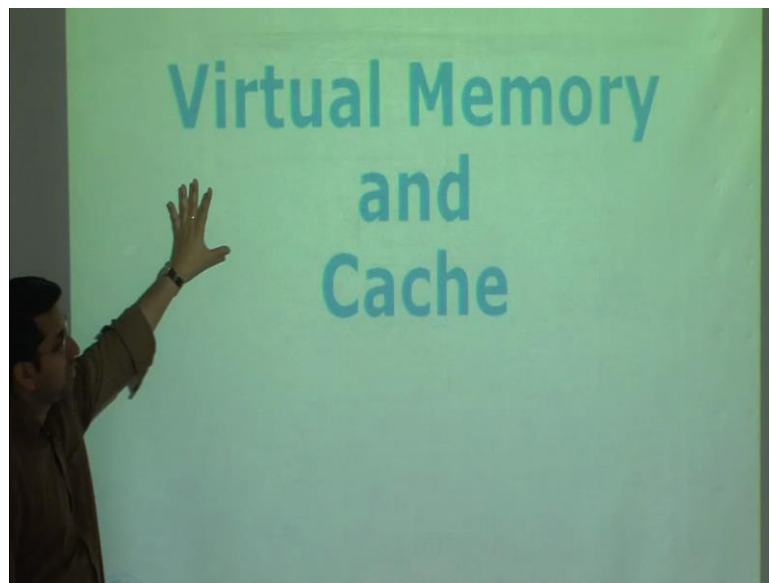


Computer Architecture
Prof. Mainak Chaudhuri
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 25
Virtual Memory and Caches

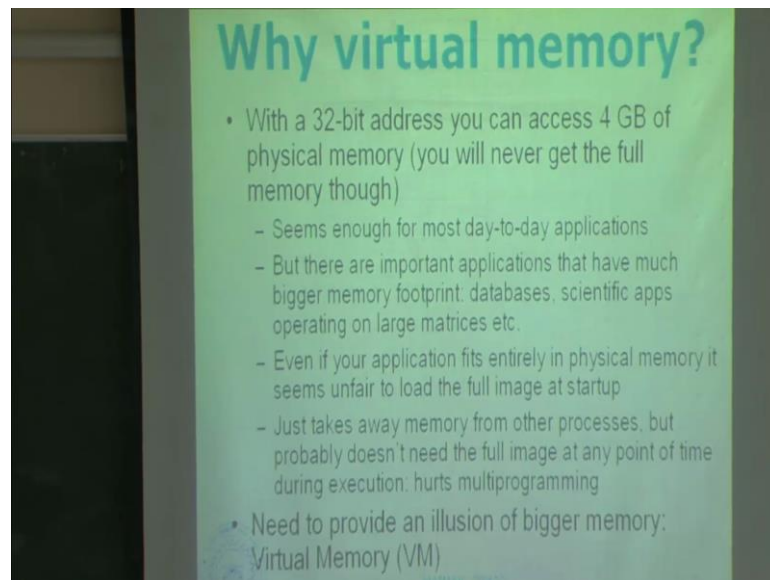
Today, we will start discussing something new. On assuming that, most of you know about these topics at least have seen something in some problem.

(Refer Slide Time: 00:34)



So, the first portion of this lecture is nice will be just recapped. That is, what I think. And then, you get deeper into cache architectures. So, where I talk about virtual memory here, we will not really get into the details of virtual memory that really belongs to an operating system. Here, we are really talking about what a processor needs to offer to facilitate keeping a list of architecture.

(Refer Slide Time: 01:08)



So, why virtual memory, a very basic question. And most of you probably feel that, because if it is an illusion of bigger memory. That is not exactly true. That was not the reason, why virtual memory was invented. So, here is a question put in a different way. In a 32 bit address space, we can access 4 Giga byte of physical memory. Of course, we will not get the full memory anytime, because operating system will have some result ((Refer Time: 01:43)).

But, still even if I assume that, you get 50 percent of 4 Giga byte, that is a huge amount of memory ((Refer Time: 01:50)). And the day to day applications, that you run which probably happy with 2 Giga bytes. So, when the question arises is, why virtual memory right today every machine comes with an operating system, which has an elaborate virtual memory unit or a memory management unit. We will talk on this virtual memory, let us start.

So, virtual memory was not invented. Because, you already get the user of illusion of ((Refer Time: 02:21)). Does anybody know, why?

Student: Multiple.

Multiple exactly, so that was the main reason. So, the point is that, there are important applications that are much bigger than memory footprint. Like, databases, scientific applications, operating on large matrices. However, the point is that, even if your

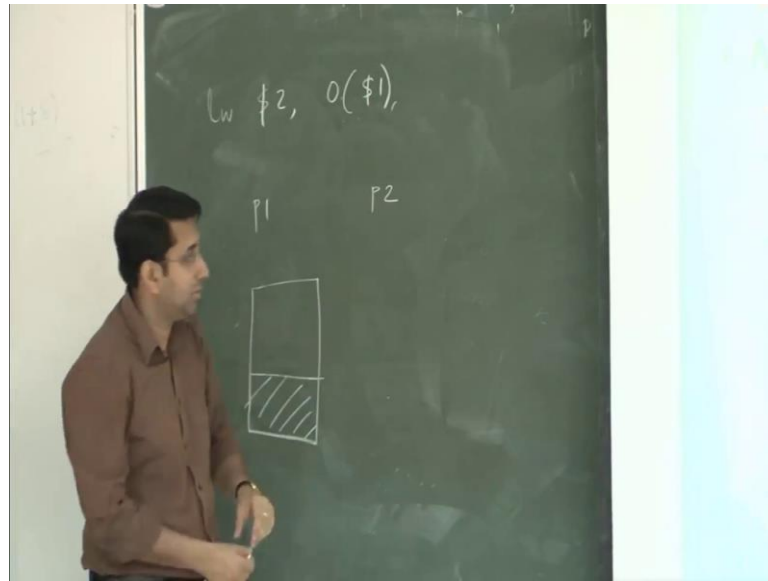
applications fits entirely in physical memory, it seems unfair to load the full image at startup, because none of these applications even these works.

You will require the entire memory foot print at any point in line. It will require the small cache. So, that is the locality principle that we believed in. So, however if you do that, so if you actually tried to load the entire memory footprint that start of, for application. What will happen is that, it will take every memory away from other processors, that could draw in the system. But, probably it does not need the full image at any point of time, during execution.

So, it hurts multiprogramming. So, what is your memory actually allows to do is, to take this particular physical load. And also, if you multiplex is on multiple processors. And that essentially requires a mapping processes. You have a virtual address space that is the application space. We have a physical addresses space. And we want multiple processors to shade these physical addresses. Essentially what this means is that, you take a virtual address of some process.

That will be a mapping of physical addresses. And this the mapping that, virtual memory actually establishes. And this allows you to bring small amount of data from the disk to the memory process. What was that? And then, when you done eventually you gets soon a memory. It gets request by where this some of this only done or some other process is received. So, that allows you to do more multiprogramming. If you did not have got some memory, there will be no way of doing that. A process starts, because of fixed physical address space. And you have to load the data in instructions exactly at the addresses.

(Refer Slide Time: 04:34)



Just to give an example, suppose you have two processes p_1 and p_2 . And if you assume that, the process address space is fixed in the sense that, certain range of memory will be assigned to instructions. Certain range of memory may assign to keep data, certain range of memory may assign to stack, etcetera, etcetera. Then, p_1 and p_2 will probably have the same address range for the instructions. We will probably have the same address for the same range, etcetera.

So, now if I want to run p_1 and p_2 together, there is no to do here. p_1 starts up, it fills up. So, let us suppose the physical point is physical memory. And let us say that, the lower portion of physical memory is deserve for code. So, p_1 starts up ignores the entire code. Keep fills up the code area of the memory. Now, when p_1 gets context switched out to be a part p_2 , I have to remove this entire data from memory to disk.

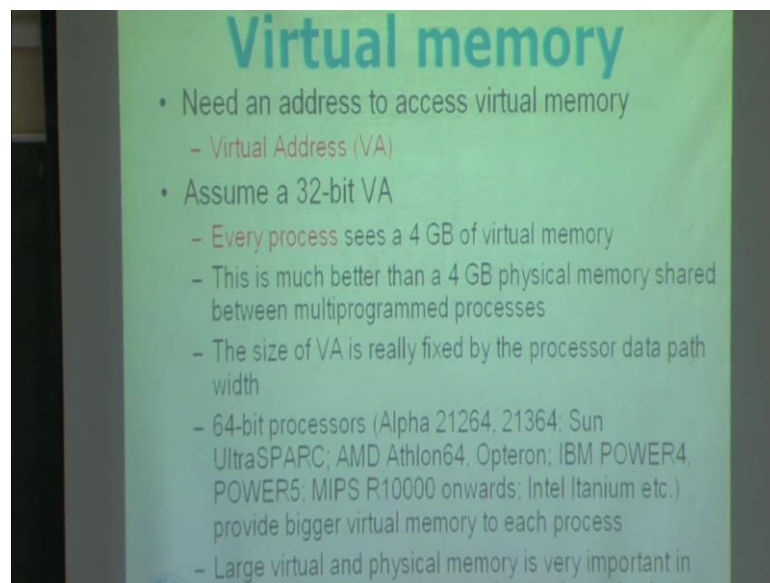
Even though, may be this entire portion is actually free, I have no use in that. Because, there is no way to translate this address into this address. It is a fixed address. So, that is why it hurts multiprogramming, because to round p_2 now you have to remove everything back to disk. You bring the data instruction of p_2 into here. p_2 transfer a while and then again it raise the context switch. Again you have to do the same thing for p_1 .

So, every context which takes a lot of time now, because you are removing the data in instruction of one process, making the data instruction into another process and do the

same thing. Even though, you may have a huge amount of memory left on use completely. There is no use. So, virtual memory allows you to do these things. And that is a main reason, why we do not. It is not that but, yes it has in certain implementations.

However, this is not always true. You can easily desire a machine, which has a smaller virtual at a space perfect compare to the physical address space ((Refer Time: 06:41)). Any question on this?

(Refer Slide Time: 06:55)



So, virtual memory since it is a some form of a memory. It needs an address, access that. and that is called a virtual address. So, if you assume a 32 bit virtual address, every process will see 4 Giga byte of virtual memory. And this is much greater than the 4 Giga byte physical memory share, between multi program processes. This now at least process gets an exclusive 4 Giga byte virtual memory.

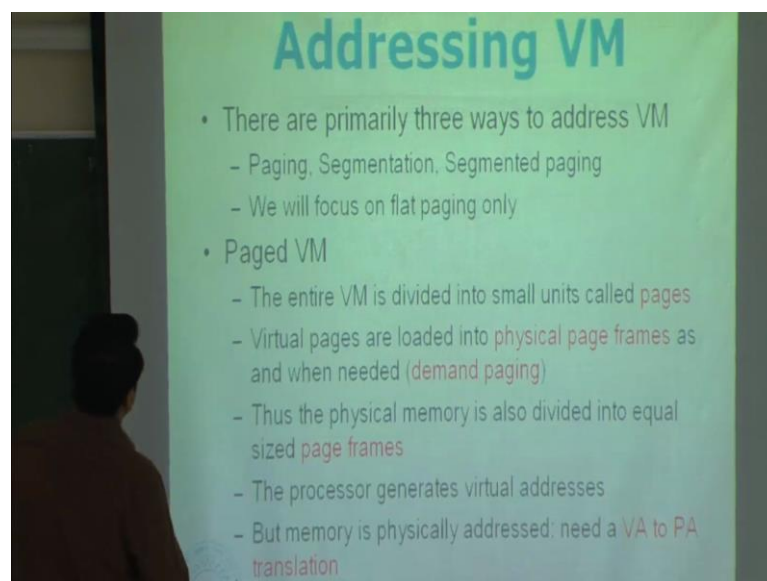
But, when you draw this of course ultimately, this is what is going to be used to multiplex between multiple processes. The size of virtual address is really fixed by the processor data path width. So, that is the only thing that decides how big the virtual access going to be? For example, a 32 bit machine you cannot generate a virtual address which is bigger than 32 bits. So, there is no need to go beyond 4 Giga byte of virtual memory in such processes. We can see that, how virtual addresses get generated? ((Refer Time: 08:00))

For example, when you makes, when you say something like load ((Refer Time: 08:02)) dollar 2, 0 dollar 1. Whatever this gets completed to, this is the virtual access. So, that is what the processor is going to generate. And its width is decided by the width of this register which is 32 bits and 32 bit processor. So, 64 bit processors like the Alpha liner processors which were there longer there. Sun Ultra SPARC also not there anymore.

AMD Athlon 64 onward, Opteron etcetera, IBM POWER4, POWER 5, 6, 7 MIPS R 10 onwards, Intel Itanium and todays all Intel processors. Provide bigger virtual memory to each process, because potentially if it have a 2 to the power of 64 bytes of virtual address space. Although, normally you will never get 64 bits, because there are certain space deserve for certain segments and all these things. It will probably small bit.

Large virtual and physical memory is very important in commercial server market, because there you need to run large databases. So, they are important actually. But of course for day today applications may be that does not matter. So, if you have really a new machine for everyday applications, you probably can do away with your virtual to physical memory, certain aspects on the virtual and physical memory translation layer. The only thing that you need to support is multi programming important. We do not have to worry about the address spaces size and all these things.

(Refer Slide Time: 09:52)



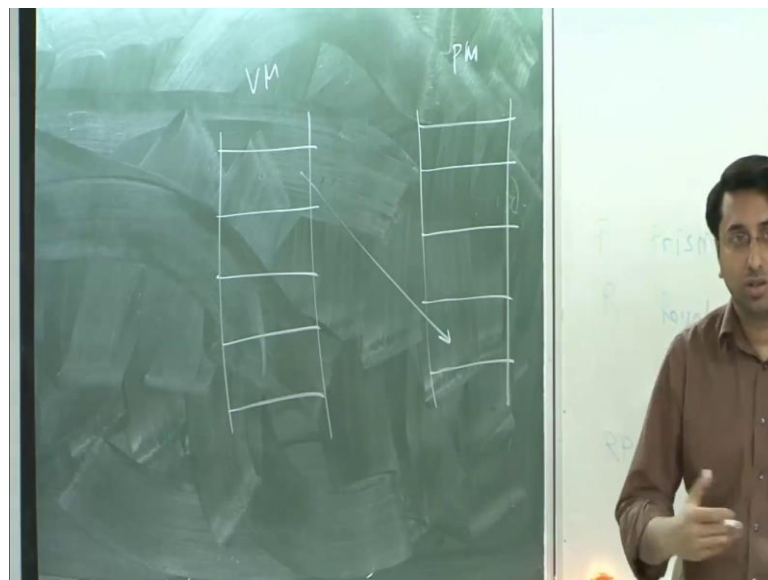
So, how do I address the virtual memory? There are primarily three ways to address virtual memory. These are called paging segmentation and segmented paging. How

many of you have a thought of paging systems? Paging anybody? Are you heard as a thought of paging? We are not talking about your pagers which you send message. Paging is the counter operating systems. Segmentations, segmented virtual memory and communication of these segmented paging.

So, what we will do is, we will not focus on any these two at all. We will look at flat paging only, because again I trait this is not really a natural virtual memory as such as. I want to talk to you about the architectural support that is need to you. So, what is paged virtual memory? That entire virtual memory is divided into small units called pages. Virtual pages are loaded into physical page frames as and when needed is called demand paging.

Whenever process needs a particular page, it will be loaded from the next level of memory, which is normally the disk into your physical memory. And that portion of memory is called physical page frame. So, essentially we were talking about a mapping process, here.

(Refer Slide Time: 11:18)



If this is my virtual memory and this is my physical memory, I divide the virtual memory into pages. I divide the physical memory into equally sized physical page frames. And then, if I need this particular virtual page, I looked up here. Find out a hole and I will put that data here. For example, may be this is a hole that we can go that. So, physical memory is also defined into equal size page frames. The processor generates virtual

addresses. But, memory is physically addressed. So, we need for virtual address to physical address translation.

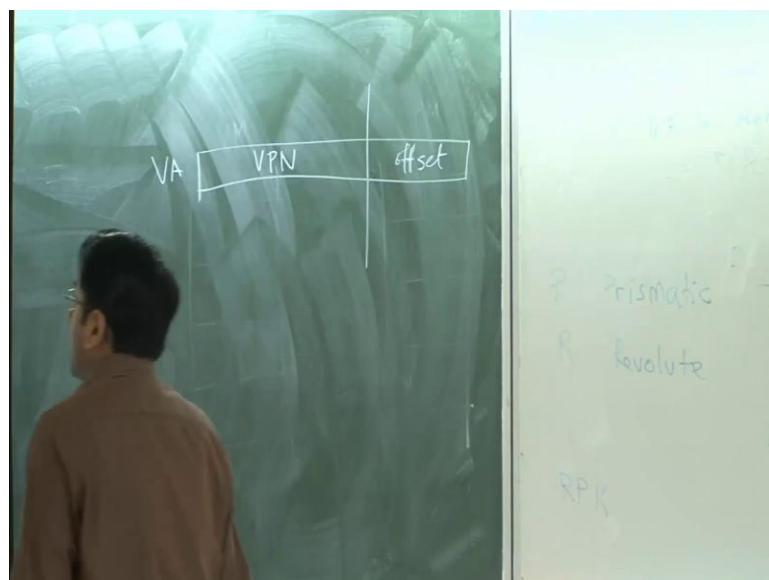
(Refer Slide Time: 12:04)

VA to PA translation

- The VA generated by the processor is divided into two parts:
 - Page offset and Virtual page number (VPN)
 - Assume a 4 KB page: within a 32-bit VA, lower 12 bits will be page offset (offset within a page) and the remaining 20 bits are VPN (hence 1 M virtual pages total)
 - The page offset remains unchanged in the translation
 - Need to translate VPN to a physical page frame number (PPFN)
 - This translation is held in a **page table** resident in memory: so first we need to access this page table

The virtual address generated by the processor is divided into two parts, page offset and virtual page number.

(Refer Slide Time: 12:26)



So, here is an example. Assume that, you have a 4 kilo byte page size ((Refer Time: 12:45)). So, that means I need 12 bits to access any byte, nearly the page. So, within a 32 bit virtual address, last 12 bits will be page offset. That is offset within a page. And the

remaining 20 bits are virtual page number. So, that gives you one million virtual pages total. So, one million times each page is 4 kilo byte. You get 4 Giga byte. The page offset remains unchanged in the translation.

Because, whether it belongs to the virtual space or physical space, the offset within a page cannot change. What changes is the position of page, where you move from virtual space to the physical page. So, essentially you need to ask with the virtual page number to a physical frame number. And this translation is held in a page table, resident in memory. So, first we need to access this page table. So, whenever you want to access any data, you need to do this translation.

Because of, virtual space to the physical space and to do that, you have to access this page table which holds this translation. And how is this translation established? Who established this translation? I started program. How does a translation get established? Who populates this table and how? To initiate the table should get and run by when the program starts and how.

Student: ((Refer Time: 14:33))

How and why?

Student: ((Refer Time: 14:36))

Is that right?

Student: ((Refer Time: 14:44))

Are you so? ((Refer Time: 14:49)) At the time, the bits allocated the populated certain data structures will start assign ((Refer Time: 15:01)). But, when and how with this table get populated? New question was taken in operating system course. We has not taken operating system course. So, let us must have seen this. So, what would be a meaning of full way doing this? What is common sense?

Student: ((Refer Time: 15:49))

Here you require. So can you?

Student: ((Refer Time: 15:52))

Fine.

Student: ((Refer Time: 16:13))

Is how,

Student: ((Refer Time: 16:25))

This is how, the page gets populated. You did not say anything about this table. Then, how I do, how do I do, where is the entry allocated is the same. So, every processor gets a page here.

Student: ((Refer Time: 16:38))

When the requirement arises. So, whenever a process needs a page, it will be brought from the disk, allocated a physical page frame and at the time the operating system establishes the translation and allocates in entry table. So, now since we need to access this particular table, before this we can go and access the data. There has to be a way to get to locate this table, where this is the table locate? That is the first question will be asked. So, ask the first question.

Student: ((Refer Time: 17:22))

How did you find the page table? So, you have gone into a new residential complex, actually. You have the central location where you can go and ask. I am looking for this friend of mine, forget which flat he is in. So, I want to know that location is central thing. How to get that? What is common sense here?

Student: ((Refer Time: 18:06))

So, what is that? It does exactly what you are looking for. So, if I know the starting address of the page table up and down, are is that belong?

Student: Yes

Why?

Student: Sir, when you need an offset and all we can get to the ((Refer Time: 18:24))

Which offset?

Student: The page number offset.

So, we have the virtual address. So, let us assume that you know the starting address on the page table. So, this is my page table.

Student: Using the offset you can

This offset?

Student: Yes, this offset

What is this offset?

Student: This offset gives the offset with respect to the page, next page

No. That is not the. What is that offset supposing?

Student: Offset within a page

Offset within a page. So, let us assume that we know the starting address locates tables.

Student: Using the map to the number and then the page length of page, you get the address.

No, I am not going to locate at this point. I just want to locate the page table country in this table.

Student: We can add the VPN

Add the VPN. Well, not exactly the VPN.

Student: VPN ((Refer Time: 19:14))

VPN multiply the size of whichever,

Student: ((Refer Time: 19:17))

Exactly,

Well, how do you ((Refer Time: 19:20))

Student: Page size, particularly the number of pages and all that

Number of pages

Student: The number of where to address the pages

No, what is the meaning of that?

Student: Sir, here 10 percent where ((Refer Time: 19:34))

So, what is the meaning of a sentence? Tell me ((Refer Time: 19:37)) for the page

Student: Sir, what was the question?

((Refer Time: 19:41)) good.

Yes, that is the right question was. So, question was how I find out the page table entry correspondent to these virtual page? So, that I know the translation.

Student: Should be the index to the VPN and then you have this register?

Is it a VPN entry? You are assuming something.

Student: By putting a van page ((Refer Time: 20:03))

Here is only van pages. Yes, that is true.

Student: ((Refer Time: 20:07))

No, forget about that. Nothing, that is way probably. We will not even talk about that. You are almost correct. But, you are assuming something in you and saying. That starting from this, go down by a VPN amount. And this is the entry, that I am looking for that has an assumption ((Refer Time: 20:30)). What is this entry contain?

Student: The address

What address?

Student: Counter page, the page where

What address? You have to very

Student: Physical address

Physical address or the physical frame number, I said in the offset does not change in the translation. What does this entry contain?

Student: Physical frame

Physical frame number, anything else?

Student: ((Refer Time: 21:13))

Sorry,

Student: ((Refer Time: 21:15))

Offset does not change. It cannot change. I know page in the virtual address space. I have the same page in the physical address space. See, if I am looking for within the page, the offset here will be exactly same in the offset here. That cannot change. What we change is the location of the page. And that is what your physical page per number is, I need that. What else is required? So, you mentioned about a valid bit or something?

Student: If it is valid bit, if I tell the valid bits ((Refer Time: 21:46))

No, I am not trying to interpret the concrete. Understand, what are the contents of the entry? There is no valid bit. There is a physical page frame number. What else? What is common sense? It is all over a common sense. It is nothing great about it. What else you need here?

Student: ((Refer Time: 22:22))

Some, no wait ((Refer Time: 22:28))

Can you elaborate that?

Student: ((Refer Time: 22:34))

Exactly, so there are some replacement same bits. This, what else is needed in the page table entry?

Student: ((Refer Time: 22:59))

That is the valid page. When the translation is valid in the pages in your virtual pages valid bit. Do you need any permission bits? Needs you have? You heard of that? Need some permission bits, need the high permissions. Need the high execution about all are problems. And there is one more bit that is needed and that is coded with and 30 bit. We signified the pages qualified, why is that needed? Somebody, can you think of any use of that page?

Student: When your, if some other process ((Refer Time: 23:56))

Exactly done, so that total bit tells you when the page gets replaced. But, the conference should be little back to that. So, there are several things in this particular entry. So, remind back to the previous question. Is it enough to go down by VPN entries here? VPN what? VPN number of entries or VPN bytes.

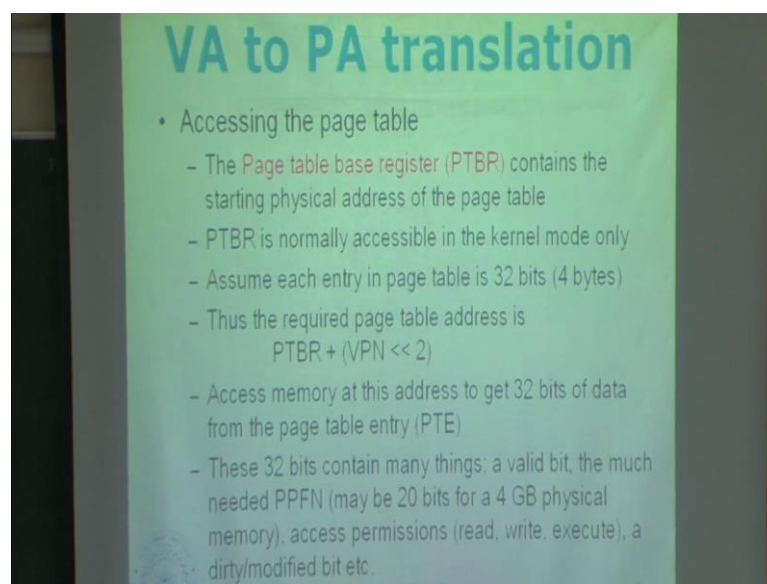
Student: ((Refer Time: 24:28))

VPN entries, correct. So, if I want an address in terms of a byte address. How will I get that? Starting address plus VPN multiplied by

Student: Size of an entry

Size of an entry, thank you.

(Refer Slide Time: 24:56)



VA to PA translation

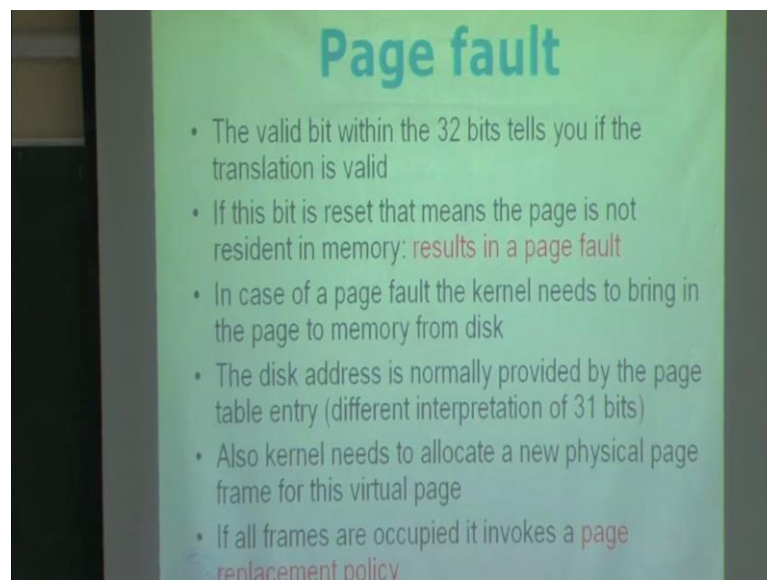
- Accessing the page table
 - The Page table base register (PTBR) contains the starting physical address of the page table
 - PTBR is normally accessible in the kernel mode only
 - Assume each entry in page table is 32 bits (4 bytes)
 - Thus the required page table address is $PTBR + (VPN \ll 2)$
 - Access memory at this address to get 32 bits of data from the page table entry (PTE)
 - These 32 bits contain many things: a valid bit, the much needed PPFN (may be 20 bits for a 4 GB physical memory), access permissions (read, write, execute), a dirty/modified bit etc.

So, the page table page register PTBR, that takes the starting physical address of the page table. PTBR is normally accessible in the kernel mode only. And this is set up by the operating system, when a particular process is loaded within a memory, starting a process. So, you need to know this. There is no other way to get to the correct entry. So, PTBR has to be known and operating system knows it.

So, assume that each entry in the page table is 32 bits. And we have just discussed what these bits might be? So, this gives you that following formula for completing the page table entry address. PTBR plus VPN shifted by 2, VPN marked to other points actually. So, this gives you the byte address, where I should read the page table entry corresponding to this particular virtual page. So, you access memory at this address to get the 32 bits of data from page table entry.

These 32 bits contain many things like a valid bit. The much needed physical page frame number, which could be 20 bits for 4 Giga byte physical memory in the 4 kilo byte page size, because that will have one million pages. Access permissions like read, write, execute and a dirty or modified. And that would be very argues, actually I can find it out. You may need some replacement statements. Then, there will be many other things.

(Refer Slide Time: 26:25)



Now, the valid bit within the 32 bits tells you, if the translation is valid or not. If the bit is reset that means, the translation is invalid. So, that is interpreted as the page is not resident in memory. And that result in something called a page fault. So, we have to do

something to correct that fault. And just to make a correspondence with what we have been discussing, a page fault has to be a precise exception.

So, when a particular load has stored instruction takes a page fault, pipeline has to be meaning that all instructions have to remove. And we have to get the exception come back and restart instruction. When you restart hopefully this time, there will be your page has been broadly removed. So, in case of a page fault the kernel needs to bring in the page to memory from disk. That is what is really needed to be done.

The disk address is normally provided by the page table entry. So, these are different interpretation of a remaining 31 bits. So, you have this 32 bit entry. Normally, if it is 32 bit entry definitely as having a physical page frame number, permission bits etcetera, provided to the valid bit set. If the valid bit is reset in the remaining 31 bits, actually give you a disk address. How to locate this page on the disk?

So, also the kernel needs to allocate a new physical page frame for this virtual page. If all frames are occupied, it invokes a page replacement policy which essentially takes a page. So, the new page. So, page faults take a long time order of millisecond usually. Why just takes an order page faults?

Student: ((Refer Time: 28:19))

B sets ((Refer Time: 28:30)) that is also right. Why the disk is particularly slow?

Student: ((Refer Time: 28:41))

These are mechanical devices. So, if you rotate the disk, you fit the head to the right point. It is a mechanical process that is very slow. So, you lost your hard of the sick time and all these things, to the disk. So, in the point is that of course that means, the page replacement policy must be intelligent, because you know want to take a page fault to off which means, you have to be you must request some page which is not likely to be used in their future. That is, what you need?

So, again I am not going into detail of this, because this is not the topic of this course. I will just assume that in the black box algorithm, which will replace some page that is it. So, once the page faults finishes, the page table entry is updated with the new virtual page number to physical page frame number mapping. Of course, if the valid bit was set

to start with, you get the physical page frame number right away without taking a page fault.

Finally, the page frame number is concatenated in the page offset. This offset to get the physical access. So, these place the virtual page number by the physical page frame number that is it. So, this is the physical address that the processor can use now, to issue a memory request to get the necessary data. So, in summary you really need two memory accesses to get a piece of data, because first memory access is to get a translation one and then to access a page frame.

And once we have the physical address, then you actually... Then, you go and access the data. So, can you prove this because, it looks right. If there is one piece of data, you have to write two memory access several time.

Student: ((Refer Time: 30:33)) the recent translations translation to anything else

Exactly, so there is the name of this function.

Student: ((Refer Time: 30:42))

Sorry.

Student: ((Refer time: 30:45))

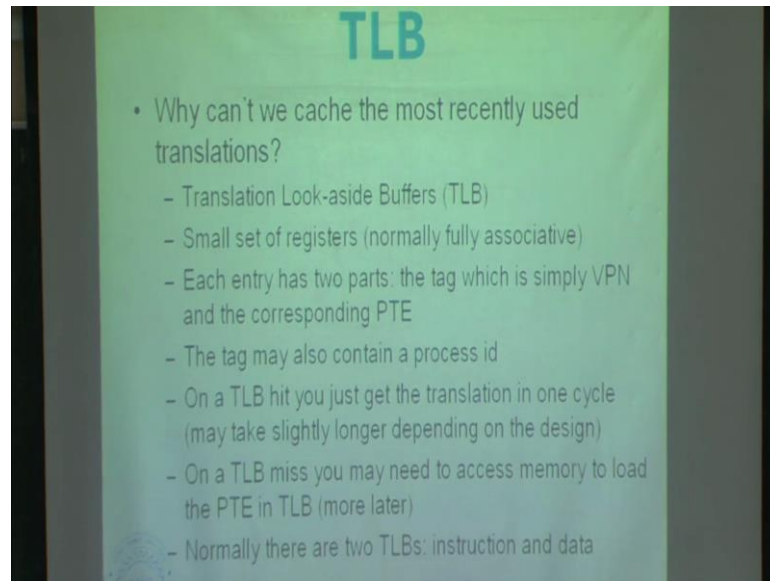
What is that stands for?

Student: Translation.

Translation looks aside buffer.

Thank you.

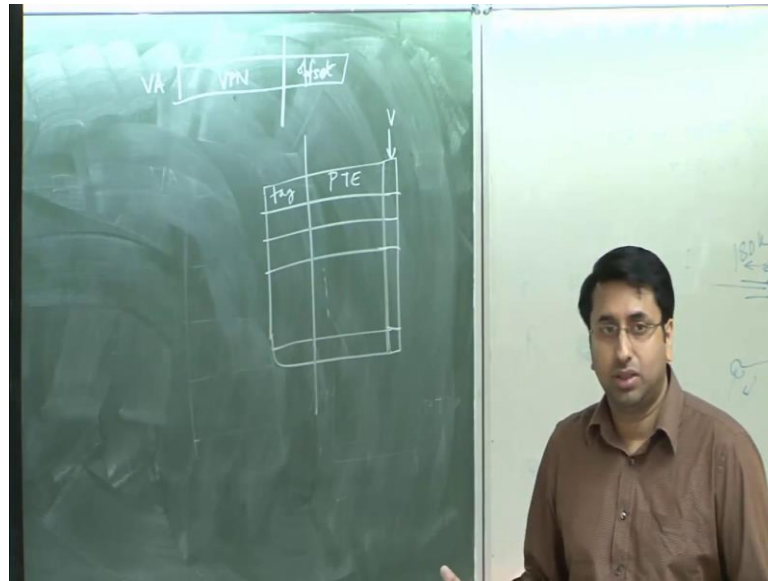
(Refer Slide Time: 30:56)



So, can here if you apply common sense that tells you that, you should be able to cache the most recently used translations. And this particular cache has a special name, it is called Translation Look Aside Buffer or TLB. It is essentially a small set of registers, which store the most recently used translations with these registers. And normally, the TLB's fully associative. But, it could be an entry. It could be a set of same in structure also.

So, it fully associative what I mean is that, whenever you have a virtual address rather than a virtual page number. And you are looking for the corresponding translation. We have to search to all these registers. So, it could be anywhere, this particular translation we provide. We search through all these registers to find out, where the translation is. You may or may not find it actually. So, each TLB entry has two parts. It has a tag, which is simply the virtual page number. And it has a corresponding page table entry.

(Refer Slide Time: 32:12)



So, TLB looks like this. It has one geometry. And this is one translation register. And each register has two pins. One is a tag, one is a page table entry. Of course, we probably have a valid bit also and if I have replacement statements. So, what happens is that, we have a virtual address. Bring here an offset. So, what I want to know is, given that VPN tell me the corresponding page frame number. This is the page frame number.

That is, what is the required requirement for the translation process? So, what you do is. As I mentioned here, the tag is the VPN. So, you store the VPN's here. So, we take this VPN, compare against all the tags. At most all will match, whichever matches you look up, you take the page table entry of that one and that is, that gives you the translation. If nothing matters, that is called a TLB miss. And if now, you really have no option but, to go and access the page table entry.

So, ((Refer Time: 33:38)) each entry has two parts, the tag which is simply the virtual page number and the corresponding page table entry. The tag may also contain a process id. Why?

Student: Shared memory ((Refer Time: 33:58))

Shared memory.

Student: ((Refer Time: 34:02))

What?

Student: Sir in case of shared memory.

Come again, in case of shared memory without any shared page.

Student: Shared page.

Two processes then,

Student: Process id might be required to valid data. This process can.

There are permission bits. Why you I need a process? I need he says, the type may contain. That means, it is not the requirement under what circumstances you require process id in tag. So, we will say what I am saying is that, I will extend my VPN by the process id as well. So, I will have another fill here which is a PID. And what I will do is now, my TLB hit process will change a little bit.

I will not only compare VPN with the tag, I will also compare the id of this process with these both must match. Then, I have a TLB otherwise not. Why do I need this?

Student: ((Refer Time: 35:28))

No, TLB is not far process. TLB is inside the processor.

Student: ((Refer Time: 35:33))

What?

Student: Any process can accept this TLB?

Right.

Student: It can access even page fault

Ok.

Student: It can ((Refer Time: 35:49))

So, you are saying that, a process was coming as cached certain translations and given that the virtual address space of all process is identical. Another process may come in

later. Generate the same virtual page number. Look up the TLB. Pick up a translation and can go and access the page. But, yes you are correct absolutely. And to prevent that, you need a process id. Can I do away with the process id extremely correct? But, I tell you that, it is not really required. I can do something to keep it up to get my process id as well, which is why it says may?

Student: ((Refer Time: 36:44))

No. It cannot be there. Virtual addresses are generated by compilers. That you cannot do. Is everyone following, what he is talking about? So, I am saying that, yes that is correct. The problem that he has mentioned can be avoided by having the PID. But, I am saying that, I will take picks in the other way also. It has to be between two processes. There is no problem with the single process and between the two processes.

Between running these two processes, there has to be some 144 event that has to happen. What is that called or context which has happen? And I do some extra work here, we got it.

Student: ((Refer Time: 38:00))

Exactly, I can just evaluate all the ((Refer Time: 38:04)). So, that is not a table flash. So, on next week if I take class clearly, then I would not require the PID. What is the down side of it? What do you do side by? Do you anyone know anything? Of course, I gain back something, then come to store it. I told you to store the PID input. I simplify the TLB hit also. What am I losing by the ways?

Student: You need to repopulate the table.

I have to repopulate the table, exactly. So, there by code start effect. Whenever process is now blocking, there is no goal starting. And the bad thing is that, I am doing this is in a very conservative way. I am assuming that, there will be a convent which will all actually do. So, that is why if you do not have a process id, you lose in terms of performance, because every process will see a goal started hit.

Whenever we switched in here, we are having a process id. You can simplify. Can give you bad, the performance down side is that your TLB becomes bulk view? So, that is all because, this process id's are not very small id's. These are large numbers. So, on a TLB

hit, you just take a translation in one cycle, may take slightly longer depending on the design. On a TLB miss, you may be to access memory to load the page table entry.

Let me say, again there is a may here. Shall we guess, what else can I do? So, I will miss the TLB which means I did not find this virtual page number, anywhere area of these entries. That the obvious solution is to go and access the page table and get a translation. I am saying that, I can do something else also which is why there is a made. Here, can I store the translation somewhere else? I look up the TLB first, I miss the TLB.

And immediately before going to paste him, can I look after somewhere else, anyway. So, the answer is that, nothing stops me from catching the translation in the cache in my data cache for example, because this is just a normal data. It has an address, it has a value. So, I think I just put it in the cache. So, I miss the TLB and go by cache. If this is not there, then of course I do not. So, we talk about this one later to be almost all processors actually do this.

They put the translation in some level of the cache. So, we will talk about that. And normally there are two TLB's, instruction and data wise. Why is that? Why not is given? Your TLB can be keep you bending utilization.

Student: ((Refer Time: 41:07))

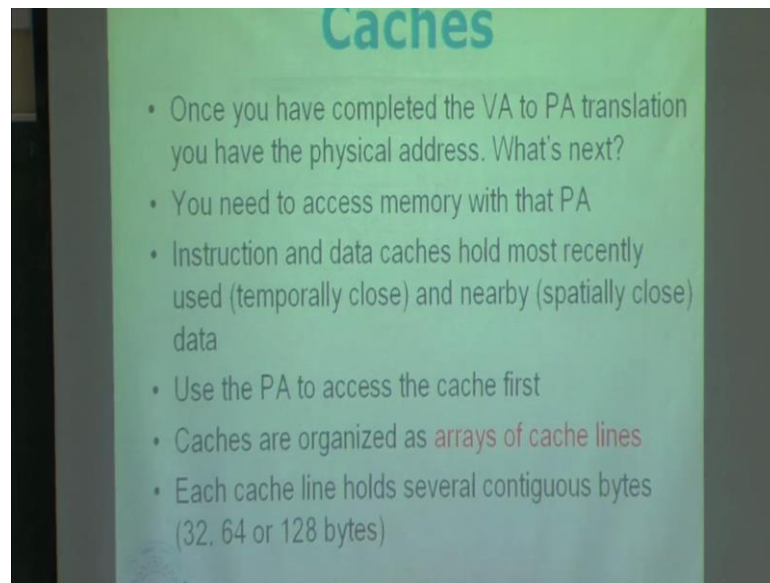
Sorry.

Student: To avoid the structural hazards

Exactly to avoid structural hazards, because your pipeline processor have a single cycle. I may have to access the instruction TLB. I may not access the TLB for instruction as well as for data as soon. I will fetch in the memory stage, it will be aligned to save cycle. They would not be in the pipeline processor, this structure has a... So, another is that you have a single TLB in which two access force. I will allow the access to the instruction later.

So, but anyways today all processors actually have separate instructions. So, once you have completed the virtual address to physical address translation, you have the physical address what is next?

(Refer Slide Time: 41:48)



So, you need to access memory with that physical address. But, you actually do not directly access memory. You first access your caches. So, instruction data caches are essentially a small memory structures, inside your processor. That hold most recently used and nearby, that is spatially closed data. So, what is this mean? That means, if I am accessing a forty data points, I assume that I will require this data point again in future.

So, it makes sense to cache. So, that is called temporal locality. And that is, what is being by the caches. And also whenever I access a particular data point, I usually bring in subsequent few data points together, because assumption is that in some accessing this data point. I will probably need by data points also, that is also spatial locality. And so, this is what the cache is build these two. So, we talk more about this.

How actually we achieve this? So, use the physical address to access the cache first. Caches are organized as arrays of cache line or cache blocks. Each cache line holds several contiguous bytes, 32, 64 or 128 bytes.

(Refer Slide Time: 43:19)

Addressing a cache

- The PA is divided into several parts

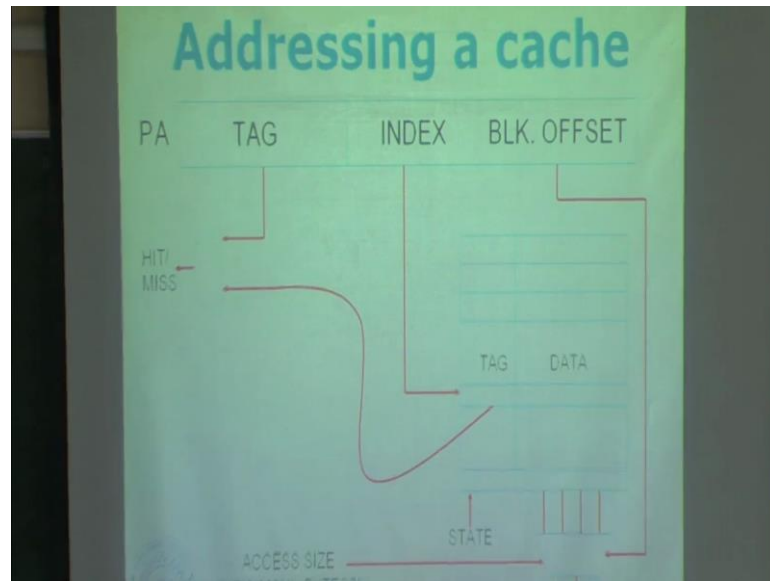
TAG	INDEX	BLK. OFFSET
-----	-------	-------------

- The block offset determines the starting byte address within a cache line
- The index tells you which cache line to access
- In that cache line you compare the tag to determine hit/miss

So, how you address a cache? The physical address divided into several parts. Usually, you tag index and a block offset. The block offset determines the starting byte address within a cache line. The index tells you, which cache lines to access? So, remove that cache within array of cache lines. So, this index tells me which cache line, we should have to access?

And this one tells you, within the cache line which bytes I should access? Where my access should start? Put that cache line, you compare the tag to determine hit miss. So, this is what it looks like.

(Refer Slide Time: 44:04)

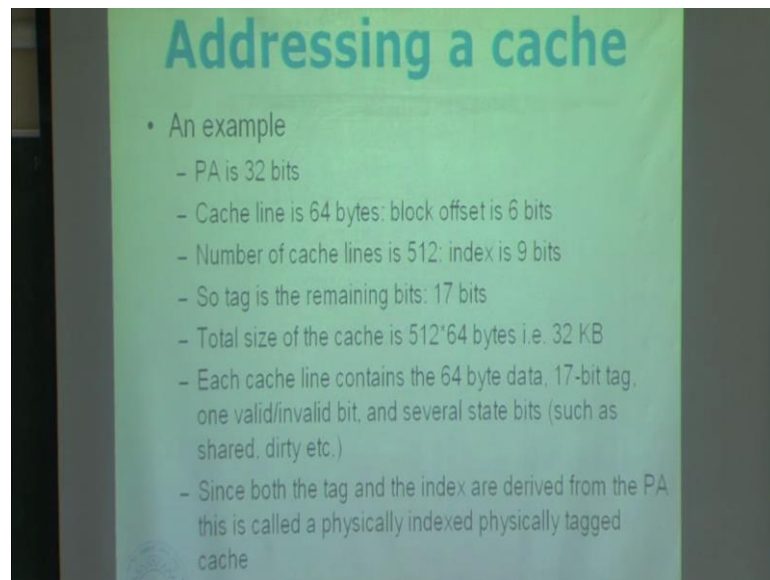


So, you take the physical address. Divide it into three parts. And the index tells you, which cache line to access. This is my cache and you have cache lines. Each cache line has two things, tag and the data. So, the index tells me which things to access? I take the tag, compare it with this tag. When the comparison passes, then only I pick up the data not other. So, if I pick up the data, then the question arises.

This is the large chunk of data and you are saying, that is 32 bytes 64 bytes and 128 bytes. But, it will look at the miss instruction that we have misses it. These are 32 bit processor in the at most on my 4 bytes. You all know, what is 4 byte access? Question is which 4 bytes I should access? Which is a lower body structure? So, block offset tells you, from where I should start within this 32 bytes?

And my instruction tells me, the access size that is how many bytes? If it is no word, 4 bytes if there is no have, it will be 2 bytes. If there is no word, it will be 1 byte. So, starting from there, I will access those many bytes and that is what is my final data, that is has to the process. There are some state bits as well, which tells me for example, valid invalid and many of the things which we will discuss very soon. So, this is how the cache was? Any question? So, let us take an example.

(Refer Slide Time: 45:36)

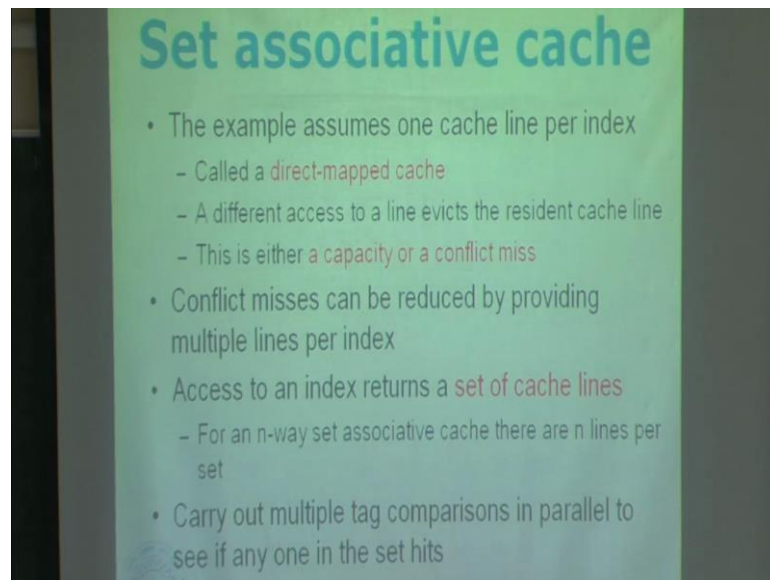


So, let us suppose that we have a 32 bit physical access cache lines. Let us say 64 bytes, which means a block offset is 6 bits. With 6 bits, I can represent any starting point in the 64 bytes. And let us assume that, the number of cache lines is 512 which means I need 9 bits of index. Decide which cache line to access? So, remaining 17 bits are going to be tagged. How would I get that? So, I have 6 bits of block option, 9 bits of index some. Let with 17 bits out of 32 bit address.

So, what the size of this cache? I have forward to a line. Each lines 64 bytes. So, the cache size is this much, which is 32 kilo byte. Each cache line contains the 64 byte data. So, I save the line 64 bytes in it, 17 bits of tag. What valid invalid bit? And several state bits, that the shared, dirty, etcetera. So, we will probably not talk about this particular bit at all in this course or little bit. This one tells you, if this cache block is modifiable.

Since both the tag and the index are derived from the physical address. This is called a physically index physically tagged cache. So, we will see other variants very soon. So, essentially what we do is, we will take the physical address in this, 17 bits of tag go to this particular index, whichever index you need to access. Cooperate the tags with a localization passes. It is a cache hit. We access the data. Otherwise, you have to do an access data from the ((Refer Time: 47:19)).

(Refer Slide Time: 47:26)

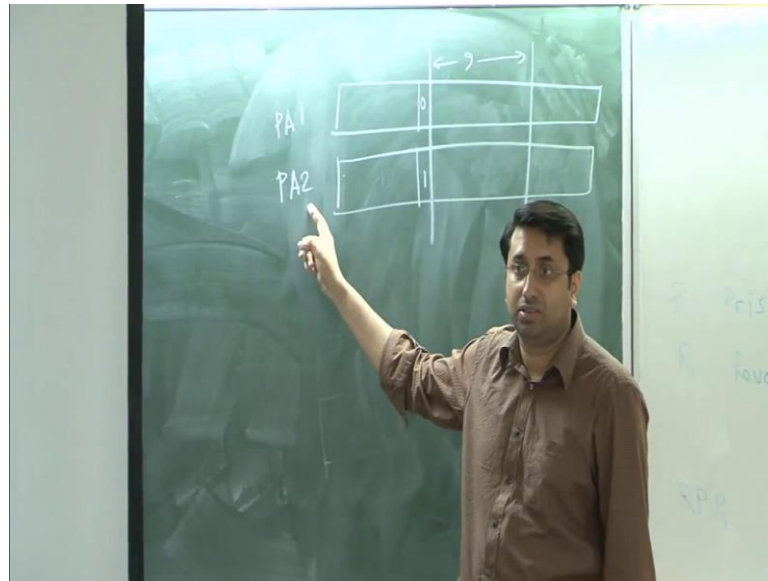


So, the example assumes that you have one cache line per index. But, it does not happen essentially. So, this is called a direct mapped cache, which maps to a particular index to one unique cache width. The problem here is that, a different access to a line evicts the resident cache line, because if we have that there are two physical addresses with the same index base. If you have and then, what will be a problem? The problem is that both of these cache lines ((Refer Time: 48:01)).

We will map to the same index in the cache. Then, there is a collision now. So, only one will find out, other has to be replaced which is not very good. So, how do you solve this? The solution is around, so this is called the capacity or a conflict miss. Because, see you have two addresses calling on the same index, which is at any calling time. You can cache from the, wherever you access the other block where you get cache miss.

And cache miss is because of, these are conflict between these two addresses. Constantly, you can involve the set as a capacity, because we have a bigger cache, where these two blocks would map to different in cases. Because, if you look at this example, if I double the number of lines in the cache 1024 lines, then we take in bits now in the caching bits. So, suppose I have two physical addresses, so this is line bits.

(Refer Slide Time: 49:12)

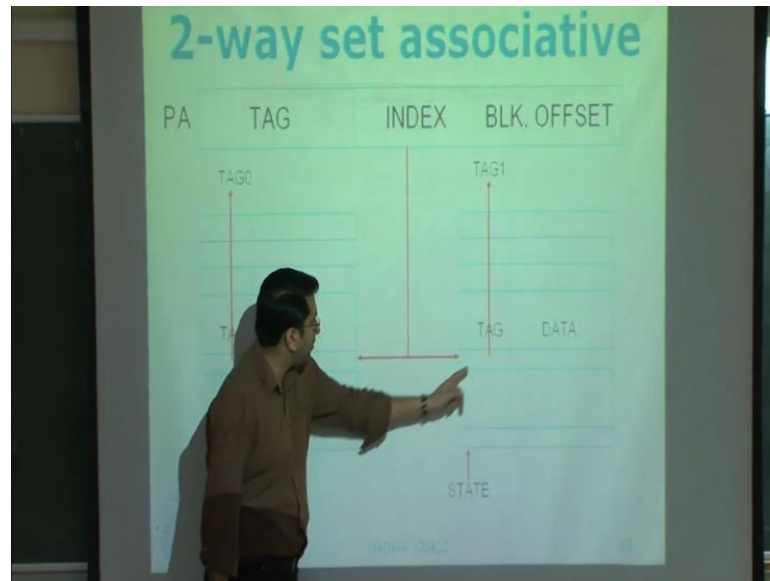


These two portions are exactly same. So, then these two addresses will map to the same cache line. But, suppose it is a 10th bit, here is zero. 10th bit here is 1. So, now you can double my cache. So, suppose I make 1024 lines in my cache ((Refer Time: 49:50)). My index bits will be 10. Then, these two addresses will actually map to different caches and the conflict to deserve it. So, a very difficult to say whether, it is a conflict miss or capacity miss.

So, of course there is a way to categorize them correctly. So, we will talk about that soon. But, for now we just say that the problem is either a capacity miss or a conflict miss. Is this reliable? So, conflict misses can be reduced by providing multiple lines per index. That is one solution here. We have talked about one solution that is, double the cache line increase the cache. The other one is, we keep the size unchanged.

But, you will allow multiple lines per index. So, access to index now returns a set of caches instead of just one cache. For an n way set, associative cache there are n lines per set. And now, you have to carry out multiple tags comparisons in parallel to see, if any one of the set hits? So, here is an example.

(Refer Slide Time: 51:03)



So, this is a two way set associated cache. A particular index corresponds to two different caches. So, which means we access both of these tags, top way both of these against this one. At most, one we match. If none of them match that means, you have a cache miss and whichever matches will be in the data. So, now this problem is gone actually two lines will reside here. One go here and two will go here.

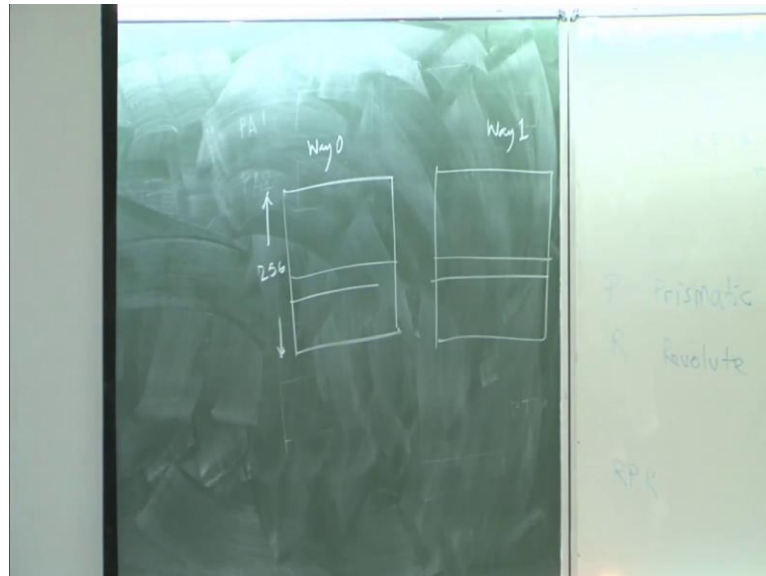
And they will coexist in the cache. So, now when you need to evict a line, that is because if it happen that now, free access that will save index. So, when the third address shows up, we have to make room called may be placing one or the other two. And now, you have option which one is less, which you did not have, so you need some algorithm here to decide which one to replace. So, that is the cache replacement policy.

So, you run a replacement policy. LRU for example, used is a good choice. Keeps the most frequently and most recently used lines, favors temporal locality. So, that is how you reduce the number of conflict. So, we try to convince. So, we will look at some of the replacement policies. But, again I will not go with the detail of that. So, there are two extreme subset sizes. One is direct mapped, which is a one way set of associative cache and fully associative where all lines are in a single set like, the TLB example.

It is a single set, you have to make all comparison. So, here is an example. Suppose, you have a 32 kilo cache, which is two ways set of associative. A line size is 64 bytes. So,

what is the number of indices or number of steps? So, essentially how does it look like, this particular cache?

(Refer Slide Time: 53:22)



So, there are two arrays of cache lines. This is the way 0. This is way 0. So, lines are 64 bytes. So, each ways there is 16 kilo byte here, 16 kilo byte here. So, how many lines do we have? So 16 kilo byte divided by 64 bytes. So, that is basically a 256. So, this is 256. So, you only carry 8 index bits. So, whichever index you get, that will give you two lines. So, that is my set. So, that is 2 a 0 0 32 kilo byte cache.

Another example is, suppose you have the same size, a line size 32 kilo byte capacity in 64 byte lines size. But, it is a fully associative cache. What does it mean? We have number of sets equal to 1. So, all in your cache blocks are in a single set. So, this looks like this, etcetera. So, here all the cache blocks in the single set. So, within the set, there are 512 lines. And you need 512 to tag comparisons for each access.

Here you need just 2. So, that is why fully associative caches are expensive. You need to make a lot more comparisons, because what is a advantage does not be seen. Is there advantage of doing a fully associative cache over the set associative cache? Otherwise, why talking about these? So, in general what happens if I increase the associative between the caches? So, give me mind up, my goal of design the cache is to maximize the number of guess.

That is what I want, because caches are slow. Would will I get that by increasing the associative I do. Because, number of conflicts normally should go down, that is actually not true. It happens only after some point. So, we will get into the detail of that very soon. The curve looks like this, if we plot associatively against miss rate, the number of misses it will look like this. So, there is an optimum point beyond which the number of misses will again start increasing. So, I will explain that very soon.