COP 4600 – Summer 2013

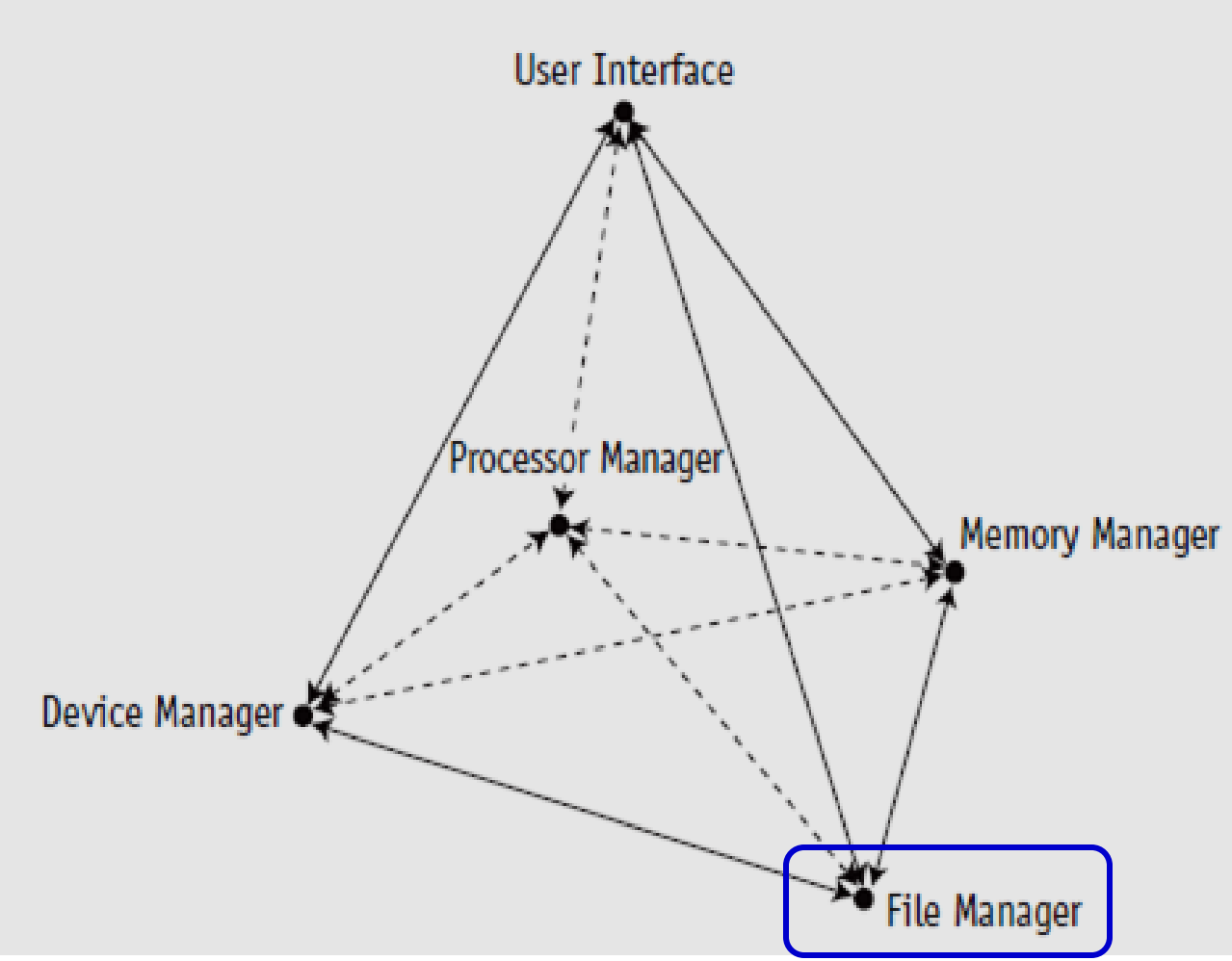
Introduction To Operating Systems

File Management

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop4600/sum2013

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

File Management In The OS



File Management In The OS

- The File Manager (FM) controls every file in the system.
- The file management system is responsible for creating, deleting, modifying, and controlling access to files – as well as for managing resources used by files.
- The FM provides support for libraries of programs and data to online users, for spooling operations, and for interactive computing.
- The functions of the FM are performed in conjunction with the Device Manager.

Responsibilities of the File Manager

- The FM has a complex job. It is in charge of the system's physical components, its information resources, and the policies used to store and distribute files.
- To carry out its responsibilities it must perform four tasks:
  1. Keep track of where each file is stored.
  2. Use a policy that will determine where and how the files will be stored, making sure to efficiently use the available storage space and provide efficient access to the files.
  3. Allocate each file when a user has been cleared for access to it, then record its use.
  4. Deallocate the file when the file is to be returned to storage, and communicate its availability to others who may be waiting for it.

Responsibilities of the File Manager

1. The FM keeps track of its files with directories that contain the filename, its physical location in secondary storage, and important information about each file.

2. The FM's predetermined policy determines where each file is stored and how the system, and users, will be able to access them simply – via commands that are independent from device details. In addition, this policy must determine who will have access to what material, and this involves two factors: flexibility of access to the information and its subsequent protection. The FM does this by allowing access to shared files, providing distributed access, and allowing users to browse through public directories. Meanwhile, the OS must protect its files against system malfunctions and provide security checks via account numbers and passwords to preserve the integrity of the data and safeguard against tampering.

Responsibilities of the File Manager

3. The system allocates a file by activating the appropriate secondary storage device and loading the file into memory while updating its records of who is using what file.

4. Finally, the FM deallocates a file by updating the file tables and rewriting the file (if it was modified) to the secondary storage device. Any processes waiting to access the file are then notified of its availability.

Some Basic Definitions

- A field is a group of related bytes that can be identified by the user with a name, type, and size.
- A record is a group of related fields.
- A file is a group of related records that contains information to be used by specific application programs to generate reports. This type of file contains data and is sometimes called a flat file because it has no connections to other files; unlike databases, it has no dimensionality.
- A database appears to the FM to be a type of file, but databases are more complex because they're actually groups of related files that are interconnected at various levels to give users flexibility of access to the stored data. If the user's database requires a specific structure, the FM must be able to support it.

Some Basic Definitions

- A program file contains instructions and data files contain data.
- As far as storage is concerned, the FM treats both program files and data files exactly the same.
- Directories are listing of filenames and their attributes and are treated in a manner similar to files by the FM.
- Data collected by the system to monitor system performance and provide for system accounting is also collected into files.
- In reality, every program and data file accessed by the computer system, as well as every piece of computer software, is treated as a file.

Communicating With The File Manager

- The user communicates with the FM via specific commands that may be either embedded in the user's program or submitted interactively by the user.
- Examples of embedded commands are OPEN, CLOSE, READ, WRITE, and MODIFY.
  - OPEN and CLOSE pertain to the availability of a file for the program invoking the command.
  - READ and WRITE are the I/O commands.
  - MODIFY is a specialized WRITE command for existing data files that allows for appending records or for rewriting selected records in their original place in the file.

Communicating With The File Manager

- Examples of interactive commands are CREATE, DELETE, RENAME, and COPY.
  - CREATE and DELETE deal with the system's knowledge of the file. Actually, files can be created with other system-specific terms: for example, the first time a user gives a command to SAVE a file, it's actually created. In other systems the OPEN NEW command within a program indicates to the FM that a file must be created. Likewise, an OPEN…FOR OUTPUT command instructs the FM to create a file by making an entry for it in the directory and to find space for it on the secondary memory.
- These commands and many more, which are the interface between the user and the hardware, were designed to be as simple as possible to use so they're devoid of the detailed instructions required to run the device where the file may be stored.

Communicating With The File Manager

- Examples of interactive commands are CREATE, DELETE, RENAME, and COPY.
  - CREATE and DELETE deal with the system's knowledge of the file. Actually, files can be created with other system-specific terms: for example, the first time a user gives a command to SAVE a file, it's actually created. In other systems the OPEN NEW command within a program indicates to the FM that a file must be created. Likewise, an OPEN…FOR OUTPUT command instructs the FM to create a file by making an entry for it in the directory and to find space for it on the secondary memory.
- These commands and many more, which are the interface between the user and the hardware, were designed to be as simple as possible to use so they're devoid of the detailed instructions required to run the device where the file may be stored.

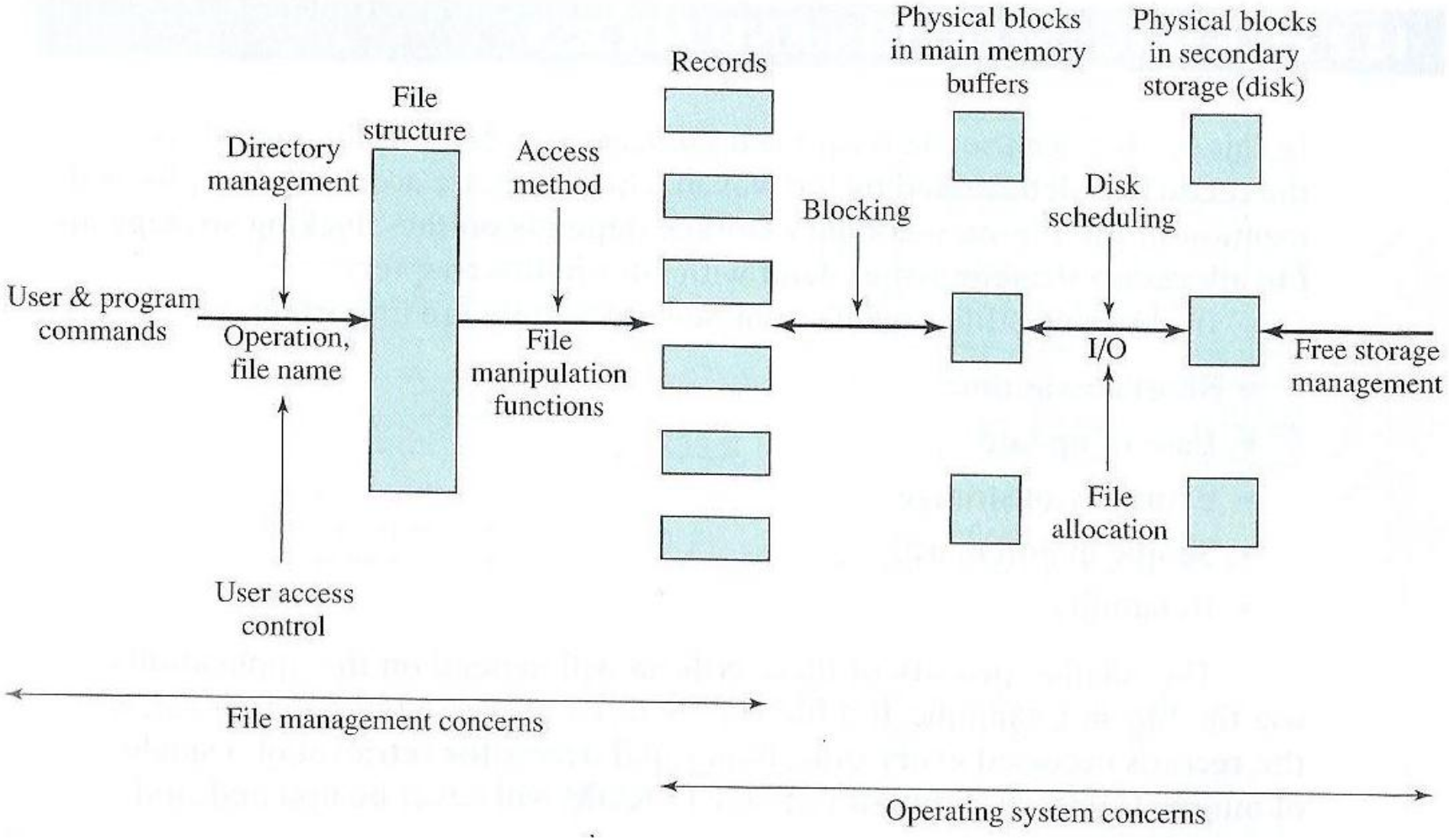
Communicating With The File Manager

- Since the commands to communicate with the FM are device independent, this frees the user from the burden of knowing, or needing to specify, where the file's exact location on the disk page (the cylinder, surface, and sector) or even the medium in which it is stored (tape, magnetic disk, optical disc, etc.).
- This is fortunate for the user because file access is a complex process.
- Each logical command is broken down into a sequence of low-level signals that trigger step-by-step actions performed by the device and supervise the progress of the operation by testing the device's status.

Functions of The File Manager

- Consider the diagram on the next page. Follow the diagram from left to right.
- Users and application programs interact with the FM by means of the commands like we just saw for creating and deleting files as well as operations on the files. Before performing any of operation, the FM must identify and locate the selected file. This requires some sort of directory that serves to describe the location of all files, plus their attributes as well as ensuring that only authorized users of the file are permitted to perform an operation.
- The basic operations that a user or application may perform on a file are at the record level.

Communicating With The File Manager



Functions of The File Manager

- The user or application views the file as having some structure that organizes the records, such as a sequential structure. To translate the user command into specific file manipulation commands, the access method appropriate to this file structure must be employed.
- Whereas users and applications are concerned with records or fields, I/O is done on a block basis. Thus, the records or fields of a file must be organized as a sequence of blocks for output and unblocked after input.
- To support block I/O of files, several functions are needed:
- Secondary storage must be managed. This involves allocation files to free blocks on secondary storage and managing free storage so as to know what blocks are available for new files and growth for existing files.

Functions of The File Manager

- In addition, as we saw when we covered device management, individual block I/O requests must be scheduled on the device in question.
- Both disk scheduling and file allocation are concerned with optimizing performance. Thus, these two functions need to be considered together. Furthermore, the optimization will depend on the structure of the files and the access patterns. Accordingly, developing an optimal file management system from the point of view of performance is an exceedingly difficult task.
- The previous diagram suggests a division between what might be considered the concern of the FM as a separate system utility and the concern of the OS itself. The point of intersection between the two being record-level processing.

File Organization and Access

- File organization refers to the logical structuring of the records as determined by the way in which they are accessed.
- The physical organization of the file on secondary storage depends on the blocking strategy and the file allocation strategy, issues that we'll address later.
- In choosing a file organization, several criteria are important:
  - Short access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability

File Organization and Access

- The relative priority of the criteria on the previous page will depend on the application that will use the file.
- For example, if a file is only to be processed in batch mode, with all of the records accessed every time, then rapid access for retrieval of a single record is of minimal concern.
- A file stored on a CD-ROM will never be updated, and so ease of update is not an issue.
- Some of the criteria will conflict. For example, for economy of storage, there should be minimum redundancy in the data. On the other hand, redundancy is a primary means of increasing the speed of access to data. An example of this is the use of indices.

File Organization and Access

- The number of different file organizations have been implemented or just proposed is unmanageably large, even for a course that just considered file systems alone.
- We'll restrict ourselves to five fundamental file organizations that are commonly implemented in many systems today.
  - The heap, or pile file
  - The sequential file
  - The indexed sequential file
  - The indexed file
  - The direct, or hashed file
- The table on the next page summarizes the performance aspects of these five file organizations.

File Organization and Access

File Method	Space Attributes		Update Record Size		Retrieval		
	Variable	Fixed	Equal	Greater	Single record	Subset	Exhaustive
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

A = Excellent, well suited to this purpose $\approx O(r)$

B = Good $\approx O(o \times r)$

C = Adequate $\approx O(r \log n)$

D = Requires some extra effort $\approx O(n)$

E = Possible with extreme effort $\approx O(r \times n)$

F = Not reasonable for this purpose $\approx O(n^{>1})$

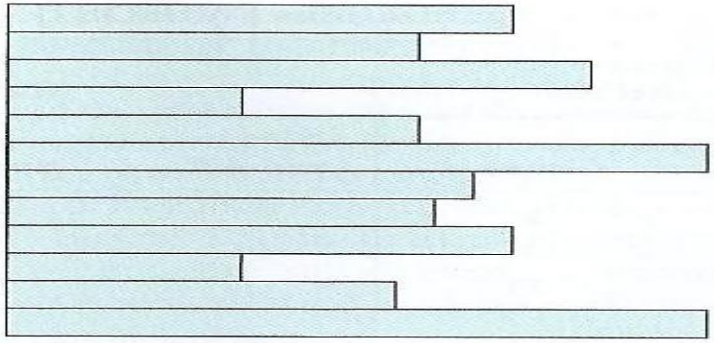
where

r = size of the result

o = number of records that overflow

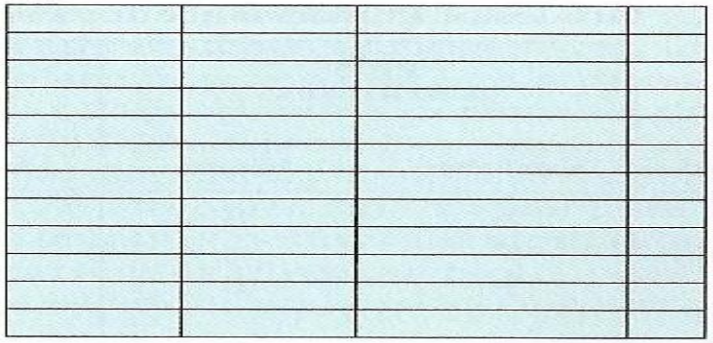
n = number of records in file

File Organization and Access



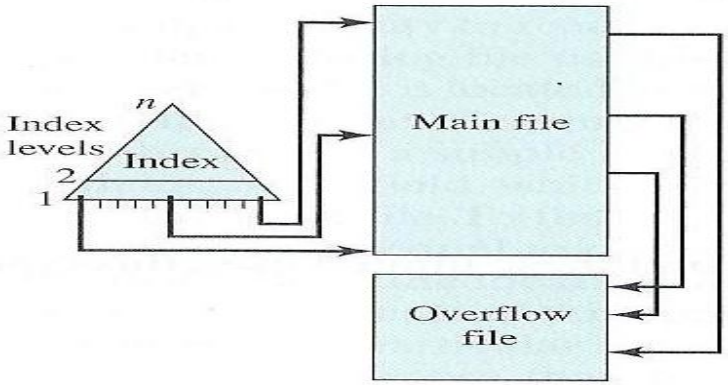
Variable-length records
Variable set of fields
Chronological order

(a) Pile file

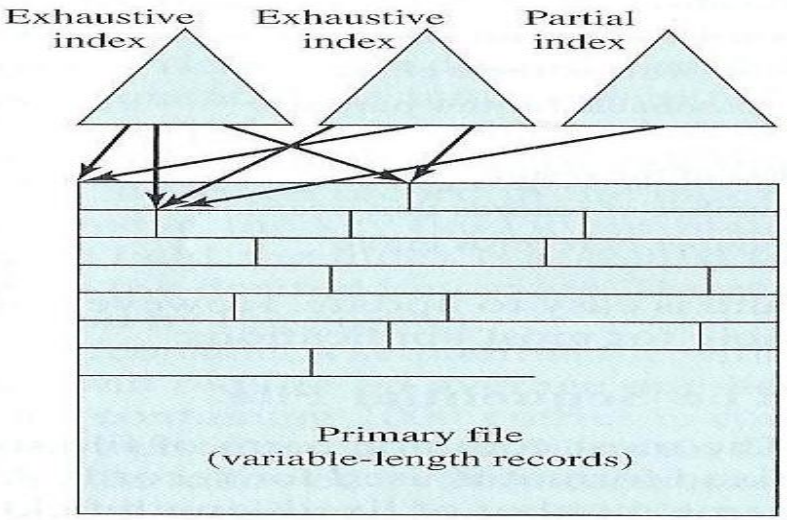


Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential file



(c) Indexed sequential file



(d) Indexed file

File Organization and Access

The Heap (Pile) File

- The least complicated form of file organization is the heap file.
- Data are collected in the order in which they arrive.
- Each record consists of one burst of data and there is no requirement that records contain the same information or be of the same length.
- The purpose of a heap is simply to accumulate the mass of data and save it.
- Thus, each field must be self-describing, including a field name as well as the value. The length of each field must be implicitly indicated by delimiters, explicitly included as a subfield, or known as default for that type of field.

File Organization and Access

The Heap (Pile) File

- Because there is no structure to the heap file, record access is by exhaustive search.
- If you wish to find a record that contains a particular field with a particular value, it is necessary to examine each record in the heap until the desired record is found or the entire file has been searched.
- Heap files are used when data are collected and stored prior to processing or when the data are not easy to organize.
- Heap files utilize space well when the stored data vary in size and structure and is perfectly adequate for exhaustive search scenarios, and is easy to update. Beyond these uses it is unsuitable for most applications.

File Organization and Access

The Sequential File

- The most common form of file structure is the sequential file. In this type of file, a fixed format is used for records. All records are of the same length and consist of the same number of fixed-length fields in the same order.
- Because the length and position of each field is know, only the values of the fields are stored; the field name and length for each field are attributes of the file structure.
- On particular field, usually the first field in each record, is referred to as the key field. The key field uniquely identifies the record; thus key values for different records must always be different.
- Records are stored in key sequence.

File Organization and Access

The Sequential File

- Sequential files are typically used in batch applications and are generally optimum for such applications if they involve processing of all the records.
- The sequential file organization is the only file structure that is as easily stored on tape as it is on disks.
- For interactive applications that involve queries and/or updates of individual records, the sequential file provides poor overall performance.

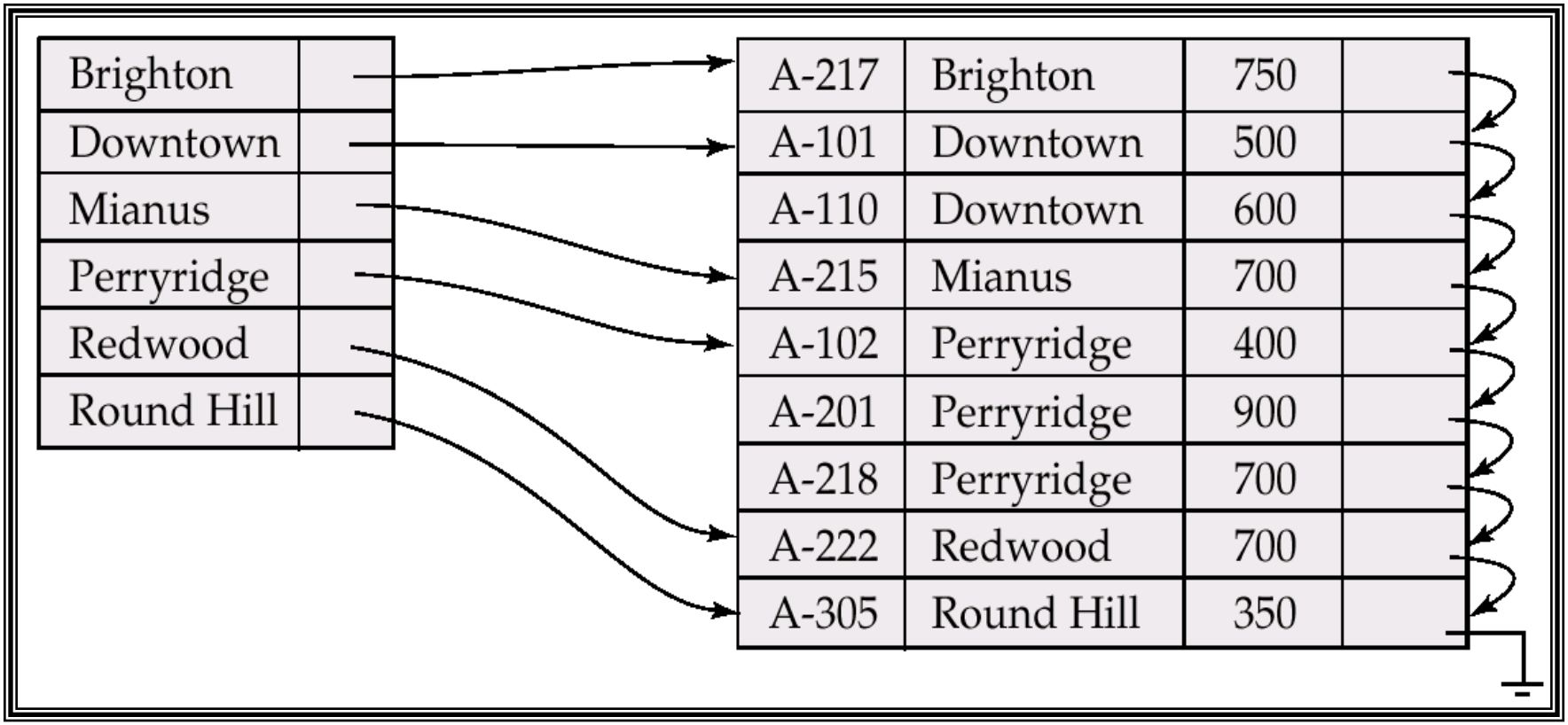
File Organization and Access

The Indexed Sequential File

- A popular approach to overcoming the disadvantages of the sequential file is the indexed sequential file. This file organization maintains the key characteristic of the sequential file in that records are organized in sequence based on a key field. However, two additional features are added: an index to the file to support random access, and an overflow file. (See page 21.)
- In the simplest case, a single level of indexing is used. In this case the index is simply a sequential file.
- Each record in the index file consists of two fields: a key field, which is the same as the key field in the main file, and a pointer into the main file. To find a specific record, the index is searched to find the highest key value that is equal to or precedes the desired key value. The search continues in the main file at the location indicated by the pointer.

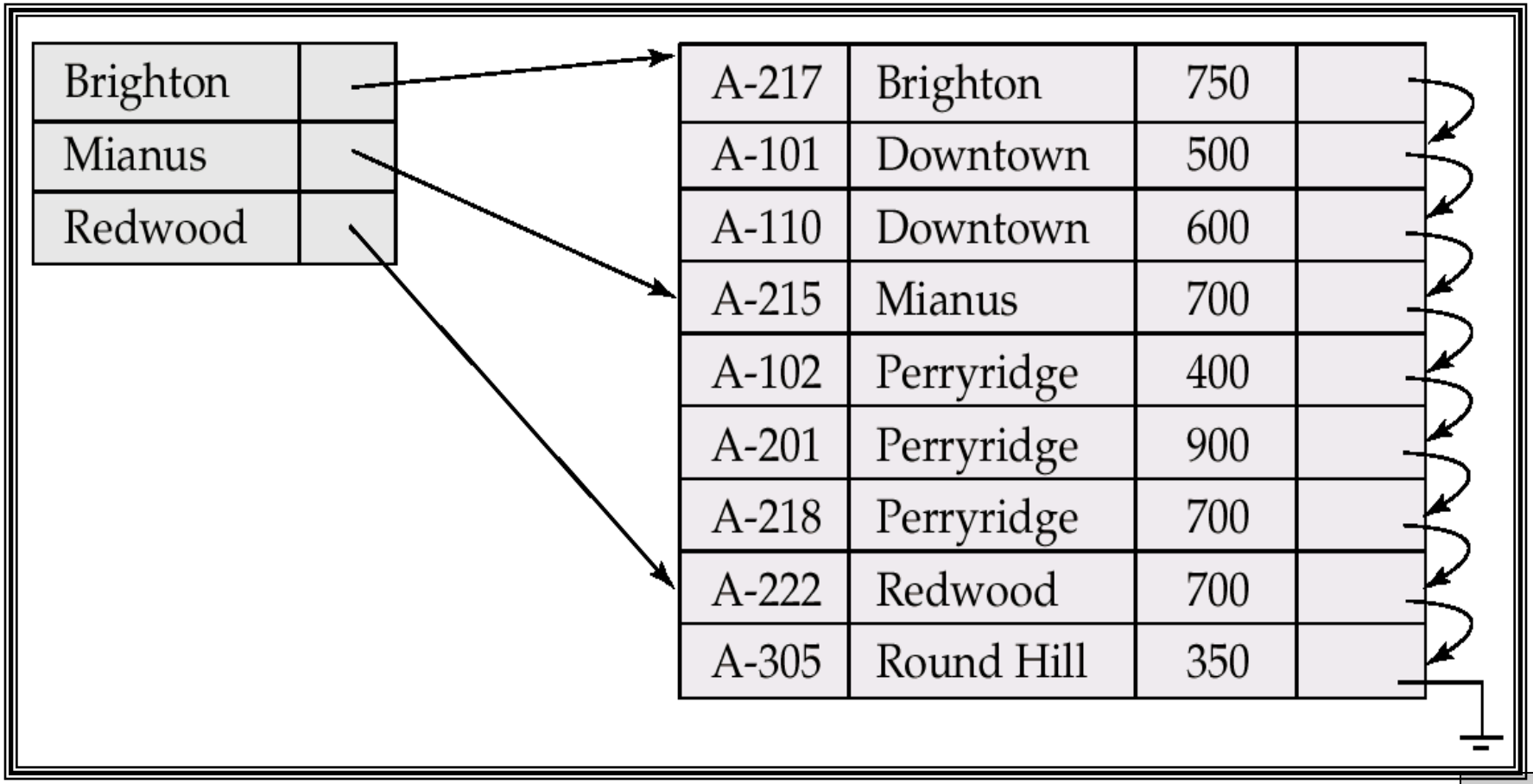
File Organization and Access

Dense Index – an index record appears for every search-key value in the file.



File Organization and Access

Sparse Index – an index record appears for some search-key values in the file.



File Organization and Access

The Indexed Sequential File

- To understand the effectiveness of the indexed sequential file organization, consider a sequential file with 1 million records. Searching for a particular record will require on average accessing ½ million records.
- Instead suppose that a sparse index is built containing 1000 entries with the keys roughly evenly distributed. Thus, each key value covers approximately 1000 records. Now, on average it will require 500 accesses to the index file followed by 500 accesses in the main file to find a specific record.
- The average search length is thus reduced from 500,000 records to 1000 records!

File Organization and Access

The Indexed Sequential File

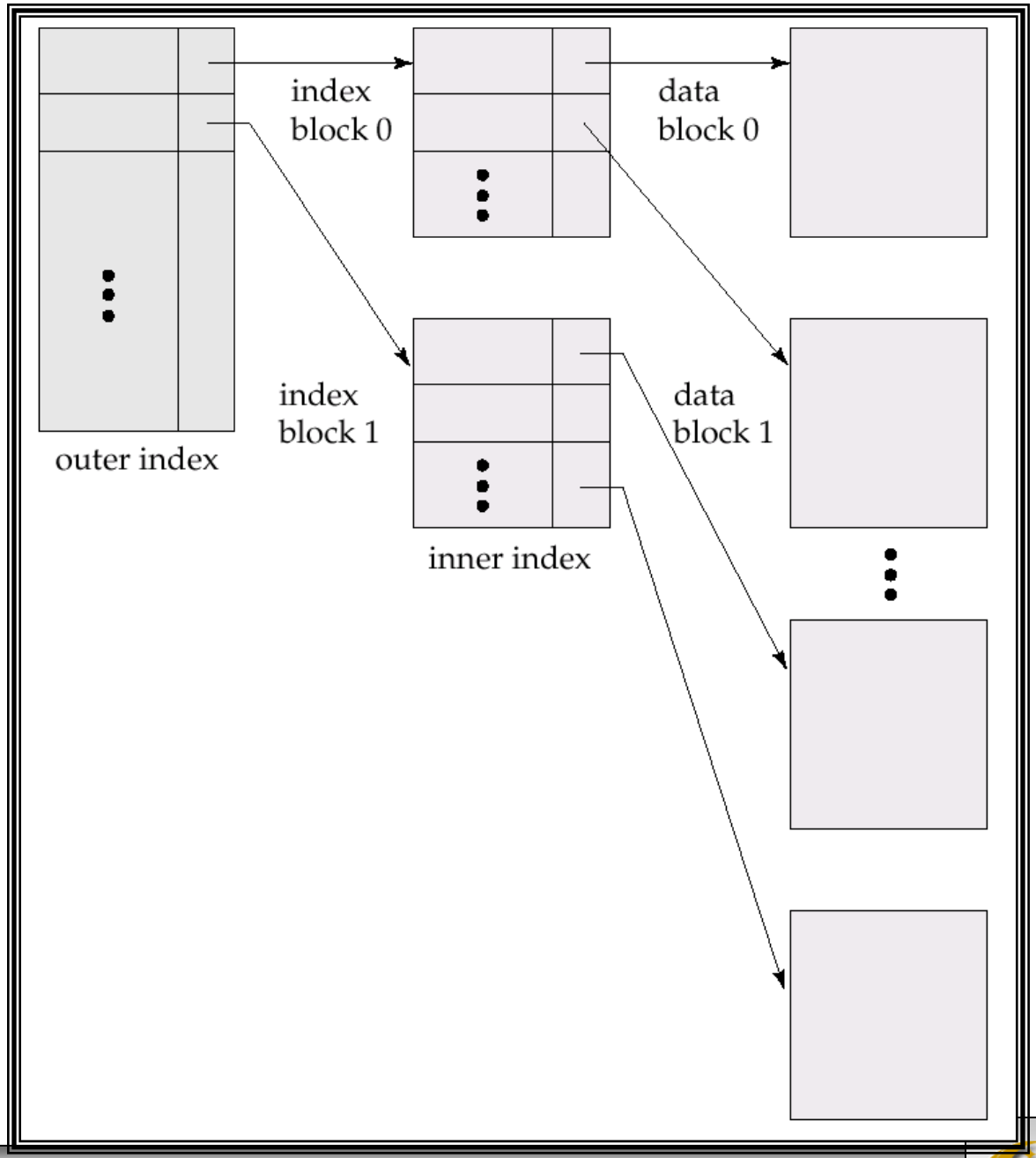
- Additions to the main file are normally handled as follows:
  - Each record in the main file contains a pointer field (not visible to an application), that points to the overflow file.
  - When a new record is to be inserted into the file, it is added to the overflow file.
  - The record in the main file that immediately precedes the new record in logical sequence is updated to contain a pointer to the new record in the overflow file. If the immediately preceding record is itself in the overflow file, then the pointer in that record is updated.
- Periodically, the overflow file is merged into the main file in batch mode during a file reorganization.

File Organization and Access

The Indexed Sequential File

- To provide even greater access efficiency, multiple levels of indexing can be used.
- The lowest level of index file is treated as a sequential file and a higher-level index is created for that file. (See next page for an illustration.)
- Using the same example file with 1 million records, assume that a low-level index is built containing 10,000 entries and a higher level index is built into that lower-level index which contains 100 entries. The search begins at the highest level index, so on average 50 records will be accessed, within the lower level index an average of 50 more records will be accessed leading to an average access of 50 records in the main file, or a total of 150 records accessed.

Example of multi-level indexing



File Organization and Access

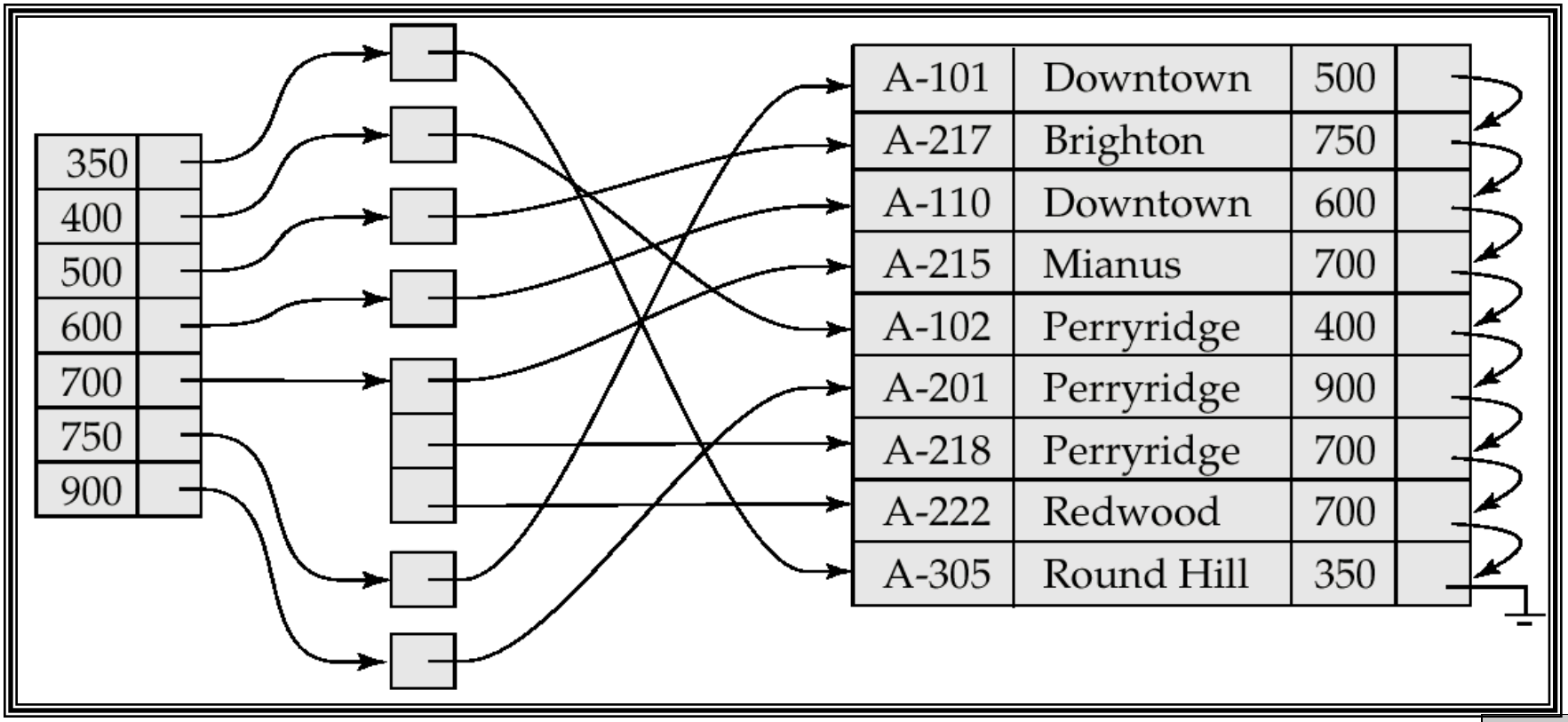
The Indexed File

- The indexed sequential file retains one limitation of the sequential file: effective processing is limited to that which is based on a single field of the file, i.e., the key field.
- When it is necessary to search for a record on the basis of some other attribute in the file, both forms of sequential files are inadequate.
- In many applications, the flexibility to efficiently search by various attributes is desirable.
- To achieve this flexibility, a structure is needed that employs multiple indices, one for each type of field that may be the subject of a search.
- In a general indexed file, the concept of sequentiality and a single key are abandoned. Records are accessed only through their indices.

File Organization and Access

The Indexed File

- Both sparse and dense indices are commonly used in indexed files. The illustration below shows a dense non-key index.



File Organization and Access

The Indexed File

- Indexed files are used mostly in applications where timeliness of information is critical and where data are rarely processed exhaustively.
- Examples are airline reservation systems and inventory control systems.

File Organization and Access

The Hashed or Direct File

- The hashed, or direct, file exploits the capability found on disks to access directly any block of a known address.
- As with sequential and indexed sequential files, a key field is required in each record. However, there is no concept of sequential ordering.
- The hashed file makes use of a hashing function on the key value.
- Hashed files are often used where very rapid access is required, where fixed-length records are used, and where records are always accessed one at a time. Examples are directories, pricing tables, schedules, and name lists.
- The next couple of pages illustrates static hashing.

File Organization and Access

The Hashed or Direct File

- A **bucket** is a unit of storage containing one or more records (a bucket is typically a disk block).
- In a **hash file organization** we obtain the bucket of a record directly from its search-key value using a **hash function.**
- Hash function *h* is a function from the set of all search-key values *K* to the set of all bucket addresses *B.*
- Hash function is used to locate records for access, insertion as well as deletion.
- Records with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate a record.

File Organization and Access

The Hashed or Direct File

Hash file organization of *account* file, using *branch-name* as key (See figure in next slide.)

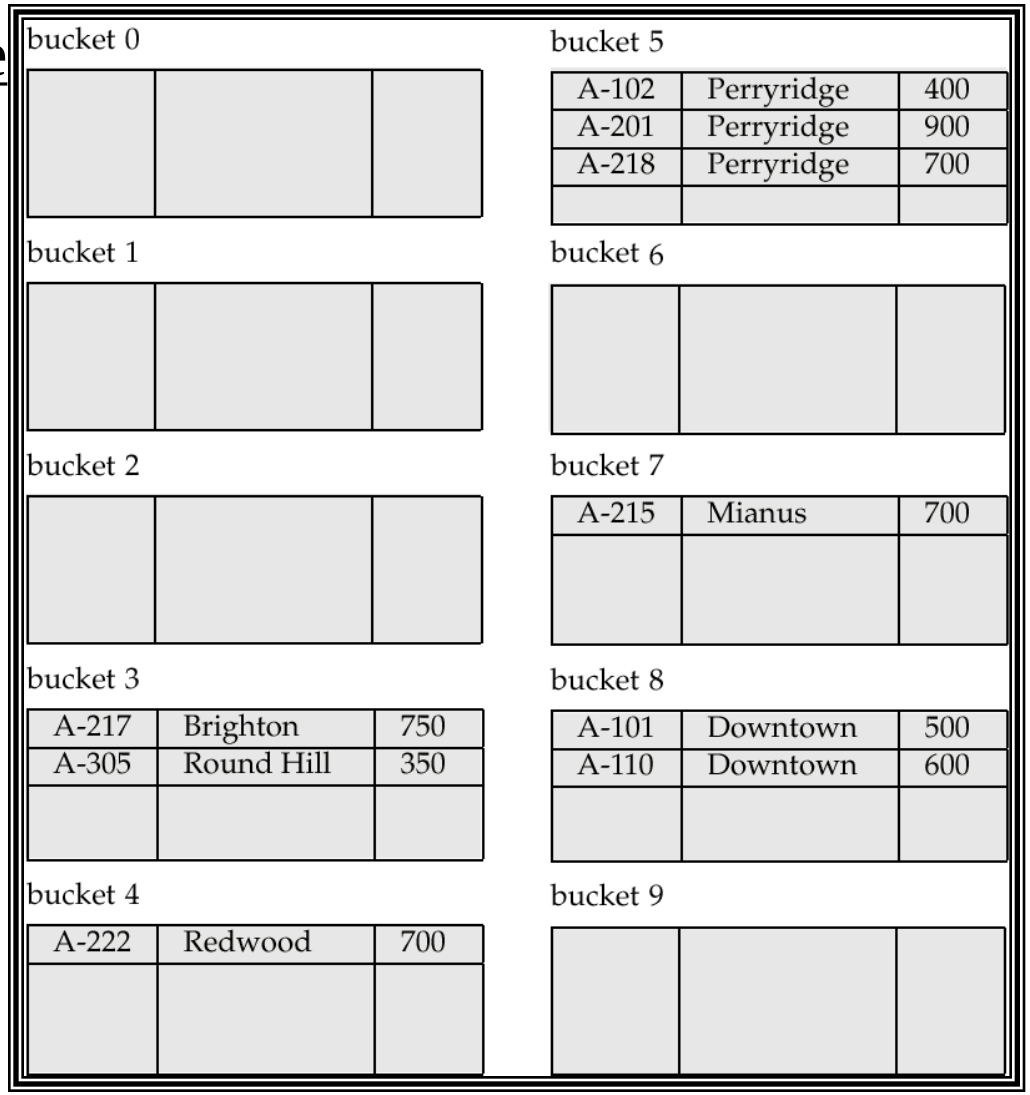
- There are 10 buckets,
- The binary representation of the *i*th character is assumed to be the integer *i*.
- The hash function returns the sum of the binary representations of the characters modulo 10
  - E.g. h(Perryridge) = 5 h(Round Hill) = 3 h(Brighton) = 3

File Organization and Access

The Hashed or Direct File

Hash file organization of *account* file, using *branch-name* as key

(see previous slide for details).



File Organization and Access

The Hashed or Direct File

- Worst has function maps all search-key values to the same bucket; this makes access time proportional to the number of search-key values in the file.
- An ideal hash function is **uniform**, i.e., each bucket is assigned the same number of search-key values from the set of *all* possible values.
- Ideal hash function is **random**, so each bucket will have the same number of records assigned to it irrespective of the *actual distribution* of search-key values in the file.
- Typical hash functions perform computation on the internal binary representation of the search-key.
  - For example, for a string search-key, the binary representations of all the characters in the string could be added and the sum modulo the number of buckets could be returned. .

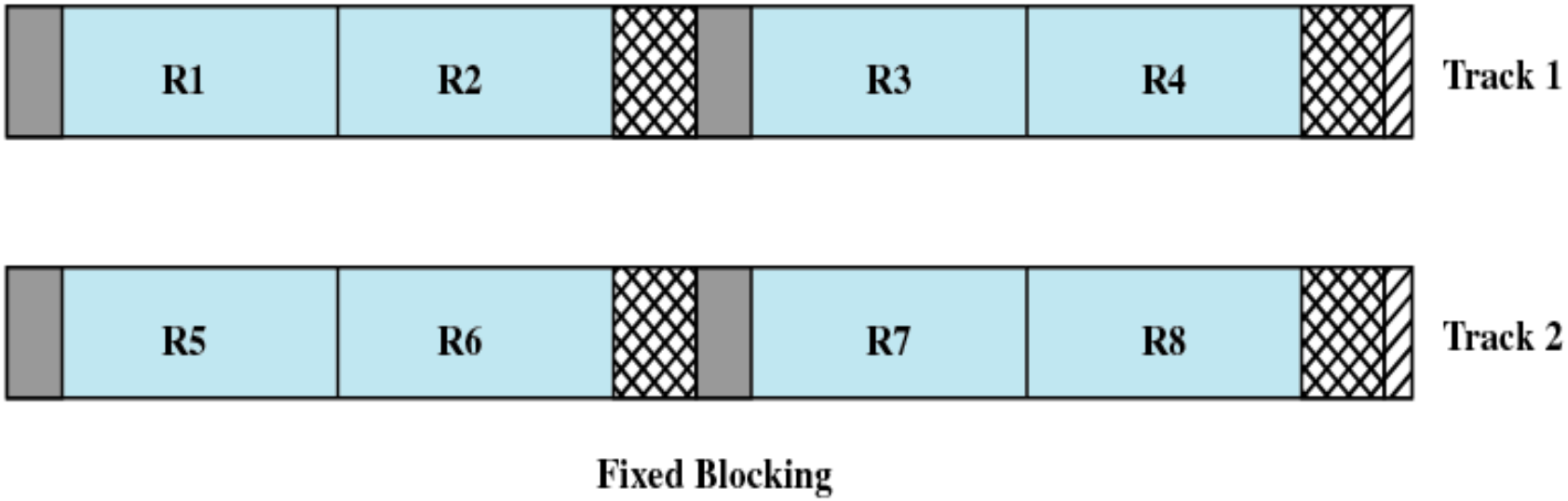
Record Blocking

- As the illustration on page 14 indicated, records are the logical unit of access of a structured file, whereas blocks are the unit of I/O in secondary storage. For I/O to be performed, records must be organized as blocks.
- There are several issues to consider:
  - First, should the blocks be of fixed or variable length? On most systems, blocks are of fixed length. This simplifies I/O, buffer allocation in main memory, and the organization of blocks on secondary storage.
  - Next, what should the relative size of a block be compared to the average record size? The tradeoff is this: The larger the block, the more records that are passed in one I/O operation. If a file is being processed sequentially, this is an advantage because it reduces the number of I/O operations. On the other hand, if records are being accessed randomly and no particular locality of reference is observed, then larger blocks result in unnecessary transfer of unused records.

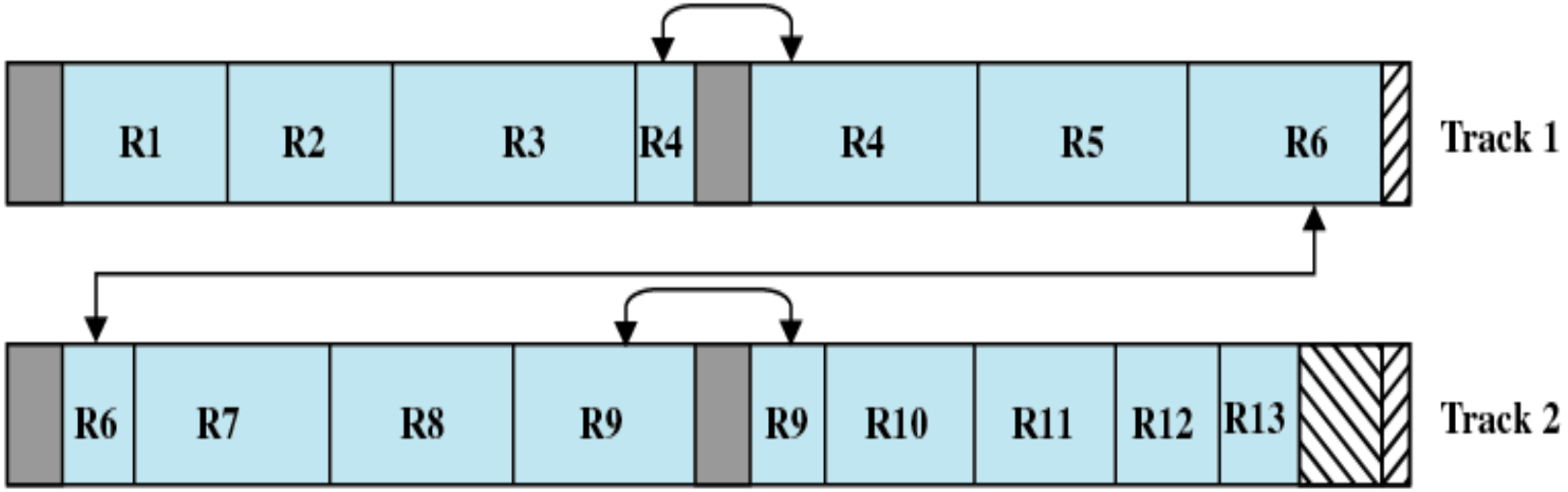
Record Blocking

- Given the size of a block, there are three methods of blocking that can be used:
  - Fixed blocking: Fixed length records are used, and an integral number of records are stored in a block. There may be unused space at the end of each block. This is internal fragmentation.
  - Variable-length spanned blocking: Variable-length records are used and are packed into blocks with no unused space (no internal fragmentation). Thus, some records must span two blocks, with the continuation indicated by a pointer to the successor block.
  - Variable-length un-spanned blocking: Variable-length records are used, but spanning is not employed. There will be wasted space in most blocks because of the inability to use the remainder of a block if the next record is larger than the remaining unused space.
- These three techniques are illustrated on the next three pages. Assumption is that the file is stored in sequential blocks on the disk.

Fixed Blocking



Variable-Length Spanned Blocking



Variable Blocking: Spanned

Data

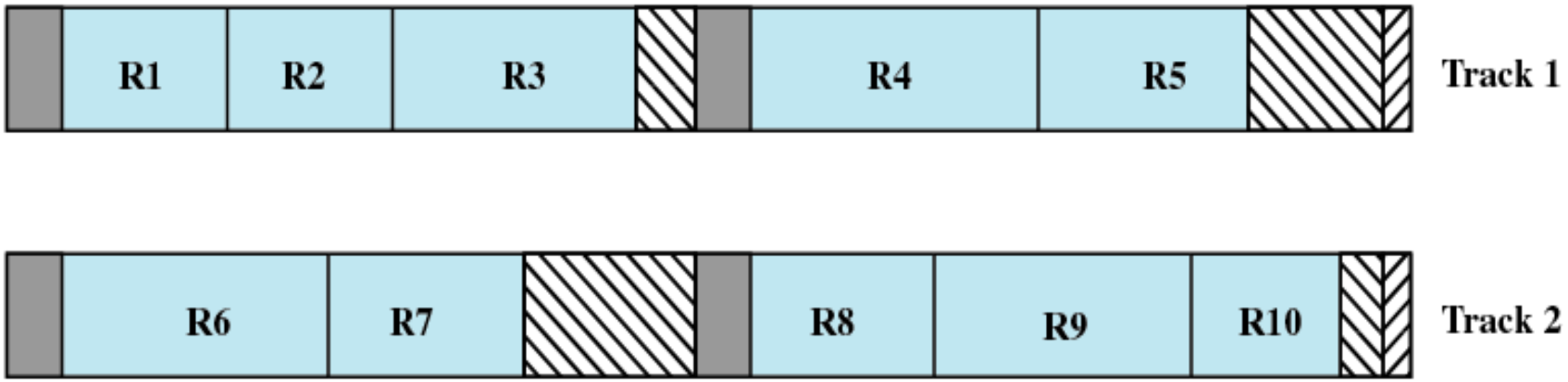
Gaps due to hardware design

Waste due to block fit to track size

Waste due to record fit to block size

Waste due to block size constraint from fixed record size

Variable-Length Un-Spanned Blocking



Variable Blocking: Unspanned

Data

Gaps due to hardware design

Waste due to block fit to track size

Waste due to record fit to block size

Waste due to block size constraint from fixed record size

Record Blocking

- Fixed blocking is the common mode for sequential files with fixed-length records.
- Variable-length spanned blocking is efficient in terms of storage space and does not limit the size of records. However, this technique is difficult to implement. Records that span two blocks require two I/O operations, and files are difficult to update, regardless of the file organization.
- Variable-length un-spanned blocking results in wasted space and limits the size of a record to the size of a block.

Record Blocking

- Another consideration with record blocking is the impact it may have on the virtual memory hardware.
- In a virtual memory environment, the basic unit of I/O transfer is a page. Pages are generally quite small, so that it is impractical to treat a page as a block for un-spanned blocking.
- Accordingly, some systems combine multiple pages to create a larger block for file I/O purposes.

Secondary Storage Management

- On secondary storage, a file consists of a collection of blocks.
- The FM is responsible for allocating blocks to files.
- This raises two management issues. First, space on secondary storage must be allocated to files, and second, it is necessary to keep track of the space available for allocation.
- The two tasks are related in that the approach taken for file allocation may influence the approach taken for free space management.
- There is also some interaction, as we will see, between the file structure and the allocation policy.

File Allocation

- There are several issues involved in file allocation:
  1. When a new file is created, is the maximum space for the file allocated all at once? Preallocation or dynamic allocation?
  2. Space is allocated to a file as one or more contiguous units (which we'll call portions). Thus, a portion is a contiguous set of allocated blocks. The size of a portion can range from a single block to the entire file. What size of portion should be used for file allocation.
  3. What sort of data structure or table is used to keep track of the portions assigned to a file? As example of such a structure is a file allocation table (FAT) found on DOS and some other OS.

Preallocation Versus Dynamic Allocation

- A preallocation policy requires that the maximum size of a file be declared at the time of the file creation request.
- Some applications can reliably determine this value, others cannot, and would thus probably tend to overestimate the file size required so as to not run out of space. This would clearly be wasteful in the long term.
- Thus, the advantage leans heavily toward dynamic allocation schemes. These will allocate space to a file in portions as needed.

Portion Size

- The second issue deals with portion size allocated to a file.
- At one extreme would be a portion large enough to hold the entire file. At the other extreme would be a portion allocation of one block at a time.
- In choosing a portion size policy there is a tradeoff between efficiency from the point of view of a single file versus overall system efficiency. Four items must be considered:
  - Contiguity of space increases performance, especially for `retrieve_next` operations.
  - Having a large number of small portions increases the size of the tables needed to manage the allocation information.
  - Having fixed-size portions (e.g., blocks) simplifies the reallocation of space.
  - Having variable-size or small fixed-size portions minimizes waste of unused storage due to over allocation.

Portion Size

- Since the four items on the previous page interact and must be considered together, the result is that there are only two major alternatives:
  - Variable, large contiguous portions: This will provide better performance. The variable size avoids waste, and the file allocation tables are small. However, space is hard to reuse.
  - Blocks: Small fixed portions provide greater flexibility. They may require large tables or complex structures for their allocation. Contiguity has been abandoned as a primary goal; blocks are allocated as needed.

Portion Size

- Either option on the previous page is compatible with either preallocation or dynamic allocation schemes.
- In the case of variable, large contiguous portions, a file is preallocated one contiguous group of blocks. This eliminates the need for a file allocation table; all that is required is a pointer to the first block and the number of blocks allocated.
- In the case of blocks, all of the portions required are allocated at one time. This means that the file allocation table for the file will remain of fixed sized, because the number of blocks allocated is fixed.

Portion Size

- With variable-size portions, there is concern over the fragmentation of free space. This is the same issue we considered when we looked at memory management in a partitioned main memory.
- Some possible strategies are:
  - First fit: Choose the first unused contiguous group of blocks of sufficient size from the free block list.
  - Best fit: Choose the smallest unused group of blocks that is of sufficient size.
  - Nearest fit (Next fit): Choose the unused group of blocks of sufficient size that is closest to the previous allocation for the file to increase its locality.

File Allocation Strategies

- Now that we've dealt with the issues of preallocation versus dynamic allocation and portion size, the file issue to consider is the specific file allocation strategies.
- There are three methods in common use today:
  - Contiguous
  - Chained
  - Indexed
- The table on the next page summarizes some of the characteristics of each strategy.

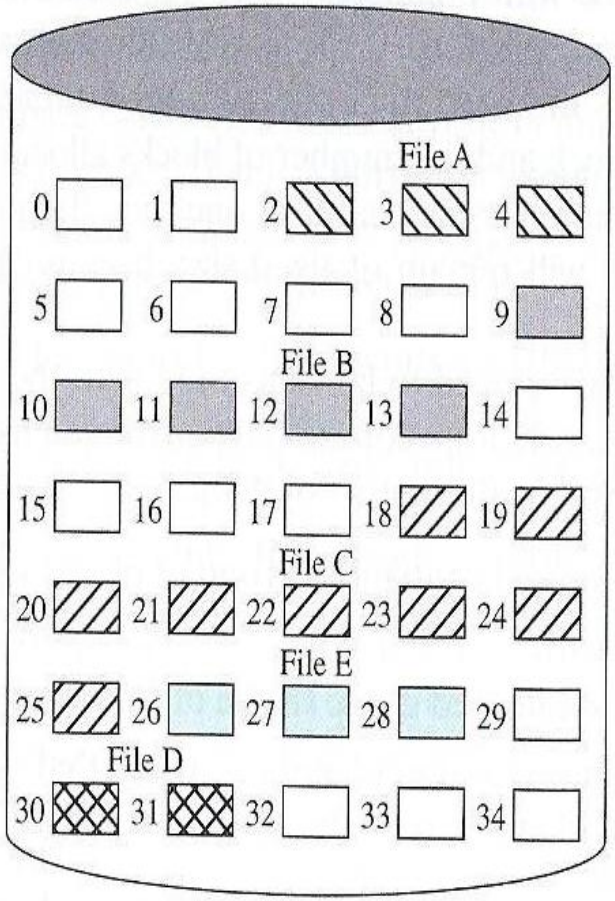
File Allocation Strategies

	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium

File Allocation Strategies: Contiguous

- With contiguous allocation, a single contiguous set of blocks is allocated to a file at the time of file creation.
- This is a preallocation using variable-size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file.
- The figure on the next page illustrates a contiguous allocation strategy.

File Allocation Strategies: Contiguous



File Allocation Table

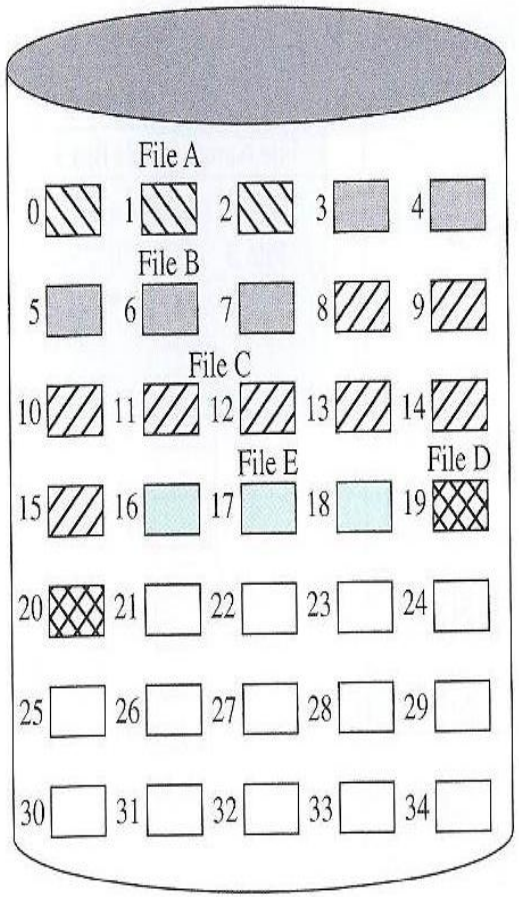
File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Note the external fragmentation on the disk. Currently there are no sets of contiguous blocks of size 5 or larger.

File Allocation Strategies: Contiguous

- Contiguous allocation is best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential processing.
- It is also easy to retrieve a single block. For example, if a file starts at block `b`, and the `i`th block of the file is desired, its location on the secondary storage is simply `b+ i - 1`.
- Contiguous allocation does allow for external fragmentation, making it difficult to find contiguous blocks of space of sufficient length. From time to time, it will be necessary to perform a compaction algorithm to free up additional space on the disk.
- The previous scenario is shown on the next page after memory compaction has occurred.

File Allocation Strategies: Contiguous



File Allocation Table

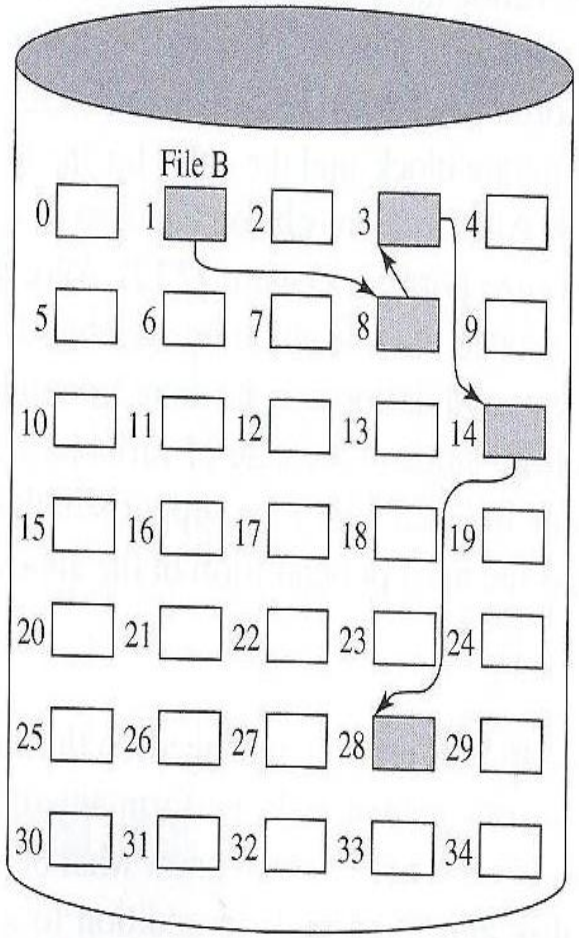
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

After compaction has been done there is now a set of contiguous blocks of size 5 or larger.

File Allocation Strategies: Chained

- At the opposite end of the strategies from contiguous allocation is chained allocation.
- Typically, allocation using this strategy is on a block by block basis (dynamic).
- Each block contains a pointer to the next block in the chain.
- Again, the file allocation table needs just a single entry for each file, showing the starting block and the length of the file.
- Block selection is simple, as any available block can be added to any chain.
- There is no external fragmentation to worry about since space is added to a file one block at a time.

File Allocation Strategies: Chained



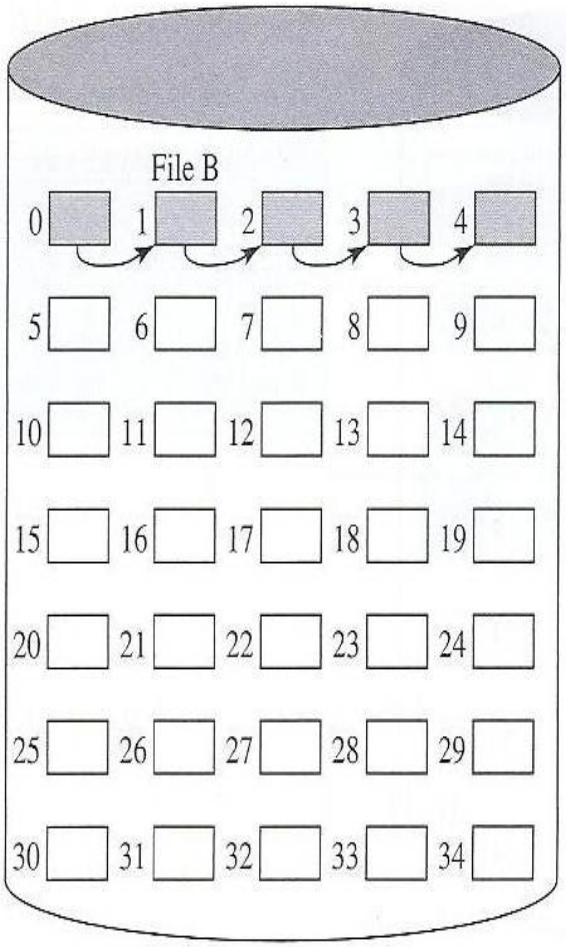
File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

File Allocation Strategies: Chained

- One consequence of a chained allocation scheme is that there is no accommodation of the principle of locality. Thus for sequential processing, the following of a chain of pointers can drastically slow down I/O processing.
- To overcome this problem, the system may periodically consolidate files. This is a technique quite similar to memory compaction, but it is done to bring file portions together rather than to create large unallocated blocks of space.
- The illustration on the next page shows the disk after the file on the previous page was consolidated.

File Allocation Strategies: Chained



File Allocation Table

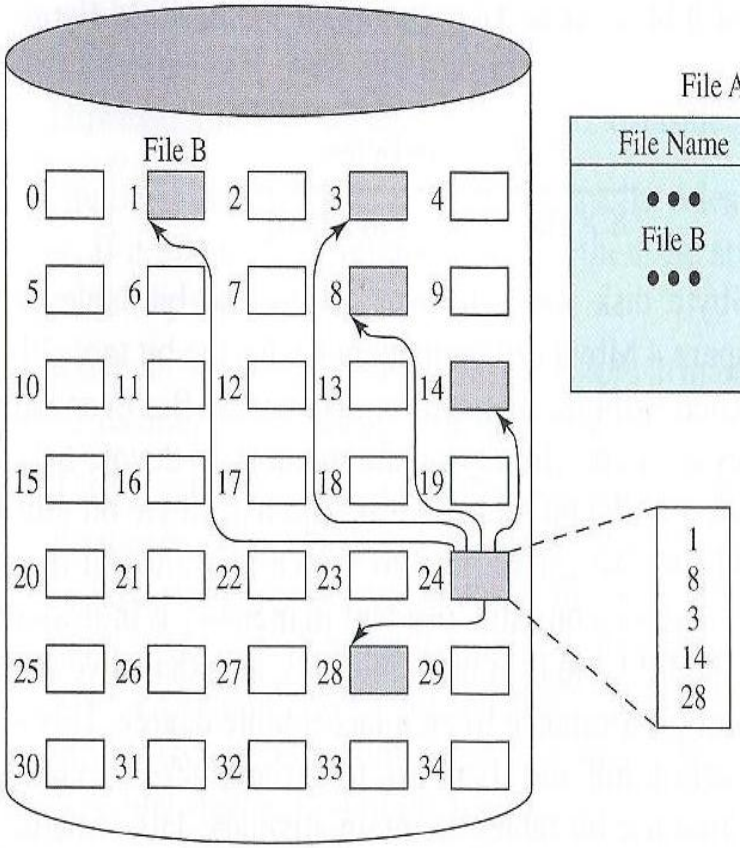
File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

After the file was consolidated

File Allocation Strategies: Indexed

- Indexed allocation schemes attempt to address the problems of both contiguous and chained allocation schemes.
- In this case the file allocation table contains a separate one-level index for each file; the index has one entry for each portion allocated to the file.
- Typically, the file indexes are not physically stored as part of the file allocation table. Rather the file index for a file is kept in a separate block, and the entry for the file in the file allocation table points to that block.
- Allocation can be based on either fixed-size blocks or variable-size portions. Allocation by blocks eliminated external fragmentation, whereas allocation by variable-size portions improves locality.

File Allocation Strategies: Indexed

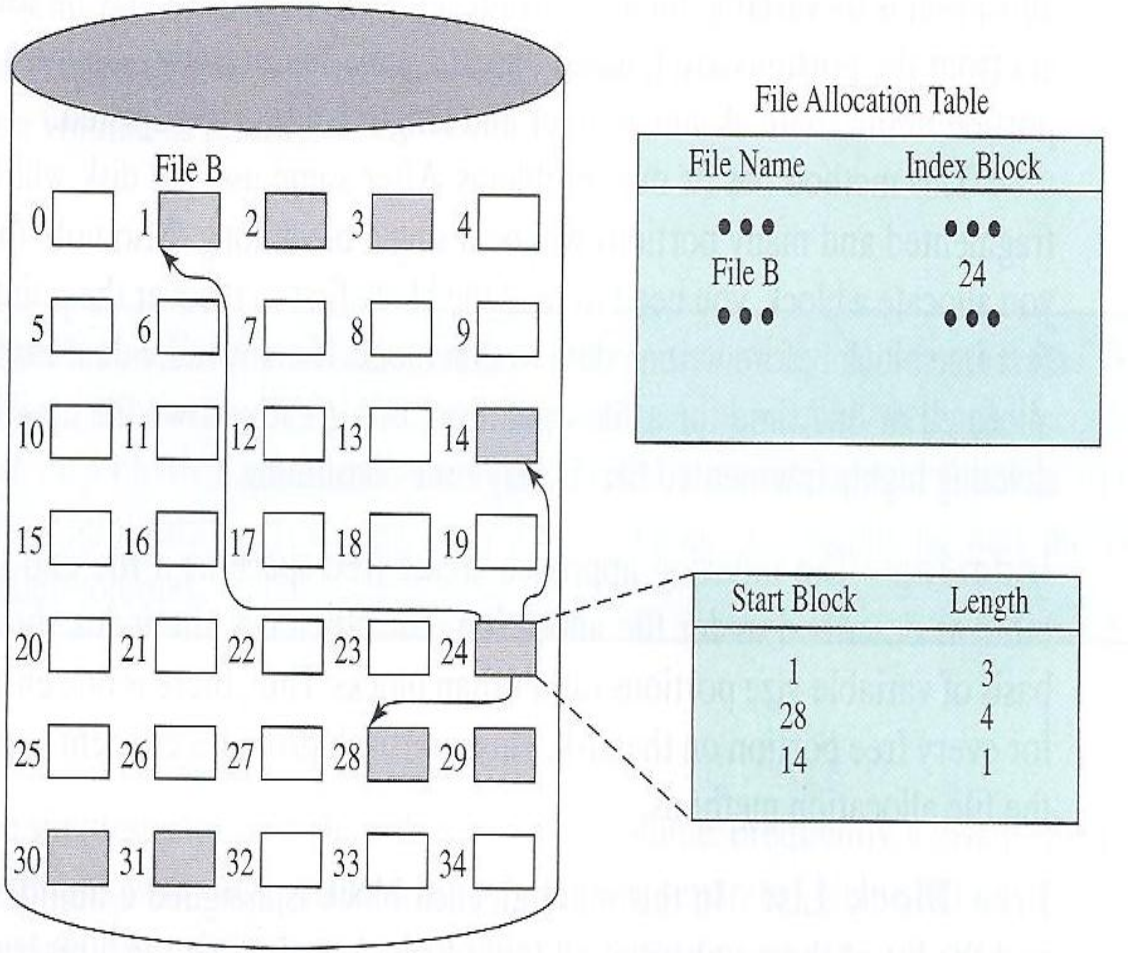


File Allocation Table

File Name	Index Block
• • •	• • •
File B	24
• • •	• • •

Indexed allocation with block portions

File Allocation Strategies



File Allocation Table

File Name	Index Block
•••	•••
File B	24
•••	•••

Start Block	Length
1	3
28	4
14	1

Indexed allocation with variable-sized portions