. So, this way it continues. So, this will generate 128 page fault for the first row itself and then there are for the j j loop is running for 0 to 127 so, another 128 iterations. So, total number of iterations is so, for which page fault will be generated is this 128 into 128. So, this many page faults will be generated.

(Refer Slide Time: 20:55)

Other Issues – Program Structure

- Program structure
 - `int [128, 128] data;`
 - Each row is stored in one page
 - Program 1

C → Row major
FORTRAN → Column major

```
for (j = 0; j < 128; j++)  
  for (i = 0; i < 128; i++)  
    data[i, j] = 0;
```

128 x 128 = 16,384 page faults

- Program 2

```
for (i = 0; i < 128; i++)  
  for (j = 0; j < 128; j++)  
    data[i, j] = 0;
```

128 page faults

So, on the other hand if we consider the second program so, here also I assume that my memory is having the data in this format. So, this is having 0 0 to 0 127 the first page ok. Then, the next page is having from 1 0 to 1 127 so, it is going like this. Then, in the second program what is happening it is first referring to data 0 0 that is this location, then it is referring to this j loop is varying. So, it is referring to 0 1 that is second location. Now, when this first reference is made so, assuming that the page was not present in the memory; so, there will be a page fault.

Now, when this page fault occurs this entire page it will be loaded from secondary storage into main memory. So, in successive 127 references there will be no more page faults. The next page fault will occur when this program will come to the next iteration of the i loop. When i value will become equal to 1 then i equal to 1 j equal 0. So, at 1 0 there will again be a page fault. So, this way the page fault will occur at this point. So, every page when the first reference is made so, it will generate a page fault and then rest of the differences so, they will capitalize on the page that has been loaded into memory. So, there will be no page fault then.

So, this way I can have all together 128 128 page faults in this program. So, you see just changing the 2 loop. So, if you look into the structure of this program. So, by changing the a loop order. So, a number of page faults can drastically change. So, and also it depends on the organization of the data, if you look into different programming

languages so, this if you look into programming language like C. So, they use this row major organization of this array a 2 dimensional array. So, row major organization means the array will be stored, 2 dimensional array will be stored row wise, as we are discussing here like row 0 row 1. So, like that.

On the other hand so, if you look into the language like Fortran, then they use a column major organization. So, they use a column major organization. So, then they are the data stored in a column wise fashion. So, it will be first storing this 0 0 in the then it will be storing the 0 so, then there is a 1 0, then 2 0 going up to 127 0, then it will be storing 0 1. So, it goes in this order. So, it go it stores the data in a column wise fashion. So, if it stores in a column wise fashion, then you can understand that this program in that case will give rise to large number of page faults.

Because, now it will be accessing this 0 0 from here, then it will be accessing 0, then it will be accessing 0 1 and 0 1 is in the next column; so, that in the page, then it will be accessing this 1 0 2 that is in the third page. So, like that. So, in that case this program will generate 128 into 128 page faults whereas, this first program will generate only 128 page faults. So, it also depends on the programming language the number of page faults that are generated, a program structure that is being used and that can change the number of page faults.

And, as we can understand that change in the number of page fault can affect system throughput significantly, because as and when a page fault occurs, then there will be page replacement, it will be accessing the secondary storage and all so, that will require lot of time. So, this program behavior this locality and all so, they have got a very good role to play in deciding these performance of this demand paging situation.

(Refer Slide Time: 24:56)

Other Issues – I/O interlock

- **I/O Interlock** – Pages must sometimes be locked into memory
- Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm
- **Pinning** of pages to lock into memory

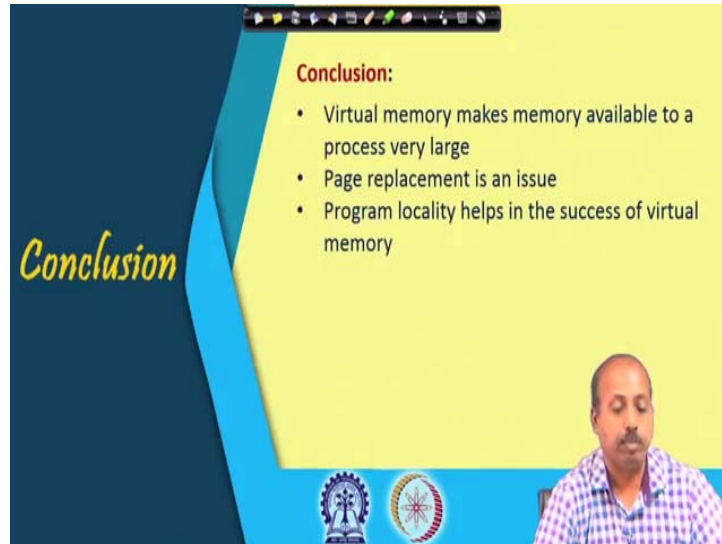
So, that is another very important issue. Other issue is something like I O interlock. So, pages must sometimes be locked into memory. So, why so, if I am consider some I O operation the pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm. So, suppose we have to do some data transfer? So, we are copying a file from a device ok. So, we are copying this file from this device into this buffer. Now, due to this page replacement algorithm and if I particularly assume that it is a global page replacement, some other process might have generated a page fault and it is trying to select the victim.

And, we may see that this among the different memory page replacement policies you can you understand that ok, we have got the policies like whichever page was not there for most of the times so, that can be moved. So, that way this page may become selected as a victim page. And, if this is selected as a victim page then, there will be difficulty because this I O transfer will get will get obstructed.

So, again the process for which this I O got initiated so, that will again generate a page fault in that case. So, we consider the case that. So, these pages they must be locked from being selected for eviction by a replacement algorithm. So, you have to pin the pages pinning of pages to lock into memory. So, you have to somehow air mark that this page should not be removed from memory for the time being. So, this is required. So, this I O

interlocking has to be done. So, that this page replacement policy does not select this page for replacement.

(Refer Slide Time: 26:40)



So, coming to the conclusion so, virtual memory makes memory available to a process very large. So, virtually infinite, because it is that directed by this logical addresses that the CPU can generate. And also so, it can even if physically I do not have that much of memory. So, I can make the, I can make this memory very large. And, then a large number of processes can be there and this individual process it can see the address space to be as large as the address generated by the memory by the CPU address lines.

So, page replacement is definitely an issue, because we have to do some allocation of frames to the processes. And, those pages are to be replaced as and when they are needed and this appropriate page is not available. Appropriate free frame is not available. So, that page we have to select a page and replace that from the frame, making the frame, free for loading the next page and this program locality it helps in selecting giving the success of the virtual memory. So, if the program locality is not there, then this entire discussion on virtual memory.

So, this is what place. Because, if you run if you write a program that makes random jumps from one place to another very frequently, then no locality will be there and the program will generate large number of page faults. So, whatever be the your whatever be your replacement policy. So, it will generate large number of page faults as a result

performance of the system will suffer. So, this entire discussion is based on the policy that ok, if I have got this locality of the processes. And that that is the behavior of the process is following this locality and that is why this virtual memory can be successful.

So, with this we end the discussion on this virtual memory environment, we will continue on this secondary storage access that is the disc particularly in the next class.