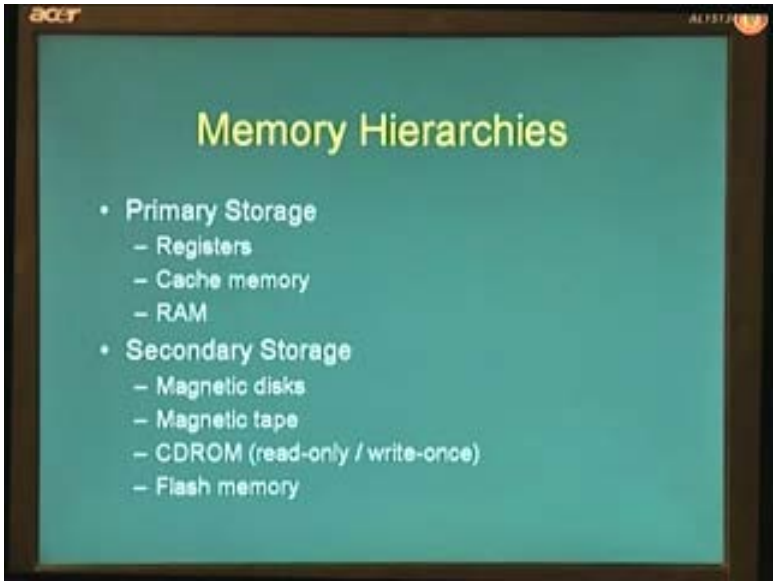
Database Management System
Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 10

Storage Structures

Looking at what might be termed as the logical aspects of database design, we looked at different data models, how data can be represented and how the relationships between them can be represented. We saw the ER schema which is meant primarily for human comprehension and we also saw the relational schema which we claimed is meant for the physical schema that is meant for machine comprehension. But when we say physical schema, it is still a kind of misnomer because the relational data model does not say anything about how data is actually stored on computers or storage device like disks or whatever, wherever databases are implemented.

So in this session and the next few sessions, we are going to actually ripe a part in a sense look inside a DBMS or inside an implementation of a DBMS and see how are these data elements actually stored on computers. And what is it mean when we say that we have stored a table? How is a table actually residing on disks or any other kind of storage device? So that is what we are going to be concerned with in the next few sessions. So let us begin the session on storage structures. In order to be understanding storage structures, one of the first things we need to understand is what might be termed as the memory hierarchy.

(Refer Slide Time: 00:02:39)



When we talk about storage or storing data, the first question we need to ask is where is data stored.

There are different kinds of devices which are capable of storing data. You might have obviously come across hard disks, I mean data that are stored in hard disks, floppy disks, CDROM and even the computer RAM that is the random access memory in the computer, the cache memory within the machine and even the registers are within the CPU, all of them are meant for storing data. And all of these can be organized, these kinds of, different kinds of memory devices can be organized in the form of a hierarchy which is termed as a memory hierarchy.

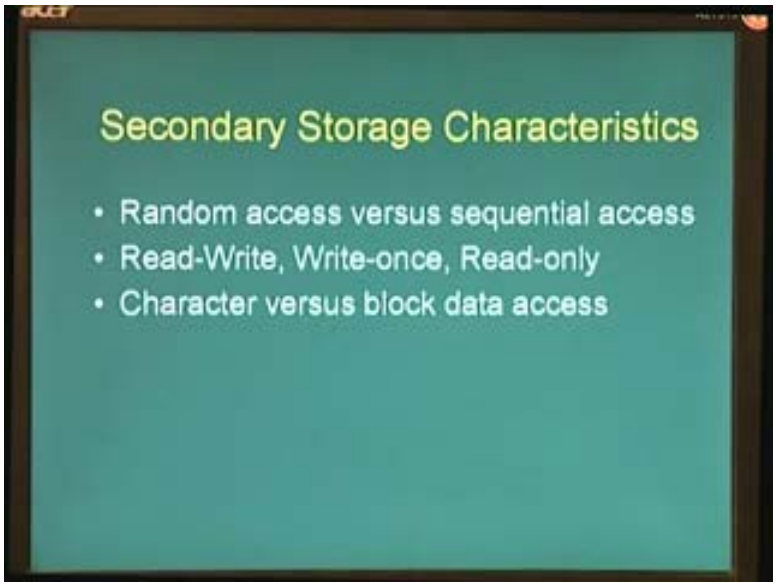
This slide shows such a memory hierarchy but it divides this hierarchy into two kinds of storage devices what are called as primary storage and what are called as secondary storage. If we were to draw a hierarchy, primary storages devices would appear up in the hierarchy and below them would be the secondary storage devices. What is a difference between the primary and secondary storage devices? Primary storage devices, some examples are shown in the slide here like CPU registers, cache memory, RAM, DRAM, SRAM and so on.

All of them are extremely fast memory devices, you can address or retrieve data elements extremely fast from these devices. However all of these are volatile memory devices that means once the power is switched off, they no longer can hold data. Data that are stored in primary storage devices cannot be persistent in nature. On the other hand secondary storage devices of which some examples are magnetic disks like your hard disk on your PC, magnetic tape which is primarily used in many locations for archiving data or taking backups of data.

Then there are CDROMs, there are read-only CDROMs, there are write-once CDROMs, there are even some kinds of read-write CDROMs that are being available today. And there are also what is a more recent phenomenon what is called as the flash memory. Flash memory is a kind of, is made of what are called as EEPROMS that is electrically erasable programmable read only memories which can store data persistently even after the power is switched off and they can perform or they can perform data transfer in a rate that is much faster than existing storage devices like say magnetic disks or tapes and so on.

So the common theme in secondary storage devices is that data can be stored in these devices in a persistent fashion and usually secondary storage devices are much cheaper than primary storage devices and they can store much more data than can be stored in a primary storage device. However usually secondary storage devices are much slower to access, they inquire much more over heads during access than accessing a primary storage device like say RAM or cache memory or so on.

(Refer Slide Time: 00:06:06)



Now for the most part when implementing a database management system, we shall be concerned mainly with secondary storage devices. We do not concern hassles with what are called as main memory database which are databases that are completely held in main memory. There are not many implementations of main memory databases simply because it's much more expensive to have large main memories which can implement databases of sizable of a pretty large size. So secondary storage devices have certain kinds, certain characteristics which are important or which influence the kind of storage structure that we are going to use to access and store and access data in these devices.

We can either categorize secondary storage devices either random access device or a sequential device. Something like the PC hard disk or magnetic disk or random access devices that is you can access any given block of data in a magnetic disk, they are called sectors. So you can access any particular sector directly, it is not purely random access because it does performs some kind of sequential searches. However for most practical purposes or magnetic disk is a random access device where you can address any block directly and move to that block read or write to that block directly.

On the other hand something like a magnetic tape is a sequential access device. If you have to access the hundredth block and the tape is rewound, you have to run through the first 99 block before being able to access the hundredth block. Therefore it's important what kinds of, how efficient can data access be when we are using a device which is either random access or sequential. For example we cannot implement a storage structure that has to perform a lot of pauses on the data mainly because especially when we are using a sequential device.

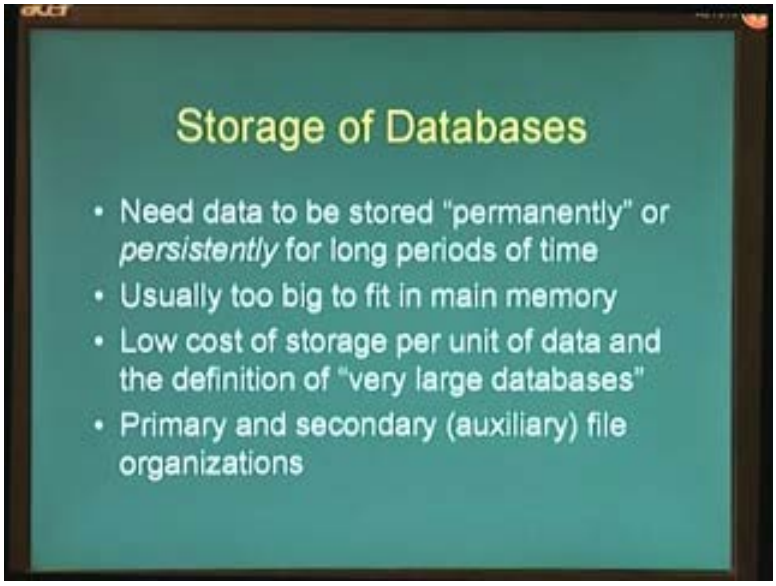
Similarly we can classify devices as either read-write devices, write-once devices or read-only devices. Read-write devices are those where you can read and write data any number of times hard disks, floppy disks, magnetic tapes and so on are examples.

On the other hand there are write-once devices where you can write data once but you cannot erase it. Once it is written, it becomes a read-only device. So such kinds of devices cannot incorporate data structures which need to be modified during runtime for example. Similarly there are read-only devices where data is stored during the manufacture of the device itself and it can never be altered and so new data cannot be stored on such devices again.

So there are devices especially in embedded systems which stores small databases within read-only devices and it's extremely important that such storage is performed correctly the first time because there is no scope for any kind of modifications, once data is written on to these devices. Then there are devices that can be classified as either character devices or block devices. Character devices are those where you have to read data character by character which can be extremely inefficient when we are dealing with large amounts of data. Some kinds of tape devices and so on are character devices.

On the other hand there are block data access devices where usually the unit of data transfer is a set of characters called a block. So every time any read request is given, a read request reads an entire block of data into memory and writes back an entire block of data on to the device and usually there block access is coupled with what are called as read aheads that means its not just one block, it's a set of blocks that are read into memory at a given point in time in order to increase data transfer efficiency.

(Refer Slide Time: 00:10:40)



What are the requirements for storing databases, what kinds of storage requirements do databases pose? Firstly we note that almost all databases required data to be stored permanently or what is called as persistently for longer periods of time. It should not be the case that once the computer is switched off, all the data in the database is lost, it has to be stored permanently or in a persistent fashion.

Usually databases are far too big to fit in main memory. It is not realistic to be able to search a database by loading the entire database into memory and then use such techniques that are mainly useful for main memory. We have to store the database on disk and involve strategies that can search data on to the disk and use memory for this purpose. Over the years the cost of storage has dropped drastically, in fact there are many claims that the progress in storage has beaten what is called as Moore's law. That is the amount of storage that can be packed into a given cost, for a given cost or on a given square area of physical dimensions.

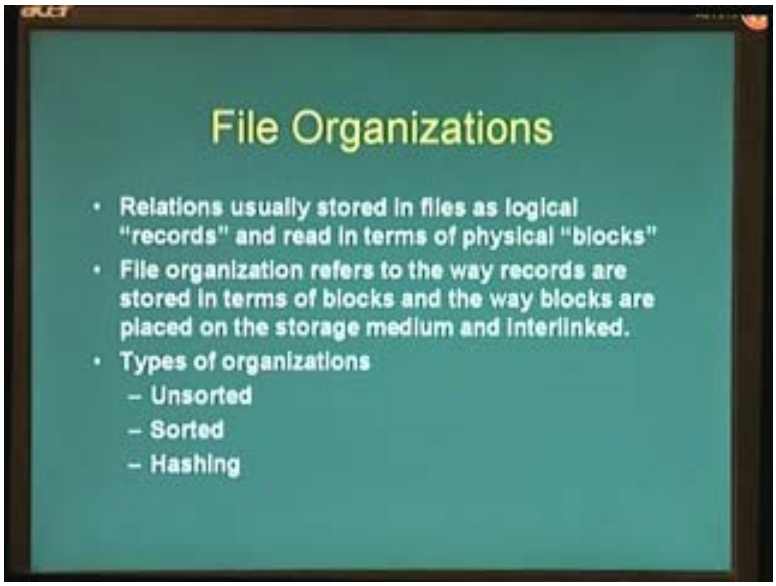
Today we can have gigabytes of data stores in a very small device that one can store within your in pockets or on in a very small area. We have gigabytes of data stores that are embedded within watches for example and within pens and so on. So the amount of the data store or the storage available per person has increased dramatically over the years. And one of the byproduct of this is the changing definition of what might be termed as very large databases. In fact the term very large databases when it was coined was meant to refer to databases which are of hundreds of megabytes large.

Slowly very large databases came to mean several gigabytes of a data and now we have databases that are several hundreds of gigabytes or terabytes which are 10 to the power of 12 bytes and then even petabytes of data. Petabytes are 10 power 15 bytes of data where especially databases that work on web related data like Google or AltaVista or search engines which collect data from all over the web actually work on petabytes of data. So the definition of very large databases has been changing continuously to include more and more data storage requirements.

Therefore what is being, what has become imperative today is to design extremely agile data structures that can store and manage data between main memory and secondary storage devices in an efficient fashion. When we store data on to secondary memory, we usually distinguish between two kinds of data storage; the one is what is called as primary data storage. The primary data storage talks about how we store the data itself, how data itself is organized on to disks or any storage device and how are they accessed.

Secondary file structures are those are also called as auxiliary or augmenting file structures are those sets of files that are used to speedup access to these data elements and this is especially important, secondary file organizations become especially important when the size of the database starts growing by leaps and bounds. When we have terabytes or petabytes of data, it is the role of auxiliary files or secondary files that provide pointers that help us in locating the required data element becomes more and more important.

(Refer Slide Time: 00:15:04)

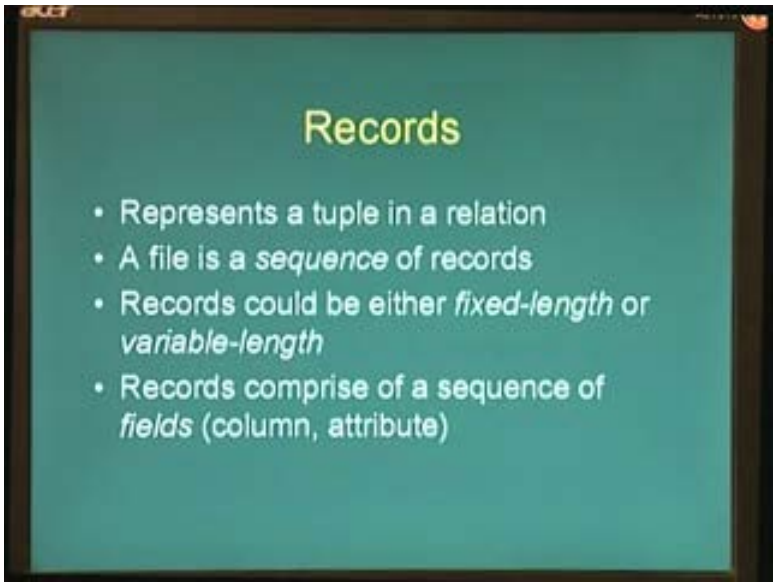


Usually when we talk about data storage on secondary devices, we are talking about what is termed as file organizations. Data is stored in logical structures called files on disks. The way files are organized on disk is called the file organizations. Usually files are stored as a sequence of records and a record is analogous to the notion of a tuple in a relation in relational algebra or a row in SQL parlance. So, a file is stored as a sequence of these physical or logical records and are stored in terms of what are called as physical blocks.

As we saw before in previous slide, the block is the unit of data transfer between main memory and the storage device. So there are two different things here within a file, one is the logical ordering of data which is in the form of records and then the physical storage of data which is in the terms of blocks. There could be 1:1 correspondence between records and blocks which is very rare which means to say that each block is one record or there could be many records per block or many blocks per record depending on how we define our record structure.

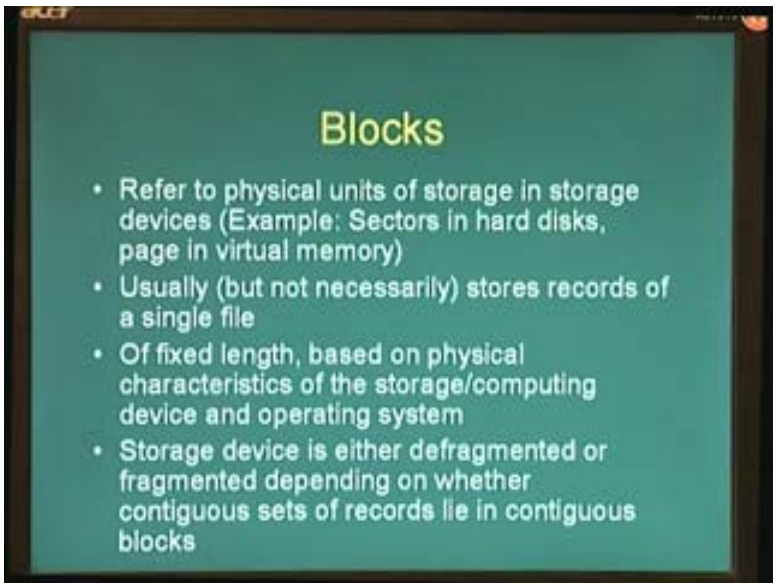
The term file organization refers to the way in which records are stored in terms of blocks and in the way blocks are placed on the storage medium and are interlinked so that they can be accessed from wherever. There are three different kinds of file organizations that we are going to see in this session today. The first one is what is called as the unsorted or the pile file or organization. And the second kind of organization is what is called as the sorted file organization and lastly we are going to look at hashing file organization.

(Refer Slide Time: 0017:09)



Let us briefly look at the notion of record and blocks which is important for us to understand these different kinds of file organizations. A record like we have mentioned earlier represents a tuple in any relation. It is a logical unit of data which of inter related data which is of interest to the user or the database management system. A file is defined as a sequence of records and records could either be fixed length or variable length. Remember in SQL for example you can use variable length strings and variable length integers and so on. So the length of a given record could also be variable or fixed and records comprise of a sequence of different fields and fields is the same as, a field is the same as a column in SQL parlance or in attribute in relational parlance.

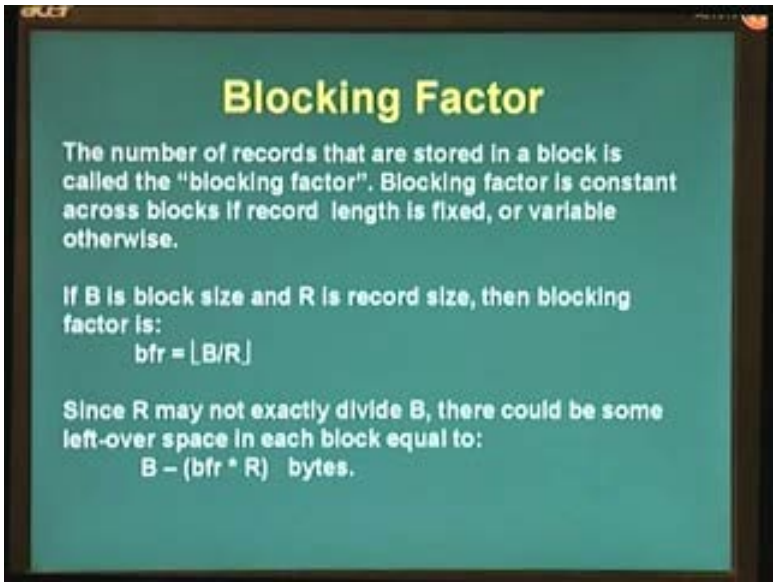
(Refer Slide Time: 00:18:09)



Blocks: So blocks like we mentioned earlier is the physical unit of data transfer or data storage in storage devices. They correspond to for example sectors in hard disk or page in virtual memory systems and so on. They store, usually a block stores records from a single file but it need not necessarily be such a case it depends on the file system structure. Blocks are usually of fixed length, blocks cannot be of varying length unlike records and the length of a block is dependent upon physical characteristics of the storage device and also of the operating system and the RAM and so on.

So many times the database management system itself does not have much control over the size of a block. A storage device is termed to be either defragmented or fragmented depending on whether contiguous sets of blocks on the storage device belong to the same file or to different files. We now define a term called the blocking factor which is important to determine how records are packed within blocks. The blocking factor is the number of records that are stored in a block on a given storage device.

(Refer Slide Time: 00:19:19)



It is constant across blocks, this blocking factor is constant across blocks if record length is fixed. On the other hand if record length is variable then the blocking factor is also variable because the number of records per block may vary from block to block. Blocking factor is simply defined as the number of blocks divided by or the size of block divided by the size of the total number of records. So bfr as shown in the slide is B divided by R where B is block size and R is record size and this is a floor function that is which takes a lower integer value of this division.

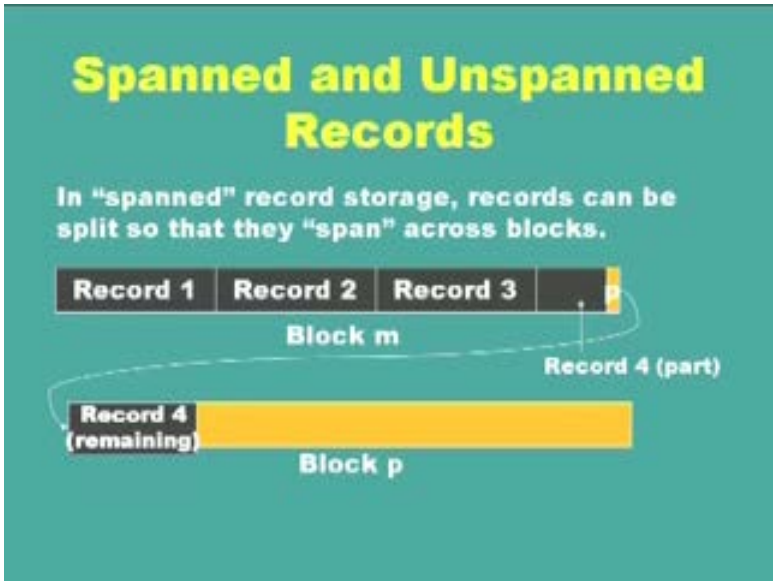
Since the record size may not exactly divide block size, there would be some amount of wastage which is given by this formula that is in each block there is a wastage of this much amount of bytes that is blocking factor times the record size number of bytes. So how is this wastage managed? There are two kinds of approaches to managing this wasted block area when records are stored within blocks.

The first is to do nothing that is don't use, let wasted spaces be. Such kinds of techniques are used in what are called as un spanned records that is a record may not span multiple blocks.

(Refer Slide Time: 00:20:41)

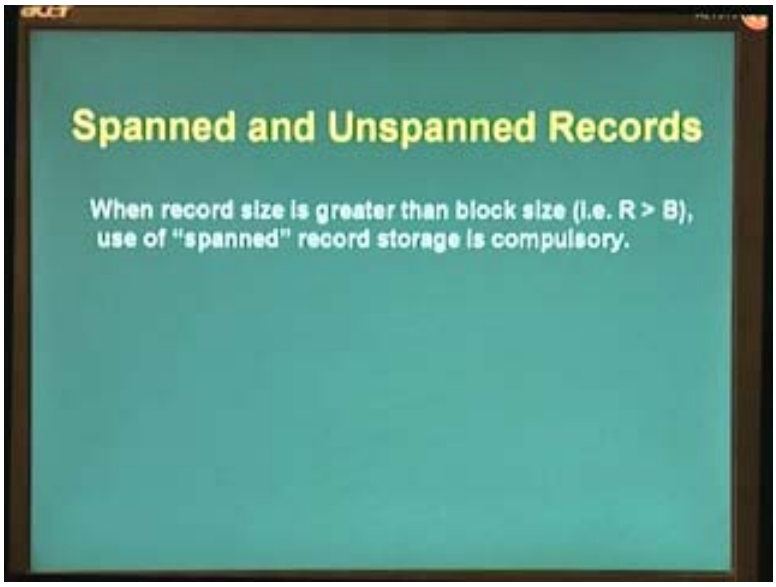
This slide shows such an example. The slide shows one block which spans from here to here comprising of three different records and there is no space for a fourth record. However there is some extra space that is left in the block which is left unused.

(Refer Slide Time: 00:21:28)



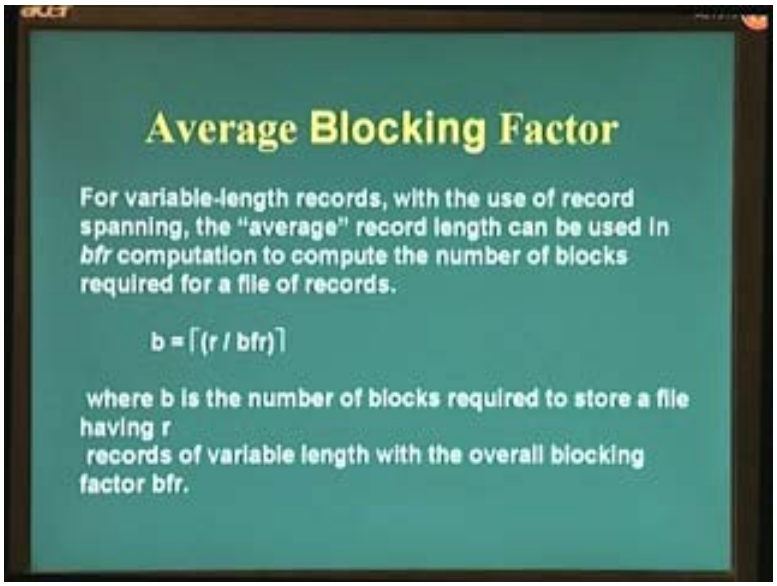
On the other hand there are what are called as spanned records. A spanned record is something that can span across different blocks. This slide shows such an example, this block contains three records and there is not enough space for the fourth record. However part of the fourth record is placed in the remaining space leaving a small amount of space for a pointer to point to the next block in the logical sequence of record. And wherever the next block begins, the remaining part of the fourth record is stored and then the next records are stored here. So such kinds of record organizations are termed to be spanning organizations where a record can span across multiple blocks. I am sure you would have noticed that if record size is bigger than the block size, we have to necessarily use spanning organization for storing records because we cannot store records into blocks otherwise.

(Refer Slide Time: 00:22:30)



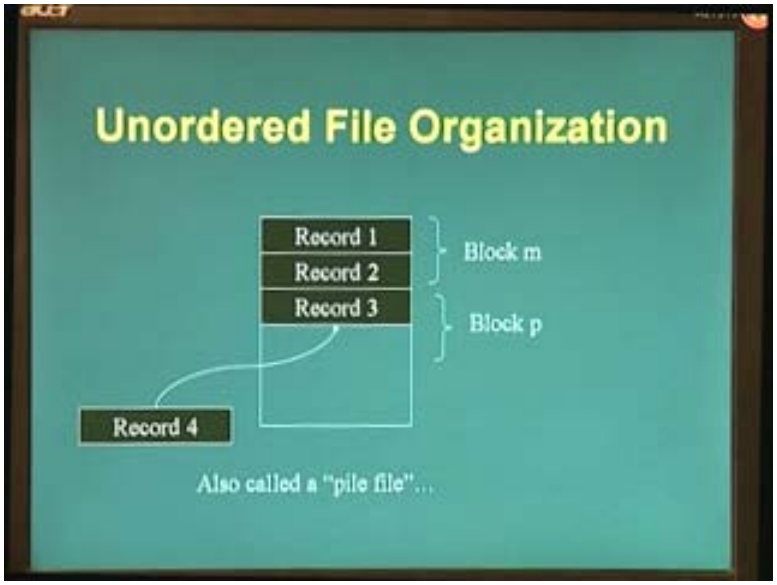
So this is what this slide says that is when record size is greater than the block size that is R is greater than B then usage of spanned records is compulsory. When we have variable record sizes or when we use spanned record allocation, we can term what is called as the average number of blocks that are required per or we can compute what is called as the average number of blocks required for storing a collection of records. So this can be, in order to compute this we first simply compute the blocking factor that is the block size divided by the average size of each record.

(Refer Slide Time: 00:22:41)



Then the following formula where r is the number of records divided by the blocking factor will give the number of blocks that is required for storing a particular database that is particular set of records.

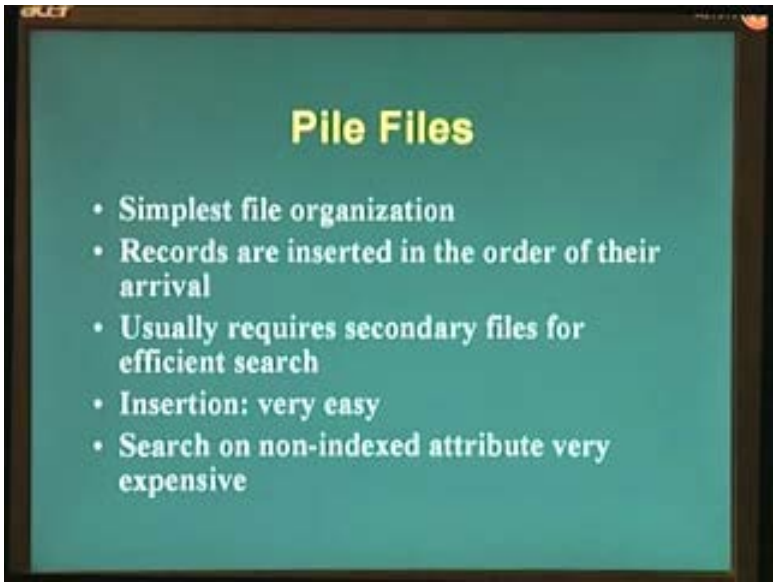
(Refer Slide Time: 00:23:30)



Let us now start with the different kinds of file organization techniques. The first kind of file organization technique that we are going to see today is what is called as the unordered file organization. This is also termed as a pile file where the term meaning that records are just stored as a pile inside the file. This slide shows such an example that is records are coming into the system and they are just being appended to the file record 1,

record 2, record 3, record 4 without request to what data that are contained in these records and how they are going to be searched. And of course these data, these records eventually go into different blocks and they are managed in some fashion by the operating system underneath in the machine.

(Refer Slide Time: 00:24:23)



A pile file is the simplest form of file organization, we don't have to do anything for organizing this file and insertion of records is the simplest that is records are inserted in the order of their arrival. And on the other hand if we have to search this file, we usually need some kinds of auxiliary files or we require some kind of help in order to efficiently search these files for a given data element. Therefore insertion is very easy however searching is extremely expensive because we will have to do a linear search, we have to just search through the entire file in order to find the data element that we require.

(Refer Slide Time: 00:25:07)

What about deletion in pile files? Deletion possesses yet another tricky problem in pile files and which can create certain kinds of fragmentation problems. Have a look at the slide here, the slide shows a pile file containing three records and some part of the block is empty and there are other blocks as well in the file. Now suppose that record two has to be deleted. Now once we delete record two and empty the space, we cannot reclaim back this space because the insertion algorithm for a pile file is not cognizant of this extra space that it can use.

The insertion algorithm simply inserts records at the end of the file, it just appends records to the file. Therefore such a kind of deletion strategy is inefficient in terms of space usage. When we are using variable length records in a pile file, we encounter another unique problem and this is of record modification. Whenever some data is modified in a record as long as the record is of fixed length, it does not matter we can make the modification in place and write it back into the file. However if we allow for records to have variable size and the modification results in the size of the record to grow, there may not be enough space to write back the record. This slide shows such an example.

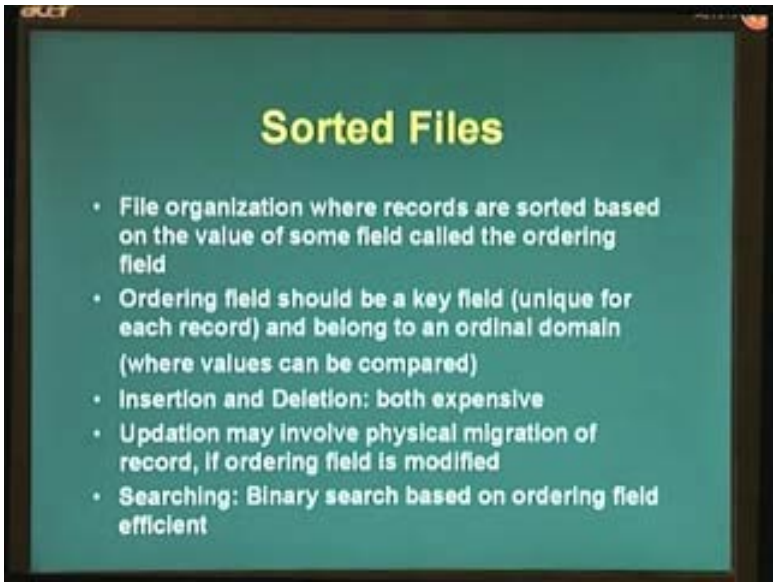
(Refer Slide Time: 00:26:06)

There are three records in this file or rather four records in the file record 1, record 2, record 3 and record 4 and record 2 is modified. Now the modification is such that the size of record 2 increases. Now we cannot write back this record at the same place where it was earlier. Therefore we will have to mark this, mark the earlier record as deleted or unused or something like that and write back record 2 at the end of the file, this is shown in the figure here.

And of course we need to update any kind of auxiliary data structures that point to record 2, so that it points to the new location in the file. The second kind of file organization that we are going to consider are what are called as sorted files. Sorted files are those files which are physically sorted on the disk based on some field called the ordering field. So the file is actually or rather physically sorted on the disk. So when you read the file on disk in a particular order, it provides or it returns back records which are sorted based on the ordering field.

Ordering field should be a key field or I should or it is recommended that the ordering key, ordering field should be a key field that is it should be unique for each record and should belong to an ordinal domain.

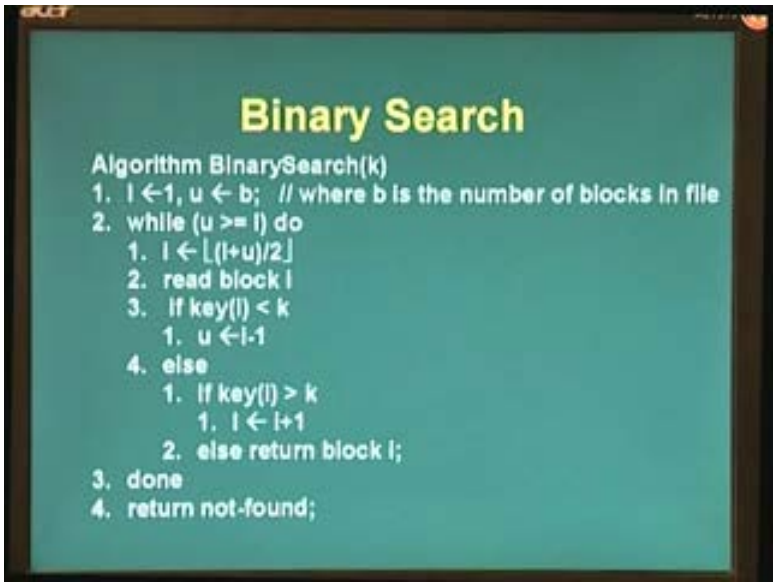
(Refer Slide Time: 00:27:27)



What is an ordinal domain? An ordinal domain is something where you can establish a total order among elements of the domain. For example the set of all integers is an ordinal domain. The set of all names for example is not an ordinal domain, we cannot place one name with respect to the other unless of course we impose some kind of an ordering like say lexical ordering. We say, we order the names as per the lexical rule that is a comes before b, b comes before c and so on. In sorted files insertion and deletion are both expensive because we have to ensure that the file remains sorted at all times. Especially when a new record is inserted with ordering field which has to go somewhere in the middle of the file rather than at the end of the file. And updation of a record may actually involve physical migration of the record, especially if the ordering field is modified.

However searching in a sorted file is made simpler because we can use what is termed as binary search. What is binary search? We shall not go into too much details of binary search, let me just give a small algorithm of what a binary search looks like. Essentially the binary search technique is a technique where we divide the search space by half that is into in each iteration of the set that is we reduce the search space to half of the previous search space in each iteration.

(Refer Slide Time: 00:29:34)



So this slide shows a simple algorithm for binary searches. We see in the first step here that we start with two bounds left and or lower and upper bound. The lower bound is at one that is the first element or the first record and the upper bound is said to b that is where b is the number of blocks in the file or the last block in the file and in each iteration we are going to compute the mid point of these bounds that is l plus u divided by 2 is the midpoint.

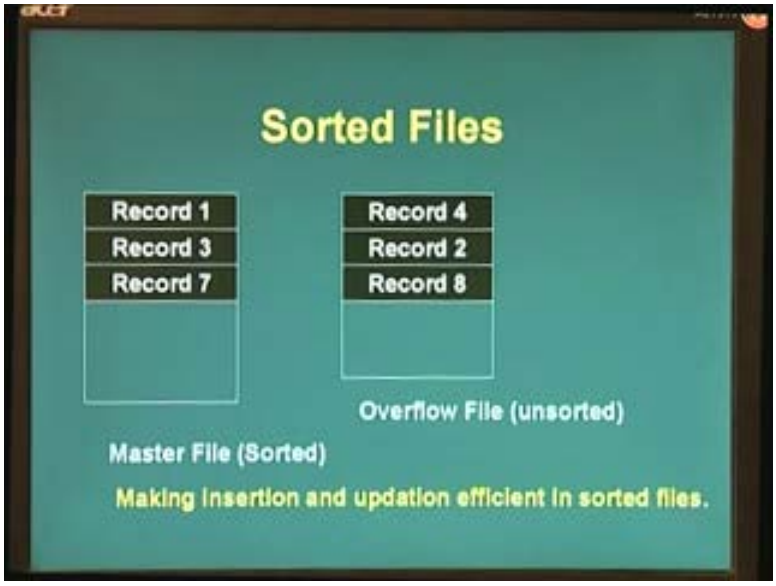
Now suppose we have to search for a given key value k, we say we read the records from form block i that is a midpoint and then see if and compare that with the required key attribute. Now there could be three different options that is one is the key attribute is equal to the key that is read from the block in which case we have found the record therefore return, we return a success.

On the other hand a key attribute could be less than the midpoint in which case we have to search the lower half of our search space. That is we have to search between 1 and i minus 1 that is l and i minus 1. On the other hand if key attribute is greater than the midpoint, we have to search in the upper half of the database that is we have to search between i plus 1 and u. So this series of steps is performed until and unless or as long as u is greater than or equal to l that is the upper bound is greater than or equal to lower bound.

Whenever they cross that is whenever the upper and lower bounds cross without having found the given record, we are able to conclude that the record does not exist in the file and then we say it is not found. So binary search, we are not going to detailed analysis of binary search here. However one can verify that a binary search technique requires an order of what is termed as log n where n is the number of blocks in the file. A binary search requires an order of log n number of disk accesses where as a linear search which actually searches through the file requires an order of n number of block accesses for

searching. That is on an average n by 2 number of block accesses have to done for a linear search whereas only log n to base 2 number of accesses need to be performed for a binary search.

(Refer Slide Time: 00:32:45)



Let us now look at one more technique by which sorted files can be made more efficient in terms of insertion and updation. Note that whenever a sorted file has, whenever a new record has to be inserted into a sorted file, it is always a problem because the file has to be always sorted physically on the disk. It's not a logical sorting that is being performed here. Therefore whenever a new record is being inserted as shown in the slide here, let us say record with key one is already in the file, next key value of 3 is already in the file and key value with 7 is already in the file. And now we receive a record whose key value is 4.

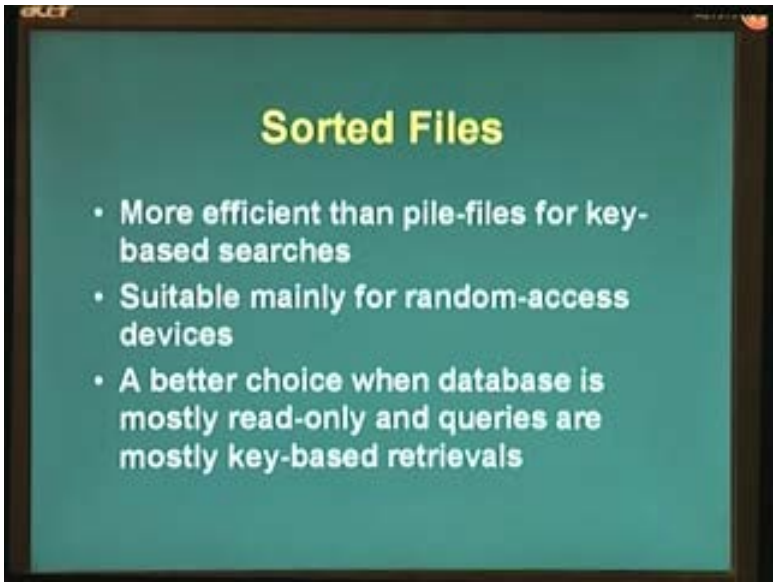
Now what do we do with this record? In fact this record has to appear between record 3 and record 7 in the file physically on the file. That means that we have to physically move record 7 below and then insert record 4 here so that the file remains sorted. This is an extremely expensive operation, especially if lots of insertion operations are taking place. In order to mitigate this problem another technique what is called as the overflow file is used.

An overflow file is a secondary file or another file where records are stored in an unsorted fashion. Whenever a new record is being added to the database, it is just added to the record or the overflow file in the form of a pile file that is it's just appended to the overflow file. And periodically that is in a less frequent fashion, let us say once in a month or once in a weak or on a weekends or something like that, the overflow file is merged into the master file or the actual sorted file.

That is the overflow file is first sorted and there are a number of merge algorithms that can take two sorted files and merge them together in an efficient fashion and using this,

the over flow file is merged back into the master file. When such a technique is used for sorted files, searching also becomes a little bit different from a pure binary search. That is whenever a key has to be searched, we can perform a binary search and the master file and incase the key is not found in the master file, we have to perform a linear search in the overflow file. So there are two kinds of searching that has to be done when overflow files are used in sorted file organizations.

(Refer Slide Time: 00:35:35)

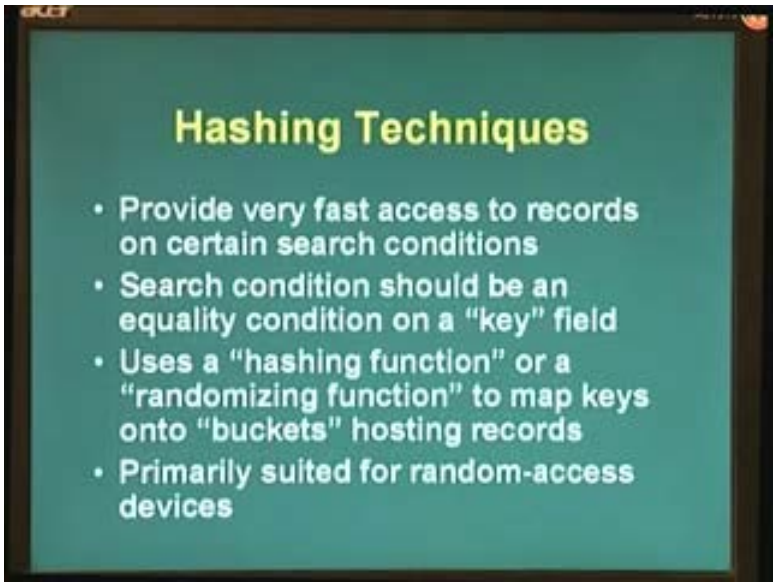


So to summarize sorted files, we see that the sorted files are more efficient than pile files especially for key base searches. However they are suitable mainly for random access devices. Note that if we have to perform a binary search and binary search over a tape device, we may have to keep moving back and forth in the tape device quiet often which makes it more, which makes it terrible inefficient. Therefore binary searches are more suitable for random access devices rather than sequential devices.

It's a better choice where a database is mostly read only because insertion is always a problem, insertion and updation is a problem. We have to use overflow files or physically move records and merge and so on and mostly queries are key based retrievals. The third kind of file organization that we are going to see today, the third and the last kind of file organization that we are seeing today is what is called as hashing file organization.

What is meant by hashing? Hashing is a means of providing very fast access to records on certain search conditions. And what are these search conditions? These search conditions are usually the equality condition based on a key field. That is whenever I what to search a record having a particular key attribute, note that it is not something like whose keys are less than a particular key attribute are greater than a particular key attribute and so on. This is useful only when we are searching for records whose key attributes are equal to the attributes attribute that is given in the query.

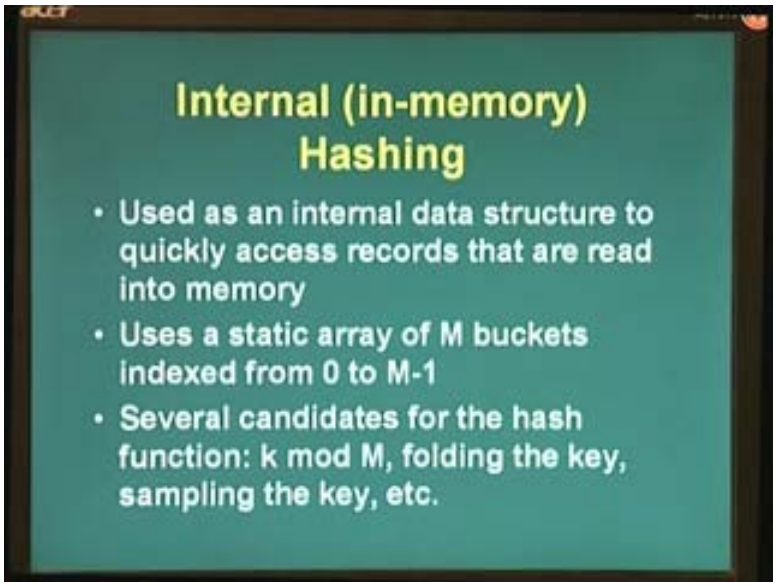
(Refer Slide Time: 00:36:28)



Hashing techniques uses what are called as hashing functions or which are also termed as randomizing functions that map particular keys into buckets for hosting records. And just like sorted file techniques, even hashing techniques are primarily suited for random access devices. Before we go into how hashing is performed on disks, let us have a look at what is called as internal hashing. Internal hashing is hashing that is performed entirely in main memory and in most database management systems, hashing is used extensively in main memory in order to quickly access a given data element among a set of data elements that have been loaded onto memory.

So usually such kinds of hashing techniques uses an internal data structure that has something like static array of M different buckets and they are indexed from 0 to M minus 1 and they are several candidates for such a hashing function. A simple candidate is to just compute the mod or the remainder of the key with M that is the M where M is the number of buckets for the hashing function.

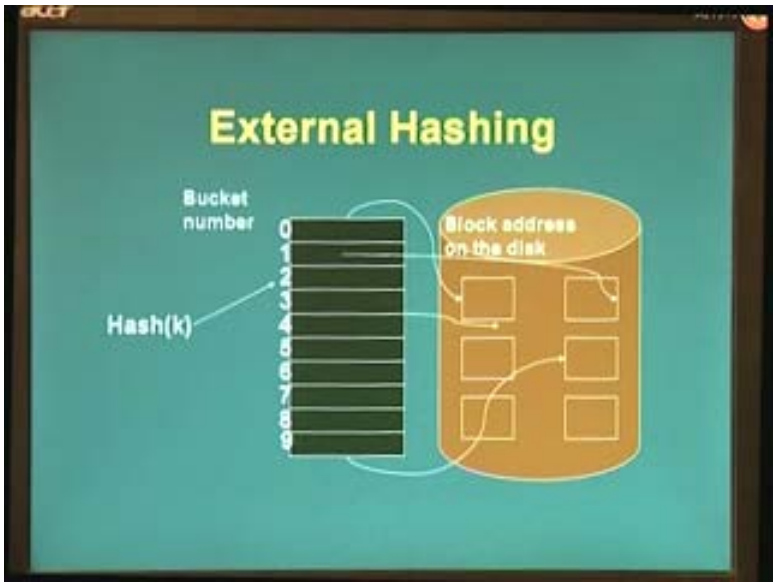
(Refer Slide Time: 00:37:39)



There are also other hashing algorithms called folding the key where a given key attribute is twisted and folded in different ways in order to come out with a number, that is uniformly randomly distributed across the set of all buckets that form the array static array of hash buckets. There are also other techniques like sampling the key and so on which we are not going to be seeing here. External hashing is the hashing technique that is used for managing data on disks rather than in memory.

External hashing comprises of blocks on the disks which act as buckets and in turn are augmented by one or more blocks that hold the hashing array itself or the set of buckets that form the hashing array itself. So the figure here shows a typical hashing process given a particular record with a key attribute K, it is first put through a hashing function which is called hash of K here and this hashing function maps it onto a particular index entry in an array of buckets. Each index entry here in turn has an address of one or more blocks which form this bucket.

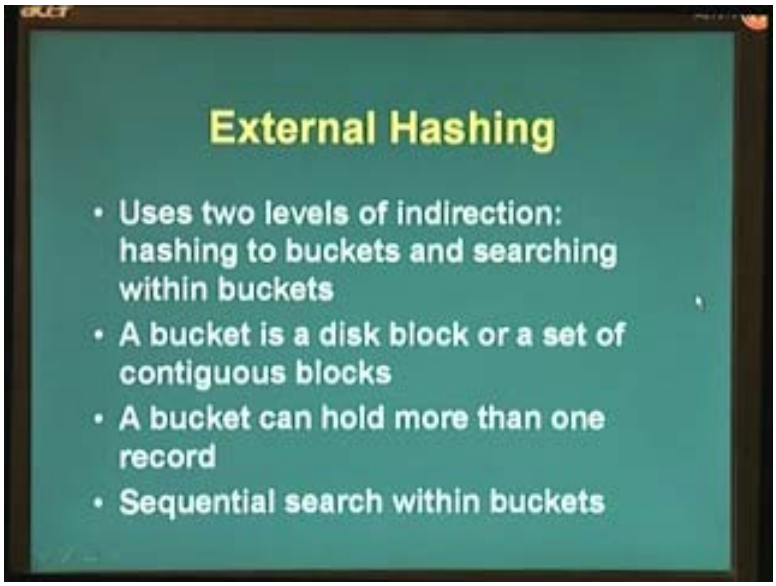
(Refer Slide Time: 00:39:02)



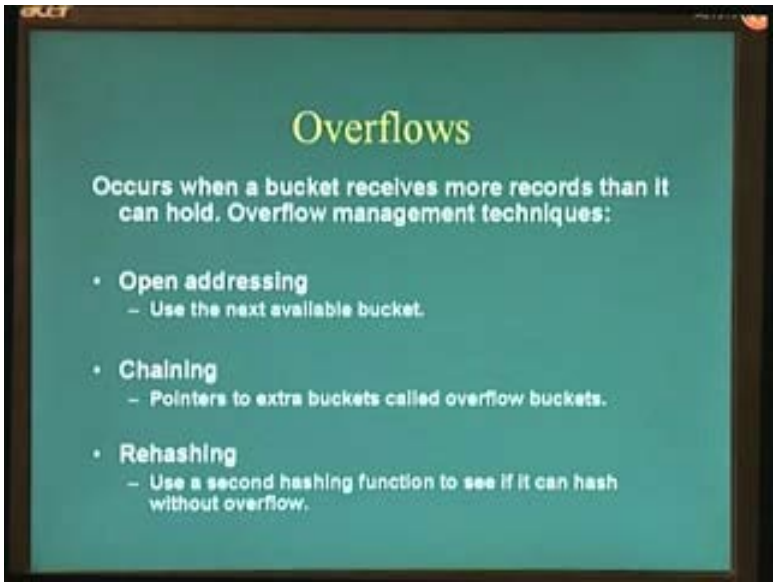
Usually one block can be allocated to one bucket or it could also be more than one blocks that are allocated to one bucket. Once the block is identified, this record is just appended to this block. So whenever a search has to be made on this key, we have to make a sequential search within the block. However we can reduce a search space drastically especially when there are large number of records on to a single block or a set of blocks that form a given hash bucket.

So to summarize external hashing again, external hashing uses two levels of indirection that is hashing into buckets and searching within buckets. A bucket is usually one disk block or a set of contiguous blocks which is also important here that is there is no point having non-contiguous blocks as part of hashing because we again need to store some information on how to access these blocks from one another. A bucket can hold more than one record obviously and also this depends on records size and we have to perform a sequential search within a bucket.

(Refer Slide Time: 00:40:31)



(Refer Slide Time: 00:41:10)



Hashing has to content with a contentious issue of what is called as overflows. What happens if we choose a hashing function that tries to hash every key onto a very small number of buckets. It especially, this can especially happen when the dataset itself could be could be skewed even if the hashing function that we have chosen is a reasonable one. That is given a set of keys that are uniformly distributed over a given range, this randomizing function uniformly distributes it over the set of all buckets.

However if a dataset itself is skewed, we have a large number of key values near a particular value rather than all across the range then the hashing function would also be

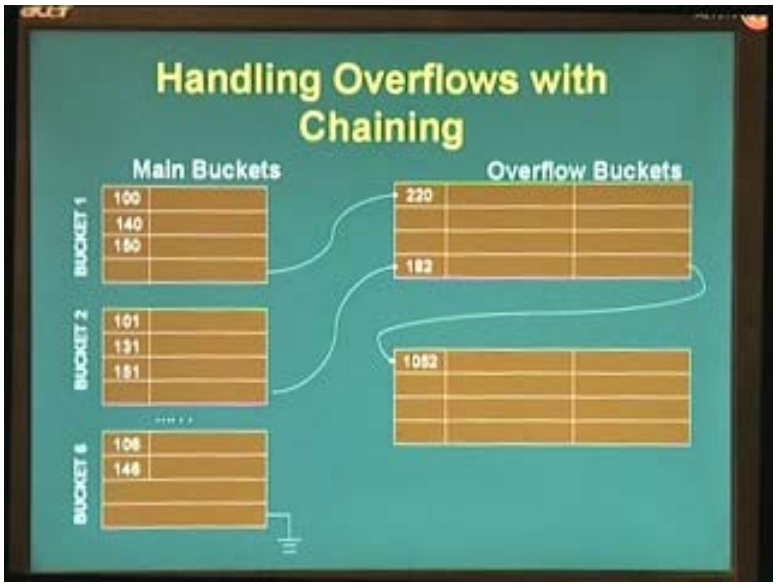
correspondingly skewed. In such cases what happens is that buckets could get overflowed that is the number of records stored in a bucket could go beyond the capacity of the bucket itself. In such cases there are several techniques that are used for overflow management. There are three different techniques that are primarily used; the first one is what is called as open addressing. Open addressing simply says that if this bucket is full just use the next available bucket which has some space in it.

So once hashing function hashes on to a particular bucket and then we find that it is full. We start a sequential scan or a linear scan of the bucket space for the next available bucket in which we can store the record. The second kind of technique that is used for managing overflows is what was called chaining. Chaining is the technique of maintaining a link list of different buckets so that when a particular bucket is full, it maintains a pointer to another disk block or another set of disk blocks acting as another bucket which can hold some more data in them and so on and when that becomes full there is another chain and so on.

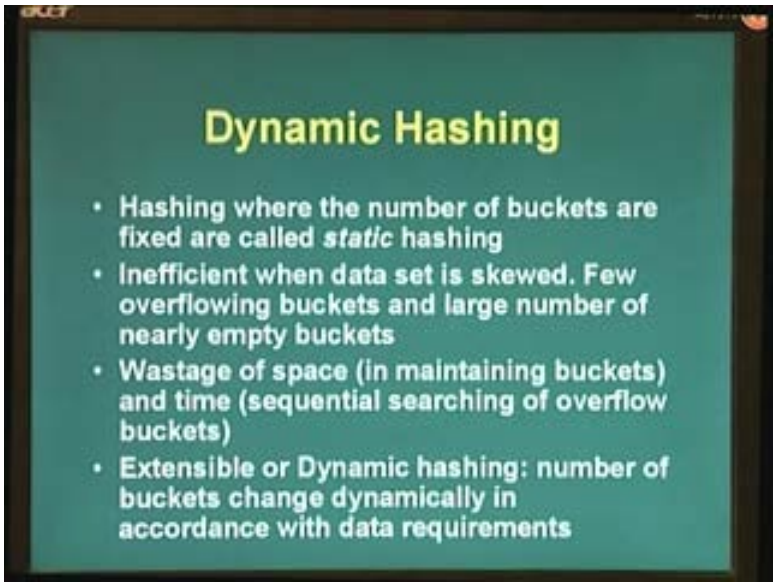
However because hashing has to perform sequential searches within buckets, if we encounter or if we end up with a long chain of buckets it becomes terribly inefficient in terms of searching. Then the third kind of technique is what is called as rehashing where we try to use another hashing function. If the first hashing function doesn't work that is maps to an overflow bucket, we use hashing function two and then see if it works and then hashing function three or whatever and then combine it with something like open addressing or chaining in order to manage overflows.

So this slide here shows the concept of chaining where there are main buckets here which in turn have pointers to overflow buckets and these pointers point to exact records in these overflow buckets and each record here has a next record pointer which points to the next overflow record that are managed by this overflow buckets.

(Refer Slide Time: 00:43:57)



(Refer Slide Time: 00:44:25)



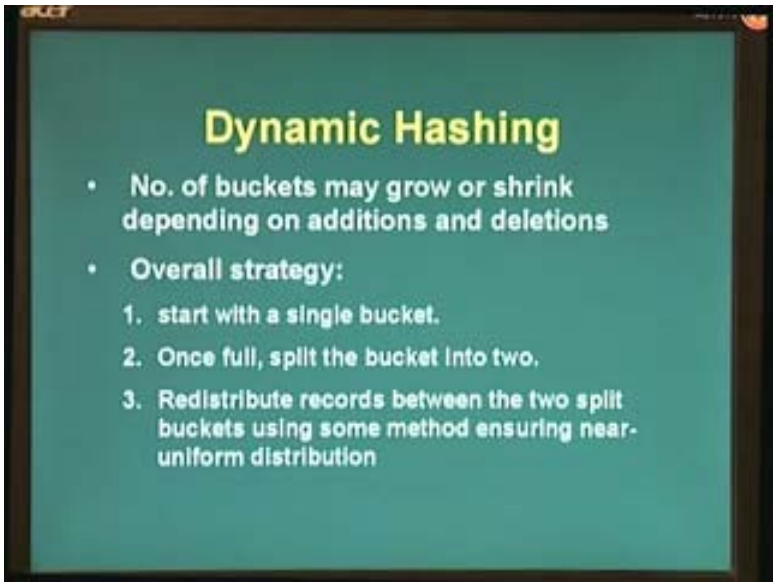
Until now we have been looking at kinds of hashing where the bucket space or the number of buckets is fixed. Such kinds of hashing techniques are what are called as static hashing techniques. And we have already seen what is the limitation of a static hashing technique. A given hashing technique might work generally that is a given hashing function might be good enough so that if the sets of keys are uniformly distributed, the hashing is also more or less uniformly distributed.

However when the set of keys are skewed, when the data itself is skewed using a static hashing might be terrible inefficient because some amounts of buckets could be overflowing while a large number of other buckets could be more or less empty. In order to obviate this need, we use what is called as dynamic hashing. Dynamic hashing is the process where the number of buckets can change dynamically can grow or shrink with time as and when keys are being added or deleted from the database. The overall strategy in dynamic hashing is quite simple and it is shown in the following three steps.

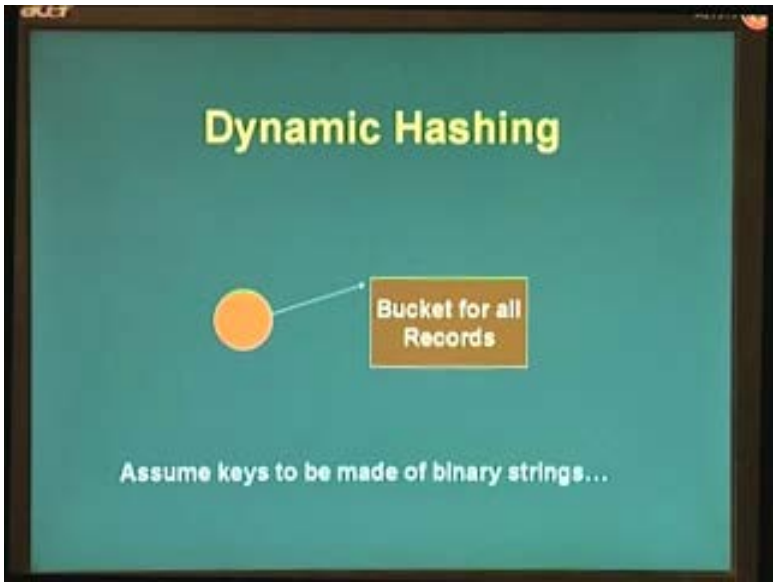
The first is we just start with a single bucket to begin with and we hash everything on to this bucket. Once the bucket is full we split the bucket into two separate buckets and then continue with the hashing process. This is and then we redistribute the records or the data that is stored within a bucket such that they are more or less uniformly distributed across the two different buckets. This process continuous, the process of splitting continuous whenever there is an overflow and then there is the process of merging that happens whenever there is an underflow that happens that is when a bucket becomes empty. We now look at a simple dynamic hashing technique where, which was how buckets can be split and merged.

The slide shows one such technique here we see that we have a small diagram here which shows two different kinds of nodes or data structures. This kind of data structure, the circle here is what is called as an internal bucket and the square or the rectangle here is what is called as the leaf bucket or an external bucket. The leaf buckets or those which actually store the data. Initially all data is stored in a given, in a single bucket that is this is the bucket for all records.

(Refer Slide Time: 00:46:17)



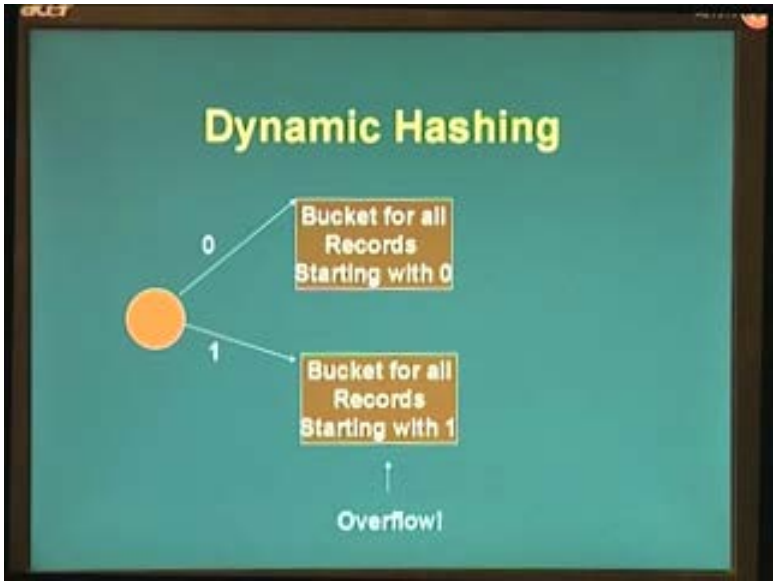
(Refer Slide Time: 00:46:19)



And assume that now we our keys are made of binary strings and then we are storing all of our data with these keys in these records.

Now suppose there is an overflow that happens here in this bucket. Now what happens when there is an overflow?

(Refer Slide Time: 00:47:19)

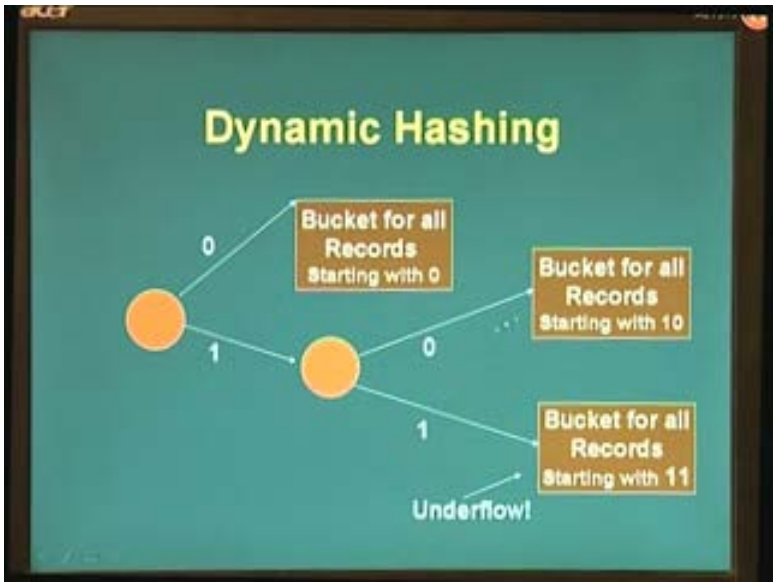


This slide shows such an technique that is when there is an overflow, the bucket is split into two different buckets and you can notice the labels here for the edges joining these buckets. The first bucket is the set of all records whose keys start with 0 and the second bucket is the set of all records whose key start with 1 assuming that our keys are made of binary strings.

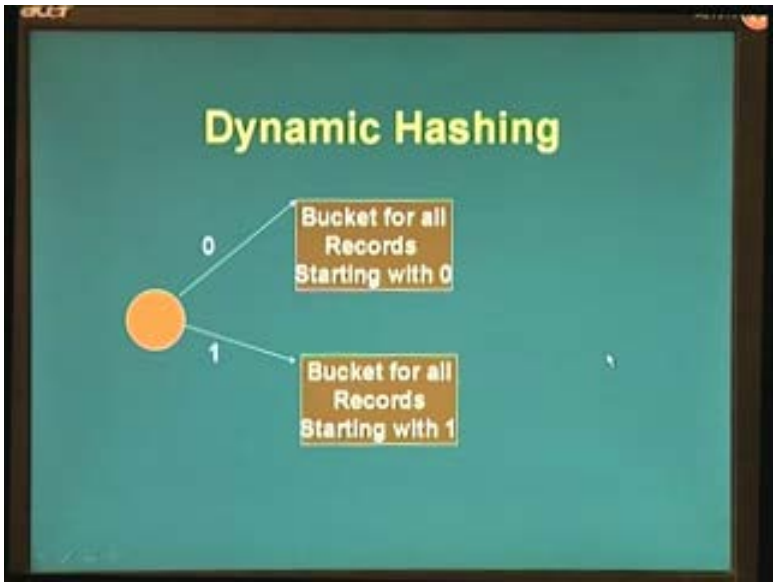
Now suppose there is again an overflow in this bucket and there is no overflow in the upper bucket here, what happens to the hash table then. The hash table then changes to this following data structure where the overflow bucket is split and a new internal node is created and two different buckets are then created. So this bucket now is the set of all, holds the set of all records whose key start with 1 0 whereas this bucket holds the set of all records whose key start with 1 1. So we can trace that starting from the start node here, so 1 0 takes us to this bucket and 1 1 takes us to this bucket.

What happens now if one of this bucket encounters an under flow that is it becomes empty when records are deleted from the database. We just have to merge this bucket with its partner so to say that is seen in the diagram here that this is the bucket who whose edge is labeled as one. We have to just merge it with its partner whose edge is labeled zero.

(Refer Slide Time: 00:47:56)

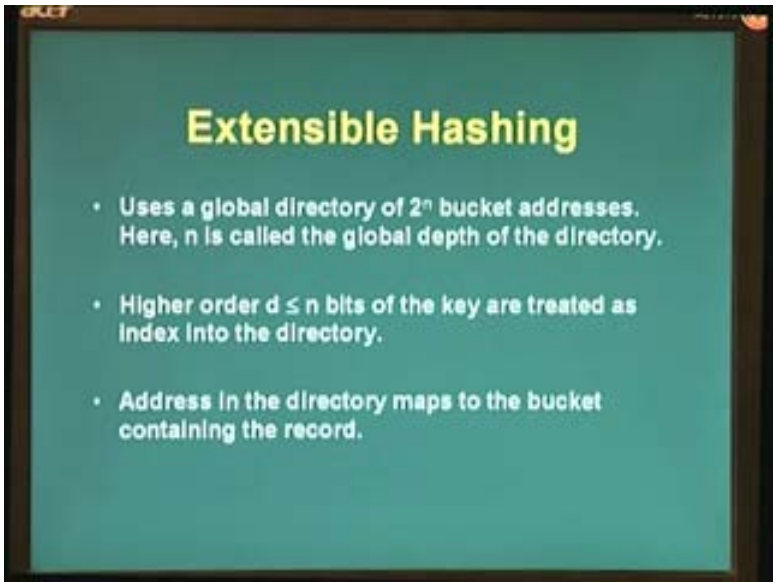


(Refer Slide Time: 00:48:57)



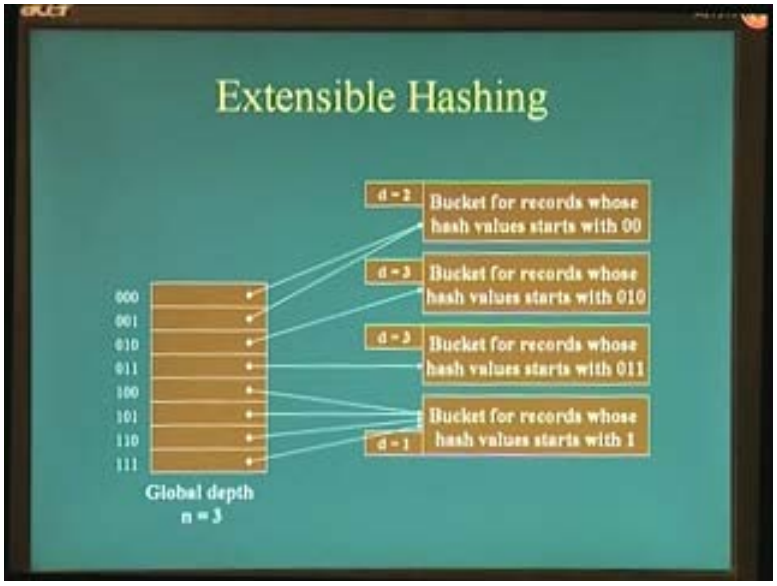
So this takes us back to the previous configuration where we had only two buckets in the hash table. There is another kind of dynamic hashing what is called as extensible hashing which also uses a similar kind of hashing technique in order to grow and shrink buckets.

(Refer Slide Time: 00:49:03)



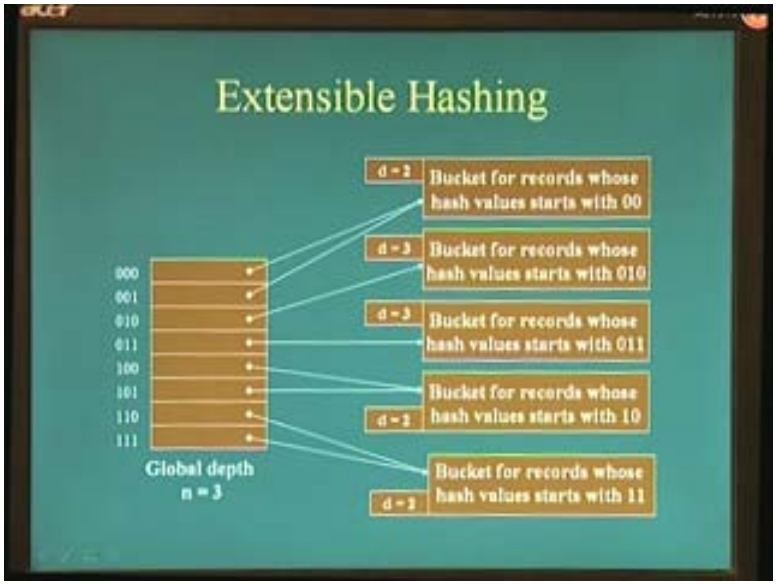
Extensible hashing uses what is termed as a global directory of 2 power n number of bucket addresses where n is called the global depth of the directory. And then each bucket is uniquely identified by some set of higher order bits d number of bits which is less than or equal to n which can uniquely identify each bucket. And of course buckets are split and merged whenever they overflow or underflow and correspondingly n is changed that is the global depth is either increased or decreased in a corresponding fashion.

(Refer Slide Time: 00:49:50)



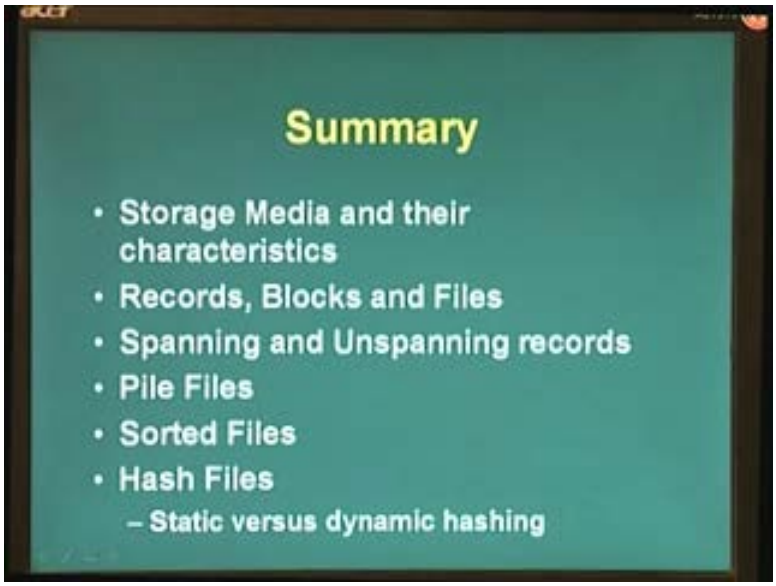
This slide shows such an example. Here we have a global depth of 3 that is n equal to 3 and there are several different bucket pointers that shows 000, 001 and so on. And there are several different buckets each with differing capacity where here this bucket says that d equal to 2. That means this bucket can be uniquely identified with just the top two bits that is all keys starting with 0 0 can go into this bucket. Here for example d equal to 1 that is all keys starting with 1 go into this bucket but here these two buckets have a large number of data elements that is where d equal to 3 that is this bucket contains all keys starting with 0 1 0 and this one contains all keys starting with 0 1 1.

(Refer Slide Time: 00:50:43)



Now suppose what happens if the last bucket overflows that is where d equal to 1? This bucket is then split so that d becomes 2 and then instead of instead of just one bit, we have to use two bits in order to uniquely identify this bucket. Therefore the top bucket here, the upper bucket here is the set of all keys which start with 1 0 and the lower bucket here is the set of all keys that start with 1 1.

(Refer Slide Time: 00:51:12)



So that brings us to the end of this session where we have looked at several kinds of file organizations and for physically managing records on storage devices. Let us quickly summarize what we have learnt in this session. We first looked at different kinds of storage media and what are their characteristics. We can classify storage media into different kind's volatile, non-volatile, primary, secondary and so on. In fact they can be placed in a hierarchy and then there are different characteristics like random access, sequential access or read-only versus read-write or write-once and so on and then there could be either character devices or block devices and so on.

Each of them, each of these characteristics impact the kind of data structure that we can use for storing records. We then looked at the concepts of records, blocks and files which are the terminology we use for dealing with data that are physically stored on to disks. Records are the logical unit of data that are stored while blocks are the physical unit of data that is used for data transfer and file is the set of records or a sequential records in which typically a relation is stored. We also saw the notion of spanning and un spanning of records in terms of how they affect the blocking factor or the number of blocks that are required to store a particular file of records. We then saw three different kinds of file organizations, the first one was the pile file organization which is the simplest where we just append records into a file.

However which poses problems with insertion or rather with deletion and updation and also with search. We then also saw sorted files which are files that are physically sorted on disks based on some ordering attribute. Sorted files are much more efficient for search because we can use binary search on sorted files. However they pose very tricky problems in terms of insertion and updation of records and they become especially tricky when records can be of varying lengths and updations can change the key value of on which it is sorted.

We also saw the last kind of file organization called hash files where hashing function is used in order to identify the block number or the bucket in which particular record is stored. We also saw how or what are the challenges that are faced by hash functions especially when we use static hashing techniques because static hashing techniques cannot work efficiently when the dataset is skewed in which case we have to use dynamic hashing where the number of buckets in the hash table can dynamically grow or shrink as and when data is inserted or deleted. So that brings us to the end of this session.