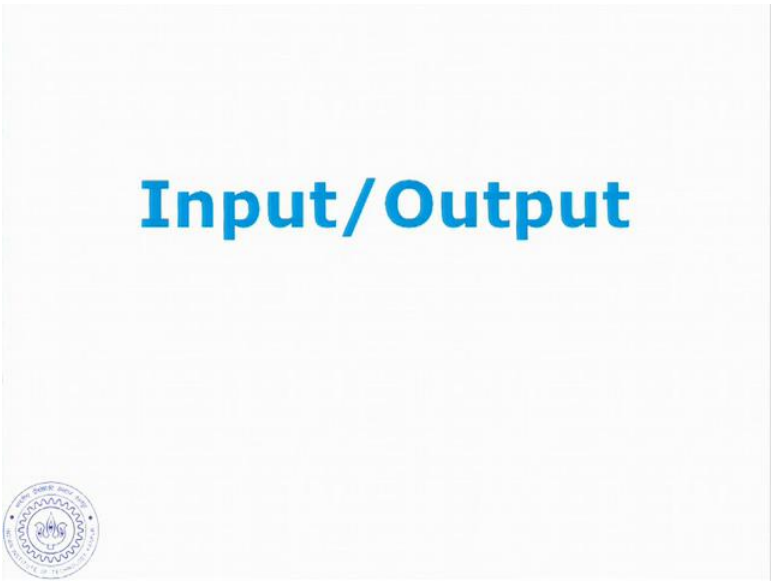
Computer Architecture
Prof. Mainak Chaudhuri
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

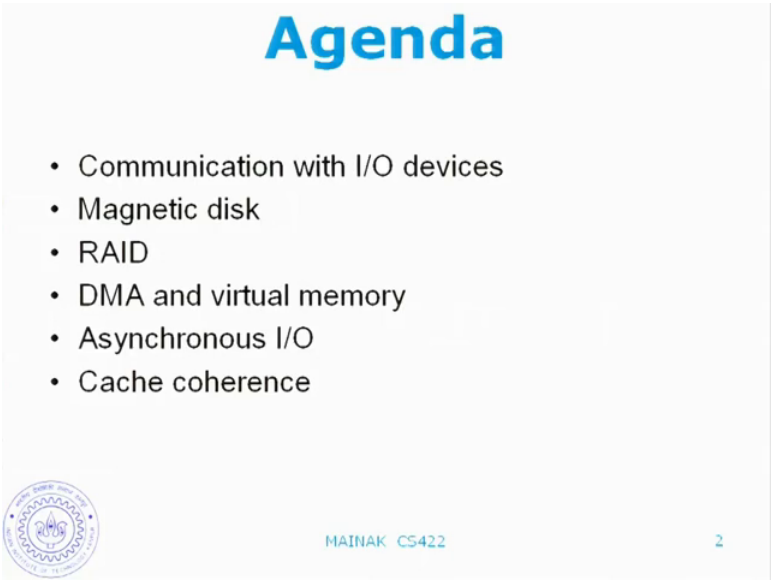
Lecture - 35
Input/Output

(Refer Slide Time: 00:13)



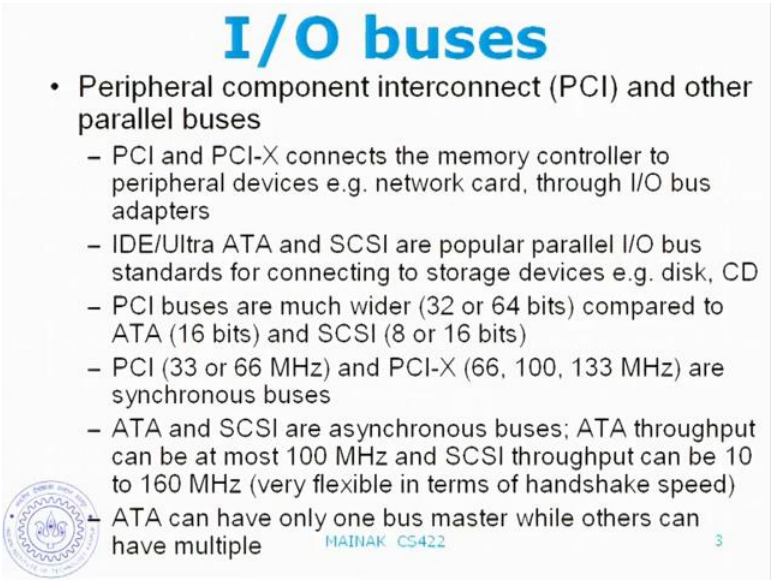
So, today we will discuss little bit about IO. Today our lecture may be a next lecturer. So, I want really go deep in today of the IO devices. Just show you how IO really happens ((Refer Time: 00:40)) with the, on the CPU of the ((Refer Time: 00:42)) on we really talks that pretty much.

(Refer Slide Time: 00:49)



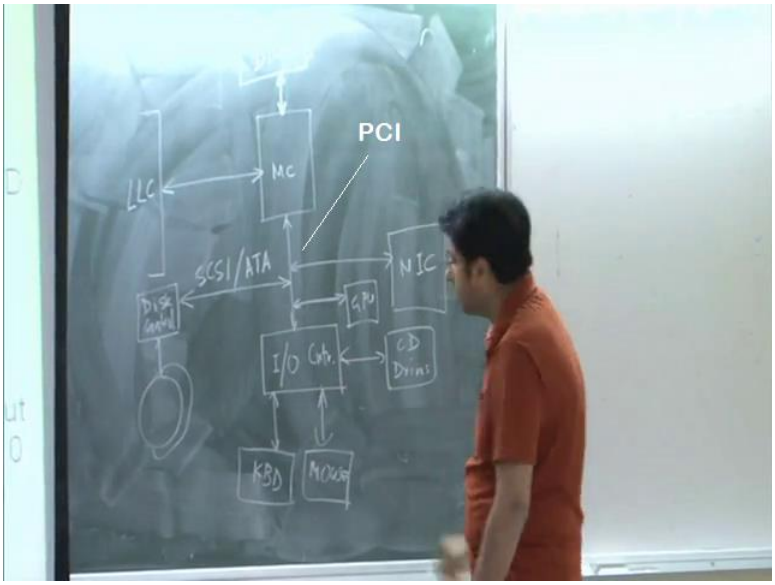
So, here is the agenda. So, we look at communication protocols with IO devices. Talk a little bit about magnetic disk, which is more time device that similarly goes. Talk about how to communicate disks angular dependable. It is you know raid, it stands for redundant array of inexpensive disks. We will talk about direct memory access, and virtual memory. We will see talk about memory, and we will look at one way of I mean IO and how interaction protocol. Talk a little bit about asynchronous IO and cache coherence

(Refer Slide Time: 01:39)



So, little bit about the hardware IO buses. This only slight talks about the standard enormous things, there are many. So, that is really the summery. Here lists we have given some of the common bus that you see. So, you must have heard about the PCI bus, that is stands for peripheral component interconnect. So, it is (( )) before that a particular computer system looks something like this.

(Refer Slide Time: 02:15)



So, let us say this is your last level cache controller. So, you have seen that side very much, we have spent most the time looking at that side, the processor pipeline and the cache and all the sides. And we talk little bit about memory controllers as well. There are the dim cards that are connected to the memory control, and then you have your IO controller. And this is typically the PCI bus, it connect the IO controller to the memory controller, and this also connects to other things. For example here network interface card. It also connects to the to the disk controller, and to the array of disks; that is handling from here etcetera. This one is often this particular bus uses for another protocol like SCSI or ATA. Also it would be handling your graphics from the PCI bus g p u. although nothing stops you from connecting the CPU to the memory controller direct nodes. It used to be the case we have told inter graphics, the a g p connected so on. And your IO controller also had connects to the other IO devices, like an IO device used frequently, that is may be the, not the most important things, and keyboard.

The mouse, CD drives sorry, what the printer exactly yes. So, there is other thing. So, this is what roughly looks like, and after this particular protocol is called the south bridge. It is an it is an old nomenclature. it has originally when that the personal computers came out, this one was called a north bridge; memory controller, and this one was called a south bridge just based on the location of these, but memory controller today is actually, got chosen to the LLC. In fact, this whole 3 will be in the single check. So, after this particular bus as the front side bus, is again an old name coming from the Intel chip sets, when memory controller was the north bridge, it will connect to the processor in the front side bus, but anyway what I really mean is the bus that connects the LLC controller to the memory controller. So, these going to be looks like.

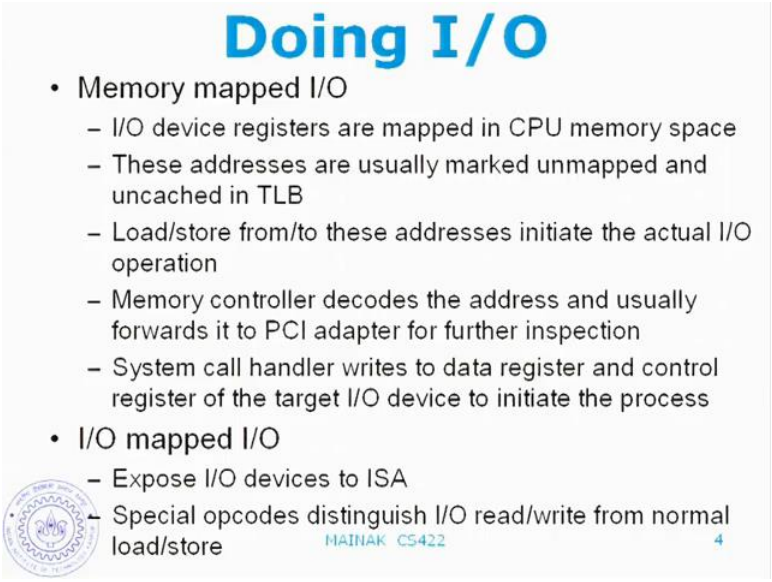
So, the PCI. So, this is the PCI bus, this particular one. So, the PCI and PCI x. PCI x stands for PCI express, it is it is a faster, the next generation PCI bus, personal general PCI. It will tell if the memory controller peripheral devices like network card through IO bus adapters. So, whenever you want to connect, you usually require adapter or a bridge; that is what it call actually, when you are connecting the device to PCI slot of your computer you get a bridge which would transmit the protocol from the PCI protocol to whatever protocol you are connecting to. For example, here you transmit PCI to SCSI. So, here are some buses that you are often used with the magnetic disk. So, i d e or ultra ATA.

So, does anybody know what this thing stands for Integrated design electrons, what does this one stands for; Advanced technology attachment. What does this one stand for; SCSI Small computer system interface. So, essentially these are of your parallel IO bus standards or connecting to storage devices like this can you see. So, you often find ATA comes with 2 2 data, serial ATA like SATA and parallel ATA. You will probably c parallel ATA, because when you say just ATA that actually these parallel ATA, otherwise we would actually mention SATA; serial advanced technology attachment. So, PCI buses are much wider 32 or 64 bits compared to ATA or SCSI. PCI and PCI x are simpler buses, that you can see that the frequency requires PCI is the (( )) it can higher rate, actually pretty much all machines routine which PCI x. ATA and SCSI are asynchronous buses.

So, what we speak is that, these are normally access to an asynchronous action. There is no clock that the synchronizes accesses. So, when this asynchronous we can talk about

to, we can talk about the frequency. So, ATA throughput can be at most 100 micro hertz, whereas, SCSI throughput can be 10 to 1 60 micro hertz. So, it is very flexible, and it also the handling speed. And ATA can have only one bus master, while others can have multiple. So, I will actually stop here about these protocols. It is generally we can go on talking about this with standard definitions, and all things are there. You can look wherever you want.

(Refer Slide Time: 09:05)



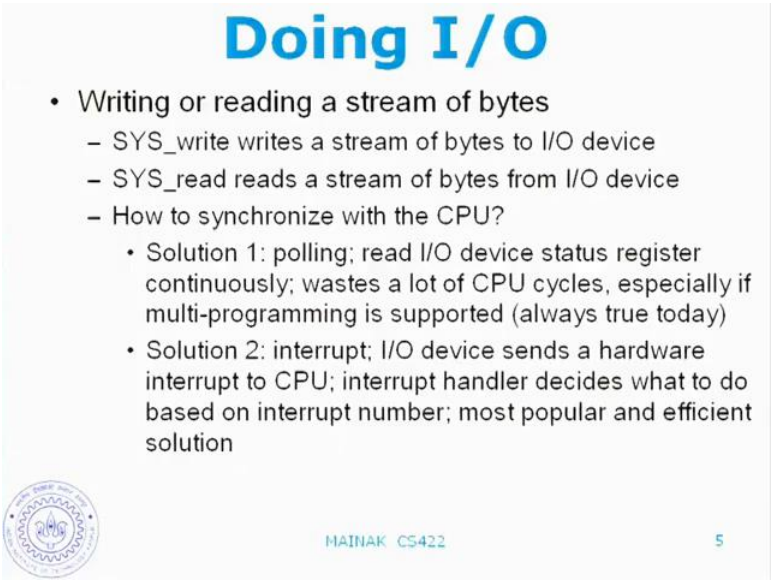
So, let us see how actually we do IO. there are normal two ways of the limit; one is called memory mapped IO and other one IO mapped IO. So, let us look at memory mapped IO first of all. So, IO device registers in this case are mapped into CPU memory space. So, essentially whenever you talk about let us say, speakers command register that will actually have an address. So, we will mapped to given some memory address, which the CPU can directly address. So, these addresses are usually marked unmapped, because these are actually physical addresses not virtual addresses, you cannot translate, and uncached in T L B address. So, these are memory cache for obvious reasons, because where we would not have any rules of these particular locations. You set the command and just find out the bus. And of course, there is a point caching these read write, because we really do not talk device directly. We do not want to put the store in the cache, that is not really done, program that command registers the printer you would send the bytes to the command register to the printer.

So, load store from to these addresses initiate the actual IO operation. So, the faster exactly saying, which will actually starts from the processor pipeline, this load store instructions, but since this is the uncached, we bypass the caches, and directly go on this bus. So, the memory controller will collect them this request. Look at the address and decode, we find that well, they do not belong to this particular space, it will first find the adapter, and will hand over the address to the IO controller, to figure out what to do next.

So, memory controller decodes the address and usually forwards it to PCI adapter for further inspection. And of course, (( )) special find out which device it actually addresses, whether it addresses your the g p u or it addresses your mouse or keyboard, or whatever it is. And these load store instructions would probably part of some system call handler. For example, which I do initiate freed command, you probably be doing some system call which would be evoking this load store operations to initiate the printer. So, system call handlers would write to the data and control registers of the target IO device to initiate the process.

So, for example, to start a printing operation, you would probably send the particular command to the printer, saying that start printing, before that we probably send the data also. So, have to data (( )) and printer will start writing. So, is this clear. So, this is fairly simple, does not require much change in the processor, they will look like load store operations, only thing is that this load store operations when they look up the data t l b for the first time, you know that these are uncached addresses, and these are unmapped addresses. So, that will allow that the bypass the cache, go directly on the bus, and the rest of the thing should be handed over the memory controller, the IO controller.

(Refer Slide Time: 13:08)

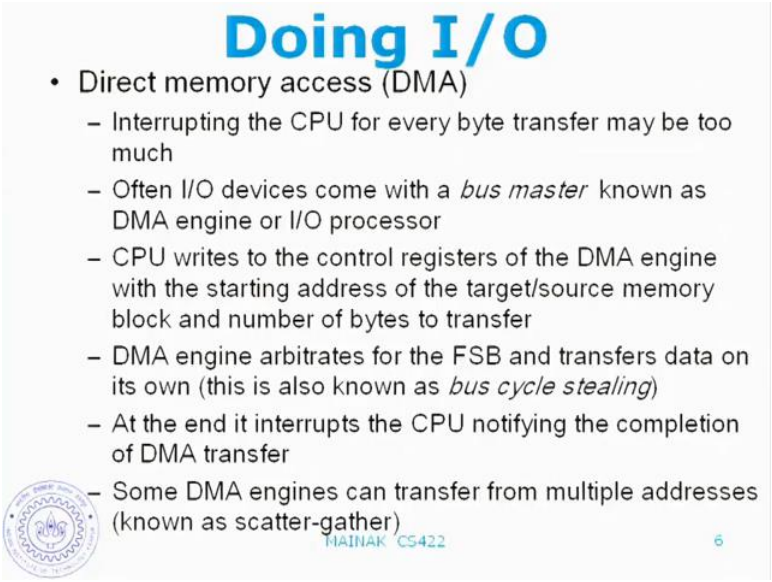


The second type is IO mapped IO this used to be supported in old machines, and it get supported today in certain machines. So, here you expose the IO devices to the i s a. So, essentially what this means is that you get special opcodes that distinguish IO read write from its (( )). Instead of saying load world some value to some address, you probably have a new instruction that would say, you know write this particular word to the particular command register it. So, there will be a different instruction. So, the path will still be same identical exactly, except that the posture will not be look up the t l b to figure out what to do instead the instruction will tell you to bypass caches, and the rest of the things would be same on this side. So, how do you write or read a stream of bytes. So, usually the sys write system call writes a stream of bytes to IO device, and sys read reads a stream of bytes from IO device. For example, when we say scan f at some point it gets transfer it to a read system, from your standard input device standard input stream. Similarly writing on writing on a bit something to a monitor at some point, that will pass to write system call, which would write to your read memory.

So, question now is that how to really synchronize with the CPU, the IO devices are doing certain things how to synchronize with the CPU. So, first solution is to polling that is read IO device status register continuously, to know whether the particular, for example, let us say your readings certain twice you are no way to disturb. So, you will pole certain IO device status register continuously, and find out that that status register will change the status whenever the read continuously. So, that wastes the lot of cycles

especially multiprogramming is supported which is always true today. Second solution is to send an interrupt IO device in the hardware interrupt the CPU. So, what really happens in this case is that, whenever interrupt arrives the program counter will change to something, a constant value, where the first few instruction of the interrupt handler has to be interrupt, and from there you can jump to your chosen something which actually does certain things. So, that already called interrupt certain thing analysis. So, interrupt handler decides what to do based on interrupt number, and this is most popular and efficient solution and normally used with IO device polling is almost never used.

(Refer Slide Time: 14:56)

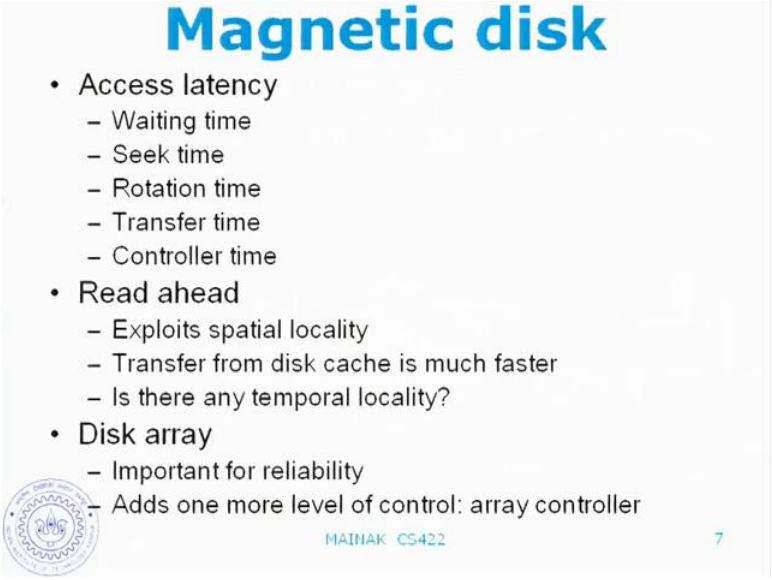


So, often for transferring data from into memory usually use direct memory access, which does not interrupt the CPU. It can interrupt the CPU for every byte transfer may be too much. So, often IO devices come with a bus master known as DMA engine or IO processor. So, how do you do, how do you really operate the DMA engine. So, CPU writes to control registers of the DMA engine with the starting address of the target or source memory block and number of bytes to transfer. So, its specify the source memory address, specify the target memory access and the figure out the DMA engine. It will copy from here to there, when the CPU can notify something.

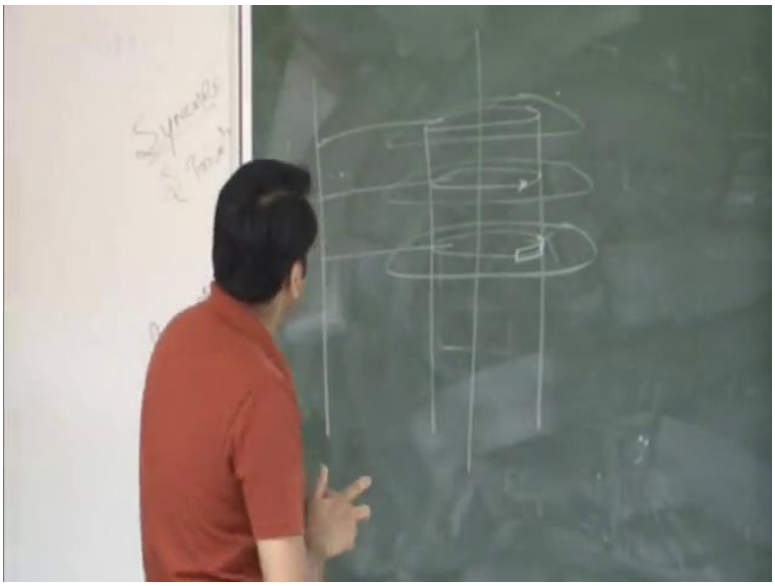
So, DMA arbitrates for the front side bus and transfers data on its own, this is also known as bus cycle stealing remember that this arbitration is needed, because the DMA engine may have to range the caches, to make sure that most data is copied from the

source address to the target address, because certain data may actually decides in the cache on the processor which may not really be here, certain things may actually be on this side. The DMA engine would actually send interventions down the caches retrieve the data, and write to the command. And when it is done it sends an interrupt to the CPU notifying the completion of DMA transfer, and it interrupts something work and that completes the some DMA engines can transfer from multiple addresses also known as scatter gather. For example, you can specify a list of addresses at the source, and another list of addresses at the target. So, it will actually copy words from this address to this address to there, this address to there and so on. So, we can scatter certain bytes and also it gathers some bytes

(Refer Slide Time: 16:52)



(Refer Slide Time: 17:21)



So, what are the IO devices that you often use, it is the magnetic disk. So, the access latency for a disk is normally drop down into this five components; the first one is the waiting time where the request comes and waits in the queue, then you have seek time. So, how does this look like. So, we have here bunch of pattern and if I look at a particular track here, and if I take each of this track on the different surfaces, so each surface will have a head. So, this is normally called a cylinder, that connects not really physically connect, but logically that connects the same track on each of the surfaces, and each track is normally divided into blocks of data called sectors, and this could be for example, types of bytes 1 kilo byte depending on the operation system configures sector size.

So, the first thing that you would like to read a particular data somewhere. So, essentially there are two things. So, suppose you read the data here, so the first thing that you do is, you have to meet the head on this particular circle, and then make sure that the head comes to this particular point to read the data. So, moving the head to be position on this particular circle is call the seek time. So, this particular mechanical assembly will move to make sure that the head is position somewhere on the circle, and after that head does not move what happens is that your disk assembly will rotate to bring this cross under the head. So, that is called the rotation time

And then you start reading their certain transfer time, that that takes to get the data from the disk to the disk controller. And then the controller will take some time to send the data back to your memory to this particular path. So, that is how you copy some data from this to the memory. So, usually among these 5 components, this is the largest one seek time. So, that is what normally topics, because this is the purely mechanical activity that causes your head to move to be positioned on this particular track.

And then solid occupies that need to do one of the common once, is read ahead. So, essentially what we do is that, if you are currently reading this center, you might want to read the next sequential sector as well, and depending on your on your data layout, the next sequential sector may on the next surface on the same track, or may be on the same surface contiguous next center of the same track, depending on how you really layout the sectors and on the cylinders.

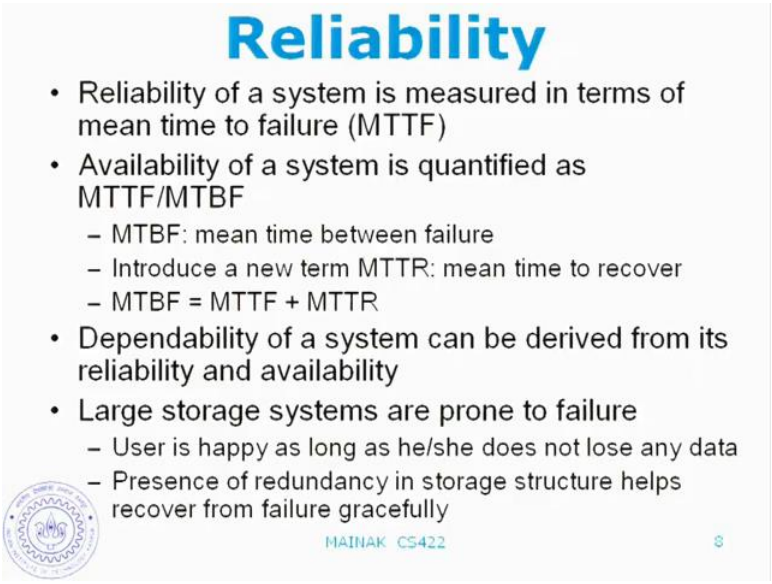
But nonetheless what the idea is that, it exploits spatial locality, and you normally when you read ahead what you reach that, you put the data from disk to an area called disk cache. So, that is these all area is the removing that is configured to be the disk cache. And this controller also has some small amount of cache, but that is not faster, and transfer from the disk cache is normally much faster. Is there any temporal locality cache.

No there is just one cache, let us assume that, we are not really worried about the exact configuration. There is just one block of data. What I am asking is that is there any rules, that step our locality. So, I am currently reading this particular sector; first it will write to the disk cache, and then it would transfer to the to the CPU. So, if I have disk cache inside my disk controller, should I do something smart about the requesting policy of disk cache, bits for temporal locality. Is there are chance that the disk cache is see some news of this particular piece of data. The answer is very unlikely, the reason is that the CPU will be pulling out data from here. It does not talk your data entry, and most likely today is given a memory sizes, the CPU will make full use of that particular sector of data before it takes return from the memory back to the disk.

So, its most unlikely that the disk cache will actually see any views of the locality. So, this is just starting as a spatial locality agent nothing else. It is just trimming in data putting it staging in there. So, that before it sees the next request, it is just nothing else .

So, that is it, as there many other optimizations for improving disk performance, both at the hardware level and at the operate system level. What we will focus our attention on little bit is on disk arrays, because this is important for reliability as you can gets, because if you have an array of disk, you can probably equal to solve the redundancy of the data, so that (( )) I can get the some else; however, this adds one more level of control that is the array controller. It is not top of the disk controller that array controller. It will send commands to the disk controller, the array of disk controller.

(Refer Slide Time: 22:58)



So, what is reliability? Reliability of system measured in terms of mean time to failure. So, essentially how frequently my system fails; that is a measure of reliability of the system. And availability of the system is quantified as mean time to failure, over mean time between failures. And mean time between failure is mean time to failure plus mean time to recover. So, here the hope is that, the recovery time will be small. So, that this number is going to be pretty much, it means that most of the most of the time this system is available, but if the recovery time is large then essentially during the recovery time my system will not be available, which is not good. Dependability of a system, can be derived from its reliability and availability. And large storage systems are prone to failure; however, user is happy as long as he or she does not lose any data. So, that is a primary goal of a storage system, that a user should not lose data. So, that is why reliability is very important. Presence of redundancy in storage structure helps recover from failure gracefully. So, given that storage system stage, it must have some

mechanism to recover data if it all takes. We are going to see some of the redundancy of this.

So, redundancy is used in other places also, it improve reliability and dependability. Not just limited to disk, second is an example from processor remainder I B M 3 90 mainframe offers a good example. So, this system it is an old system. It had 14 processors; some are built in spares to be used in case some processor fails. So, not all of them are all the time use essentially, some of them have spares. Within each processor certain pipeline resources that are duplicated like fetcher decoder reamer functional units l one cache register file. Essentially you have two pipelines, and both the pipelines are computing the same instruction in the anode. Two results coming out of the duplicate instructions pipelines are compared to take field.

Of course, here assumption is that, what is the assumptions? But it is not going to work. I compare two results, they do not match. I thought that there is failure of course, not must have failure; that is pretty of this. I compared pull up them match and could put that there is no failure, is that correct all the time. Why does not it works? In both ways exactly. So, it goes a 5 times field in the same way, it predict the same results, both of which are notb. Although the polarity of that happen is very low, which is why most the time it will work. So, if the results match, the processor state is check pointed, in case the next instruction fails. So, they can roll back.

If the results do not match the processor rolls back to the previous checkpoint which is the previous instruction, and retires the failed instructions many times to see if the fault was actually transient or permanent. So, if really the fault was permanent, then a hot swapping takes a second. Hot swapping means that you swap the failed processor with a spared 1, and that happens automatically that takes the hot swapping, which is actually change. So, that is the time when your system will not be available; that the mean time to recover anyways. So, that was just vibration which shows that redundancy helps in all cases. Could you improve this a little bit instead of having two pipelines, if I give you three pipelines; does it help.

Student: (( )).

No it can recover the two pipelines we can reject the failure and recover its transient failure. If both the factors retain then of course return. You cannot reject of course, there

are question on recovery. The three pipelines what else can you do. We cannot use three pipelines.

Student: Sir how do we know which pipeline is failure.

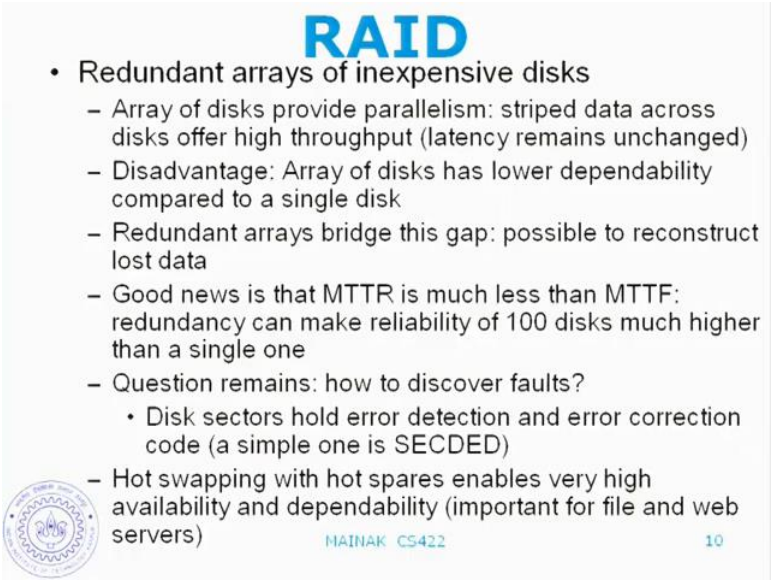
We does not know, diplomacy does it help remember. So, in three pipeline what will I do, I get three runs and then. What if they do not match. So, suppose three results are not attending.

Student: If two of them match then we can restart that.

So, I can take the majority vote, I can take the majority vote. So, if one result does not match with other two, I can still go ahead and assume that the other two are correct, and take the majority vote, but the question is, what way does it help operate to two pipelines. I am giving you more redundancy, does it help.

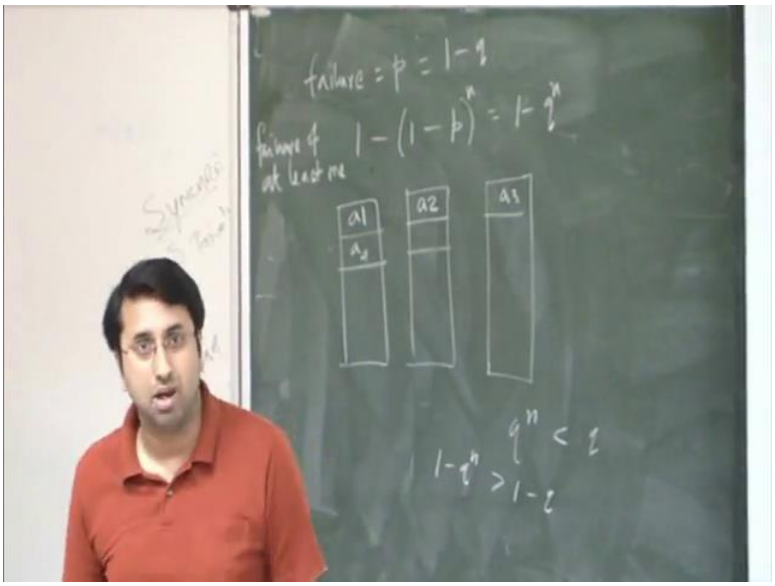
Student: It is saving our time. We do not have to re-execute the instruction.

(Refer Slide Time: 29:31)



So, in this case if one phase if they do not match, I will actually have to roll back and re-execute, if we get the correct result. If I have 3 then I can take the majority vote, and believe the vote, and go ahead, I can say some overreact. So, redundant array of inexpensive disks. So, this is the proposal from University of California, and overtime which has actually become industry standard.

(Refer Slide Time: 30:17)



So, as a name suggest, it is basically an array of disks and with some redundancy bit. So, the array of disk provide parallel, that was the main goal of. If you just remove the r on the beginning, if you just have 8 array of inexpensive disks what you gain parallel, it gets you more throughput. So, what you can do is you can stripe data, across disks to get more throughputs. So, what we mean by study.

So, let us suppose these are 3 bits. So, I will be actually instead of 3 sequential data want this pulling and then we want other disk, what I will do is, I will go like this. I will put a block of 512 byte here, the next block here, next block here, next block here and so on. So, then I can make sequential accesses to parallel disks, and I can get the throughput. I have got latency remain unchanged, because n 1other this are actually really faster than. So, then it does not really improve the speed of the disk. They are still remain as slow as they were, what about disadvantage. Array of disks has no dependable compared to a single disk, why is that. If the probability of failure of a single disk is p then what is it for n disk.

Student: P by n.

What access.

Student: P 2 the power of n.

P 2 the power of n, it is smaller or larger.

Student: Smaller.

Smaller why

Student: Because probability of (( )).

Exactly. So, as you raise the power n the value actually goes down. So, that is what is mentioned here, that we have an array of disks the probability. So, probability of failure of one disk is p. see if I ask you if I give you an array of n disks, what is the probability that at least one disk will fail, what is it. This is the probability of failure of single disk. In n disk what is the probability that at least one disk will fail.

Student: p into 1 minus p to the power n.

Sorry say again what is the probability that nothing fails.

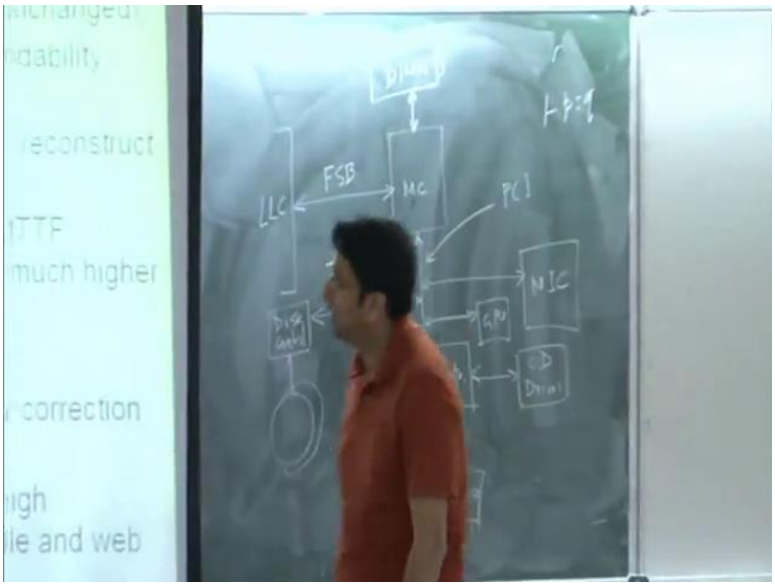
Student: (( )).

This question should be answered, what is the probability that at least 1 disk fails, your array of n disks.

Student: 1 minus 1 minus p to the power n.

Whole to the power n, which one is larger p or this 1. So, this one is failure of at least one which one is larger. This one is larger.

(Refer Slide Time: 34:14)



So, 1 minus p is q. So, this is 1 minus q, which one is larger.

Student: q n is less than q, 1minus q n is greater than 1 minus q.

So, this probability is much larger. So, the probability that at least 1 disk fails is much higher than failure of the single disk square, so that is what mentioned. So, a redundant array tries to bridge this particular gap. It says that well, we have an array of disk, so we have this particular problem; we have the failure both the array. So, can I bridge the gap which. So, essentially what are disk is that, there must be possibility of reconstruct the lost data I there is any. So, good news is that, m t t r is much less than m t t f. So, redundancy can make reliability of 100 disks, much higher than the single point. So, question remains how to discover faults. So, disk sectors hold error detection and data correction code.

So, a simple one could be; simple error correct double error detect. So, how do you do this actually, which means I must be codify data in such a way that have you correct one error and detects two errors. It will suggest the simple way of doing that; a simple coding scale. So, forget about disks. So, let us (( )) details and which assume that I will not transmit say n bits over a noisy channel. And I want the receiver to be able to set that, since there are correct, and I am not worried about how much extra bits I have to transmit to be able to do. Can you suggest the simple protocol to failures.

Student: Counting the number of ones.

Can you detect down errors. Actually one particular bit from 1 to 0, other 1 from 0 to 1, remain exactly same number ones.

Student: We can calculate the hash of that.

You have to give a guaranty here. I will not do that sector.

Student: count the number of ones in multiple packets parallel, like we have n packets. T coming the number of ones in the first bit of each packets, and then second one respectively.

But I am transmitting this one packet of data disk. I do not have any packets. The receiver should be able to correct one error detect two errors.

Student: we calculate three times.

So, that is not a simple thing that we can do. We transmit every bit tracks, and then the receiver checks 3 bits in blocks. If all of them are not same, then you sure that there is an error. And if there is one error you can correct it. You look at the other two and just give, you know what the correct one is. If there are two errors, then can you detect that.

Student: (()).

If you look at the…

Student: (()).

No there is no disk I am just transmitting n bits over the channel. So, if I look at

Student: (()).

I know I want to guaranty that, you cannot. So, can you

Student: one more copy.

You need one more copy, how.

Student: transmitted one more.

Sorry say again what. So, does everybody see that repeating the bit thrice does not help. It cannot detect double errors. You can detect an error; you can correct it if there is one error. What else can you do. What you may transmit the x or of all the bits also. I repeat every bit thrice. So, I said 3 bits, I said one more extra bit, which is x or bit of all the bits with that time detecting two errors.

Student: No

No why

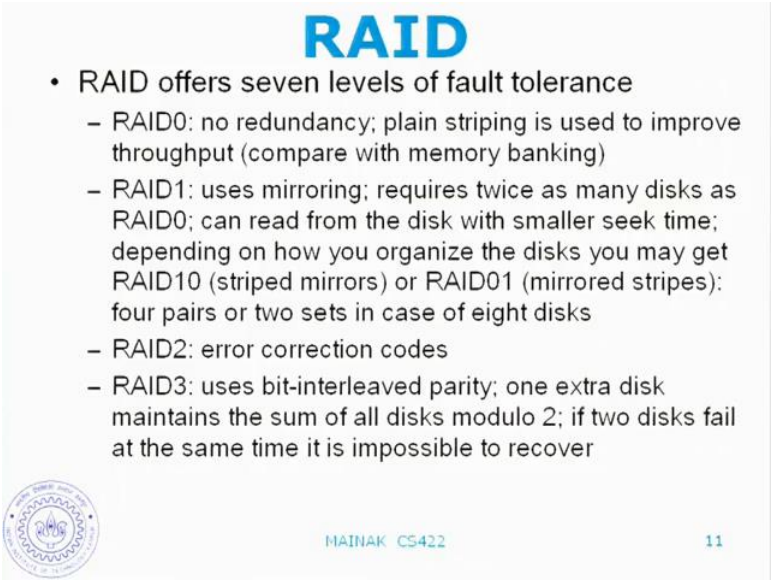
Student: I mean we can have that error in the x or bit also, when if there is an error we send one error.

So, that is an exceptional that the parity bit cannot.

Student: And 3rd bit can.

Then you can do that.

(Refer Slide Time: 41:29)



So, there are really other ways of read this, but I am not going in to that. Warning here is that, you can if attach error detection correction codes, you can recover from errors, and hot swapping with hot spares enables very high reliability and dependability. It is important for file and web servers. So, let us look at some of the raid enables, and also we will look at some of the coding examples that you know. So, raid offers 7 levels of

faults tolerance; raid 0 is no redundancy, plain striping is used to improve throughput. So, this is just very similar to memory banking, there is no redundancy at all. Raid 1 uses mirroring requires twice as many disk as raid 0. So, you can read from the disk with smaller seek time. So, essentially what I am doing is that, I duplicate every disk. So, that is one here.

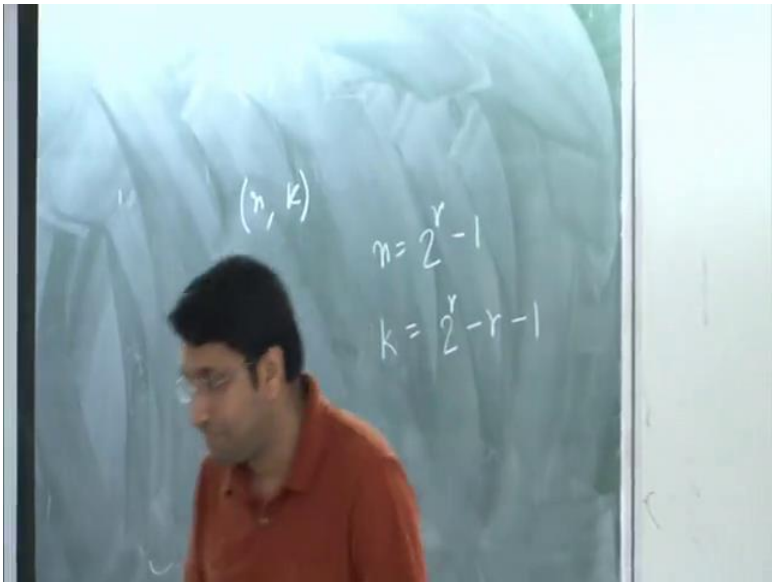
(Refer Slide Time: 42:15)

And since I am now the option of reading the data from two possible disks, I read the one that has the smaller seeker. Now depending on how you organize the disk, you may get raid 1 0 which is striped mirrors or raid 0 1 mirrored stripes. So, I will explain that. So, this I have just two data disks, and then I redundant. So, this is essentially mirrored stripes, I first write the data and then mirrored the data. So, essentially what it gives, first I do raid 0 that is what I take, I raid 0 first and then apply raid 1, from the entire disks as well . The other one which I could do is. So, I read out first and then write it; that is basically striped mirrors raid 1, whenever increase a disk I read a disk first and then correct. The raid 2 adds error correction codes on out of this. Now here the original, according to the original publication. There is really no constraint on what kinds of data correction codes you can use.

So, for example, you could use the repetition thing that we has mentioned, duplicate thrice. And different types of codes would give you different kinds of recovery data. For example, what I can do is also that. So, again this t which gives you a 50 percent over it.

So, for example, I have 4 data disk, and what I do is, I take the pairs and store the parity in the disk. So, I have two error correction disks. So, this 1; exhorts the data here inputs that here, and this one exhorts the data here inputs here. So, what kind of error correction detection can I do with this. So, let me number this; may be a b c d p q. So, p is a exhort b q is c exhort d. can I do single error correction. I can. Can I detect two errors. What if both a and b, its over to detect, because exhort will be same actually. So, I can do single error correction here, I can detect single error. So, that is all I can do essentially.

(Refer Slide Time: 46:16)



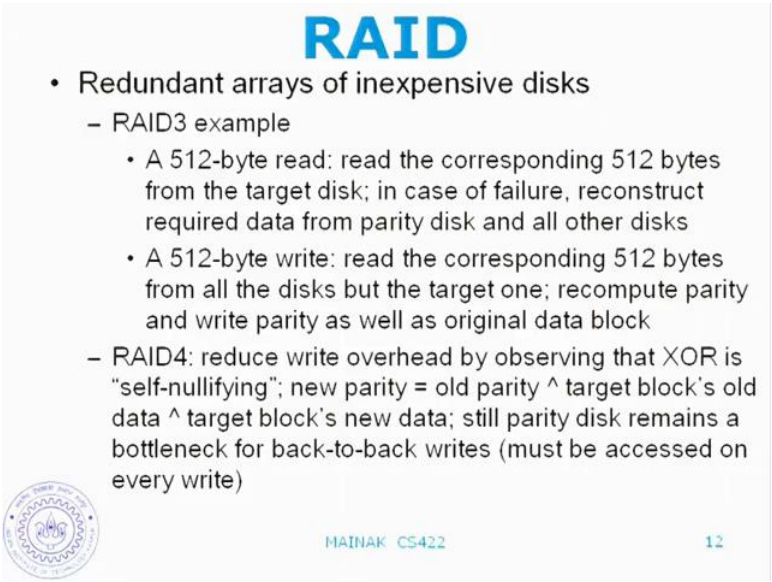
So, there is something called hamming codes; hamming codes. I am not talking about hamming distance, no. So, I would not get the details of hamming code I will say you what it is. So, the original raid 2 paper actually mentioned hamming code for this purpose. So, what hamming code does is that, normally it mentioned as total n, k, where n is the size of the total amount of data; that means, data plus your error connection code which is 2 to the power r minus 1. So, its r bit you know. So, we have 2 to the power r minus 1 bits will transmit, and out of which, the number of actual data bits is this; 2 to the power r minus r minus 1.

So, essentially what it means is that, if you have 4 data bits, you need three more bits to constant the hamming code. There will be those three bits will be essentially at the writing codes. So, 7 4 for example, in the possible hamming code, turns out that, on top

of hammy code if you add 1 parity bit; that is that exhorts all the data bits, we can actually do double error detection and single error correction .

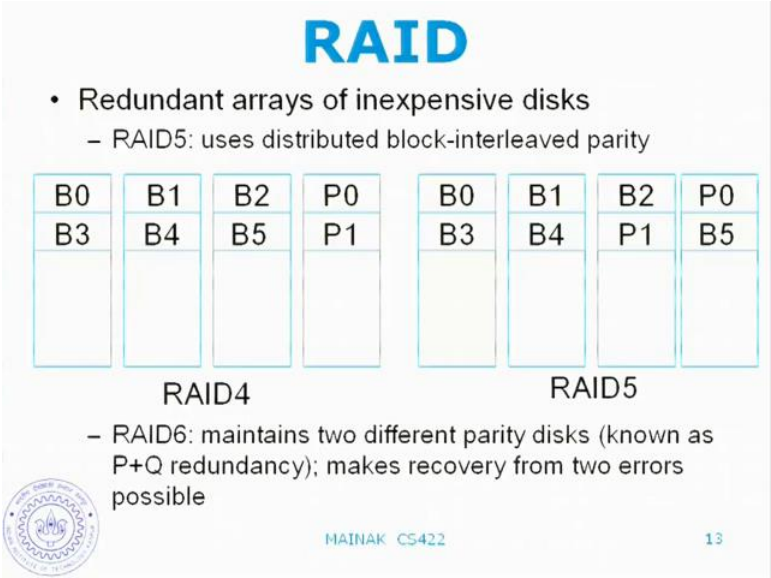
So, anyway. So, this is the more I get theoretical proposal, and industry does not really use this, any of the commercial basis. Although raid 1 0 and raid 0 are very popular in industry, and then of course, that the subsequent also raid 3 uses bit interleaved parity. Here you have one extra disk, that maintains the sum of all this modular 2, since an x or. If 2 disk fail at the same time it is impossible to recover. So, that is what we have already mentioned in this case also, so that is raid 3.

(Refer Slide Time: 48:02)



So, here is an example for raid 3. So, let us suppose I want to do a 5 1 2 byte read which is my sectors. So, you read the corresponding 5 12 bytes from the target disk, where it is located. In case of failure you reconstruct the required data from parity disk in all other cases, that you can do. So, only in case of failure we have to access all the disk, for this particular sector of data.

(Refer Slide Time: 49:24)



A 5 12 byte write is actually more expensive. You first read the corresponding 5 12 bytes from all the disk, except the target 1, recomputed parity and write parity as well, as the original data block . So, here for computing the parity when they meet all the disks, but actually that is not necessary, because of the property of the x or. So, that is what it recognize raid 4. It reduce the write overhead by observing the x or is self-nullifying. So, for competing the new parity, you could just do performing old parity exor with the bit of old data ex or bit target blocks new data, that would be able to do that. So, still parity disk remains a bottleneck for back to back writes, that must be accessed, wherever the parity stored. Raid 4 is here, so this is what is raid 4. So, these are my 3 data disk that is not parity disk, which ex or all this data table. So, raid 5 uses distributed block interleaved parity, so essentially what it does is that. It does not fix the parity disk distribute the parity already. So, p naught is here p 1 would be here p 2 would be here p 3 here and again p 4 will be here and. So, on; is it good, what does it.

Student: The parity disk becomes the problem in raid 4.

No that creates a problem.

Student: Sir that is not. So, p 0.

So, let us suppose this space, I use p 0

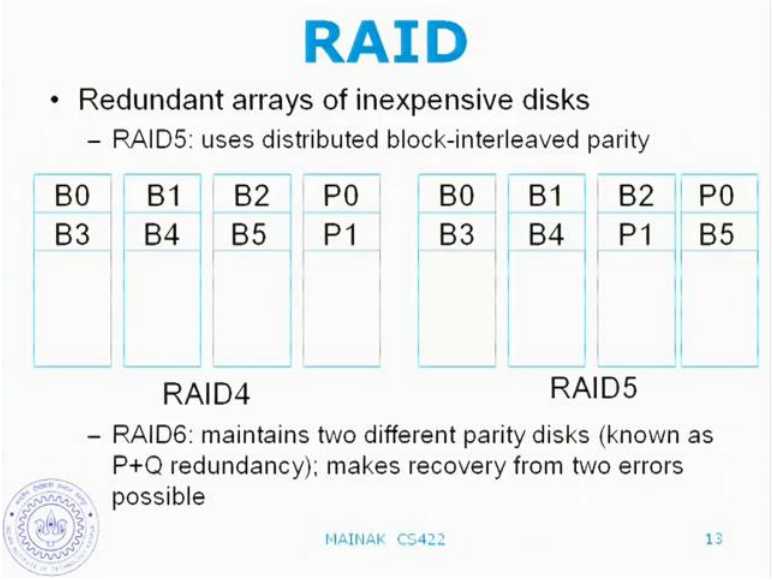
Student: p 0 parity.

How?

Student: Because on f b accesses you need to read from the parity disk. Over you do not mean that like over year it is been distributed.

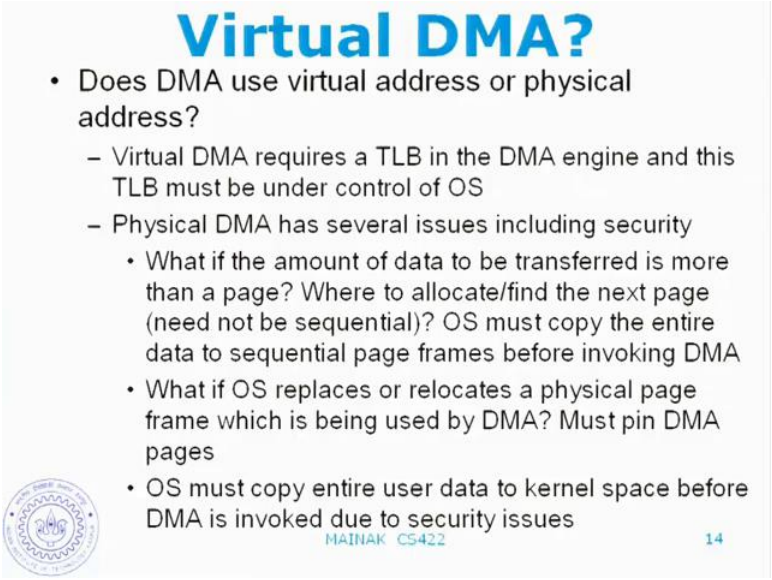
So, actually no. So, reading is not the real problem, the problem was the write. So, here if you want to do two back to back writes, this space you this would be the model actually. Here you could allow them to go concurrently, here ignore and here. Yes you are. So, for reads also you could do that, but actually the bit can be a raid where form the disk actually. So, that was not really the main problem, and raid 6 maintains two different parity disk known as p plus q redundancy. Next recovery from two errors possible.

(Refer Slide Time: 51:33)



So, here we do this. So, let us say raid 5, we are showing how we do raid 6. Here p 0. So, similarly distribute again two parity. Now p could be ex or of all the data q should be a stronger code, if you really want to recover from today, because q is also an ex or, you get nothing. So, q should be a different piece of code, which is code un-code independent of p. So, I will not get into mathematics of that, if you really interested how to generate this 2nd parity (( )). So, that is about raid.

(Refer Slide Time: 52:53)



So, we will do d m a, one question that we did not address is, does DMA use virtual address or physical address. So, the way we have described we seen that DMA should be physical address, because the CPU sends the physical address, the source and target and rest happens, without any intervention on ways or you know anything. So, you copy the physical address source to the target. So, let us see the pros and cons of the both. If you want to virtual d m a; that is DMA based on virtual addresses you will require A DMA with A DMA engine. So, that you can get the corresponding physical address, and this t l b once under the control of request, and it should be kept coherent with your processor t l b. On another hand, if you have physical DMA, which has several issues, including security. So, here just some questions that you should think about physical d m a. So, first question is what is the amount of data to be transferred is more than a page, when I down with the page where should I go now, because the physical pages are not contiguous they are normally scattered, and normally when you do DMA the CPU sets the starting address only. So, when you cross the pitch boundary, you cannot really start writing to the next page; that probably not.

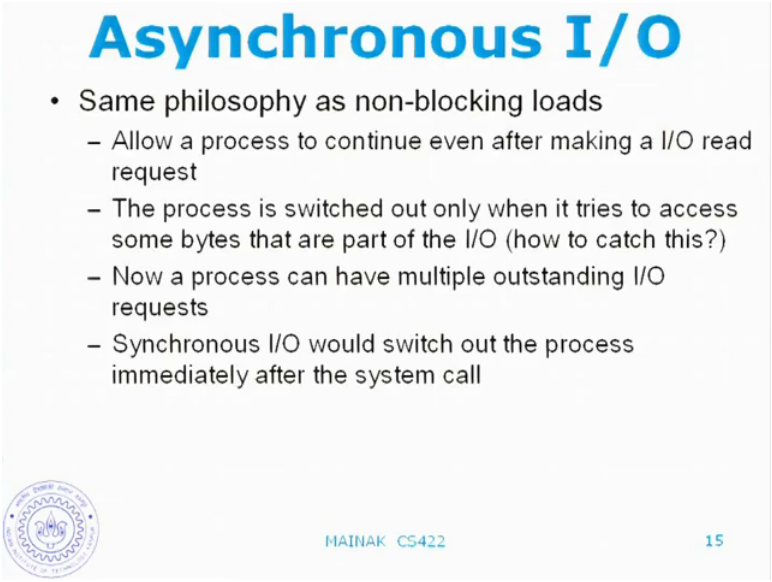
OS must copy the entire data to sequential page frames before invoking d m a. What if o s replaces or relocates a physical page frame, which is being used by a d m a; that is to avoid that you must p in DMA pages. So, that why the DMA is going on, a page should not get into this. And o s must copy entire user data to kernel space before DMA is invoked due to security issues, because what will happen is that, through the DMA A

particular user can actually start writing to some other space, which is once you start the DMA there is no security check, there is no check d 1actually. It is a white copy from one address page to another that is it. So, to avoid this problem o s must copy entire user data to kernel space before DMA is invoked.

Student: Sir DMA also has to have to read from the caches. So, that could also the caches are virtually that virtually index. That will also be a problem in physical d m a.

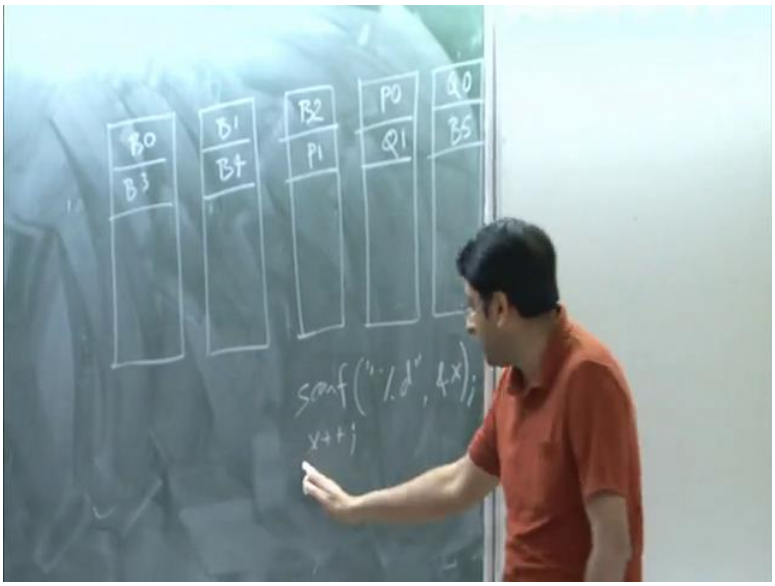
Yes. This is not special problem here.

(Refer Slide Time: 56:16)



Multiprocessors 7 in multiprocessors these problems arise, because when processor 1 sends a message to processor 2, normally it comes, it terms of physical address. In the process processor cache it is how do you really. So, within discuss anyhow this things, because I really cannot do that here, there is a lot of lire exploded in certain outsets, but anyway. So, these are the problems of you know virtual versus physical d m a, seems that virtual DMA main reason support as long as you have TLB in the rest control in the DMA engine. In physical DMA as long as you restrict your data size to a page you are done (( )). So, till now we have been mostly discussing about synchronous IO; that is whenever you start a IO operation, the calling process gets switched out in some summery of when the calling process must wait until the IO requests; however, you could asynchronous IO as well and one optimal. So, essentially here what we do is we allow a process to continue even after making a IO read request.

(Refer Slide Time: 57:02)



The process is switched out only when it tries to access some bytes, that are part of the (( )) how do you do this the process the program. So, let us suppose that I am the program that is particularly scan f and x and then I give x plus plus, and if I allow a synchronous IO for the for the s t d read, the program will continue execution come here, and here what it ask is that, this is the point where you should look at the (( )) not really, but how do I know that x is not available. Compilers generate a virtual address and look up the T L B and make it as a (( )) and do a translation, and codes access the physical page, and get some value which is protocol. So, how do I really achieve this particular thing.

Student: (( )) higher competition.

But processor will not allow instruction, how will you trap it actually

Student: (( )) standard process that this is the memory may goes to IO.

The request to the, if it rate of page.

Cannot reach why do you said so

Student: Because this does not have a t l b.

Why?

Student: This is the IO part so.

Which is the IO part, you are talking about.

Student: This one. This is the IO request, this will generate the IO request.

Which general system or yes.

Student: Sir that will be via page 4

Why should that will page 4.

Student: how will the IO request (( )).

There is a system call which will generate this read, which will read from your. So, at the lowest level the bytes will be red from the keyboard buffer, to some memory area.

Student: There main call is the load instruction.

No I am talking about part of the system call actually. System call handler will read from keyboard buffer into some memory area, and then copy from the memory area to the user space

Student: Sir System call handler knows that this is the IO. So, it can block all the other dependency.

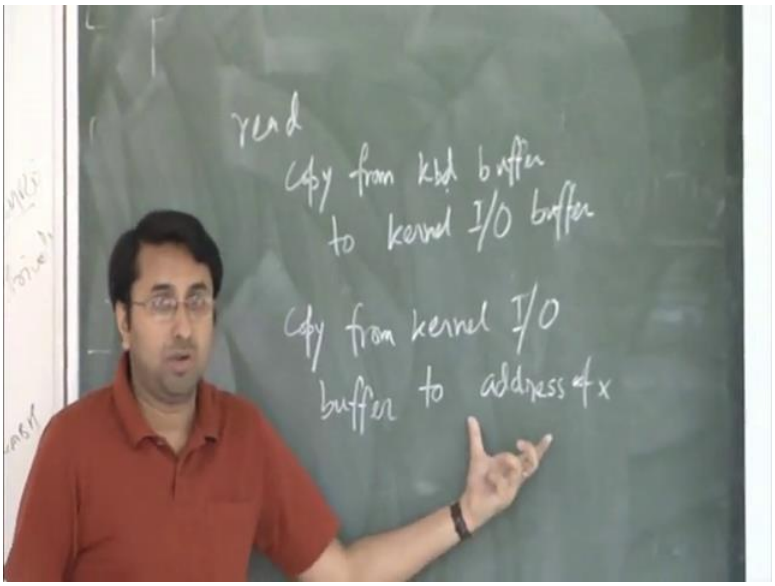
Which dependence

Student: The dependence on this.

How do you do that, that is what I am asking. So, first eventually generate this load instruction, which comes out with an address, now what would you do.

Student: remember that in the page table remember that in the page remember that there is the IO pending to this page and the next IO has completed that 0. On the next address translation, we can check whether the set or not. If it is set then we have to do a computation.

(Refer Slide Time: 61:21)

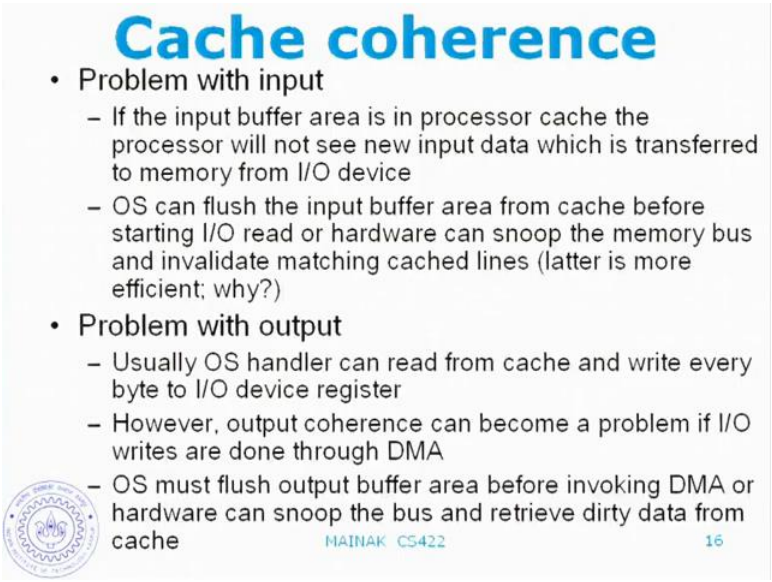


So, inside the system call handler at some point , the handler has to copy the data from the IO buffer in the controller space, in this particular address, in case of the memory that has to happen, because transistor will be generating this particular load eventually, which would go that particular address. Am I making sense. So, what I am saying is that you look at sequence of spaces that will happen, there is a read system call, which will essentially copy from keyboard buffer, whatever that is, to kernel IO buffer. And then the next step is, copy from kernel IO buffer to address of x.

Student: Sir, but how is this (( )).

So, previously what would happen is that, the process would get switched out, immediately as soon as you execute the system c. Here what I am doing is that I am allowing the process to continue essentially, why is this happen. The processor will generate that load instruction eventually. So, what you have to do is that, as you have mentioned in the page table I think we have to mark that this particular virtual page as the IO page, but you make sure that you actually get the point, before you allow the processor. So, if you want the asynchronous IO, the part of the system call handler will also have to (( )). So, that the T L B bits should happen, and then only the operating system may take the page table. So, that is one way, there could be other way also.

(Refer Slide Time: 63:13)



So, now a process can have multiple outstanding IO request, and synchronous IO would switch out the process immediately after the system call, soon as the system call actually. So, I am going to talking at this is the last slide we will finish it up. And then we will next week what we will do is, we will talk about hyper study, I am not sure if you have heard about that. Anybody heard of hyper study. No fine, which is surprising, but anyway. So, that is what we are going to invest our time on. It is kind of hardware trading, we have learned about trading at the operating system. So, we see how bring that down hardware; that is how we spend on next week.