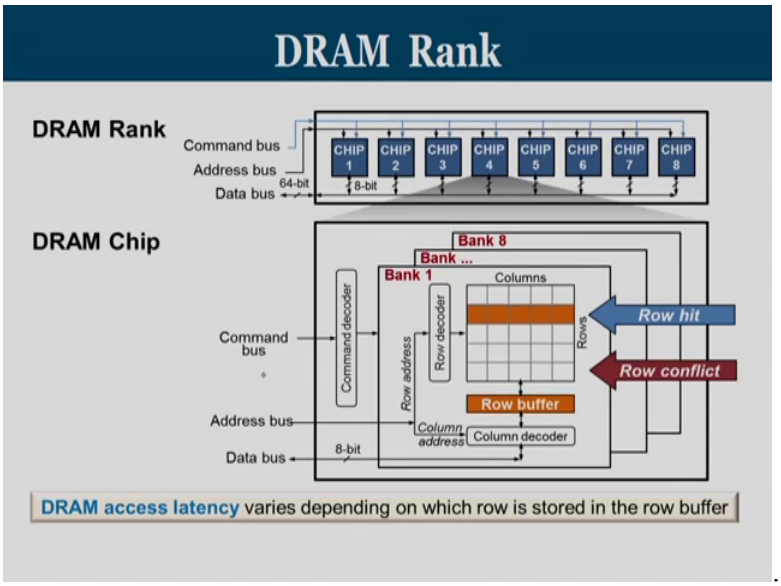
Multi-core Computer Architecture – Storage and Interconnects

Dr. John Jose
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture – 11
Address Translation Mechanisms

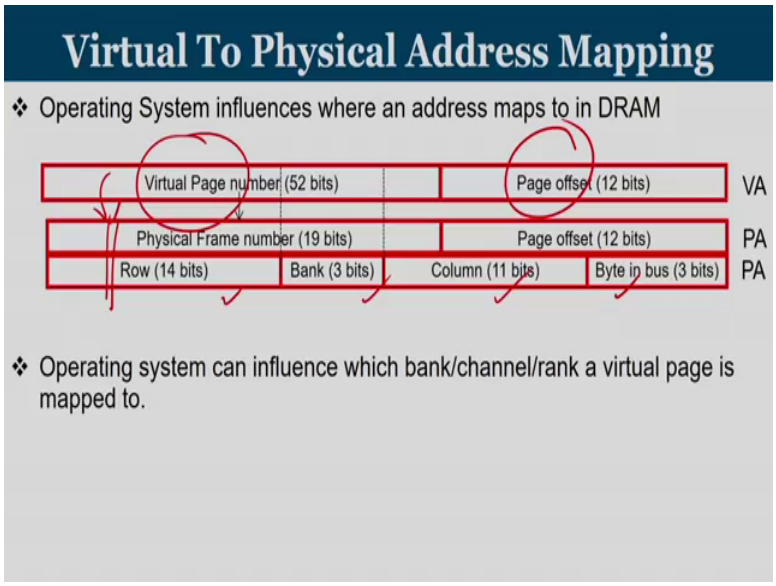
Welcome to the 11th lecture of the course today our focus of discussion will be on Address Translation Mechanisms. So, far we were trying to understand how cache memory and main memory systems are working and we see their internal organization and how cache memory interacts with the main memory. Now we will slightly pitch into one level up to understand what is the role that is played by operating system to make sure that the hardware cache memory as well as the supporting main memory is working together such that it meets the requirement of the application program.

(Refer Slide Time: 01:17)



One such important component is address translation mechanisms. This is what we have seen, already we have your main memory system in place, which we call it as DRAM and one of the level of granularity that we address DRAM is a DRAM rank which itself consists of multiple chips and each chip is going to give you some component of your data, some subset of your data and we have multiple banks.

(Refer Slide Time: 01:39)



Now, today we will focus little bit aspect on the concept of virtual address and how it is correlated with the concept of physical addresses. We have seen that operating system is going to influence significantly on the address that is mapping on the DRAM. So, we have a concept called a virtual address that is what we have seen and there is an offset.

Today we are going to discuss deeper about what is significance of offset and what is this virtual page number all about and we have the physical frame address. So, if you take up any main memory system, let us say you have 1 GB of physical memory that is available then the physical address consist of 30 bits. Here we are talking about physical frame number of 19 for old by an offset of 12; that means, total 19 plus 12 we have 20 sorry, 31 bits of address; that means, we have 2 GB of physical memory that is available. And it is this physical address is what we divide in to rows, columns, banks and many other things that we have seen already in the DRAM structure.

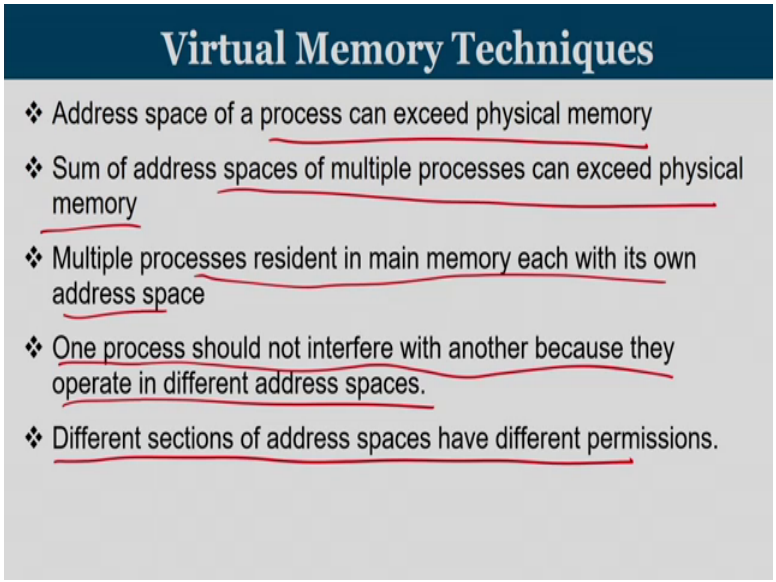
So, there is a process by which a virtual page number is translated to a physical frame number and that is being given by the operating system. So, you are going to bring a chunk of data from the secondary memory and going to place it inside main memory. So, the address of the chunk of data once it is residing inside the main memory is known as the physical address. Now it is operating system which we look through the memory and find out where in memory the space is available and if you have multiple spaces there is available inside your main memory, operating system will pick one among that. And

whatever you pick let us say, you can give the 1st frame or 10th frame or 100 frame, let us say whatever be the frame number at the end the physical frame number has a significant say in finding out what is the address that we are going to give to this chunk of data and once you know the physical address in place it is this physical address we are going to divide into tag, index and offset. So, any address that is been assigned to a chunk of data indirectly that will tell you what is going to be your tag.

Let us say if you put certain location or certain block of data in the initial few portions of your memory, let us say more significant bits all are 0. So, you have one tag let us say, tag values a if you, if the operating system is making a different choice by keeping the same data maybe at a lower end of a memory it may be having a different type. So, the index as well as a tag is dependent on what is your physical address and this physical address governs where it goes in the cache memory.

So, operating system can influence essentially which bank you have to keep or in which index in cache memory you are going to keep or which bank or which channel it is going to be kept in the main memory. So, the role of operating system is twofold, it will take a call a piece of data should be located in which bank, which rank and the which channels rows and columns. So, it governs the placement of data in the main memory similarly, once you have a physical address in place and that is what which determines in which set of the cache memory this data is mapped. So, indirectly operating system governs the placement of data in cache memory as well.

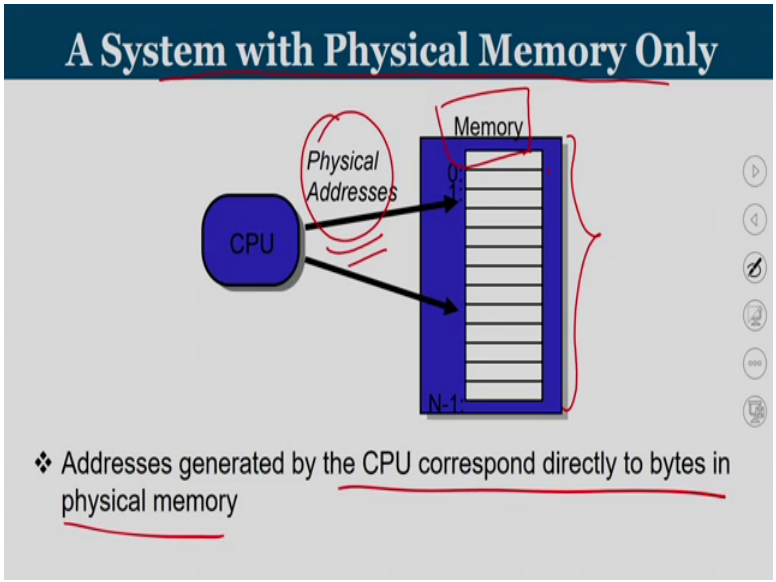
(Refer Slide Time: 05:23)



So, it is highly required to understand, what is the address translation mechanism that helps overall the operating system in getting a data done? The next topic what we are going to discuss today is the virtual memory techniques we know that the physical memory that you have it cannot accommodate all the programs. So, the address space of a process can exceed physical memory if you are going to watch a very big process. Let us say, you are going to watch a movie and the available physical memory space is 500 MB and you are going to play something which is more than 500 MB in size.

So, you are having a scenario where that the space of a process let us say in this case a movie program is exceeding the physical memory. Sometimes when you are in a multi programming environment the sum of address space of multiple process, let us say you have three program a, b and c. The sum of address space of a, b and c can exceed the physical memory as well. We have scenarios in which multiple process are resident in the main memory each with it is own address space and we sometimes require another kind of restriction, where one process should not interfere with another because they operate in different address space. You may have a program a at the same time you may have another program b, I want a to be highly restrictive on b or b should not be able to access a or a should be given permission to access b, some kind of a slave or a master relation also we want.

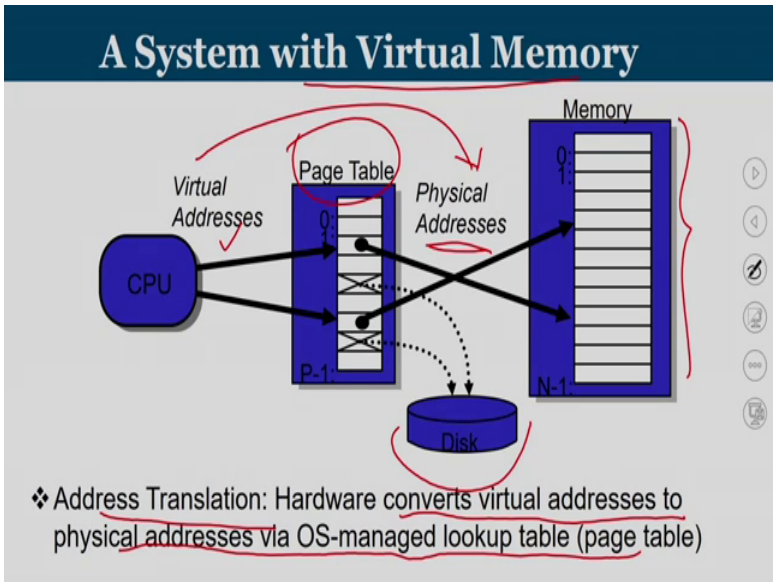
(Refer Slide Time: 07:03)



So, and sometimes different sections of address space may require different permissions let us say the first of one function anybody can access, second function only a can access. So, we can put it restrictions in accessing these locations as well. To begin our discussion on virtual memories let us first consider a system it has only one type of memory that is called physical memory. So, whatever is a address that is generated by CPU correspond directly to the bytes in the physical memory.

So, we have a system in place and this system has only physical addresses in hand. So, when you have only physical addresses whatever CPU is generating it is only physical address and you have a memory here, whatever is addressed it is generated by the CPU correspond. So, the CPU generates an address and the address reaches memory and you return the data, CPU returns our CPU generates another address that is also going to memory and memory is going to respond with the corresponding data.

(Refer Slide Time: 08:11)



So, every address CPU generates it is available inside memory or CPU generating an address which is subset of set of all possible address in memory then, we call it as a system with physical memory only, but practically there is a problem we cannot have that much big physical memory. So, reality what you see is the modern day memory it is system with a virtual memory in hand. So, you have a memory and then you have a disk that is available, the disk can accommodate all the programs and then we have something like a page table, where CPU is going to give only virtual addresses. It is a duty of the page table to translate the virtual address that you have, this virtual address is translated to physical address with the help of page table and once you get physical address that is a place where this data or instruction is loaded in memory.

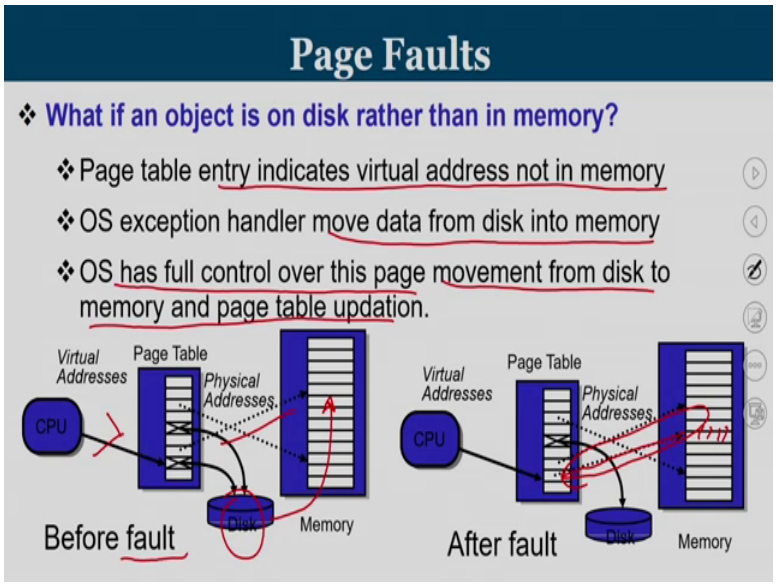
So, in this context when CPU is going to generate an address, it may not know whether where it is available. So, we need to have another hierarchy of address and that is what is called the logical address. CPU is dealing with a process and every process has a logical address that starts with 0 to CPU gives an address it is a logical address and this can be placed anywhere in the main memory. So, you need to have a translation process that converts the logical address to the physical address and that is what is known as a page table.

So, we have an address translation that takes place. There is a hardware that converts virtual address into physical address by an operating system manage lookup table and

that is what is known as a page table. So, this is what is called a system with virtual memory. So, when you have a system with virtual memory your virtual memory can be relatively large, CPU is going to generate virtual addresses and from these virtual addresses you are going to translate them into physical address and once you get the physical address then you can go and access the memory. So, in the meantime there can be movement of data inside your physical memory from one end of the memory to another, it is not a problem at all.

Similarly, any moment that happens inside the memory, the CPU is not going to get affected. Every time CPU mentions an address, it will be assisted by a structure called page table, where it will tell you this virtual address is available in this physical address and this translation process is what we look at to see today.

(Refer Slide Time: 10:27)



Now, consider the case that, what if an object is on the disk rather than on the main memory. Sometimes we have a scenario like when you access an object for the very first time it may not be available inside your main memory.

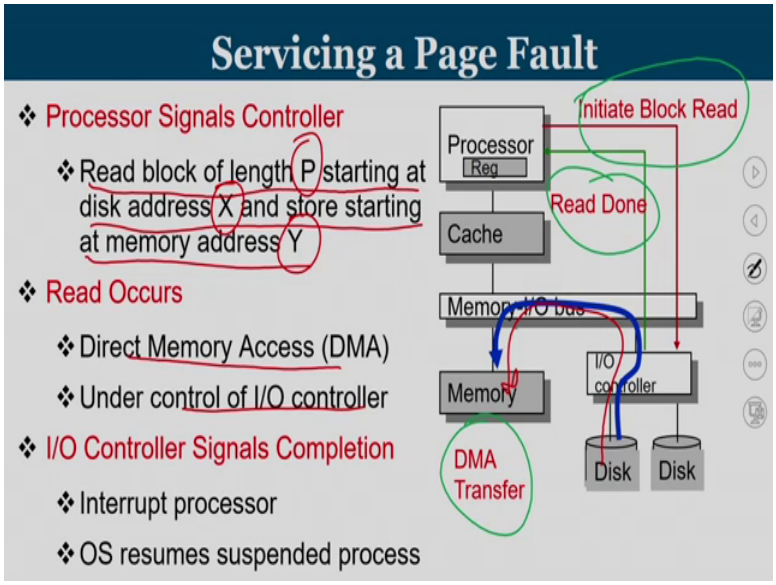
So, that particular object, the program or data whatever it is that may be available in the disk, we will deal with the scenario like that. Now a page table entry indicates virtual address that is not a memory. So, you have a scenario that is before a page fault, CPU is going to talk about an address when you look at in the page table there is no mapping from there, there is no dotted arrow from the page table into the main memory the

particular thing this known as a page fault, once you have a page fault your operating system exception handler move the data.

So, when you have a scenario where a virtual address given by the CPU is not having an equivalent physical address we call it as a page table fault, in the page table fault you have to pick up that particular data from the disk, put it inside memory and then make it ready for execution. So, the operating system has full control once the page is more from the disk to memory and there will help in the page table updation.

Now, after the page fault if you look at the scenario the, virtual address is given and when it is mapping to one location and from the location, you are actually pointing to a new main memory frame. So, previously we had a case where the virtual address given by CPU is not having an equivalent physical address and that is what is known as a page fault. Operating system takes cover whenever there is a page fault that is obtaining and find out a space inside main memory and the block is more from the disk into the main memory with proper updation in page table.

(Refer Slide Time: 12:42)



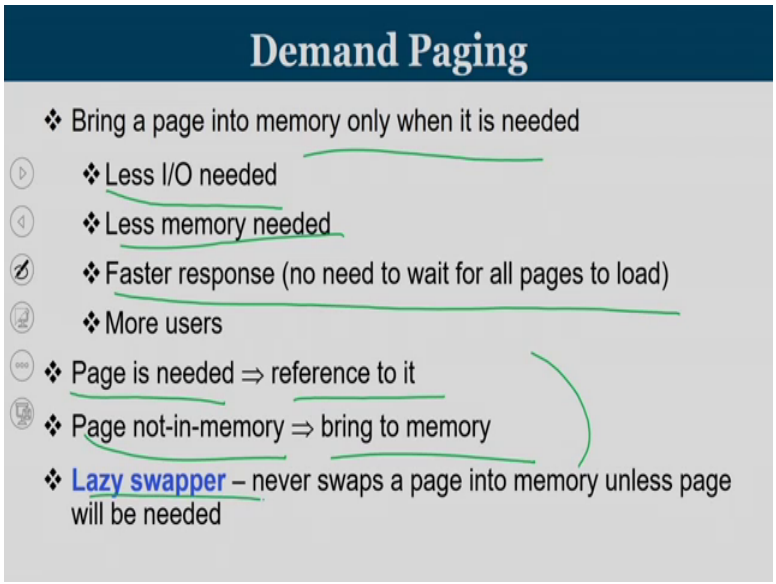
So, when you access after the page fault, wherever you encountered a page fault now you can see that there exist a mapping, a new main memory block is been assigned to this leading to further execution of the missed address. Now, how are you going to service a page fault? Consider the case you have a processor, a couple of registers are there, you have cache memory or the main memory and then you have the I O controller. So, there

is a main memory bus that it takes care of the transfer your step 1, whenever there is a page fault you have to take something from the disk and going to put inside main memory.

So, you have to read a block of length P starting at disk address X and store it at Y. So, essentially you are going to tell from the disk, how much you have to transfer, you have to transfer P bytes of data. This is going to start at location X and it should be stored from location Y onwards in your location in your main memory. So, you are initiating a block read and then using the help of a DMA controller direct memory access under the control of the I O controller the data moves from the disk all the way through the memory bus into the main memory.

So, you are transferring a block of data from the disk into the main memory and that is what is known as DMA controller and once it is done, the I O controller is going to tell the processor that he is actually done. So, there are basically three steps there is associated in this case. One is called you tell from where to read, to which location to copy, how much is the data? Second one you have a DMA transfer that is that from the disk, a byte data is copied to main memory and update the corresponding page tables.

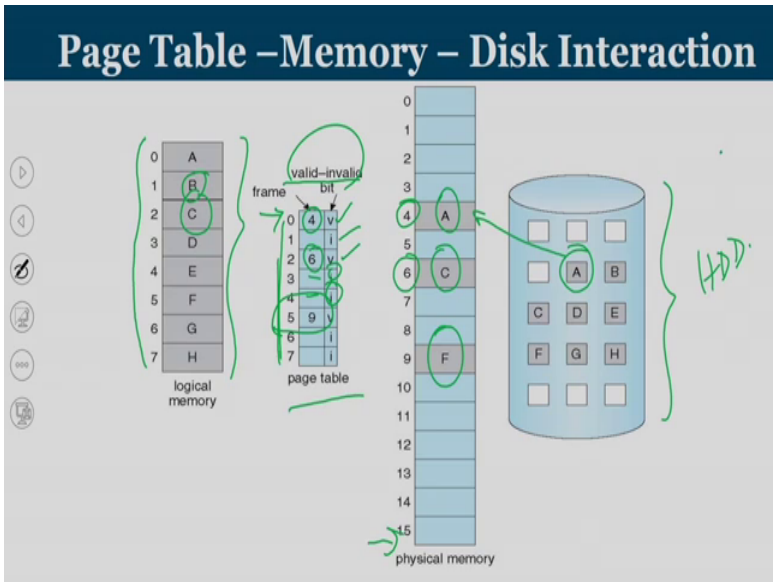
(Refer Slide Time: 14:13)



Once it is done the I O controller is going to tell back, the processor that whatever assign task is over. Now we try to introduce into the concept called demand paging. You are going to bring a page into memory only when it is needed, why it is so? If you are going

to bring only when it is needed, we have only less I O, less memory is required under response is faster because, we are not going to lode pages a priori and this is something that, we call it as a lazy swapper a page is needed you have the reference, if a page is not in memory when you reference it, you have to bring it to memory.

(Refer Slide Time: 14:54)



These two concepts are put together to known as lazy swapper. This is the physical view of what happens between a main memory and a disk interaction. So, you have a logical memory that is going to tell you, what are the contents that is associated with 8 pages of a block, then you have a page table that is going to tell you how will you map from the logical memory into the physical memory. Let us say you have a physical memory which is having 16 different frames, that is available and this is going to be your hard disk.

Now, how are you going to transfer? The meaning of this means now let us assume these are the 8 pages, the logical address page number 0 to page number 7 and these corresponds to some data, let us say 0 is basically A in the disk and it. So, happened that this A is already brought to main memory and it is located in the 4th frame. The meaning of it is the 0th page of that program is residing inside the fourth frame and it indicates valid bit is 1, but the first frame is not mapped this B is not mapped anywhere.

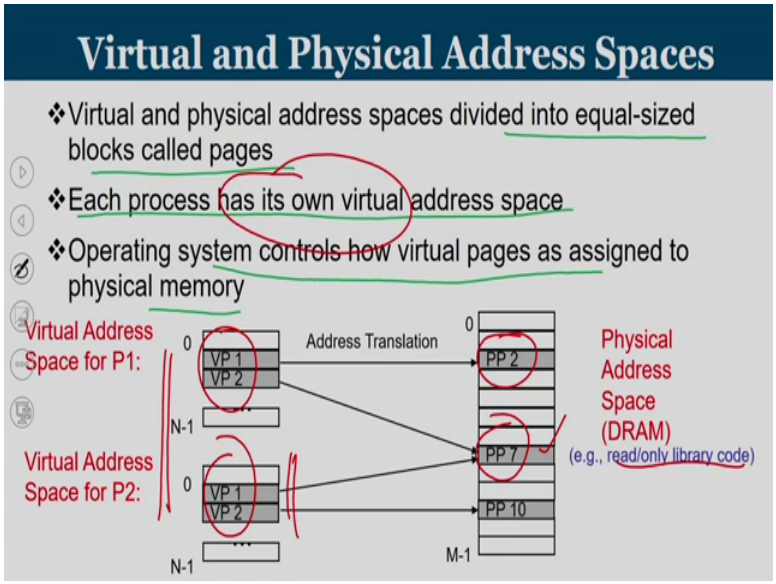
So, it is called an invalid, now the 2 to C, C is already kept and C is kept in the 6th frame that is why it is marked as valid and this so, this page table is going to tell me a mapping for the page numbers by which I am going to index into the page table. If the valid bit is

equal to 1 or if it is a valid a page table entry, we could easily see what is the frame number that is mapped so, wherever there is invalid; that means, no frame that is mapped or these particular pages are experiencing a page fault.

So, this is a visualization how a disk is going to interact with the memory. In this case whenever you have a logical address that is available, these logical addresses has to be converted to physical address with the help of a page table. Page table is indexed by the logical page number and whenever the valid bit for each of the page number is 1; that means, the page is having a mapping to main memory there exist a frame that is going to help and this valid invalid bit is going to help in this case.

So, once you get the frame number; that means, the 0th page is now available in the 4th frame, the first the second page is available in the 6th frame and the 5th page is available in the 9th frame.

(Refer Slide Time: 17:37)

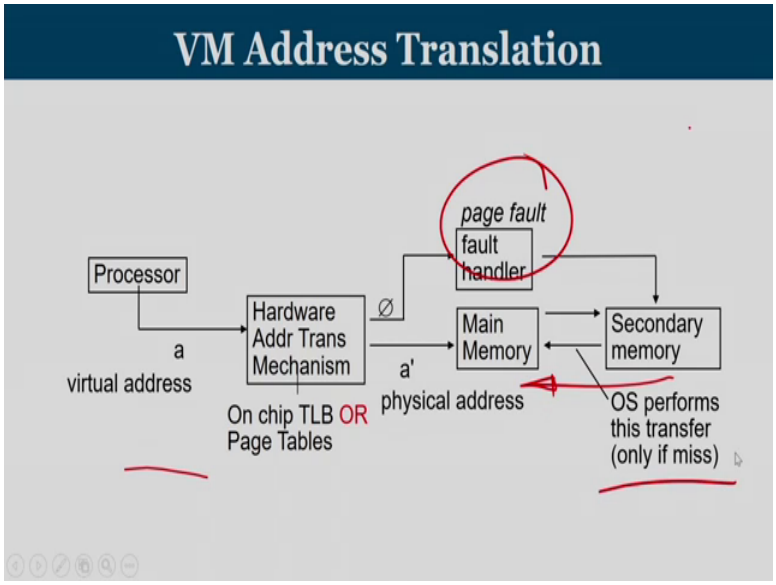


This is how the system works. Now what you mean by virtual and physical address space? Virtual and physical address space divided into equal sized blocks called pages. So, your entire virtual memory is divided into pages and physical memory is also divided into pages, but we call it as page frames and both page size and frame size are same, each process has its own virtual address.

So, each process starts on virtual address 0, 1, 2, 3 like that and operating system controls how virtual addresses are assigned to the physical memory. Now consider there is a virtual address space for P 1. So, you have a process P 1, where there is a virtual page 1 and virtual page 2 and you have another process P 2 that also has it is own virtual page 1 and virtual page 2. So, the concept of each process has it is own virtual address can be viewed here. They all have their own virtual address space starting from 0 to whatever they end.

Now, you have to consider that in the main memory these two process are having 1 page in common; that means, the virtual page 2 for P 1 and virtual page 1 for P 2 are essentially the same thing it can be a read only or a library file. So, there exists a translation your virtual page 1 of P 1 is actually located in the physical page 2 and virtual page 2 of P 1 is nothing, but physical page of 7, similarly you can see the mapping of virtual page 1 and 2 for the second process also.

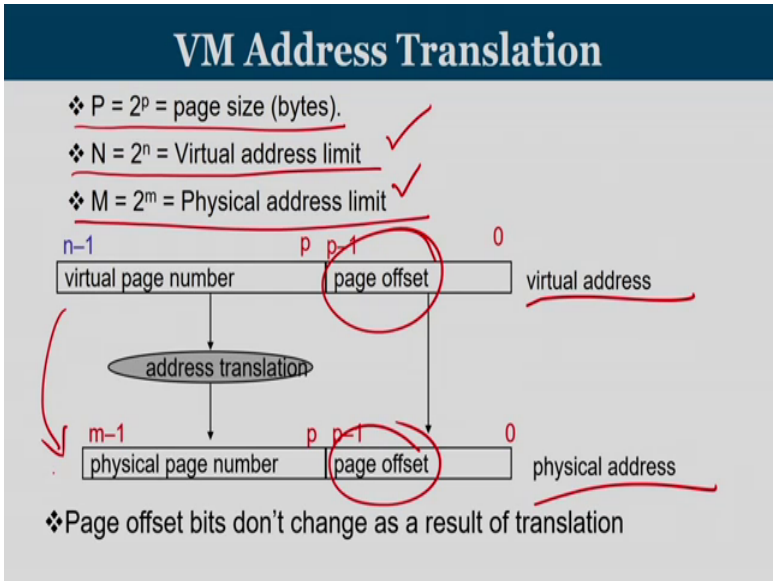
(Refer Slide Time: 19:13)



How it is being done? Typically when a processor is going to give you something, the processors will tell a virtual address you have an hardware address translation mechanism it can be an on chip TLB translation look aside buffer or a page table, what are they going to give? They are going to give you physical address. So, whatever is a virtual address a, this a is converted to a dash. So, from a virtual address you are going to get a physical address. Sometimes during the address translation mechanism you will

come to know that, there is no frame that is available and that scenario is known as page fault we call it as page fault.

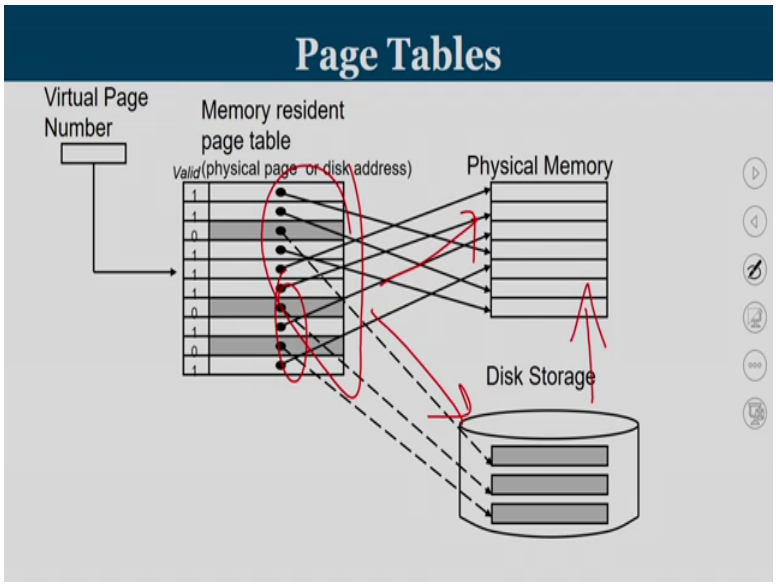
(Refer Slide Time: 20:09)



Now, once you get a page fault then you are going into the secondary memory and transfer condensed into the main memory and that is been done with the help of operating system.

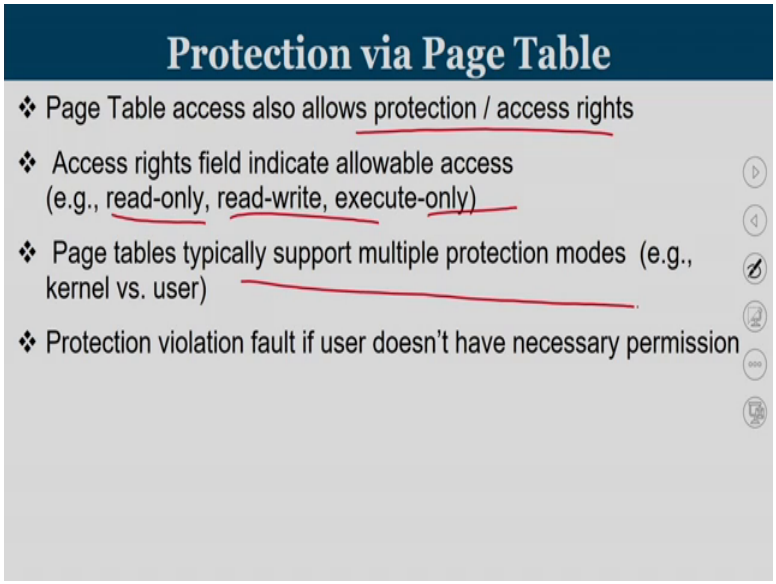
Now we try to understand the translation process little bit further let us say, I have the total of page size equal to 2 power p, which can be defined by capital P, virtual address of N and physical address of M. So, virtual address can be represent that using n bit smaller n bits, where how to power n virtual address location and two power m physical address location. So, once you get a virtual address we have a page offset and a virtual page number and then you have a physical page number that is, from virtual address to the physical address there is a translation process that is involved. You have a translation logic, but we now to understand that during this translation logic, the offset portion whatever offset portion you have, it is not going to be affected, the offset of virtual address and offset of physical address is more or less same.

(Refer Slide Time: 21:09)



So, essentially translation is a process where in virtual page number is converted to physical page number. This is how it looks like you have actual page numbers, these virtual page numbers are used to index into the table and once you index into the table that is a page table, it will tell you where to go, we can tell that it will tell you that to go into the physical memory, if something is missing there you are going to have an access to disk and from disk it gets storage to the physical memory.

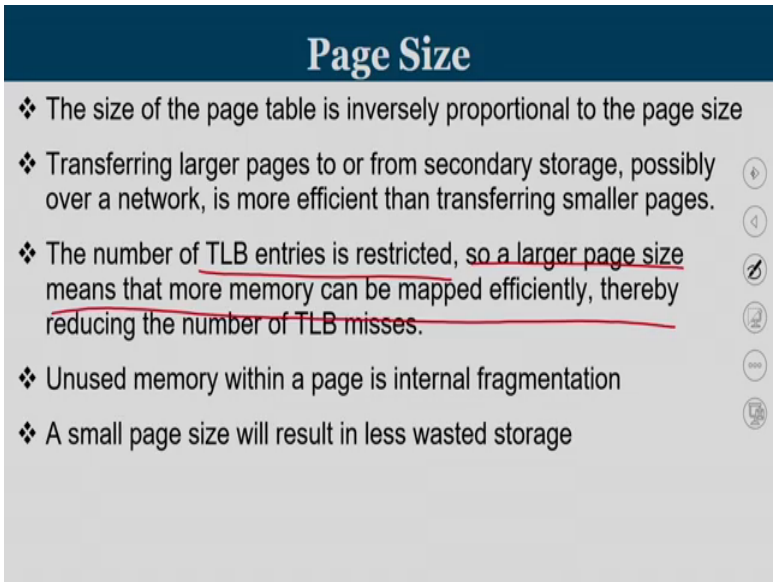
(Refer Slide Time: 21:39)



So, the idea of page table is to convert virtual page number into the physical page number. Now, you have certain scenario where the programmer wanted some kind of a protection with respect to certain segments of the program. So, in that case page table access will also permit protection and access rights. Access right means whether a program can be only in the read only mode or somebody can write on it or somebody can execute on it and page table typically contains few extra bits, which will tell you the processor can perform only read operation or read or write or execute. So, different kind of modes are there.

So, additional bits that you permit in the page table will tell you, whether a processor is allowed to perform that particular operation so, protection violation sometimes how it to be handled in separate cases.

(Refer Slide Time: 22:23)



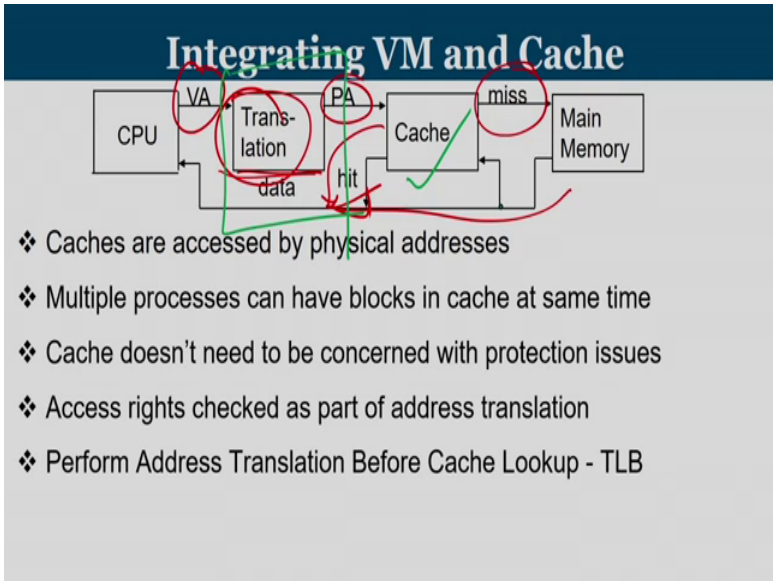
Now how will you find out what is going to be the best size? The size of the page table is inversely proportional to the page size. So, when the page size is very small you have more number of pages, when you have more number of pages your page table size is also going to increase because, we are going to have a separate page table for each of the process.

So, the more the number of pages the page table entry is also going to be higher. Transferring large pages to or from secondary storage possibly over through a network or a bus is more efficient than smaller pages, because overhead involved in going to a

secondary memory by rotating the disk and initiating something it is very high. So, once you have incured that much overhead it is always advisable to transfer little bit bulkier amount of data and the more is the page size the number of pages are going to cut down. So, we have a unit which is known as the translation look aside buffer or TLB. The number of TLB is restricted because TLB is part on chip let us say I can store only the 10 enrich there.

So, larger page size means that more memory can be mapped efficiently thereby reducing TLB misses. So, unused memory within a page will be causing internal fragmentation. So, internal fragmentation is something like let us say your page size is 4 KB and your program comes only up to 3.2 KB. So, 4 KB minus 3.2 KB, that this is 0.8 KB is internal fragmentation, I cannot assign the those 0.8 KB to any other program and when you use very small size pages you are going to waste lot of these storages.

(Refer Slide Time: 24:14)



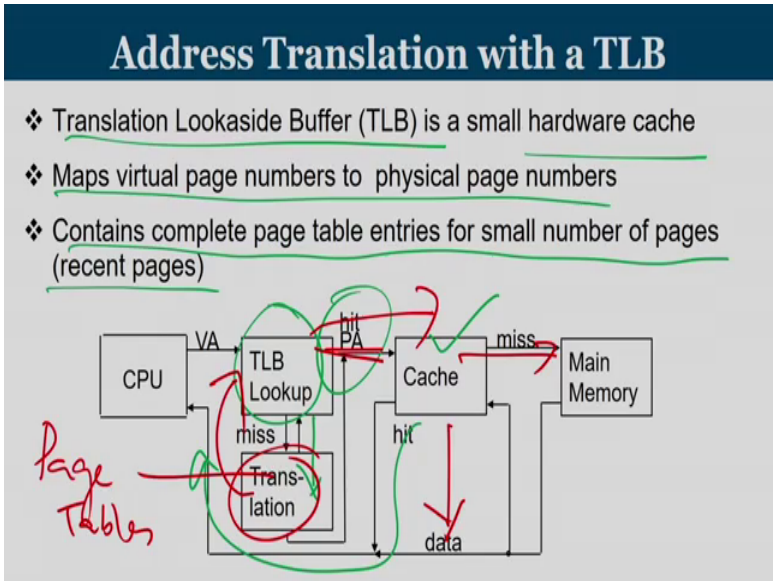
Now, we try to understand what is all of this virtual memory and the caches, we know that CPU is going to give you the virtual address. So, and the virtual address is given and there is a translation process, which converts virtual address to physical address and once you get physical address you are going to divide them into index tag and offset, you go into the index location perform tag comparison and the rest of the cache memory operation follows. So, in that case if it is a hit, we are going to transfer the data;

sometimes when you go to cache memory you can encounter a miss then you get it from main memory and then you transfer it to the CPU.

So, typically caches are accessed by the physical address so, far our notion was CPU gives an address you are going to break the address into index tag and offset, but today we realize that CPU is not going to operate on physical address, CPU is going to operate on virtual addresses. So, once you give the virtual address first you have to translate the virtual address into the physical address and then only the splitting of the address required for the cache memory is possible.

So, cache memory does not need to be concerned with the protection issues because before coming into cache anyway, I am going to the translation process. So, this translation process will surely look into the protection and all other aspects. So, as long as our translation process is there cache memory should not be bothered about what is the protection mechanisms so what we have to do? All the translation process and protection look up all these things should be done before accessing cache.

(Refer Slide Time: 25:51)



So, the position of TLB is before the caches. Now, how real address translation happens with the TLB, this TLB which is called the Translation Lookaside Buffer it is a small hardware cache it maps virtual page numbers to physical page numbers.

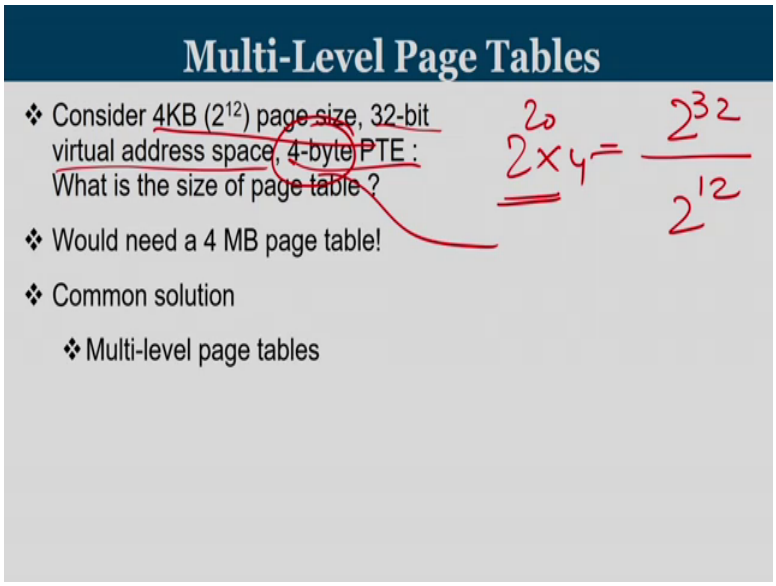
So, we have seen a page table. Every process has it is own page table, once the virtual address comes you index into the page table, page table if the valid bit is 1, it tells that for that particular logical page number there exist a physical frame number. Once you get the physical frame number go to the corresponding page and get things done, but for every access to cache memory, I need to have the physical address. So, always going to page table which it is residing inside main memory is not a cost effective solution. So, I need some faster address translation mechanisms. So, the most frequently used page table entries are kept in a hardware cache or a hardware register and that is what is known as translation lookaside buffer.

So, translation lookaside buffer contains complete page table entries of some of the pages. So, you have only 1 TLB in a system, when there is 1 processor. Whereas, you have n page table where there are n process. So, the most frequently used page table entries of all the process. Let us say, I am going to work with 10 process together, these 10 process are going to work with some 16 pages. The most frequently used page table entries are put in one register or one hardware unit and that is what is known as TLB. So, when CPU gives a virtual address, if it is a recently used to virtual address in TLB lookup you will get a hit, once you get a hit in TLB; that means, you got a physical address divide into tag index and offset, go to cache memory and return.

Sometimes when you look in TLB it may not be available there, that does not mean it is a cache memory miss. It is just that this particular virtual address was not recently used. So, any TLB lookup miss that goes into the translation logic and this translation logic is your page table. So, this translation is basically known as page tables. So, you take data from page table, update TLB and then you obtain the physical address look into the cache. It can be a hit when you return the data or if it is miss then you are going to the main memory. So, a context like a miss in the cache will surely be a hit in the main memory, you may be wondering how this happens our typical notion of understanding of cache memory and main memory is a bit tricky concept here.

So, if you go to cache memory surely; that means that you have a physical address. Once you have a physical address then the indexing and tagging and offsetting will work, but if you do not have a physical address you would not even reach cache. So, you are going to access cache only if you have a physical address and physical address is obtained by translation process.

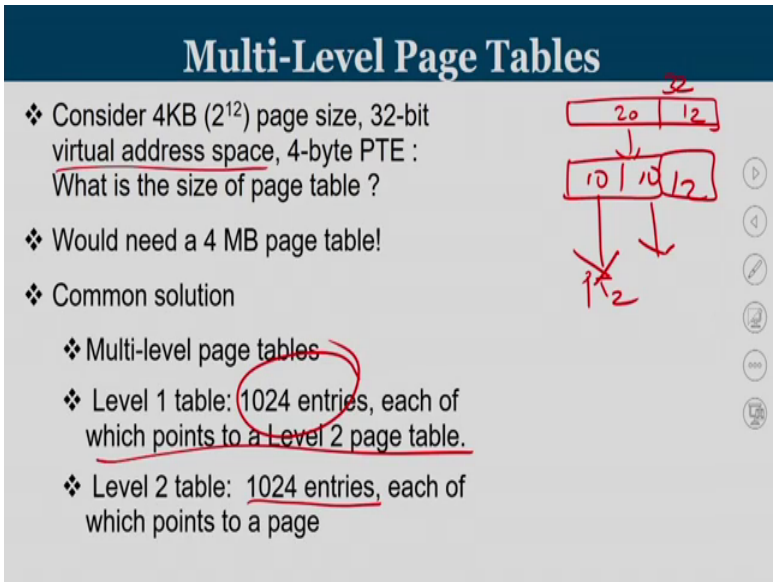
(Refer Slide Time: 28:53)



So, something is not in main memory then during translation process we would not get it that is way how it is going to work. Now, consider a scenario where you are going to work with 4 KB pages. So, 2 power 12 bits is the page size, let us say the virtual address space is 32 bit and each page table entry is going to consume 4 bits.

So, what is going to be the size of page table? So, when you have virtual address of the size of 2 power 32 locations are there and each of the page is going to be 2 power 12 locations then; that means, 2 power 20 pages are there that is 1 mega pages are there and we have seen that each of the page table entry is equal to 4 bytes. So, together we are going to have 4 MB page table, now when you have 4 MB page table the common solution is can we go for multi level page tables and that is exactly what we are going to see now.

(Refer Slide Time: 29:45)



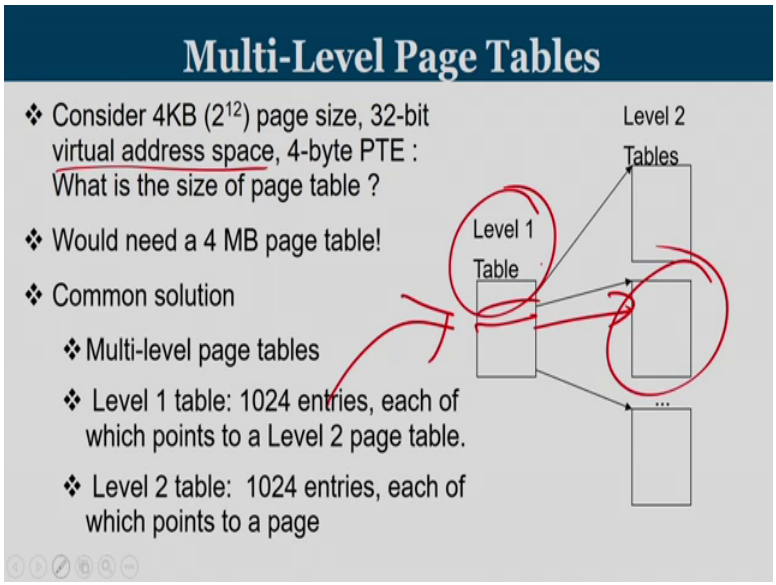
So, how are you going to get multi level page tables? First level let us say there are 1024 entries and each will, each itself is going to pick one among the level 2 table and then the level 2 table each has 1024 entries. So, we told that there are 2 power 20 entries in the page table rather than having one page table with 2 power 20 entries we have a first level table which has 1024 entries. So, only the 10 bits of the address is going to you be used to there. So, consider the case in this we have 32 bits of address out of it 20 bit will tell the virtual page number and 12 bit is offset.

So, when you have 20 bit virtual page number; that means, you have a page table with 2 power 20 rows and these 20 bits will tell out of the 2 power 20 rows, which is the entry that I am going to index and then will give you the frame number. So, that will take significant amount of space let us say, each page table entry is going to be 4 bits 4 MB is consumed by the page table of this particular process.

Similarly, you have multiple such process and all of them will be having their own page tables. So, storing page table itself will incur significant amount of overhead. The solution that is proposed is this 20, I am going to break into 2 10 each. So, it become 10 10 12. This first 10, using the first 10 bits you go into one of the entry of a 1024 first level page table. Each of the entry in the first level 1024 entry page table will point to another table that has another 24. So, this will tell you what is the next page table 2? And this 10 bits will tell you inside the so called located page table to which entry I have to

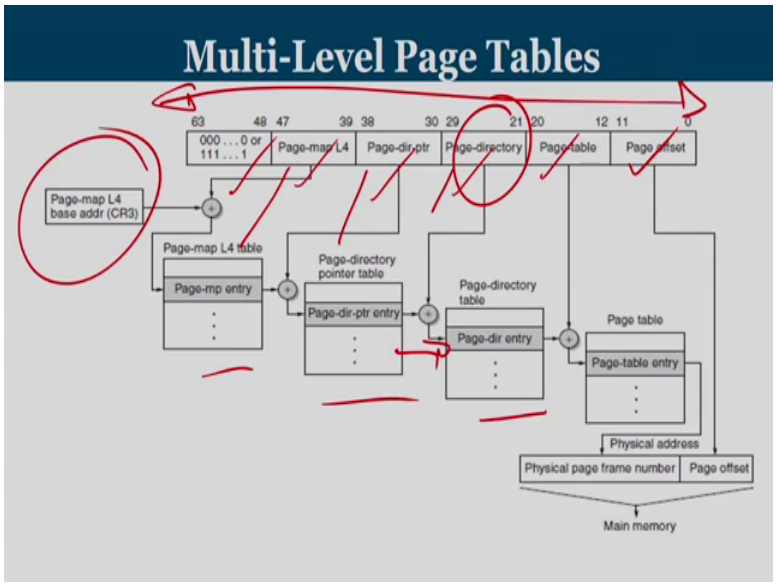
go. And that will surely help us to bridge the gap between a long compact, a long and complex page table into a combat page table. So, you have a level 1 page table that is going to help you.

(Refer Slide Time: 31:57)



So, using the most significant few bits you index into level 1 page table, that itself will choose you to level 2 page table using the next set of bits you are going to access into level 2 table as well.

(Refer Slide Time: 32:15)



So, this is a scenario that has been given where you have 64 bit virtual address and that consist of many many subdivisions of page, directory point there, page directory, page table, offset and all many levels are there.

So, you have your page map base address register will tell you the starting location with that you keep on adding at each level. So, each level is in turn pointing to another table and a few more couple of bits will tell you where to index and then you go to the next one take the next set of bits, like that you proceed. So, this is the concept of multi level page tables. So, when you have relatively large virtual address space rather than keeping one page table for the entire process, we can have a hierarchy of page table, where there is a level 1 page table which is our first point of indexing.

So, using fewer number of bits index into page table level 1 and each of the entry into page table level 1 is indirectly pointing to another page set of page tables. So, in the first example, what I have mentioned you go inducing 10 bits you go and index into one of 1024 page table entry, each of this entry indirectly point into another page table. And the next 10 bits will help us to index into second page table.

So, I have 1 first level page table, I have 1024 second level page table. So, one of the second level page table is weekly chosen and therein we are going to proceed further. So, today we were trying to understand how address translation works, to clearly get a grasp of how cache memory works. We need to know what is a physical address. Only if you have a physical address then only indexing into the cache memory is going to work. Our notion of processor is going to generate the physical address is not always practically true, when you work on large virtual space and relatively small physical memory there should be a translation process that has to be applied which will convert your virtual address back into the physical address.

So, this gives a fair idea about what is a role that has been played by operating system. So, this page table is managed by operating system, when you bring something from the disk and put it in main memory under the control of wires, some physical address is been assigned and the most frequently used page table entries are kept inside a special hardware unit which is called translation lookaside buffer. And translation lookaside buffer contains the page table entries of not only one process, but many process that are recently used. Every access into this cache memory is going through the translation

lookaside buffer. So, CPU generates a virtual address it goes to TLB, TLB gives the physical address, split the address, index into the cache and return the data. If it is a TLB miss you go to main memory, take the corresponding entry from the main memory; that means, from the page table put it inside the TLB, update TLB and then continue.

Sometimes in the process a TLB miss can result in a page table miss, that is what is known as page fault so, page fault is serviced and then the remaining process is being done. In this way processor when it gives address, in what way this address is correlating to the address translation mechanism that is from virtual address to physical address. What are the functional units that help the processor in realizing this transition like page tables and translation lookaside buffer? So, the cache memory, the main memory, the address translation mechanisms everything is closely with that together the healthy cooperation of all these is giving us access to the memories. So, with this we conclude today's session.

Thank you.