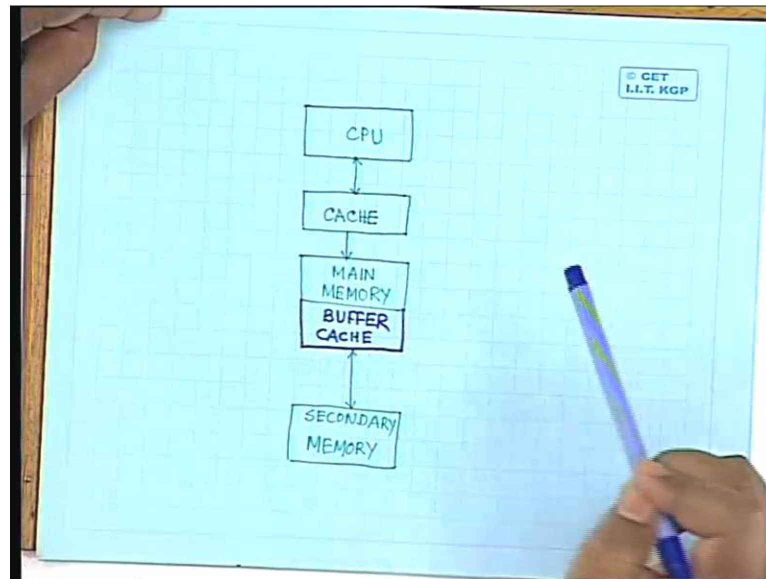
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture No. # 23
Buffer Cache

So, in the last class we have started our discussion on buffer cache and we have said that when you talk about the memory, memory is basically organized in a three level hierarchy.

(Refer Slide Time: 00:01:09 min)

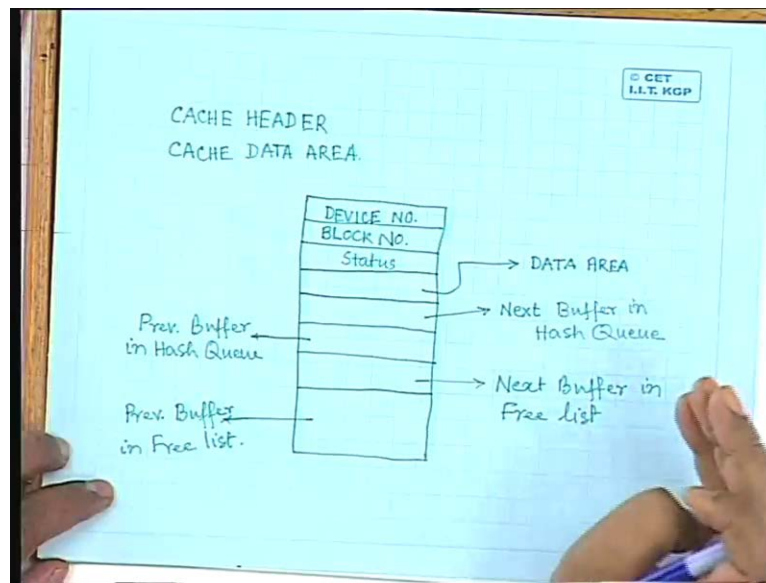


So at the lowest level in the hierarchy, we have the secondary memory which may be hard disk, which may be magnetic tape, which may be floppy drive and all those things. At the higher level we have the main memory and still higher level, we have the cache memory which has direct access to CPU. Now we have said that in between these two levels that is main memory and secondary memory, we introduced one more level which is called a buffer cache and this buffer cache is actually an interface between the main memory and secondary memory and the buffer cache is maintained in the main memory under direct control of the operating system.

So whenever we want to read any data from the secondary storage, we have already said that I cannot access an individual character or an individual byte from the secondary storage. Whenever I have to read any byte, I have to access the block containing that byte. This block may be say 256 bytes, block may be 512 bytes, 1 kilo byte and so on. So whenever the user process puts a request for any data or any instruction which is not available in the main memory or in the cache memory that has to be read from the secondary memory, that means the block containing that data or that instruction has to be read from the secondary memory and put to main memory subsequently to the cache memory.

So to improve the performance of the system, what is done is in between the main memory and secondary memory we introduce another layer that is buffer cache which is part of the main memory. So whenever a block is read, the block is read from the secondary memory and put into the buffer cache and it will remain in the buffer cache until you have a situation whether when this buffer cache content has to be replaced or overwritten by the content of another block from the secondary storage. So by this time it should be clear that I should not have or we cannot have duplicate copies of a secondary block, a block of the secondary memory into the buffer cache. There has to be only one copy of a particular block of the secondary memory into one of the buffer caches. Then we have said for proper management of the buffer cache, the buffer cache contains mainly two parts. One is called the buffer cache header and the other part is called the cache data area and it is this data area which will contain the content of a particular block.

(Refer Slide Time: 00:03:33 min)



So, naturally the data area of a cache must be of same size which is the size of a block on the secondary storage. So if the secondary storage is of 512 bytes block, every block on the secondary storage contains 512 bytes then the buffer, the data area in the buffer cache must be at least of 512 byte size, it cannot be less than that. It can be more than that but the remaining part will be wasted and we said that the cache header contains various fields. It contains a field for the device number because in an installation, we can have more than one file systems or more than one discs, even if it's a single disc but logically a single disc can be partitioned into more than one discs. So every such logical partition will have a device number. So buffer cache header will have one field which will contain the device number then the second field will contain the block number of that device. Then it has a status field, the status field contains various information, for example one of the information can be whether this buffer cache is currently being used by some process or not. So whenever any process makes use of some buffer cache then while that buffer cache is in use by that process, the buffer cache cannot be used by any other process until and unless it is released by the process who is using it.

So in the status field you maintained an information whether the buffer cache is free or the buffer cache is locked. If it is locked, no other process can access it. If it is free other processes can access it. So when a process starts using a particular buffer cache firstly, what has to be done is the buffer cache has to be locked then use it, then at the end you unlock the buffer cache and come out. Among other informations that can be stored in the status field is whether the content of the buffer cache is valid or it is invalid. There is one bit, you set that bit equal to 1 whenever a process uses it. See why we need the buffer cache. We need the buffer cache because from the secondary storage, if I want to get a new block I don't transfer the new block in single step to the user area. I first transfer it to the buffer cache then from the buffer cache it will be transferred to the user area and in the buffer cache, it will returned until and unless I face a situation that I have to bring in a new data block into the buffer cache and I don't find any other buffer cache which is free to contain this new data.

So only in that situation, one of the buffer caches will be overwritten by the new block content and whenever you overwrite, so that overwriting will be done in the data area because it is the data area which contains the content of a block. so whenever I bring in a new block content into the buffer cache, I have to put that into the data area and at the same time all this error information are to be modified. To increase the through put, to improve the performance of the system. See firstly whenever a base fault occurs, firstly the canal will try to find out whether the data which is being asked for is present in the buffer cache or not. If it is present in the buffer cache then from the buffer cache, the data can be transferred to the user area. So that being main memory to main memory transfer, the time taken is very small.

In case it is not there in the buffer cache then only you go to the secondary storage because in that case I don't have any other option. So by introducing this buffer cache in between the main memory, user area in the main memory and the secondary storage, the frequency of physical discs access is reduced so that improves the performance. And the locking means suppose the canal is making use of a buffer cache for getting the data from the secondary storage into that buffer cache. While it is doing that this status filed has to show the locked information because while the data transfer is taking place between a particular buffer cache and the secondary storage the buffer cache, that particular buffer cache is being used by the canal. So until and unless copying of data is complete and the canal releases the buffer, the buffer cannot be used by any other process. Not only that, it can so happen that two processes are trying to access, are trying to get the data from the same block on the secondary storage, so both the processes and that will happen in case of page fault.

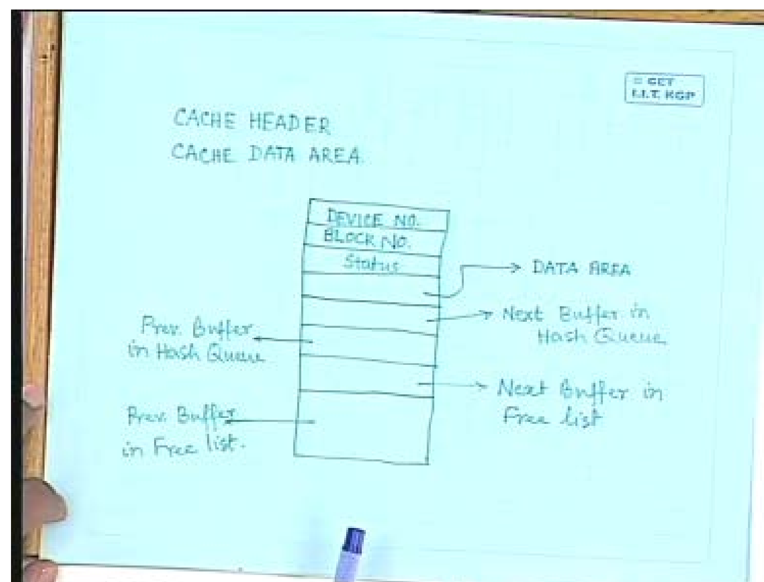
So the first process it finds that it encounters a page fault, it tries to get the data from the secondary storage, a particular block of data from the secondary storage. So how does it do it? It will invoke a system called for reading a block of secondary storage, following that system call the control will be taken over by the canal of the operating system. Now the operating system will try to find out in the buffer pool or buffer queue whether the block which is needed exists in any of the buffer cache or not. So for the first process it finds that the data exists in the buffer cache. So now the data has to be transferred from the buffer

cache to the user area because the user will be operating on it, so the data has to be transferred from the buffer cache to the user area.

Now while this transfer will take place, the status of the buffer cache has to indicate lock because until and unless for the first process, the data is transferred to the first processes user area, the same buffer cache cannot be used by the second process. Now once for the first process the data has been transferred then the status field has to be marked as free, it is no more locked because if it remains locked the second process will never get the access. So just after transferring the data to the user area of the first process, the canal will unlock this buffer so that now the request of the second process can also be entertained, may be in the user area the first process will take a lot of time to process that area but in the meantime the second process can also get access to these buffer and the data can be transferred from the same buffer to the user space of the second process.

Again while doing this transfer, the buffer has to be indicated locked so that if any other process tries to access the buffer, during the same time there should not be time overlapped. (Conversation between professor and student: Refer Slide Time: 10:38) Processing cannot be done on the buffer area, processing can be done on the user area. But before that user one's area must have been overwritten by some other buffer content. It is not that whenever user one finishes processing, it will lock the buffer because the data from the buffer is already transferred to user one's area. So until and unless it is requesting, user one is requesting for another buffer, this buffer will not be locked. Whichever buffer that one puts a request that buffer will be locked. So in the buffer pool I will have a number of buffers, different buffers will contain different data blocks. The only purpose of putting this lock is I should not allow more than one process to access the same buffer simultaneously.

(Refer Slide Time: 15:23)



For every buffer I will have such buffer header. If I have 1000 buffers in the system each, the data area of each of the buffer has to be same as the block size of the secondary storage. Every buffer will also have this header. So I will have 1000 headers, buffer headers, I will have 1000 cache data areas if there are 1000 buffers in the buffer pool. For every buffer I have to have a header. So among other informations which can also be there in the status field is whether the content of that buffer is valid. See while processing, some process may find that there is some data error in the data contained in the data area of some buffer. So moment it finds that it is invalid then the content of the data has to be invalidated, content of the buffer has to be invalidated.

So whenever such an invalid data is encountered, immediately some status bit in the status field has to be marked saying that the content of this buffer is invalid, it should not be processed by any other process. So likewise there can be many other informations put in this status field that we will come subsequently. Usually no. For the efficient utilization of the storage, that will help. What you are saying is right but the moment I go for such a flexibility, the management becomes too difficult. So usually what is done is for a particular file system, the block size is usually fixed. I can have more than one devices on the system. say for example when you are working on pc's, you must have seen that you have c drive, you can have d drive, you can have e drive, you can have f drive, each of them is a different file system. Logically they are taken as different devices. Now this device number and block number taken together that uniquely identifies a particular block in a files system. If I put only block number then I don't know that this block number belongs to what, whether it belongs to c drive or it belongs to d drive or b or belongs to e drive and so on.

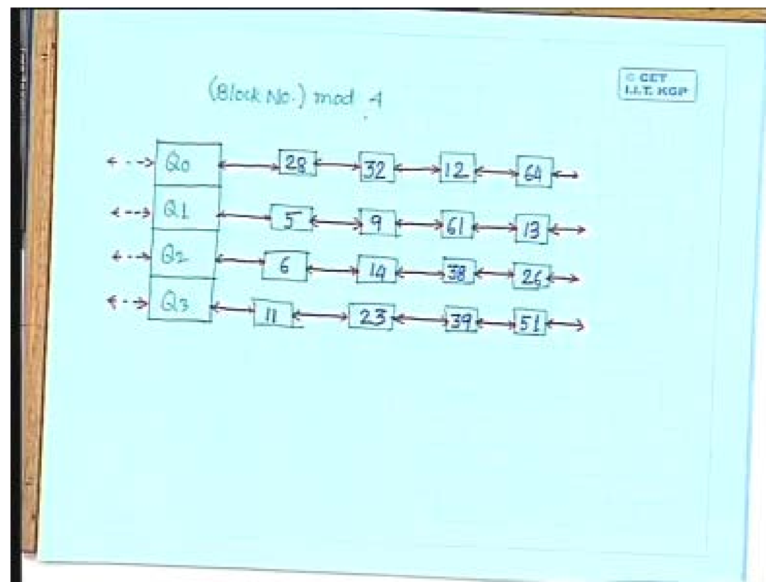
So these two together uniquely identify the particular block in the system and that is important. Then among other things in the header, we have said that we have a number of pointers, one pointer is obviously going to the data area because this is the one which actually contains the block data, data block. then we said that the buffers can be maintained, are maintained in two different queues, one is the free queue that means when I said this status field, if the status field indicates that the buffer is locked then this buffer will not be present in the free list queue. If the buffer is unlocked or the buffer is free that is indicated in the status field then this buffer will be existing in the free list queue. When it exists in the free list queue that does not mean; that the data area of this particular cache does not contain any data. It may contain a valid data which is actually copy of block number on some device. So it will contain the valid data but the buffer is not being currently used by any other, any process. so it will exist in the free list queue, simultaneously it will also exist in the hash queue, so we have two pointers pointing to hash queues and the hash queue this pointers will determine that on which of the hash queues this buffer exists.

Now why we need hash queues? Suppose in a system, we have 1000 of buffers then following a page fault interrupt what the canal will do is it will try to find out whether the block requested is present in the buffer queue or not, is present in the buffer pool or not. For that it has to go for a linear search. So if I have say 10000 buffers in the system, the system is so configured that I have 10000 buffers in the system then in the worst case before the canal says that particular block does not exist in the buffer, it has to make 10000 search operations. On an average to get a buffer, the number of search operations is half of 10000

that is 5000. So tremendous amount of search time will be wasted just to see whether the buffer is existing in the, whether the block exists in any of the buffer caches or not. So to reduce the search time what is done is the same buffer is also maintained in a queue which is called a hash queue. So this hash queue and the free list queue they are actually overlapped.

I don't maintain separate buffers, one for the hash queue and other for the free list queue. It is the same buffer which can simultaneously exist on a hash queue and also on a free queue. If a buffer contains a data of any of the block, it will always be present in the hash queue but it may not be present on the free list queue. So that is what we started saying that in a particular implementation, if we decide that we will have say 4 number of hash queues. How do you decide that hash queue? Whenever for the time being, let us forget about the device number, let us concentrate on say suppose we have a single device in the system. So, only block number gives you an indent unique identification of a data block on the file system.

(Refer Slide Time: 00:17:54 min)



So what we do is we perform some hash function on this block number to decide that on which of the hash queues, this buffer will exist, a buffer containing a particular block will exist. So the simplest search hash function is a mod function. So suppose a process puts a request for say block number 5 then hash function that will be performed is 5 mod 4 which gives you an output of 1. So if there is any buffer containing block number 5 then that buffer must exist in hash queue one, it will not exist in any other hash queues. So now instead of searching for all the buffers, the canal will search only buffers in hash queue one to see whether this block number 5 exists in the buffer or not. If it exists, it will exist only in hash queue one, if it does not exist in hash queue one then there is no buffer containing that block number 5.

So the situation will be something like this. When I go for this mod 4 as a hash function then what I will have is I will have a number of buffers which will be placed in one of the hash queues. So in this particular situation I will have 4 such hash queue. So this is hash queue 0,

this is hash queue 1, this is hash queue 2, this is hash queue 3. There will be 4 such hash queues because I am making use of mod 4 as the hash function. And suppose there are a number of buffers in this hash queue something like this. Device number is needed but that will lead to a complicated hash function. So for simplicity I am assuming that only block number, I am having a single device but that is needed. But the concept is still valid whether I go for a single device case or multiple device case.

So I have situations something like this. So let me assume that I have 4 buffers in each of the hash queues. Now this queue number 0 suppose it will contain block number say 28, say 32 12, 64 something like this. this may contains a block number 5, block number 9, say block number 61 then tell me any number say block number 13. This may contains a block number 6, block number 14 then say block number 38, say block number 26. This may contain block number say 11, block number maybe 23, maybe say 39 and say 51 something like this. So you find that if I perform 28 mod 4 that becomes 0.

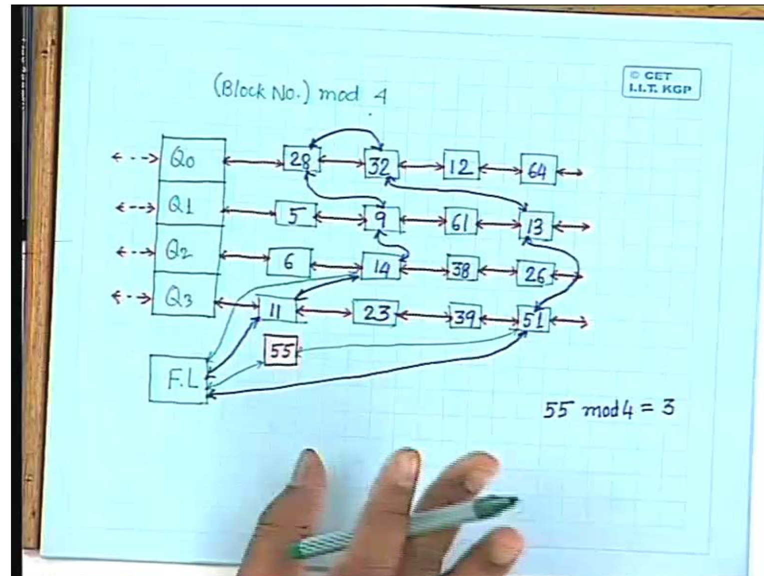
So if there is any buffer containing block number 28, it has to exist in hash queue 0. 23 mod 4 that gives you 3, so if there is any buffer containing block number 23, it has to exist in hash queue 3. Within every hash queue, all these buffers are maintained in the form of a doubly connected circular link list. So I can assume that these are the header nodes of every such hash queue with these buffers connected in a doubly connected circular link list like this, so like this it will be. So then as I said that a buffer can simultaneously exist on a hash queue and a free buffer queue. So I also have to have a free buffer list with a header node, so this is free buffer list.

Free buffer list is also maintained in the form of a doubly connected circular link list. So here because all these buffers are present in the hash queue, some of them will also be present in the free list or maybe all of them will also be present in the free list, if none of these buffers are being used by any of the process at a particular instant of time. So I can have a free list like this, it can be present in any arbitrary form depending upon the way they are used. So all these buffers are present in the hash queue they are also present in the free list; that indicates that these buffers are not currently used by any of the process. Now how this buffer management is done? Whenever a process puts a request for a particular data block following a page fault interrupt, so for example a process has put a request for a data block let us say data block which is not present in this buffer queue. For example the data block say 55 is it present? No, suppose a process puts a request for data block 55. First, what you have to do is you have to perform 55 mod 4 that gives you a value 3. That means if the block, if there is any buffer containing this block number 55 it has to exist in hash queue number 3.

So I directly come to hash queue number 3, check the buffers present in the hash queue number 3. I find that there is no buffer with block number equal to 55 and this block number will be identified by the block number field in the buffer header. So what I have to do is I have to match this block number with the block number field in the buffer header and I find that there is no buffer header in this hash queue which is having block number field equal to 55. So immediately I say that this block does not exist in the buffer but the process needs this block, so I have to have some way of getting this block from the secondary storage, put

it into one of the buffers. For doing that what the canal will do is it will simply check this free list because it knows that the buffers which are present in the free list they are not used by any of the process at this moment. And this buffer overwriting is done following LRU technique that is least recently used technique, LRU least recently used technique.

(Refer Slide Time: 29:38)



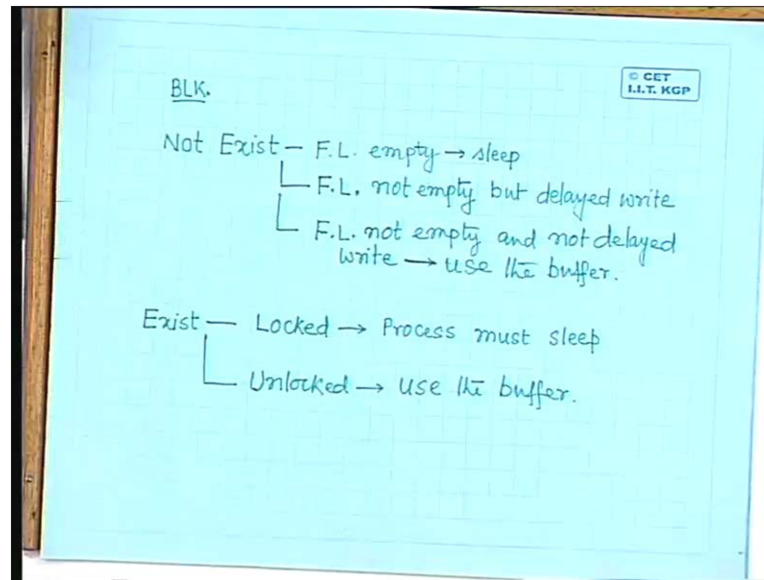
Following the same logic that we have done in case of page replacement that assuming a buffer which has been used most recently that will also be used next. So a buffer which is not used most recently or a buffer which is used least recently that will not be used for a longer period of time. So based on that assumption even this buffer replacement is also done following the LRU technique. So this free list is so maintained that a buffer which is at the head of the free list that is least recently used, a buffer which is at the tail of free list that is least recently used. When I say head or tail that is following the forward pointer of the free list buffers. So following the forward pointer the first buffer that you get that is the least recently used buffer and we should try to replace the content of this buffer that is block number 11 by block number 55.

So what you have to do is we have to simply take out from this free list, replace the data content by block number 55 and once this transferring data, copying the data from block number 5 to this buffer is complete, we have to return this buffer to the free list until and unless it is actually accessed by the process to transfer the data from the buffer to its user area. and when we return this to the free list, we have to return it to the tail of the free list because now this buffer becomes the most recently used buffer. So what you have to do is we have to simply overwrite this by block number 55. And now which one becomes the most recently used? that is 55 which has to come at the tail of the free list and the least recently used among all these buffers which are there on the free list is this buffer containing block number 14 because earlier this was the least recently used one which was at the head of the free list, now this has been taken out. So the next buffer which is least recently used in the free buffer is block number 14. So I have to modify the pointers like this and 55, it will

appear in the same hash queue because it is hash queue number 3 only. So it will remain in the hash queue, same hash queue but its position on the in the free list will be different. Now your free list will appear something like this. The other nodes in the free list remains intact.

So whenever a process puts a request for any of the data block, I can have one of the two situations either the data block is present in the buffer cache in the corresponding hash queue or there is no buffer containing that data block. So let us analyze these two different situations individually. A is the most recently used.

(Refer Slide Time: 00:30:15 min)



So whenever a process puts a request for a particular data block then I can have two situations, using this block number the canal has to search the corresponding hash queue to find out whether there is any buffer containing this block or not. Then I can have two situations, in one situation the block does not exist in the hash queue and in the other situation I can have the block existing in the hash queue. Now if the block exists in the hash queue then also I can have two different situations that the canal finds that the block exists in the corresponding hash queue but the block is currently locked, the buffer is currently locked because the same buffer is being used by some other process at that time.

So I can have a situation that it exists but locked. So if the block exists in the buffer and the buffer is locked then the only way is because the data is already existing into the buffer, so I cannot have another copy of the same data block into another buffer. So in this case the process which is requesting for it that must go to sleep mode and the process will wake up when this buffer becomes free. The second situation can be that the buffer exists and it is unlocked and that is the situation that we expect. So if the block exists in the buffer and the block is unlocked and the buffer is unlocked or the buffer is free then the process can use this buffer.

In the other case when the block does not exist in the buffer then also I can face various situations. Block does not exist that means I have to physically read the block from the secondary storage, put into one of the buffers which is free. I have to get the free buffers from the free list, from the header of the free list. Now when I try to get this block, I can have a situation like this that I find that the free list is empty because I can have a situation that all these buffers which are present which are containing some data blocks they are being used at that instant of time, all of them are being used by some process that means all the buffers are locked. If all the buffers are locked then obviously the free list will be empty because free list is a dynamic situation. So here I can have a situation that free list empty.

So if the process finds that the free list is empty then again the process has to go to sleep mode because I don't have any other option because there is no free buffer where I can put this new block. So the process has to go to sleep mode. Now there is difference between this sleep mode and this sleep mode. What is the difference? Here when the block exists into the buffer, the process sleeps until and unless that particular buffer becomes free and in this case the process is looking for any buffer which is free and currently there is no buffer in the free list. So here the process will sleep until and unless there is some buffer which becomes free. I am looking for a free buffer, this free buffer can be taken from any of the hash queues but when the buffer contains block number 55 then the buffer will be returned to hash queue number 3. See what I have done here.

In this particular situation block number 11 was at the head of the hash queue, at the front of the hash queue. So I have extracted block number 11, loaded block number 55 into this buffer. Now incidentally in this case, the previous location of the buffer was hash queue number 3, after loading the data from block number 55 the buffer has to be returned to hash queue number 3. So here it remains in the same hash queue but if my situation was like this suppose this buffer was absent. So this block number 14 a buffer containing block number 14 is at the head of the free list. Process requests for block number 55, I have to get a free buffer from the head of the free list. Now at the head of the free list, I have block number 14. so I have to take out the buffer, overwrite this with block number 55 then when I return it to the hash queue, this has to come to hash queue number 3, it cannot remain in hash queue 2 anymore.

So the new location of the buffer will be depending upon, what is the new block number that is contained in that buffer. but this buffer, free buffer may be taken from any of the hash queues because here we are interested in any free buffer which is at the head of the free list and that may exist in any of the hash queues. So that is the difference between this one and this one. Here if the canal finds that the block exists in the hash queue but it is locked then the process has to go to sleep mode and it will be in sleep mode until and unless that particular buffer becomes free, not any buffer. But in this case the free list was empty and the block is not contained in any of the buffers, so block has to be read new and any free buffer will serve the purpose because in any case I have to read the data from the secondary storage and put into one of the free buffers. So earlier position of the free buffer can be in any of the hash queues but the new position of the free buffer will depend upon what is the block number that is being read. Is that okay?

So here the process goes to sleep mode until and unless any of the buffers become free. if any buffers become free then that buffer will be returned to the free list, so free list no more is empty and it is the same buffer which will be head of the free list as well as tail of the free list because earlier there was no buffer in the free list, now one buffer has been returned. so whatever be the earlier position of the hash queues, simply take out that buffer, overwrite that with the new block and when you return to the free list of the hash queue you have to place it in the appropriate hash queue depending upon which block has been left. The other situation can be that free list is not empty but it is marked as delayed write. Now what is this delayed write? We have said that whenever we wanted to read a disc block, first we try to search that into the buffer cache just to reduce the frequency of disc operations to be performed.

Similarly if I want to write some data into a disc block or I modify any of the disc blocks then first modification is done on the buffer cache, you don't modify that immediately on to the disc. If the data, if the corresponding block content is present in any of the buffer, you do the modification in the buffer, don't modify it on the disc immediately. But once you modify any of the buffer content, you mark that buffer as delayed write and that delayed write information has to be placed in the status field of the buffer cache. So if I modify the content of any of the cache buffers, buffer cache then the status field of the corresponding cache header has to indicate a delayed write mark.

Now why this delayed write is necessary because in case I have to overwrite this buffer with the block of another data with another data block and maybe this happens to be the header of the buffer; the header of the free list. As we are taking always a free buffer from the header of the free list for overwriting then this is the block which needs to be overwritten by new data block. In that case if it is marked as delayed write that means now the actual disc content is different from the content of the buffer. When you read it, you copy the disc content into one of the buffers so they are identical. The moment you modify the buffer then the content of the disc and content of the buffer are different. So before you overwrite this buffer, what you have to do is you have to write the content of the buffer on to the disc then only this buffer can be overwritten, otherwise all this modified data will be lost. So if you find that the free list is not empty, the free list contained a number of free buffers and so what I have to do is I have to take the buffer from the header of the free list, from the head of the free list and the buffer that I get from the head of the free list that is marked as delayed write. if it is marked as delayed write then we have to initiate the process of writing the content of the buffer into the corresponding disc block and this process, writing process what is invoked is called an asynchronous writing.

Asynchronous in the sense that process who is requesting for the block has initiated this writing operation but it does not wait for the writing operation to be complete because any free block will meet the requirement of the process. So what the process does is it simply initiates this write operation but without waiting for this write operation to complete, it searches for another free block. so what will be done is in this particular case while loading this block number 55, we take this block number 11 from the free list because block number 11 is at the header of the free list and we find that block number 11 is marked as delayed write. So what we do is we simply initiate writing the content of this buffer into block

number 11 on the secondary storage, on the disc but don't wait for this buffer to become free because once you have started writing the content of the buffer to the disc, the buffer must have been locked by the canal. But you don't wait for this write operation to be complete. one way can be you let this writing operation be complete then get this same buffer to load this block number 55 but instead of doing that because in this case any of the free buffers will meet my requirement. So what I do is I simply start writing this into secondary storage but without waiting for it I will try to look for whether there is any other buffer in the free list which is free.

So initiate writing this block number 11 on to the secondary storage then you search for the free buffer on the free list, another free buffer on the free list and you find that the next buffer is block number 14 and it may so happen that block number 14 is not marked as delayed write. So you get block number 14, write this block number 55 into block number 14. Now place this buffer into this hash queue and return the buffer at the tail of the free list because it is the most recently used one. But what will happen to this buffer? This buffer because we have start initiated the process of storing this buffer back on to the secondary storage onto the disc. Now at the completion of this write operation, the device rather will give an interrupt saying that the writing operation is complete following that because this was locked only for storing the data on to the secondary storage, once this writing operation is complete this block will become free but still what I have done is I have simply copied the data, the block still contains the data it has not been overwritten. So it will be remaining in the same hash queue but now position of the in the free list is likely to be different.

See what we are doing is every time we are returning a block to the free list, we are always returning it to the tail of the free list because that is the most recently used block. But what is this situation? In true sense it is not most recently used, it was least recently used in between we have accessed this only to save the modified content onto the disc. So it is still least recently used. So in this case what will be done is when you return this block into the free list, you don't return it at the tail but you return it at the head because truly speaking it is not most recently used, it is still list recently used. So this is one situation when you returned a buffer to the free list I don't return it to the tail but I return it to the head. The status will be changed. The other situation is I get a free buffer into the free list and that is not marked as delayed write. Free list and not empty and not delayed write and obviously this is the situation that we expect when the buffer can be used by the process to write the new data, so you can simply use the buffer. And obviously now the new location of the buffer may be different depending upon which block is being read from the secondary storage and put into the data area of the buffer.

So one situation we have said that when the buffer can be placed at the head of the free list, there is another situation that a process puts a request for a buffer for a block, the block exists in the buffer, the process gets the data from that buffer puts into user space, while processing it the process finds that the content of the buffer is not valid there is some data error. I can have a situation like this. Now it is always expected that I should not maintain any buffer containing an invalid data. So I should try to overwrite it as early as possible with a valid data. In such cases also when this buffer is to be returned to the free list, it should be

returned at the head of the free list because only the buffer which is at the head of the free list that is going to be overwritten next.

So there are two situations in which we can place a buffer at the head of the buffer list, one situation is that when just writing operation has been complete because of delayed write, the next situation is when the content of a buffer is found to be invalid. In all other situations we will place the buffer at the tail of the free list but a buffer can never be inserted in middle of the free list, it will either go to the head or to the tail not in the middle. So let us stop here today.