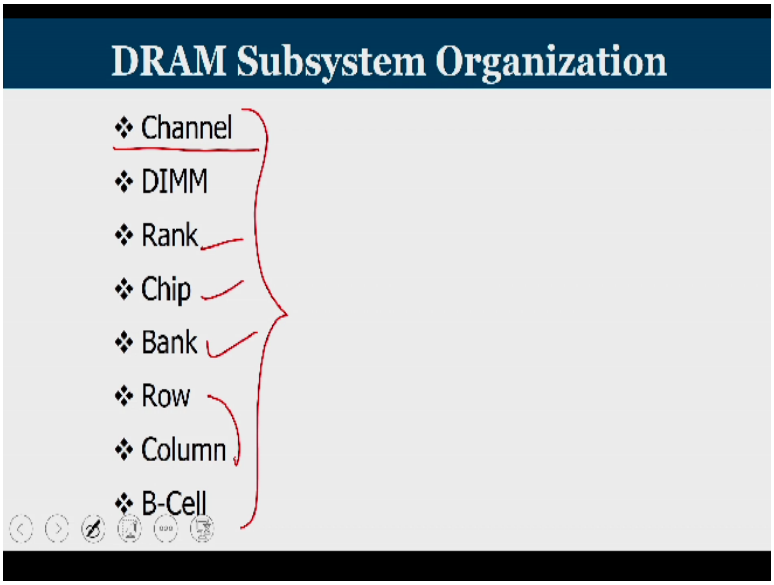
Advanced Computer Architecture
Prof. Dr. John Jose
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology-Guwahati

Lecture-26
DRAM Controllers and Address Mapping

Welcome to lecture number 18 where we are going to study about the working of DRAM controllers and the concept of address mapping in main memory system. In the last lecture we saw about what is the organization of a DRAM system and what are the typical hierarchies inside your DRAM system like channels, DIMMS, rank, bank, rows, columns and B cells. Now how this whole operation is being controlled.

We have a control circuitry which is known as the DRAM controller, which was initially located on the north bridge in the motherboard of a computer and later in modern processors, rather than placing a DRAM controller on motherboard, we are now having it in the processor itself. Let us know try to understand it more deeper.

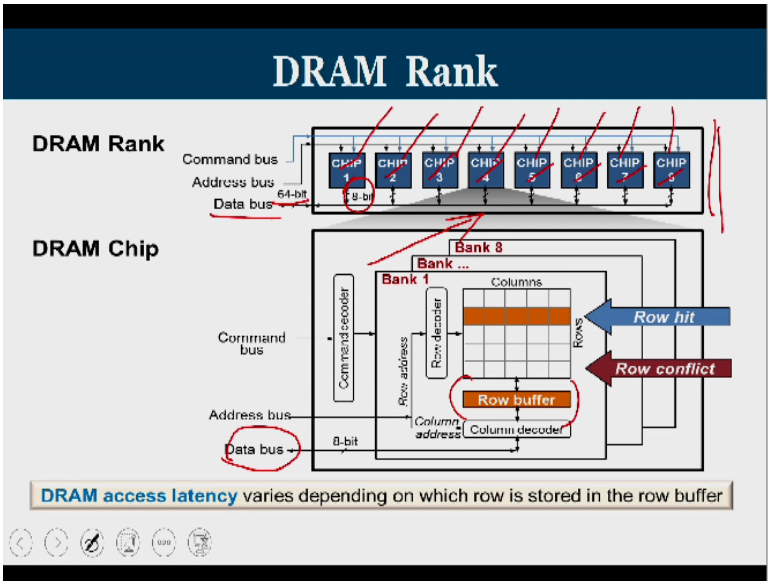
(Refer Slide Time: 01:25)



We have seen in our last lecture, this is the different levels inside the organization of a DRAM, modern multi core processors or multiple channels and we know that each channel consists of multiple DIMMs and then each DIMMs consists of ranks which itself ease of collection of chips

and then multiple banks and inside each bank we have rows and columns at the meeting point of a row and column is B cell.

(Refer Slide Time: 01:54)

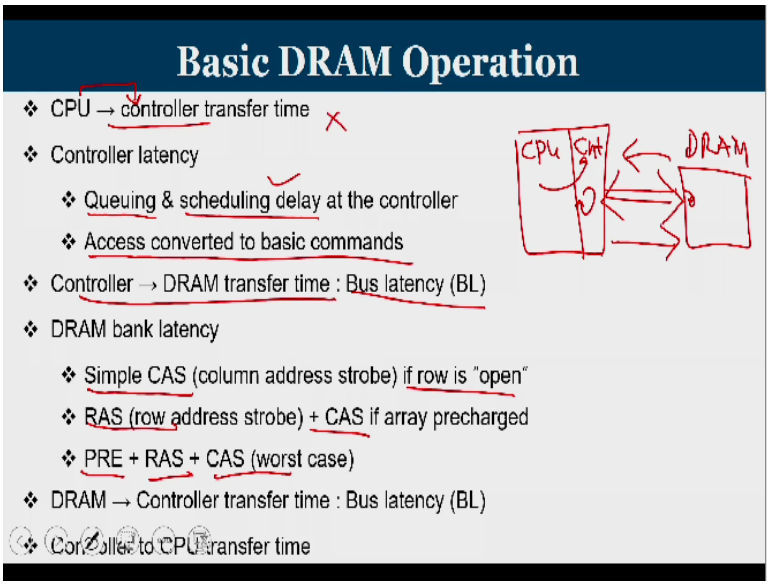


Now, look at this we are going to have a rank which has a lot of chips and each of the chip is going to give you a small fragment of data of the total. So, let us say if it is a 64 bit is my word length, every transaction from DRAM to the processor is in terms of 64 bits, meaning 8 bits are transferred from DRAM into the chip through the data bus or the data bus is 8 byte wide. Now these 8 bytes each of this chip is going to supply 1 byte each.

And together it will form the 8 byte of data. And this layering is known as the bank concert and we have separate row buffer for each of the bank. So I can give signals in such a way that each of the row buffer can be independently manipulated, but since all the row buffer are connected to 1 data bus, we have to be very careful about generation of signals as far as entry into the data buffer is concerned.

No DRAM access latency varies depending on which row is stored in the row buffer. So, if you are going to get a request to row number 20 and presently row number 20 is in row buffer, that it can be serviced very fast, but if the row buffer contains row number 30, then 30 has to be written back and then 20 has to be brought, it will take little bit more time than whether it us a row hit or not.

(Refer Slide Time: 03:23)



Let us try to see what are the basic DRAM operation. So, first CPU in terms of a cache memory miss will inform the DRAM controller and then there is a time for a signal to start from CPU and reach the controller and the controller itself will take some time to study this request and then do some processing on it which further involved there is a queuing inside of the controller because there may be multiple CPUs that is working on a single shared memory.

So there can be requested multiple such CPUs, and they are all queued up. And DRAM controller has to take a call, which of these request I am going to schedule, that is call the scheduling delay and the controller and once you have picked up this particular request you are going to process this access is been converted to a set of basic commands. And from controller there is a DRAM transfer time and that is traveling through the bus.

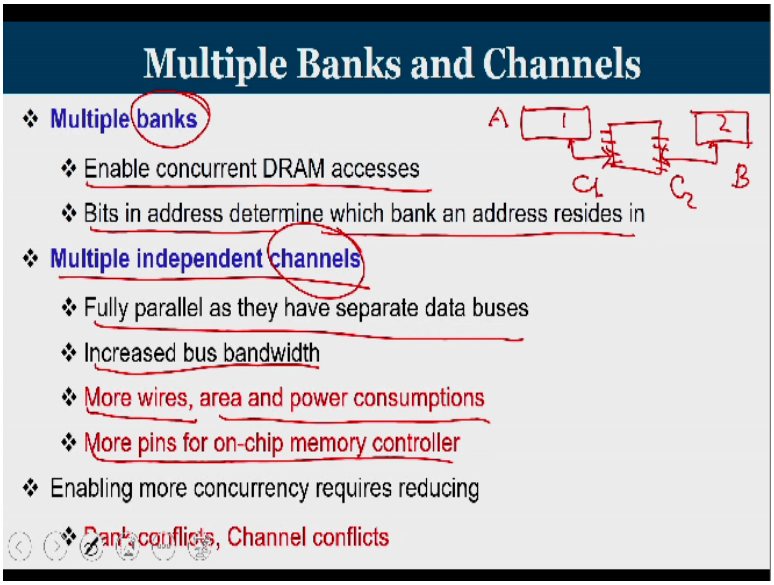
So what do you look into generally closely is let us say this is the CPU chip and out of that this is going to be the DRAM controller, which is part of the chip and then we have a bus which connect to the actual DRAM where in your CPU to controller transfer time is 1 and then the time it takes inside the controller and then the time to reach the DRAM and generally that is where the bus comes. Now, once it reaches the DRAM we know that it is consists of banks, rows, columns and all.

If it is a row hit then you apply a column address strobe that means for those requests where it is of already an open row. If the row is already precharge then we have to activate new rows. So rows and CAS is there, if it is a row conflict that means some other row is already opened and we have to precharge it then apply row address strobe and followed by column address strobe. And once this has been done, you have your data already here.

And data is ready then it is travels back. So again it is DRAM back to controller transfer time that the bus latency and then we have the controller to CPU transfer time whatever we had here, the same thing will happen towards. Now when you operate on DRAM one such important thing is we have to understand the various levels at which there can be a delay and all these are handled with circuitries.

And these circuitries are going to make some intelligent choices once you get multiple such kind of requests you have to pick one of them and this is a place where an intelligence circuitries and intelligent DRAM controller can play a very significant role in improving the performance.

(Refer Slide Time: 06:05)



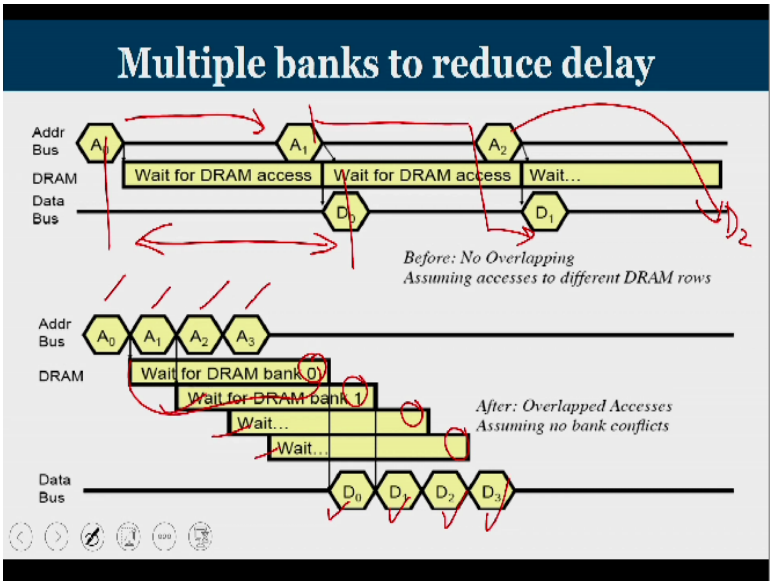
Now, we have multiple banks and channels that we have seen and what these banks are going to do is they are going to enable concurrent DRAM access and bits in the address sometimes determine what is the bank it is going to resides. So, we have multiple banks. Now, the question is how are you going to do and then similarly we have multiple independent channels. So it is

banks and channels that help us some kind of parallelism in terms of data that is being stored in DRAM.

So, when you go for independent channels, the idea of channels, if this is the processor, let us say this is 2 DRAM segment, let us say 1 and 2, and both are accessed by separate data buses, then we call it as this is channel 1 and this is channel 2. So, when you have program a which is residing in channel 1, and program B, which is residing in channel 2, then fully parallelly I can access them, I can bring contents of A and B together because they reached processor through different entry points.

So, when you have multiple channels, it is going to increase the bandwidth. But to have multiple such channels, of course we require more wire area and power consumption aspect is there. And moreover, inside your CPU chip, you require more number of pins, now, the double number of pins are being used only to interface with the DRAM. So more pins are there required for the chips. And to enable more concurrency, we requires to handle bank conflicts and channel conflicts.

(Refer Slide Time: 07:41)



Now, let us try to see what you mean by this bank conflict. So, imagine that you are going to give an address, the address will take some time for this processing whatever we have mentioned. So, if what address is given at this point, only at this point, the data is available. And

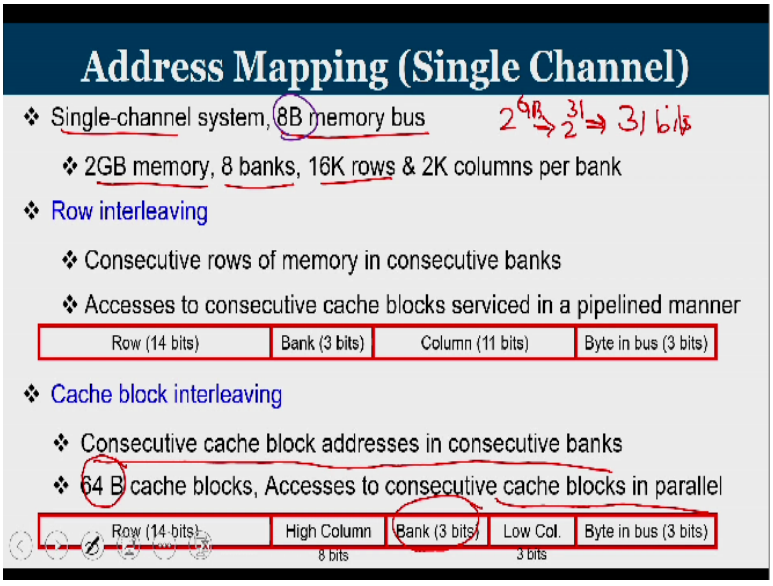
then you view the next address, it will take this time the data is available, A2 will take some time in order to get the corresponding D 2.

So, A stands for the address and B stands for the time at which the data is ready. Now, if you imagine that you are going to give addresses A0, A1, A2 and A3, each one will take some time and if D addresses were part of different banks, then all these banks will independently process take care of activities in order to activate this row buffer. And once the data is ready, then subsequently we are going to get data from adjacent banks, had A1, A2 and A3 be in the same bank as that of A0.

Then we may have to write back the row, then activate the contents in order to get A1. So, because of the mapping, which is in such a way that all these addresses are mapped to different different banks, even though they belong to different rows and columns. All the work pertaining to activating row writing back a row can be independently done because they are mapped to different banks.

And the last signal which is called column address strobe, which will transfer the data from a decide row and column that has to be carefully given such that they all are connected to the same data bus. So, in this way bank level parallelism will hide a lot of initial latency that could have happened if they are belonging to the same bank.

(Refer Slide Time: 09:29)



The next comes for address mapping. So consider a system which has a single channel and total of 8 byte memory bus you imagine it is a 2 GB memory. So, once it is a 2 GB memory you know that 2 GB correspond to 2 power 31 bytes that means you require 31 bits in address.

$$2GB = 2^{31}$$

A 2 GB memory let us assume that it has 8 banks and 16K rows and 2K columns per bank, that is going to be output.

Now, there are 2 types of ways in which you can do one is called row interleaving which you will learn first followed by cache level interleaving or cache block interleaving. The concept of row interleaving is we want consequently row of memory in consecutive memory banks and access to constituted cache blocks are serviced in a pipeline the manner. So, you just imagine, if let us say if this is going to be the address out of it, few bits will go as bite within the bus or within a word and then you have columns and then you have rows.

So, when you changes the bits in the LSB side they means that I am going to change only byte with respect to a row on the column. But when you are going to change here, these are all adjacent columns. Now, if all the values of the columns are fully over, if you add one more that naturally will result in change in the row number, that what we want here is constitute row. So, previously my row was in, now I wanted my row to be n + 1.

So, if I want consecutive rows of memory to be located in consecutive banks, then my bank should act somewhere here that is in between or immediately after the column bits. That is what exactly you see in this case we told that there are 16K rows that means it is 2 power 14, so 14 bits or 4 rows and it is an 8 byte memory bus. So at any given point of time once you give an address a total of 8 bytes are transferred 8 consecutive bytes.

So the 3 bits of the physical address they are basically byte within the bus and then you have column numbers. So here I put bank number in-between your row and the column we are introducing the 3 bits is used to distinguish the 8 banks. So, the idea is once the column numbers all these 11 bits are completely one, any increament to it that will result in a change in the bank bits. Even though we can assume this whole sector is wrong number.

What are the total number of rows in the system if you count do how 2 power 17 rows we are telling 2 power 14 rows are there per bank, since you have 8 such banks, so 2 power 14 into 2 power 3 that is 2 power 17 banks, sorry 2 power 17 rows are there.

$$2^{14} * 2^{3} = 2^{17} \text{ rows}$$

So, in the case of row interleaving the bank bits has to be placed immediately on the MSB side of the column bits. Now an interesting thing is we contact DRAM only in cases where we have a cache miss last level cache miss.

So, from DRAM we are going to transfer contents to the cache only and what is the unit that we wanted to transfer to the cache, it is basically a block of data. So, when there is a cache miss, you are going to contact your DRAM cache miss in terms of last level cache miss we are going to contact the DRAM and the DRAM has to identify the data and transfer a block of data, it is not possible to transfer a block of data you are transferring only one word at a time.

So, multiple such words, which are basically adjacent locations has to be transferred to the data bus such that the cache will receive a block of data. So, there is a very high possibility that when there is special locality I can request for the next block. So, if I request for block number n in main memory, there is very high probability that there can be a request to block number n + 1, n block number n + 2 like that.

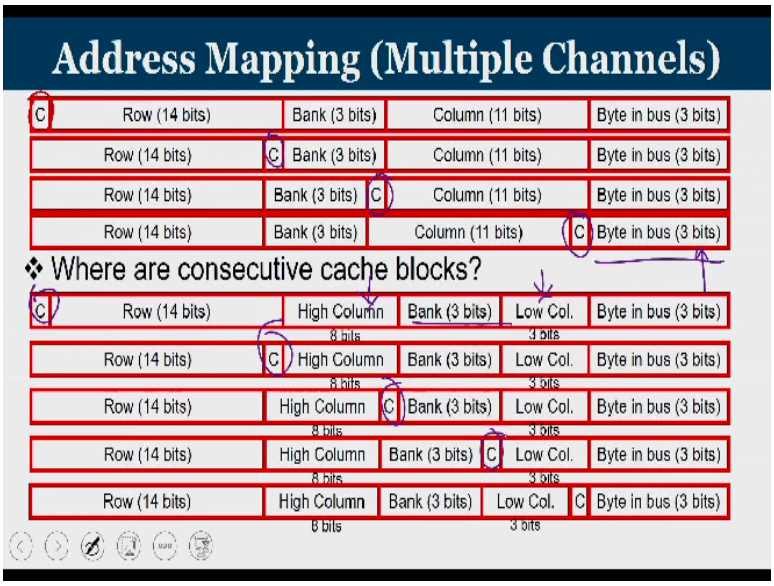
So, to service this special locality principle adjacent cache blocks should be kept in adjacent banks, so if the cache block 64 byte 1 chunk of 64 byte will be kept here. Now, as far as address is concerned, the next 64 bytes should not be kept in the same bank it should be kept in another bank such that they can do all this row and column commands in background when I am currently processing the first 64 bytes.

Then when the first 64 bytes are being supplied, I can keep the next 64 bytes ready if and only if they belong to different banks. So in cache bank interleaving the concept is adjacent cache blocks are mapped to a adjacent memory banks. So cache block interleaving constituted cache block addresses are kept in consecutive banks. So 64 byte cache block access to consecutive cache blocks in parallel.

And if you look at here, what is the cache block size, if you are going to work on a cache block size of 64 bytes and 64 bytes means you have basically 6 bits in the address, the last 6 bit of an address 6 bits LSB, if you vary them from 6 zeros all the way up to 6 ones, they all will be part of the same cache block. So, the last 6 bits. This is the last 6 bits 3 bits will represent bite within a bus and the next 3 bit will tell the lower order column number.

So, basically we are dividing the column number and introducing the banks in between. So that means once you work with the last 6 bits, then when you add up one more it is basically a change in the bank number. In this way adjacent cache blocks are kept in consecutive banks.

(Refer Slide Time: 15:36)



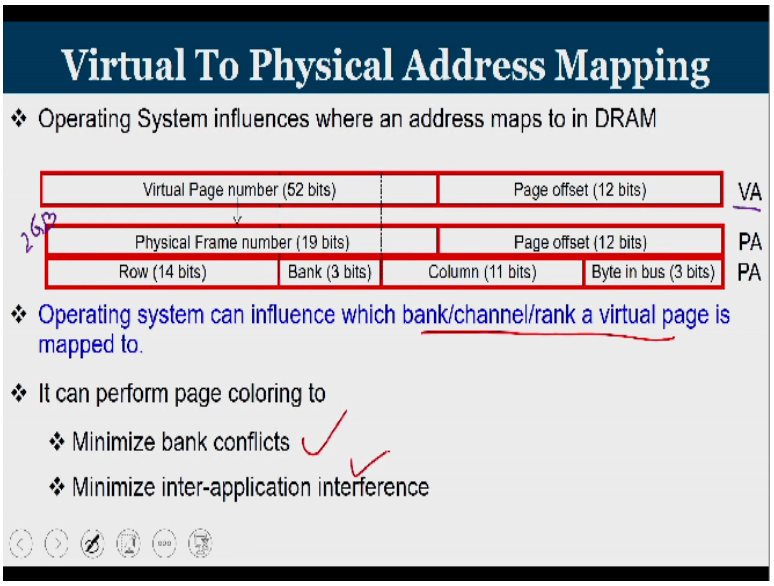
Now imagine you are going to have multiple channels. So previous question we had only 1 channel. Now if you have multiple channel, let us say this is a channel bit. Now if the channel with MSB that means all your locations are going to be continuously present. And the next thing for example imagine you have 2 channels. So MSB is 0 they belong to channel number 0, if MSB is 1 they belong to channel number 1.

So if you like care of total of 2 GB of memory per channel, and you are capacity of main memory is 4 GB. So the first 2 GB is part of channel 0 and next 2 GB is part of channel 1. The extreme case is the other one where I put channel number or I am going to assign channel number immediately after the word that is the last 3 bit aby way we cannot separate their byte within a bus.

This means that constitutive words are mapped to adjacent channels. Similarly, I can put like this constitutive rows are going to be put in adjacent channels, and here also the bank number also vary, now if you introduce cache box also into concept, so, you can know that this bank value is between your column number, lower order column number, higher order column number, that is what you can see.

And then you can put these channel numbers at whatever location you want. So, this is a scenario where adjacent cache blocks are mapped to adjacent channels.

(Refer Slide Time: 17:05)



So, so far what we have seen is we have seen a mechanism by which how we place the address as far as the cache block is concerned. Now, the next role is what is the role played by operating system, some of you might have already studied a subject or a course called operating system. Where operating systems memory management, duty is to assign a physical address. So when there is a virtual address the memory management unit is going to translate from this virtual address to physical address.

So, the moment space is allocated inside the main memory that means that particular program has a physical address. Now, what do you mean by virtual address every process assume that it can use the memory completely. So every process starts with the logical address beginning from 0 all the way up to 2 power – 1. So, there is a program A which believes from 0 to 2 power minus 1, program B which is below 0 to 2 power –1.

But when you have multiple programs working together, depending on main memory available capacity, I am going to take pages, it is a small unit, I map it page wise. So the process of translation of address from virtual to physical, that has been taken care of by operating systems memory management unit. Now in what way so, when I am going to place a program inside main memory operating system has the flexibility should I place in the upper half, lower half or middle or after 2000, after 10,000 like that.

And what is the criteria by which a place because once I keep this few bits in the address will tell you what is a bank that belongs to, let me draw your attention to an example that has been shown operating system influence where address maps in DRAM. So typically, what happens in a memory management unit is given a virtual address let us say the virtual address has 64 bits out of which 52 bit is page number and 12 it is page offset.

What your memory management unit these doing, let us say the memory the physical memory is are 2 GB. So, if it is 2 GB then whatever virtual address at the end of the map to 2 GB meaning 31 bit physical address, let us say this 31 bit physical address you are having 19 bit frame number and 12 bit offset, so 19 + 12 that is going to be 31. Now, you know that this is what we have seen all together. This 31 bits of address you know that last few bits is representing bytes and then you have columns, then you have banks and then the remaining is columns.

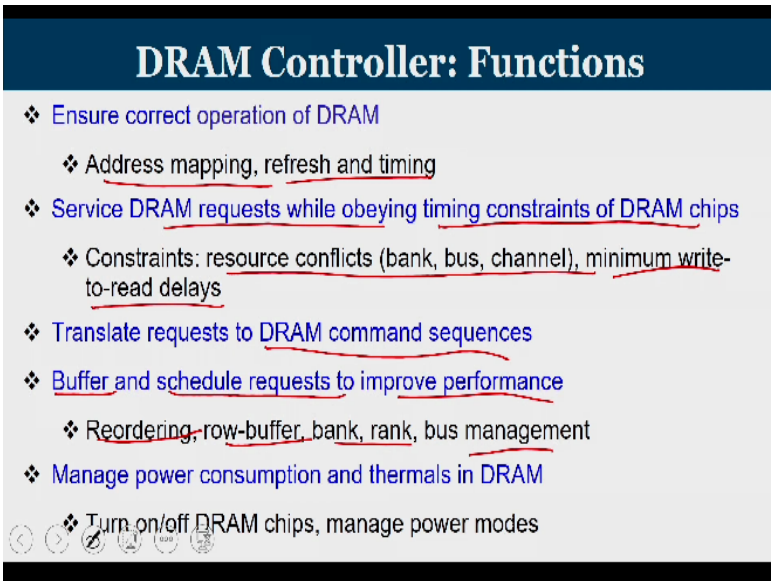
So, in this 19 bit which will tell you there are 2 power 19 page frames and operating system is going to assign one of the page frame for every incoming request, out of this 19 bits 3 bits are basically your bank. So operating system can know whether I should put in bank 0, bank 1, bank 2 like that. So, if operating system knows that there are 2 programs, which are going to be continuously used together simultaneously, that it is not a good idea to keep both of them in the same bank.

So program A should be given for example, bank number 001 and program B should be given let us say bank number 110. So, A can be given any address for this 19 bits except these 3 bits let say I give 001 but program B I can give any value as far as physical prime number is concerned,

let us say these 3 bits should not be 001, so that they want conflict each other. So that is the way by which operating system is going to interact in all these cases.

So operating system as a significant say. So, OS wanted to keep somebody in bank A or not to keep in bank A, the appropriate bits has to be properly configured, operating system can influence which bank channel or rank a virtual paper can be mapped on to. So, based on this it can perform page colouring, which will minimize bank conflicts and it can minimize in their application interference as well.

(Refer Slide Time: 20:55)



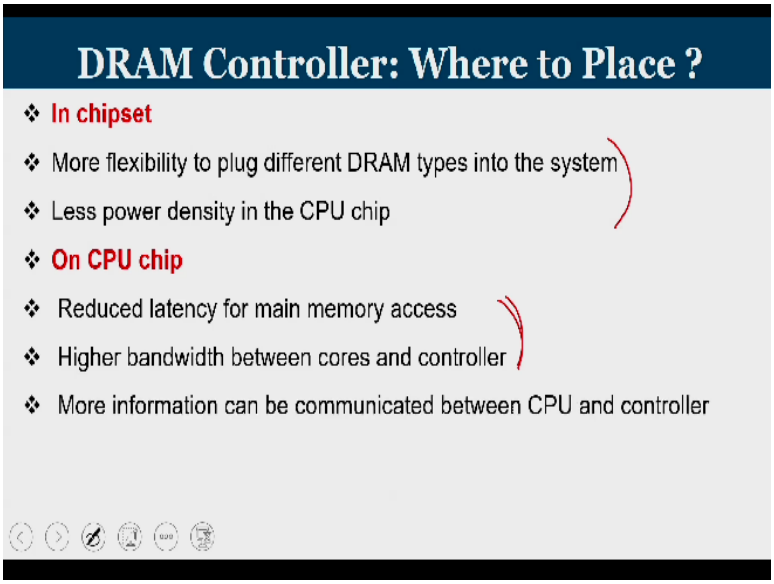
So, whatever we have seen the assigning of address everything is done by DRAM controller, we will study about what are the functions of this DRAM controller next. First thing is it has to ensure correct operation of DRAM and it has to give addresses and you know that the DRAMS are built with capacitor, So you need to refresh them and the timing aspect also has to be maintained and whatever request is coming to DRAM we have to strictly maintain the timing constraints at what time per-charge has to be given what time refresh has to be given.

And then when should we give complimentary strobe and what are the constraints that can be some resource conflicts in terms of bank bus and channels. And we have to reduce the delay between read and write. So, basically your DRAM controller translate the request whatever it gets into the DRAM command sequences, we are to buffer certain requests and we have to pick up certain requests or schedule certain requests to improve performance. So, based on the criteria

of sometimes we may have to reorder certain request managing of row buffer bank, rank and bus management.

And at the end we are to manage the power consumption and thermal aspects in DRAM, how it has been done, we can turn off certain sector of the DRAM if that is not frequently used.

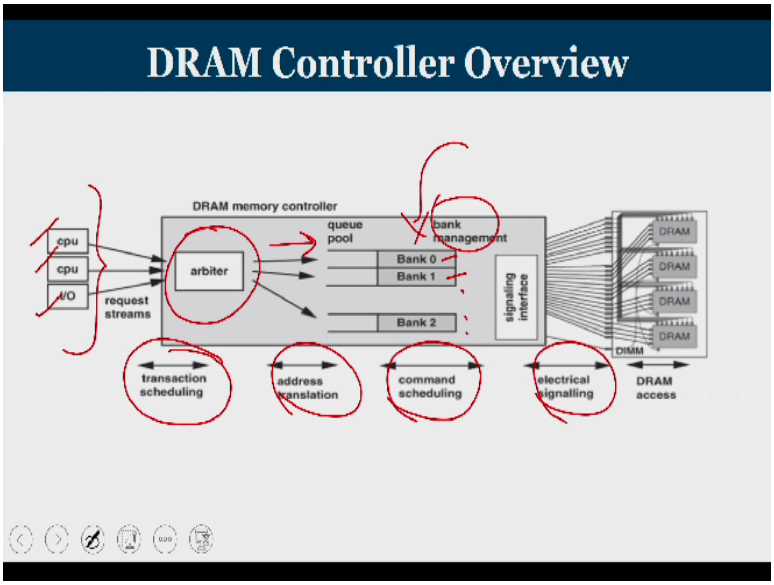
(Refer Slide Time: 22:12)



Now where can it keep DRAMs initially DRAMs are kept in the chipset in the north bridge, that gives you more flexibility to plug different DRAM types into the system even though the processor is changing to change the corresponding motherboard you will get a different DRAM controller as well, but less power density in the CPU, because it was not used that that but if you keep DRAM controller on chip, it will reduce your main memory access.

Because CPU can communicate with DRAM controller in much faster way because it is part of the CPU chip itself. And there is higher bandwidth between cores and controller. In this case CPU can communicate with DRAM controller only through pins, whereas in this case, it is an internal bus, there is no pin that has been involved, because DRAM controller is part of a CPU chip. So more information can be communicated between CPU and the controller.

(Refer Slide Time: 22:58)

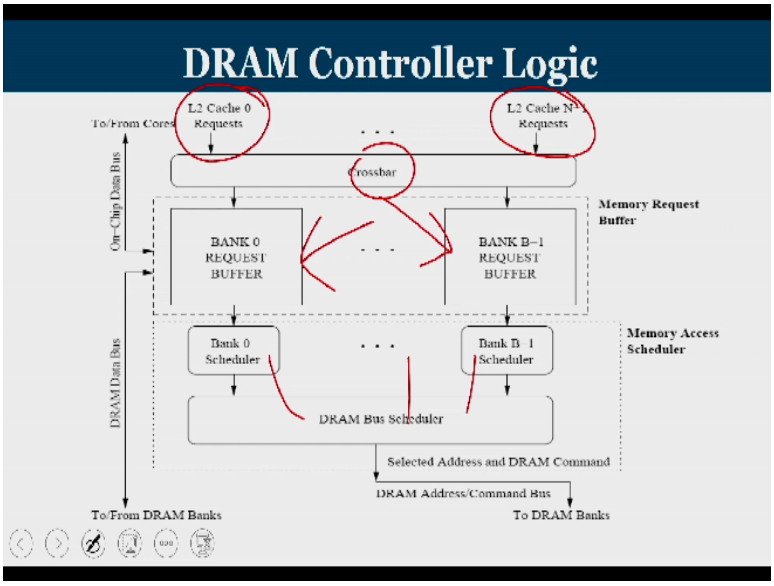


Now this is at the another view of a DRAM controller, will say you how CPUs your IO processes many things that is going to give request to the DRAM controller, and then you have an arbiter, arbiter will look into certain bits in the address and find out what are the bank bits. And based upon that and going to queue the request into bank 0. So bank 0 has a separate queue bank 1 has a separate queue like that.

So different queues are there based on incoming requests, look at the specific bit pattern and look for the appropriate bank and queue up them in different different banks. And then you have a bank management. So the basically the job is to pick up one from each bank. So this is called basically transaction scheduling, that is the set of request that is coming the transaction requests coming from various CPUs or I/O processors.

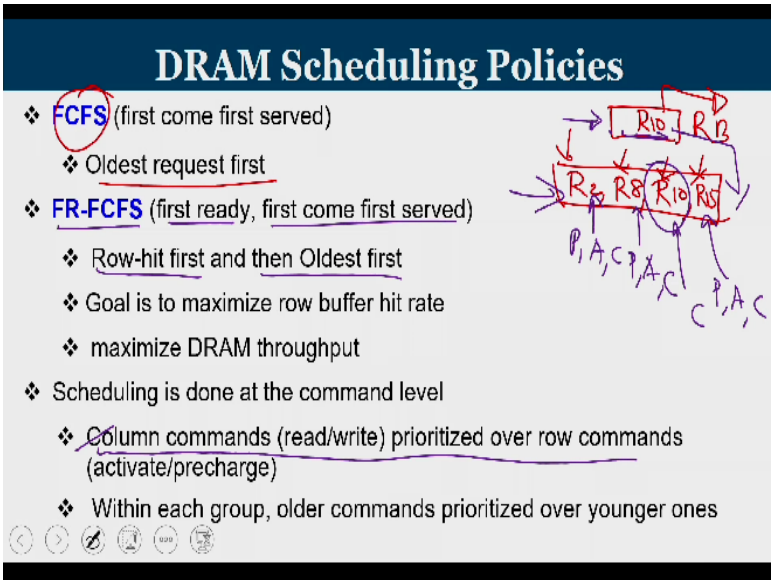
And then you translate into address and then you find out what are the banks and then you put it in the appropriate banks and do the scheduling and then you generate the signals that is electrical engineering side of it.

(Refer Slide Time: 23:54)



So basically, you have some requests from different caches are going to come and crossbar is going to put into bank 0 bank 1, it is a drop to the last level bank and then you have the scheduler which is going to pick up from each of these bank and give commands.

(Refer Slide Time: 24:10)



Now what are the policies by which you schedule, schedule means you have multiple requests that is there in your queue, you have to pick one among them. The first and foremost and the simplest one is FCFS based on the time of arrival, so the oldest request is going to be serviced to first, see consider the case of DRAM queue wherein you got requests coming from different processors are there.

So the DRAM is currently buys, once a DRAM is free now I have to pick up one among this. How will it be based on first come first serve which of these requests came first, that is called the FCFS policy, oldest request first. But the problem is we just imagine, assume that there is a row number n that is currently in the row buffer, ok. So let us say my row buffer is this is row buffer and it is carrying row number 10.

Now in the queue, let us say there are different requests one request is to row number 2 other request is row number 8, and there is another request row number 10 and then the row number 15. Let us say these are the 4 requests that has come these requests are incomplete as different column number as well. But anyway, column number is not relevant in this here. So by the time I complete some request, this is the status and we know that R2 came first followed by R 8, followed by R 10 and then R 15.

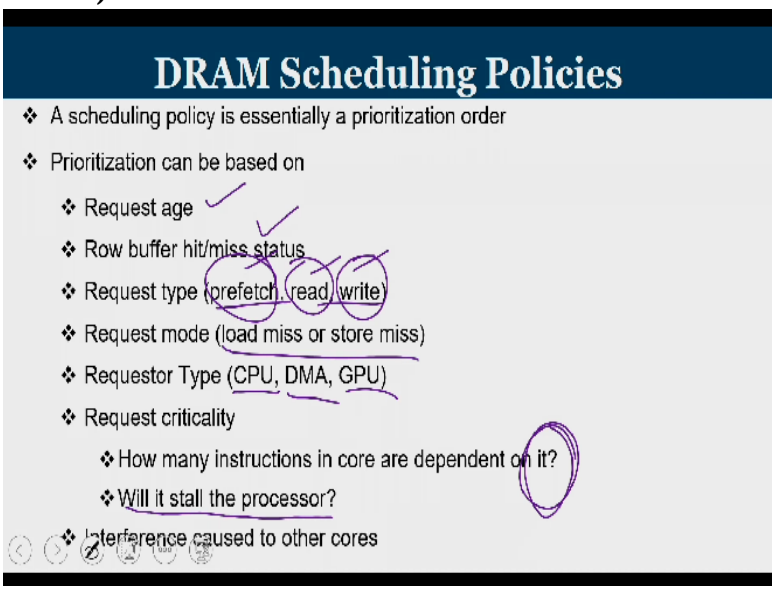
Had it been FCFS scheduling, we will always give R 2 and how are you going to schedule R 2 because of row buffer I need to get R 2 but already it is R10 I have how to precharge R 10, the contents in the row buffer has to be returned to row number 10 activate contents of R2 then you get R2 there and the new service. But you just look at the case you all to do a preacharge followed by an activate, if you pick up R10 rather than strictly going for FCFS.

Let us imagine if you are going to pick up R 10 already R 10 is there in the row buffer. So only the column number I need to apply I can service, such a kind of a mechanism wherein if already a row is open and I wish to service a request which is to maxing to that row then that is called FRFCFS- first ready first come first served request. So, first what will I do can I do any row hit first and there is no row hit, then you go for oldest first.

If there are multiple row hits, then I go for oldest first. So, the goal is to maximize row buffer hit rate and that will maximize DRAM throughput as well. And basically the scheduling is done at the command level. So, you do call them command if you can satisfy somebody only with the column command. So, how we just been possible, I already have your row buffer 10 that is active and out of these 4 request for doing this I require a precharge followed by activate followed by your column.

This also I can do by a precharge activate and column, whereas this can be done only by giving the corresponding column number. This also same precharge, activate and column number. So, I can do this one because it is only a column command that has been needed. That is my first priority, now within each group older commands are prioritized over the inner ones.

(Refer Slide Time: 27:25)

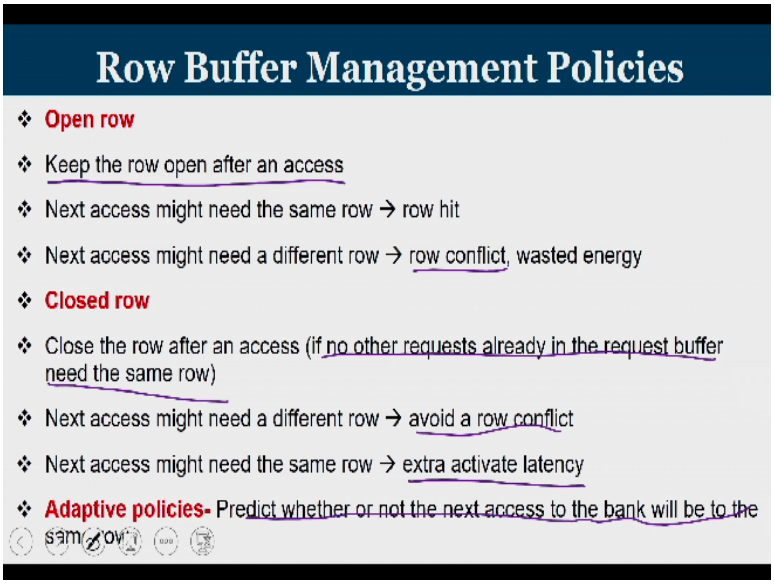


To come into scheduling policy, scheduling policy is essentially a prioritization order. A scheduling policy is essentially a prioritization order and prioritization can be based on age, that is what we have seen first, row buffer hit or miss, then certain requests are read some are write and some are prefetch request, during cache optimization, we have seen cache I can ask for prefetching packets bringing packets before they are actually needed.

So do we need to prioritize the prefetch packets when there are many other read or write there is they are called demand request, which is already there in the queue. And then there is a request mode. That is basically for the load miss and store miss. Now who requested whether it is CPU requested or GPU requested or a DMA controller who gave the request that is also a question and then you request for the criticality, How many instructions in the core of dependent on it.

So there are certain requests in the DRAM and if you service them many other waiting instructions can proceed, because CPUs already stalled because of this. So can you get the information vital information regarding the criticality of the instruction that is waiting for a DRAM miss and the interference cost to the other course as well.

(Refer Slide Time: 28:45)



The coming the row buffer management policy, we have 2 types of policies, one is called open row buffer management policy. So what we do is after the end of an axis, we keep the row buffer open. So what is the idea of keeping row buffer opened. So while you get the command to perform the operation, the data is being serviced. Now, once the data is service, what do you do we have to do a scheduling and scheduling happens when you have a buffer.

And if it is an FCFS scheduling, you look into buffer, find out is there any other request which is to the same row, you do it. Now imagine that the row buffer is empty, meaning that there is no other request that has come. Now we have 2 choices, the first choice is, I will keep the row buffer open with the hope that the next new request that is coming to the DRAM will be to the same row. But what is the next request is to a different row.

I have to precharge it and activate the new row incoming row, that is called open row buffer management policy. That means once the request is being serviced, and if the DRAM queue is empty, then leave the DRAM row buffer as it is such that if the next new request that is coming, if it is to same row, then you need to apply only column commands. And the second policy is called cross row policy.

Once you know that, the DRAM queue is empty, then we precharge it because there assumption is the new request will be always to a different row. If it is to a different row than rather than

doing a precharge at that point of time can we do precharge the way, the DRAM is now idle, there is no request there. So in my vacant time, or in my time, which I am not going to use for anything, let me complete the precharge work.

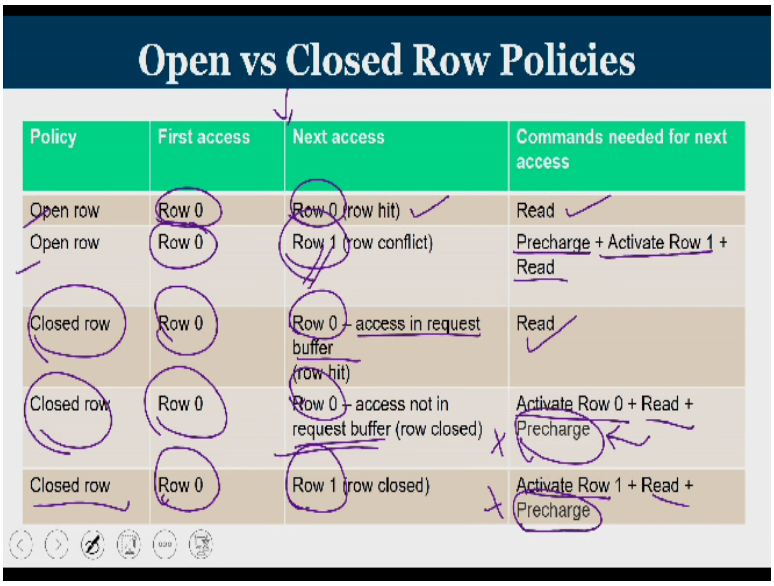
So that if the incoming request is to a different row, I need to do only activate, but what is the problem in it, I am precharging it. If the new record says to the same row, then it how to activate it again. If I have not done pre-charge, then simple column command is enough. So I am keeping somebody I hope that somebody else will come. So this is written back. Once you write it back if the new person is different, then it is ok, just activate new row and then service.

So if the new row is to a different person that close to buffer policies very good, because my precharge which would not come in the critical path, but if it is an open row policy, I keep the row open. But it is a new request that time have to precharge I have to activate and that is the time that I am going to lose. Both of it is own pros and cons. It basically depends on what are the kinds of requests that is coming in the future, whether it is for the same row or it is to a different row.

Just to summarize the points in open row policy, keep the row open after an axis, the next axis might need the same row. If that happens, it is called the row hit. If the next axis might need a different row, then it is a row conflict actually, I am wasting my energy by keeping it open. Now in the case of a closed row policy, you close the row after an access if no other request is already in the request buffer need the same row.

And next access might need a different row then that is called a row conflict we are going to avoid row conflict. Next access might need the same row than an extra activation energy is required. And both have its own pros and cons. So people are done research when you predict whether the next access to do the same row or different one based upon that you keep us open and close to strategies.

(Refer Slide Time: 32:09)



Now what is this policy here, we have an open row policy. If you are using an open row policy, let us say the first access is to row 0. Next access also used to roll 0 then it is a row bit, we just perform the read command by giving the appropriate column number, if the first access is to row 0 the next access is to row 1 then they how to perform a precharge because at this point, I do not know what is the next access I am leaving the row open.

As on when the next access come I come to know it is a different row. So I had to perform precharge, activate and then read. There is now look at close row policy letter say the first access is row 0. Next access also is rows 0. But let us assume that the next access was already there in the request buffer. So this perform read I am not going to close, in a closed row policy, let us say the first access was row 0.

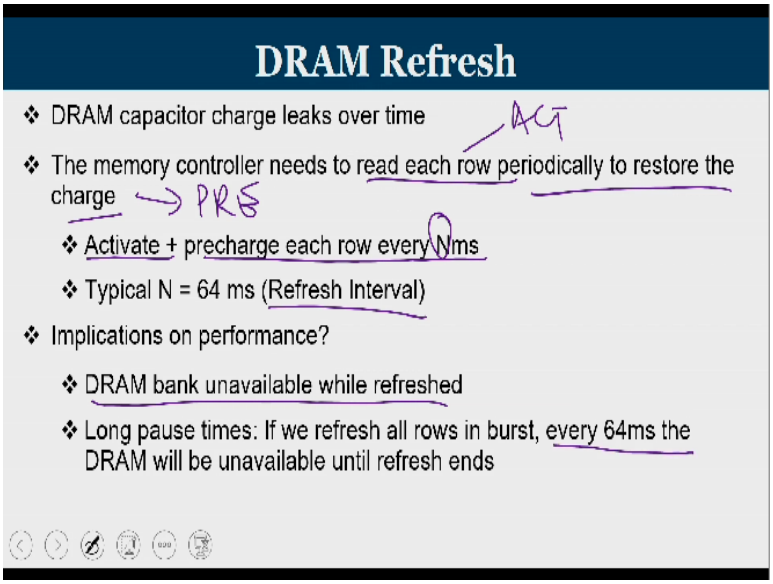
Now the request buffer is empty. So what will I do, I will close the row is the next access is to row 0 since I have already precharge I have to activate it again and then read and what I do at the end. If there is the request buffer is again empty perform the precharge, so precharge comes at the end. Now in a closed row policy if first access is rows 0, the next access is to row 1 and then I have to activate row 1 read at the end I do the precharge.

So the precharging come at the end the provided to in these 2 cases we can see you perform a precharge operation that is done provided the queue is empty. Now we know that DRAM is built

with capacitors and capacitors are a property of leakage. So if you read a data, data is lost, which I have to write it back by a precharge operation, but if you are not reading, then due to the leakage property capacitor between 2 parallel plates, the charge will leak.

And whatever value that represent logic 1 will become logic 0 if the potential differences below a threshold. So, in capacity based circuitry in order to retain value, I have to refresh it back. So DRAMs has a different thing circuitry, but then how often we do refresh let us see.

(Refer Slide Time: 34:21)

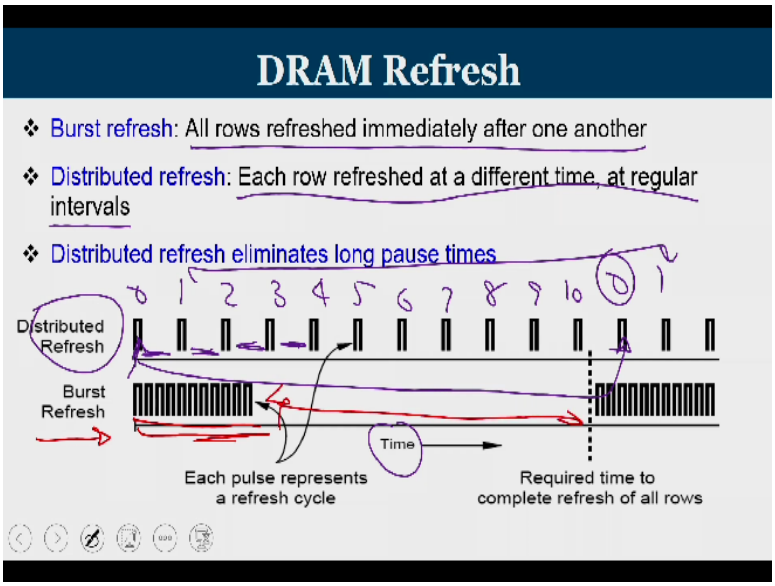


DRAM capacitor charge leaks over time. So, memory controller need to read each row, how will you do that by using an activate command periodically and restore the charge and this is done by recharge. So activator all the contents of the entire row will come to row buffer do a precharge once you do a precharge you are regenerating the value. So all these slowly decaying values once you do a precharge operation that is act followed by a pre the values are restored back to the exact logic 1.

So activate has to do precharge each row for every n millisecond, what is significance of N, N is called refresh interval. So, if you delay this operation, beyond a millisecond then the leaking of capacitor may result in a scenario where there is data loss. So refreshing has to be done once in every 64 milliseconds. And what is the implication on performance. When you do the refreshing operation you have to refresh all of those one at a time take row number 0, activate precharge.

Row number 1 activate precharge. So, during this operation, DRAM bank is unavailable for the general request from the cache. So that can be longer pastimes. So, if we refresh all rows in one shot, then every 64 millisecond the DRAM will be unavailable until the refresh operation is over.

(Refer Slide Time: 35:38)



So, basically we have 2 types of refresh, one is called burst refresh. So, at the end of this refresh period, all rows are refreshed immediately one after another. So, for some time, your DRAM is not available for rest of the service. The second one is called distributed refresh, each row is refreshed at a different time, but at regular intervals. So distributed refresh eliminates long pause times. So how can you represent it.

So distributed refresh what happened, let us say this is row 0. Now, I am going to refresh row 0 here. So between 2 rows are refreshing I have my time, now, this is row 1, row 1 has to be refreshed at this point. So, this spike indicates when row 0 row 1 is this is row 2, row 3 row 4, 5 6 7 8 9 and row 10. So, you imagine you have only 10 rows, for now, in this case 0 next up to 10 there is 11 rows, now your row 0 refreshed again.

So between the refreshing of row 0 up to the refreshing of row 0 again there is the time that I mean, so row 0 has to be refreshed every 64 millisecond. Similarly, row 1 also has to be refreshed within this time. So I am not refreshing all rows together I am refreshing rows only 1 at a time and then I am, so during this time you DRAM is available for general operation. Whereas, you see the burst refresh in this case all rows are refresh one after another.

So, during this entire segment DRAM is not available and this much time is DRAM is continuously available. And then again the repeat cycle starts. So that is called burst refresh each has shows its merits and demerits. So, with that, we come to the end of a DRAM controllers, so with these 2 lectures last lecture, we have learned the fundamentals of DRAM organization. And this lecture, we have learned the concepts of DRAM memory controllers, address mapping and scheduling.

So numerical problems in this section also will help you in understanding the topic with clarity. There is a tutorial session also in this week, and where in familiarized with some of these concepts, what you learned in the lecture, thank you.