THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

OPERATING SYSTEMS

Week 10

Much of the material on these slides comes from the recommended textbook by William Stallings

Detailed content

Weekly program

- ✓ Week 1 – Operating System Overview
- ✓ Week 2 – Processes and Threads
- ✓ Week 3 – Scheduling
- ✓ Week 4 – Real-time System Scheduling and Multiprocessor Scheduling
- ✓ Week 5 – Concurrency: Mutual Exclusion and Synchronization
- ✓ Week 6 – Concurrency: Deadlock and Starvation
- ✓ Week 7 – Memory Management
- ✓ Week 8 – Memory Management II
- ✓ Week 9 – Disk and I/O Scheduling
- ❑ **Week 10 – File Management**
- ❑ Week 11 – Security and Protection
- ❑ Week 12 – Revision of the course
- ❑ Week 13 – Extra revision (if needed)

Key Concepts From Last Lecture

- I/O function is generally broken up into a number of layers
  - Lower layers deal with detains and closer to physical functions
  - Higher layers deal with I/O in a logical and generic fashion
  - Changes in HW parameter need not to affect most of the I/O SW
- A key aspect of I/O is the use of buffers that are controlled by I/O utilities rather than by application processes
  - Buffering smooths out the differences between the internal speed of the computer system and the speed of I/O devices
  - Decouples the actual I/O transfer from the address space of the application process
  - Allows OS more flexibility in memory management

Key Concepts From Last Lecture

- Performance of disk storage subsystem is of vital concern
- Disk scheduling is to satisfy the I/O requests on the disk in a way to minimize the seek time of the disk
  - FCFS, STTF, SCAN, C-SCAN are some algorithms used
- RAID is an architecture to organize data over multiple disks to improve data transfer capacity and data reliability
  - RAID scheme consists of seven level zero through six

- .

Week 10 Lecture Outline

File Management

- ❑ Files, File Structures and File Systems
- ❑ File System Software architecture
- ❑ Elements of File Management
- ❑ File organization and access
- ❑ Indexed structure: B-Trees
- ❑ File directories
- ❑ File sharing
- ❑ Record blocking
- ❑ Secondary storage management

Videos to watch before lecture

File Management

- The file is the central element in most applications
  - Input to the application is by means of a file
  - Virtually all applications, output is saved in a file for long term storage, later access by the user and use by other programs.
- Files also have a life outside of any individual application
  - Users wish to be able to access files, save them, and maintain the integrity of their contents.
- Virtually all operating systems provide file management systems.
  - Typically, a file management system consists of system utility programs that run as privileged applications.
  - At the least, a file management system needs special services from the operating system
  - At the most, the entire file management system is considered part of the operating system.

Files

- The File System is one of the most important parts of the OS to a user
- Files are data collections created by users
- Desirable properties of files:

Long-term existence

- files are stored on disk or other secondary storage and do not disappear when a user logs off

Sharable between processes

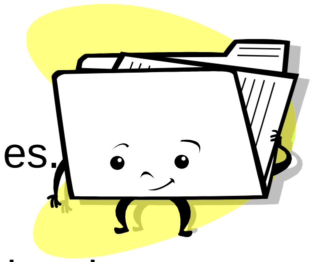
- files have names and can have associated access permissions that permit controlled sharing

Structure

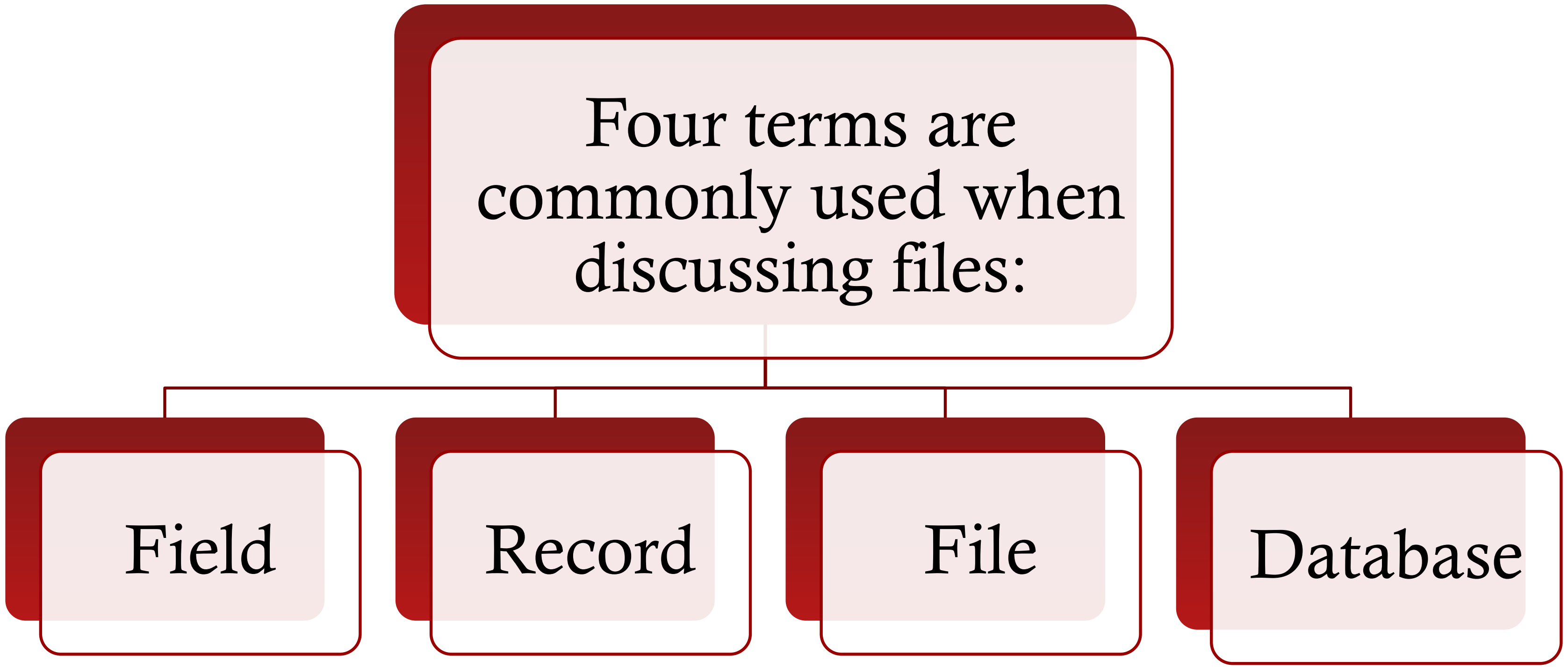
- files can have an internal structure that is convenient for particular application
- files can be organized into hierarchical or more complex structure to reflect the relationships among files

File system

- Provide a means to store data organized as files as well as a collection of functions that can be performed on files
- Typical operations include:
  - **Create**: A new file is defined and positioned within the structure of files.
  - **Delete**: A file is removed from the file structure and destroyed.
  - **Open**: An existing file is declared to be “opened” by a process, allowing the process to perform functions on the file.
  - **Close**: The file is closed with respect to a process, so that the process no longer may perform functions on the file, until the process opens the file again.
  - **Read**: A process reads all or a portion of the data in a file.
  - **Write**: A process updates a file, either by adding new data that expands the size of the file or by changing the values of existing data items in the file.
- Also maintains a set of attributes associated with the file
  - Owner, creation time, time last modified, access privileges etc.



File structure



Structure terms

Field

- basic element of data
- contains a single value
- fixed or variable length

File

- collection of similar records
- treated as a single entity
- may be referenced by name
- access control restrictions usually apply at the file level

Record

- collection of related fields that can be treated as a unit by some application program
- fixed or variable length

Database

- collection of related data
- relationships among elements of data are explicit
- designed for use by a number of different applications
- consists of one or more types of files

File management system objectives

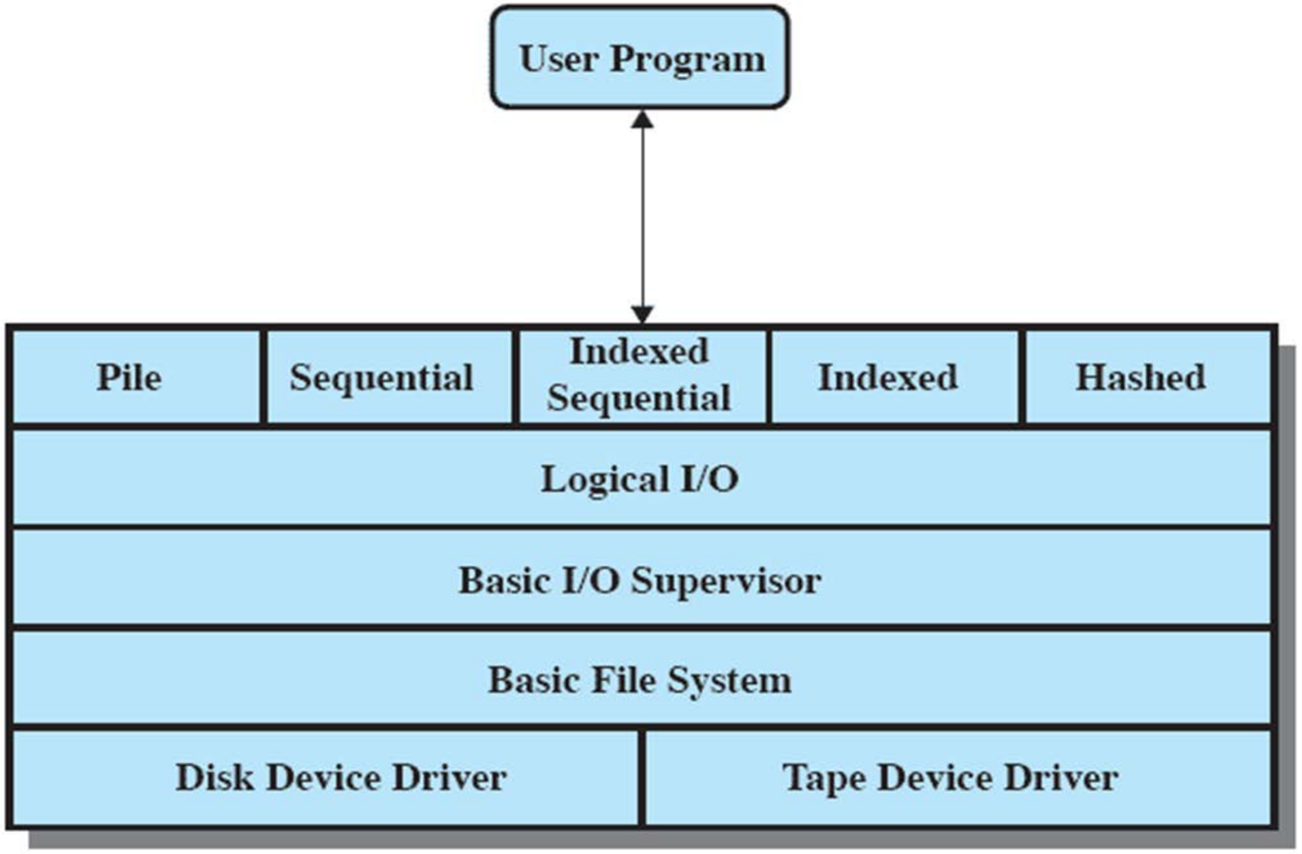
- Meet the data management needs of the user
- Guarantee that the data in the file are valid
- Optimize performance
- Provide I/O support for a variety of storage device types
- Minimize the potential for lost or destroyed data
- Provide a standardized set of I/O interface routines to user processes
- Provide I/O support for multiple users in the case of multiple-user systems

Minimal user requirements

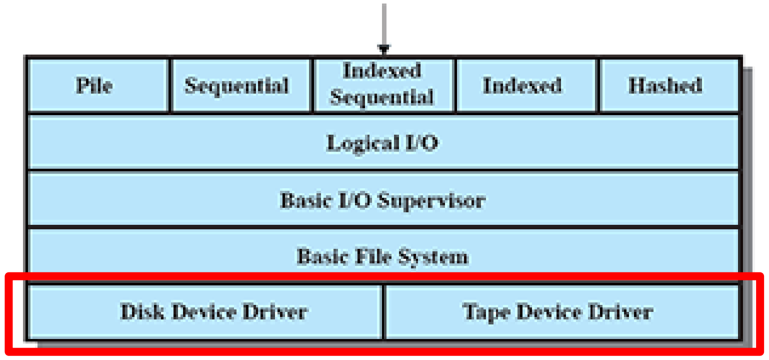
- Each user:

1. should be able to create, delete, read, write and modify files
2. may have controlled access to other users' files
3. may control what type of accesses are allowed to the files
4. should be able to restructure the files in a form appropriate to the problem
5. should be able to move data between files
6. should be able to back up and recover files in case of damage
7. should be able to access his or her files by name rather than by numeric identifier

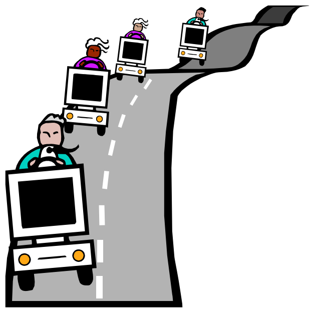
File system software architecture



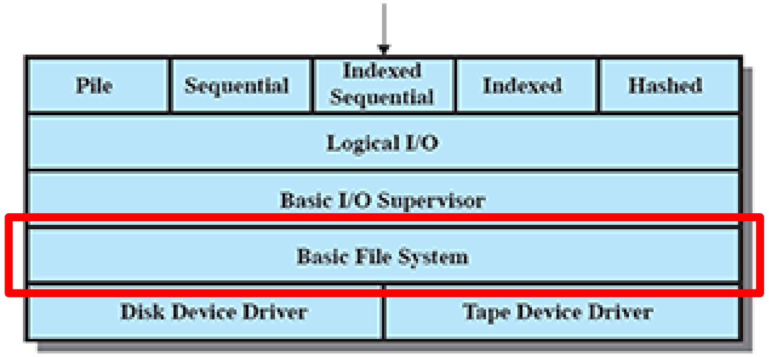
Device drivers



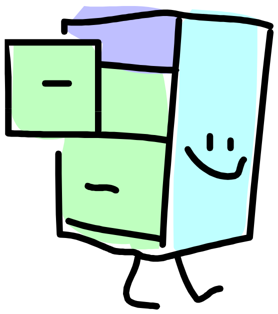
- Lowest level
- Communicates directly with peripheral devices
- Responsible for
  - starting I/O operations on a device
  - processes the completion of an I/O request
- Considered to be part of the operating system



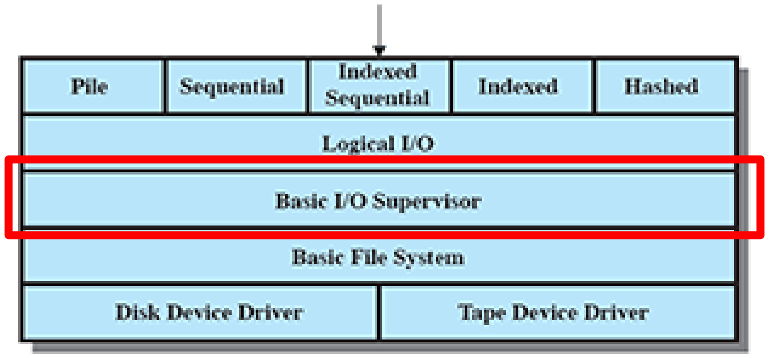
Basic file system



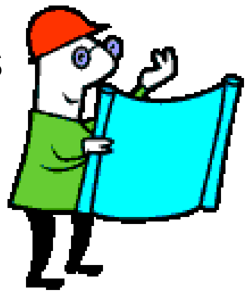
- Also referred to as the physical I/O level
- Primary interface with the environment outside the computer system
- Deals with blocks of data that are exchanged with disk or tape systems
- Concerned with the placement of blocks on the secondary storage device
- Concerned with buffering blocks in main memory
- Does not understand the content of the data or structure of the files involved
- Considered part of the operating system



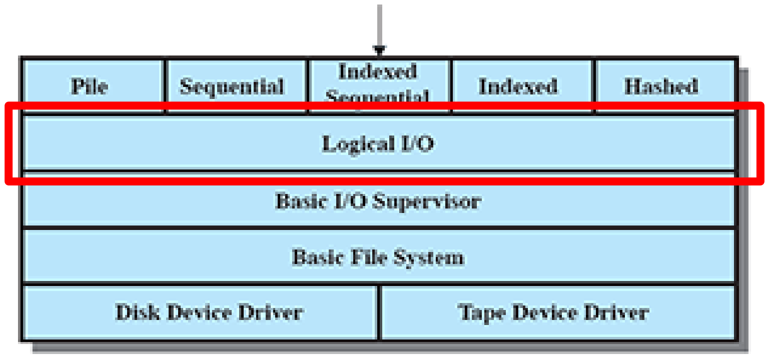
Basic I/O supervisor



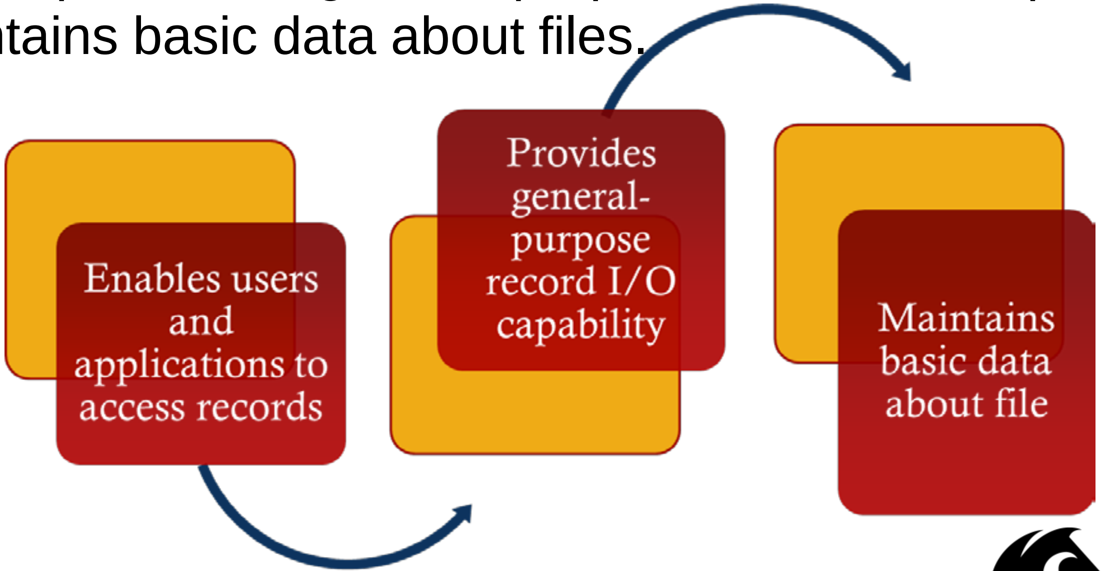
- Responsible for all file I/O initiation and termination
- Control structures that deal with device I/O, scheduling, and file status are maintained
- Selects the device on which I/O is to be performed
- Concerned with scheduling disk and tape accesses to optimize performance
- I/O buffers are assigned and secondary memory is allocated at this level
- Part of the operating system



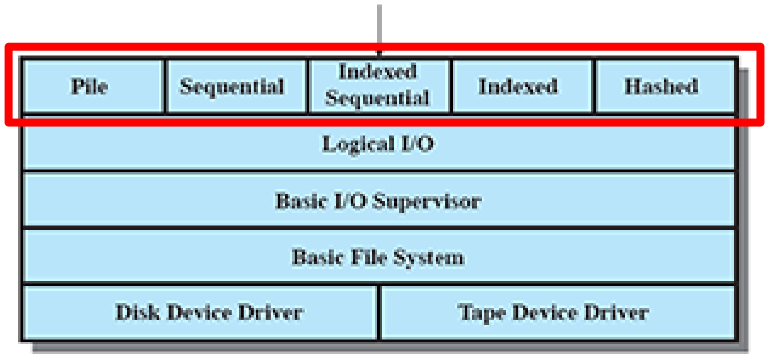
Logical I/O



- Logical I/O enables users and applications to access records.
  - Whereas the basic file system deals with blocks of data, the logical I/O module deals with file records.
  - Logical I/O provides a general-purpose record I/O capability and maintains basic data about files.

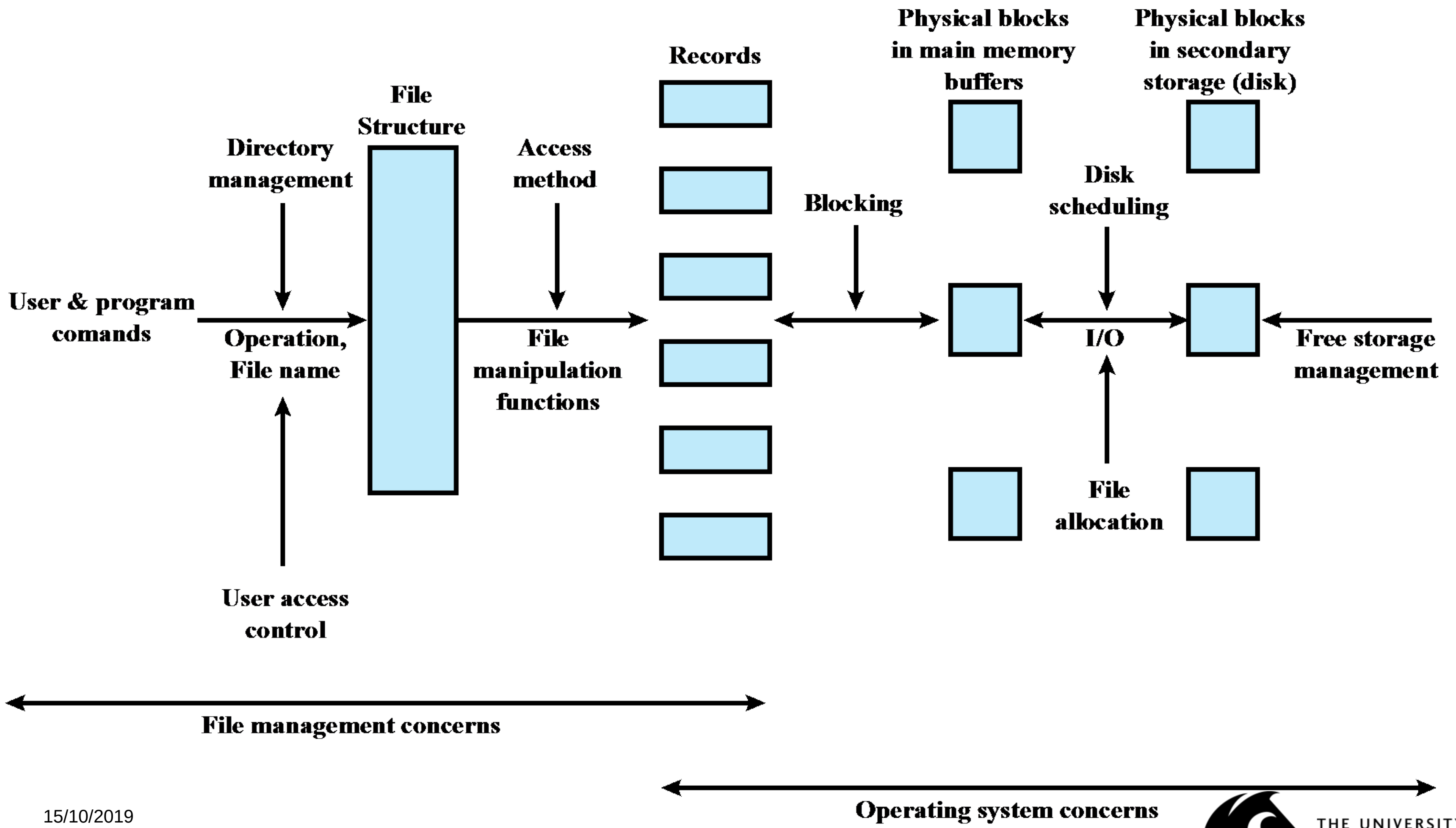

Enables users and applications to access records
Provides general-purpose record I/O capability
Maintains basic data about file

Access method



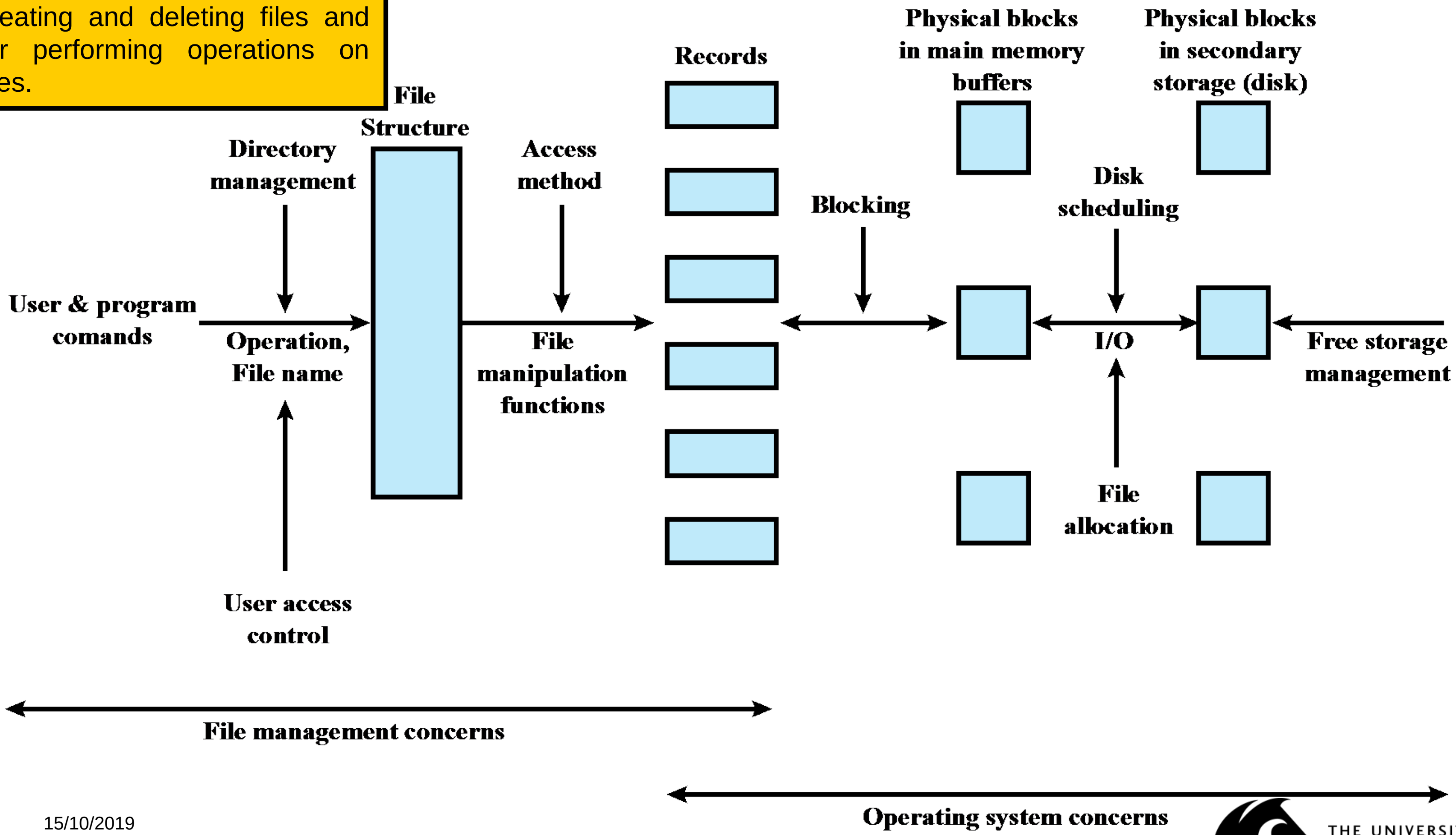
- Level of the file system closest to the user
- Provides a standard interface between applications and the file systems and devices that hold the data
- Different access methods reflect different file structures and different ways of accessing and processing the data

Elements of file management

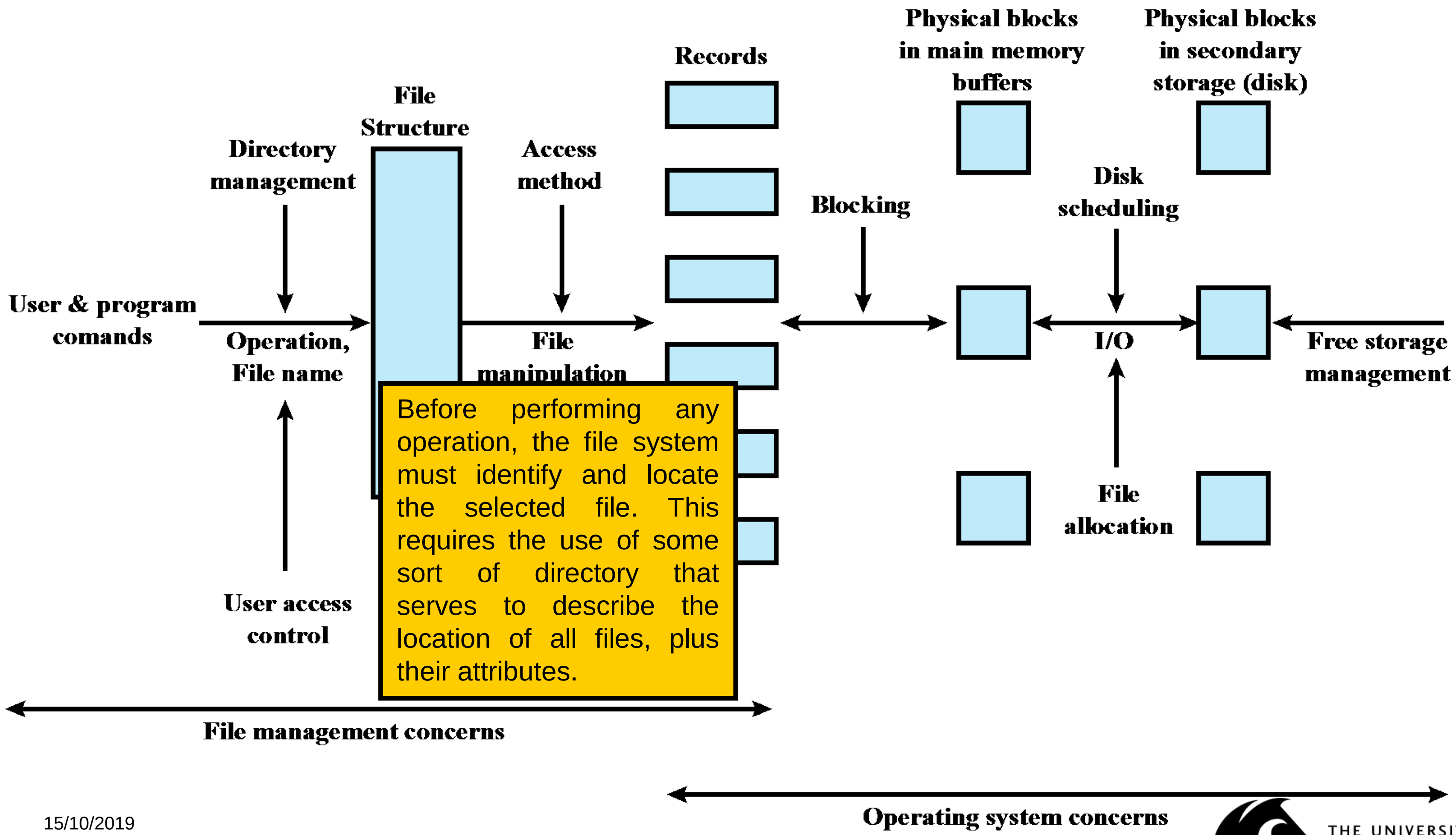


Elements of file management

Users and application programs interact with the file system by means of commands for creating and deleting files and for performing operations on files.

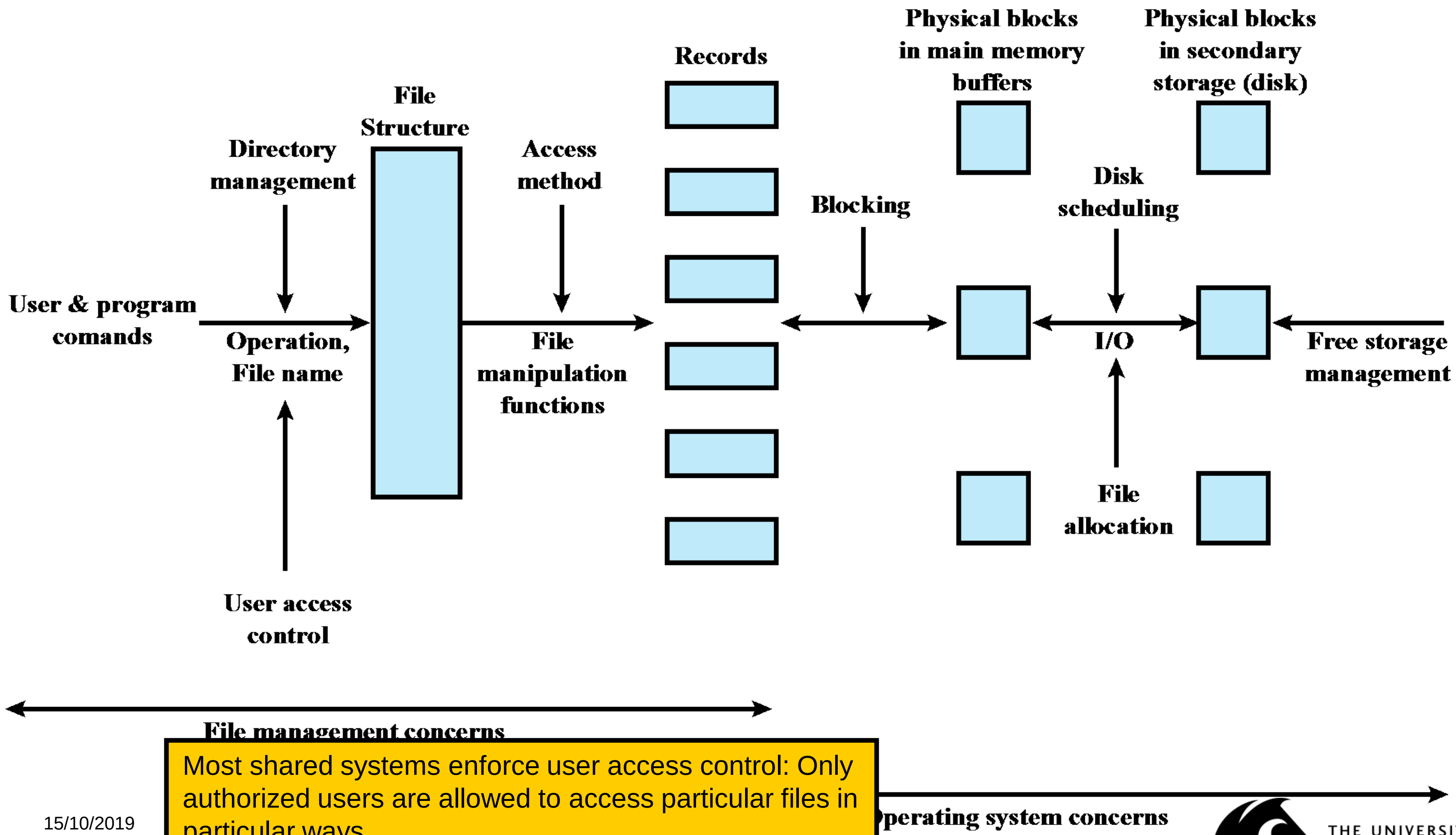


Elements of file management



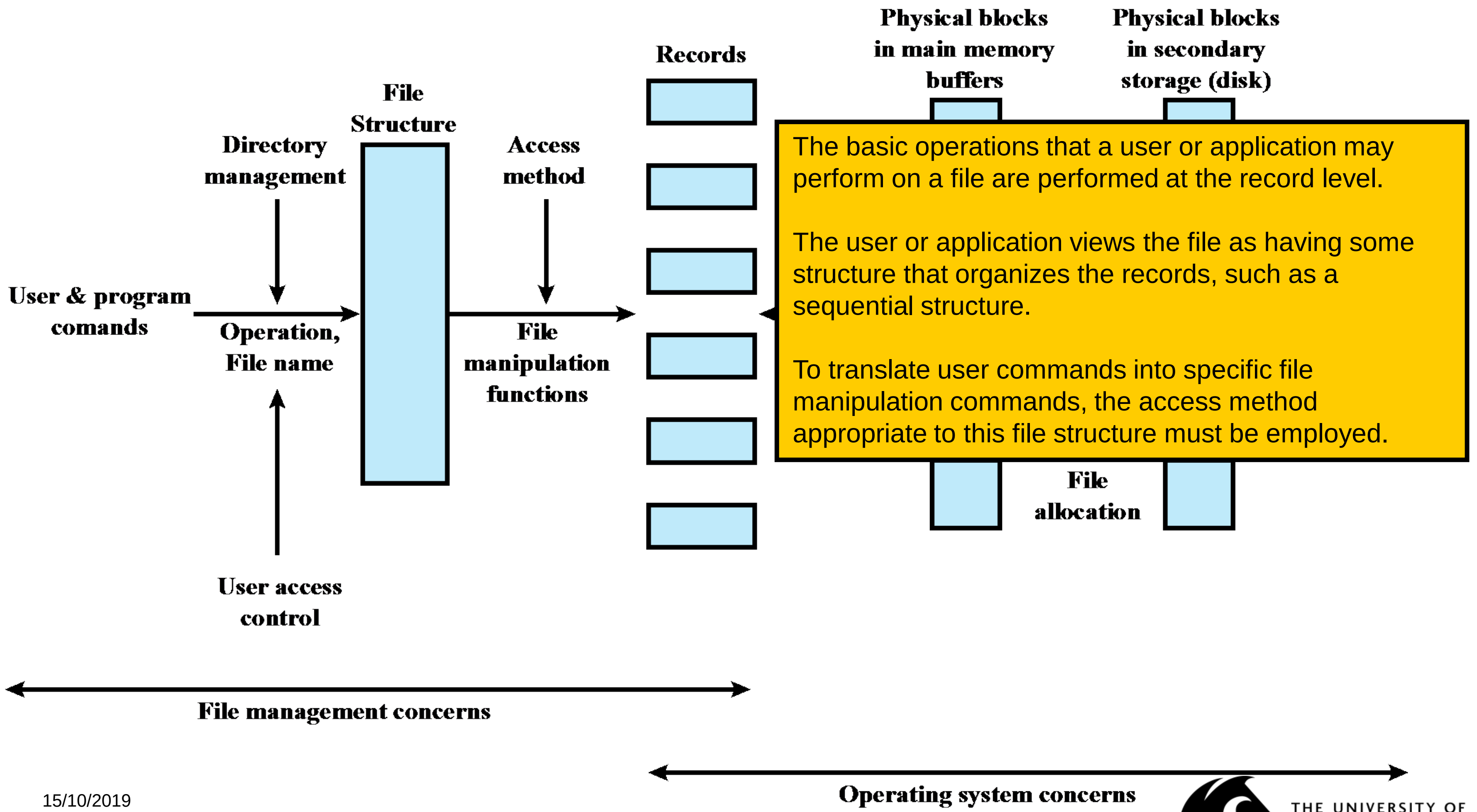
Before performing any operation, the file system must identify and locate the selected file. This requires the use of some sort of directory that serves to describe the location of all files, plus their attributes.

Elements of file management



Most shared systems enforce user access control: Only authorized users are allowed to access particular files in particular ways

Elements of file management

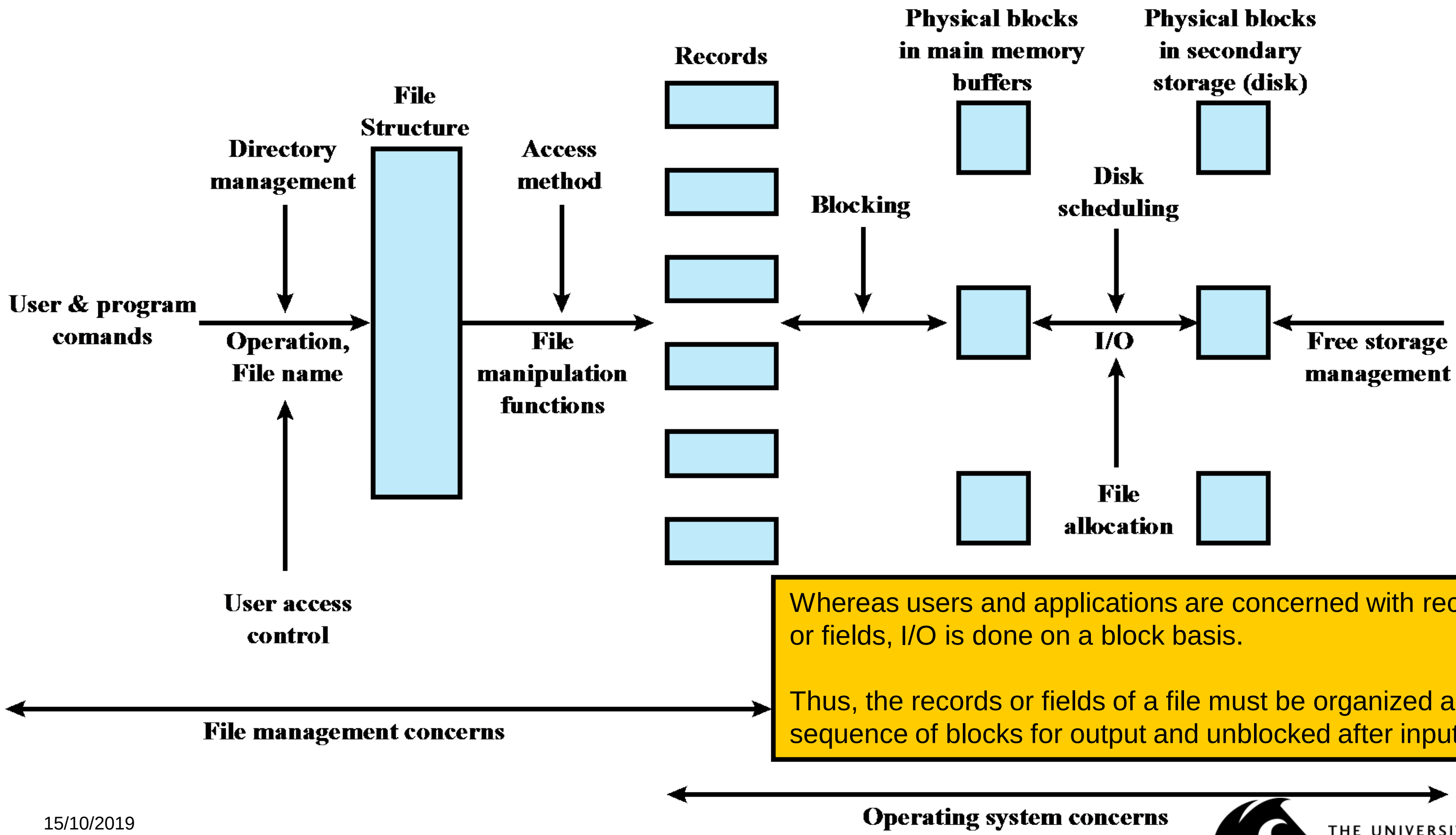


The basic operations that a user or application may perform on a file are performed at the record level.

The user or application views the file as having some structure that organizes the records, such as a sequential structure.

To translate user commands into specific file manipulation commands, the access method appropriate to this file structure must be employed.

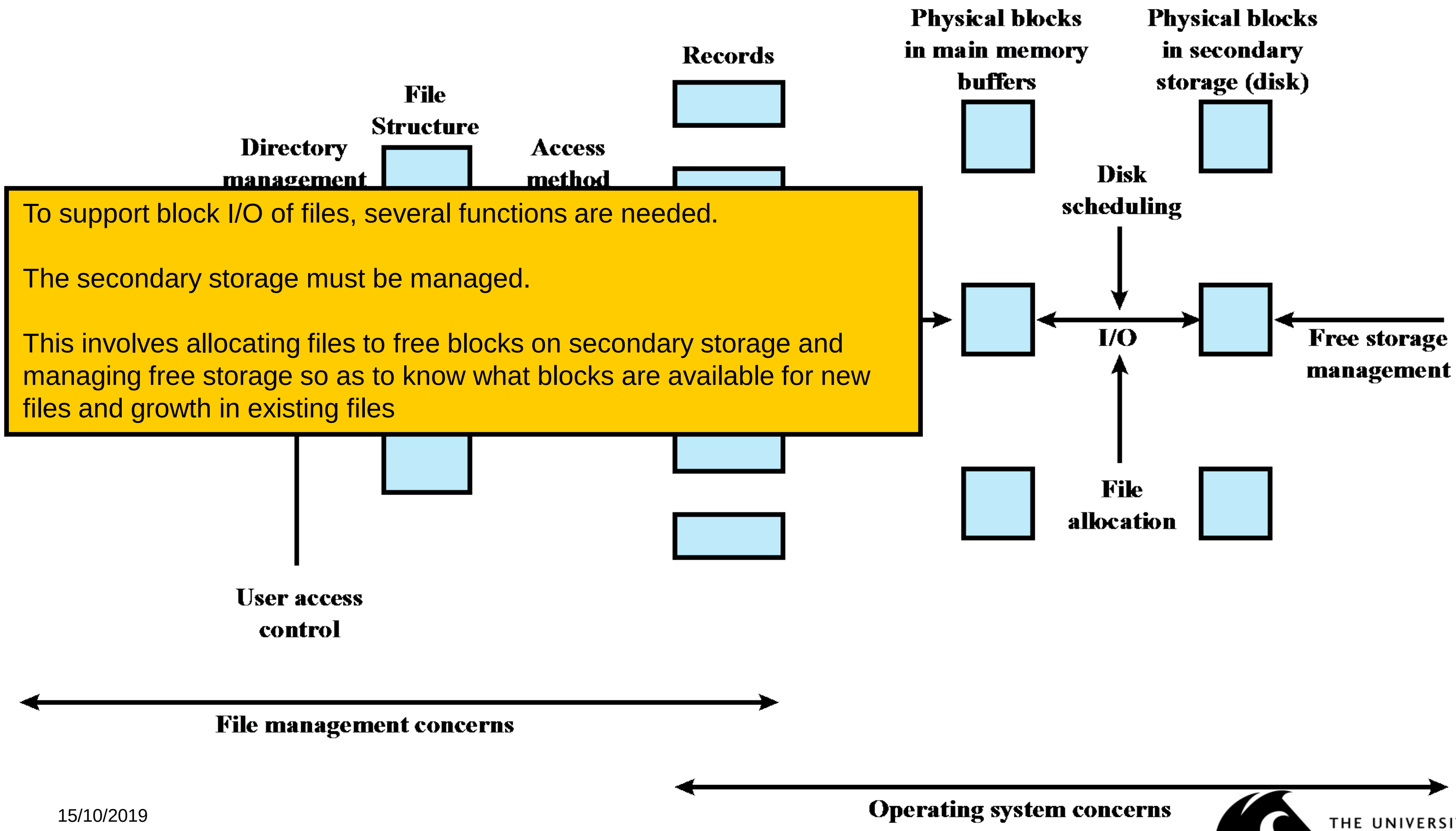
Elements of file management



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.

Elements of file management

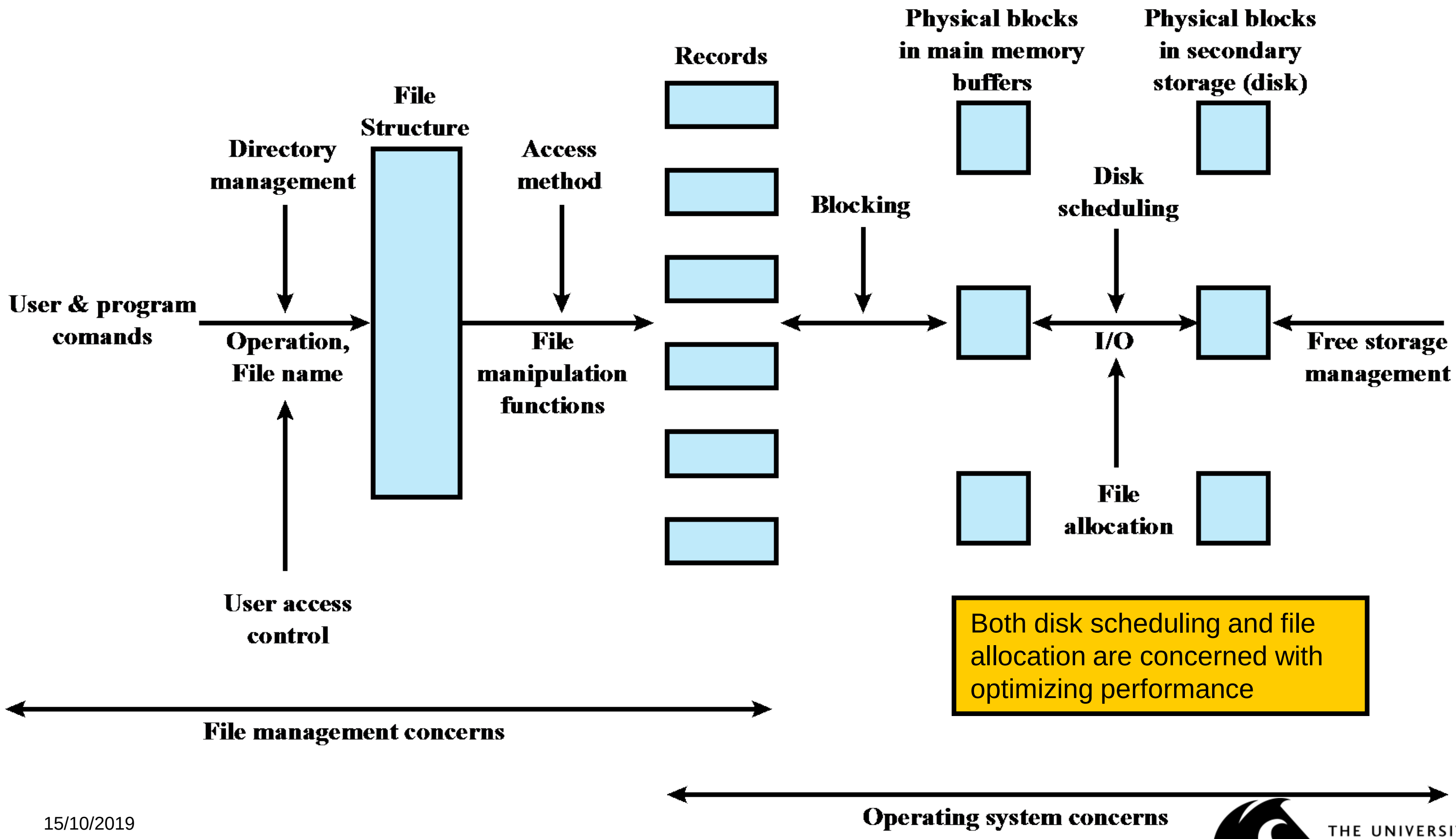


To support block I/O of files, several functions are needed.

The secondary storage must be managed.

This involves allocating files to free blocks on secondary storage and managing free storage so as to know what blocks are available for new files and growth in existing files

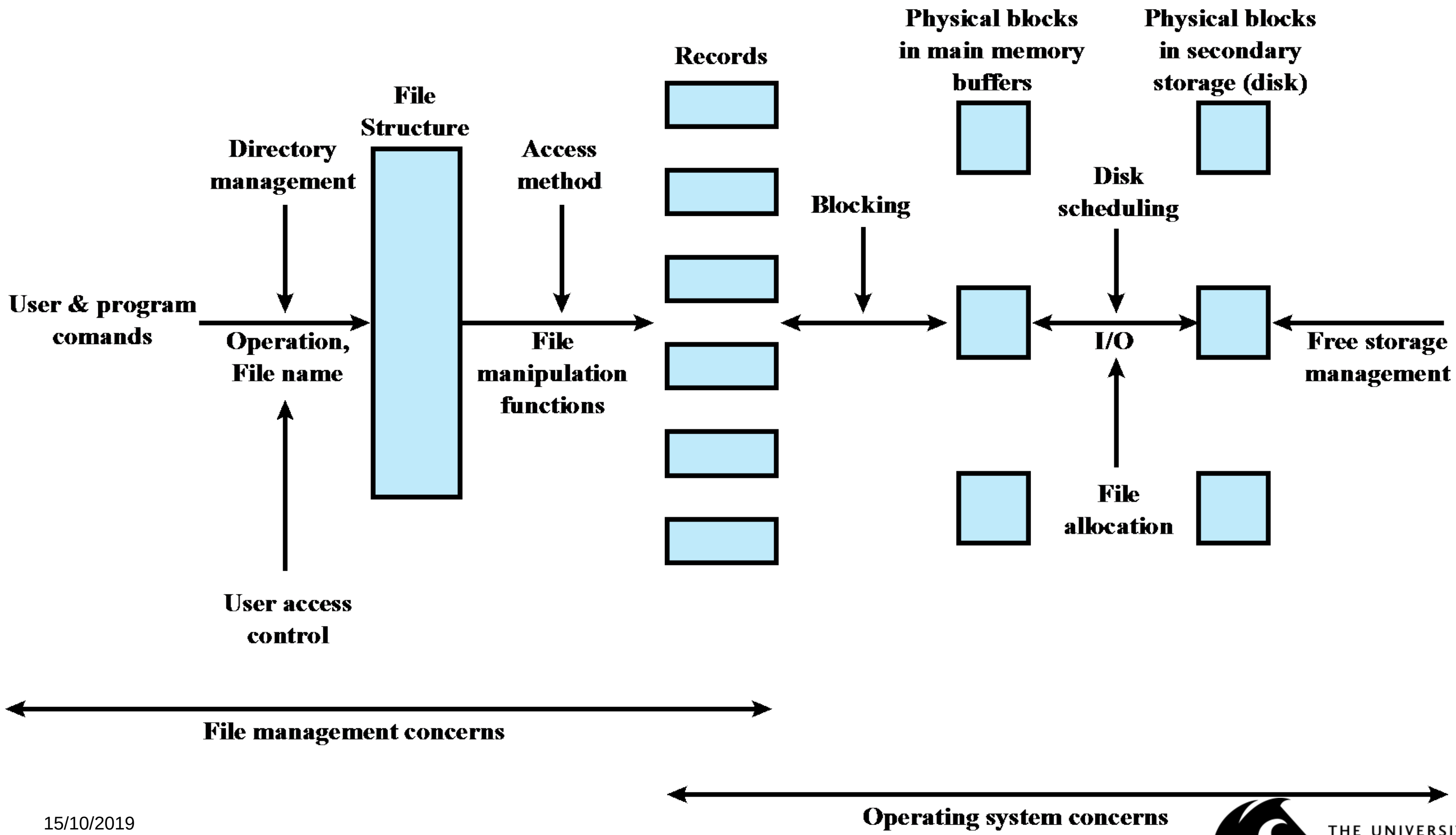
Elements of file management



Both disk scheduling and file allocation are concerned with optimizing performance

Consider the division between what might be considered the concerns of the file management system as a separate system utility and the concerns of the operating system, with the point of intersection being record processing.

This division is arbitrary; various approaches are taken in various systems.



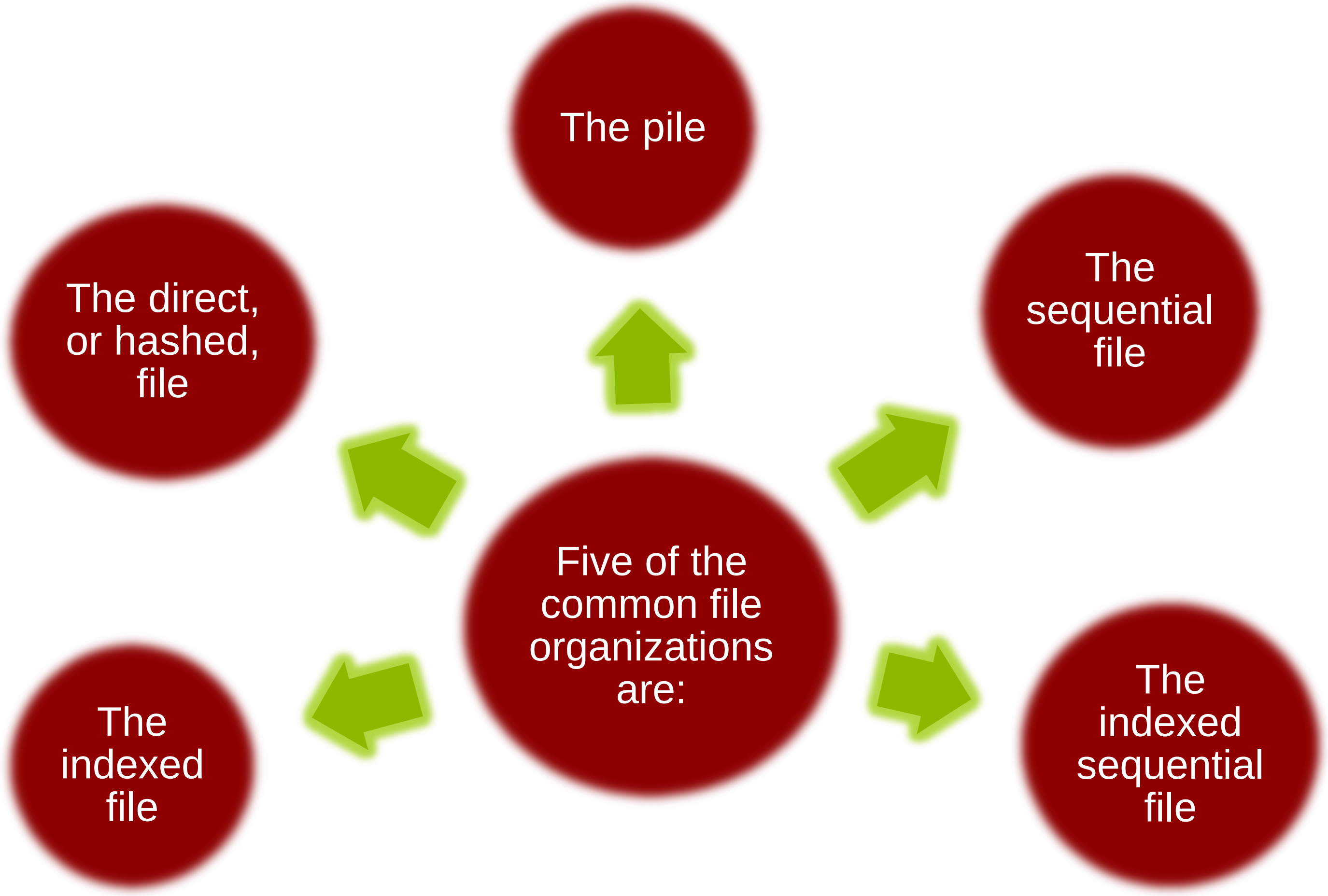
File organisation and access

- File organization is the **logical structuring** of records as determined by the way in which they are accessed

- In choosing a file organization, several criteria are important:
  - short access time
  - ease of update
  - economy of storage
  - simple maintenance
  - reliability

- Priority of criteria depends on the application that will use the file

File organisation types


Five of the common file organizations are:
The pile
The sequential file
The indexed sequential file
The indexed file
The direct, or hashed, file

The Pile

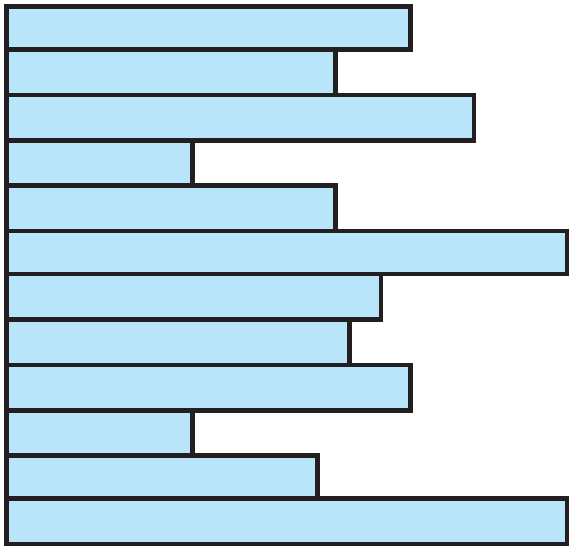
- Least complicated form of file organization
- Data are collected in the order they arrive
- Each record consists of one burst of data
- Purpose is simply to accumulate the mass of data and save it

- Records may have different fields or differently ordered fields
  - each field should have a name and a value
  - field length and / delimiter should be used

- Record access is by exhaustive search

Advantages:
- Utilizes space well when stored data vary in size and structure
- Easy to update

Disadvantage:
- Unsuitable for most application


Variable-length records
Variable set of fields
Chronological order

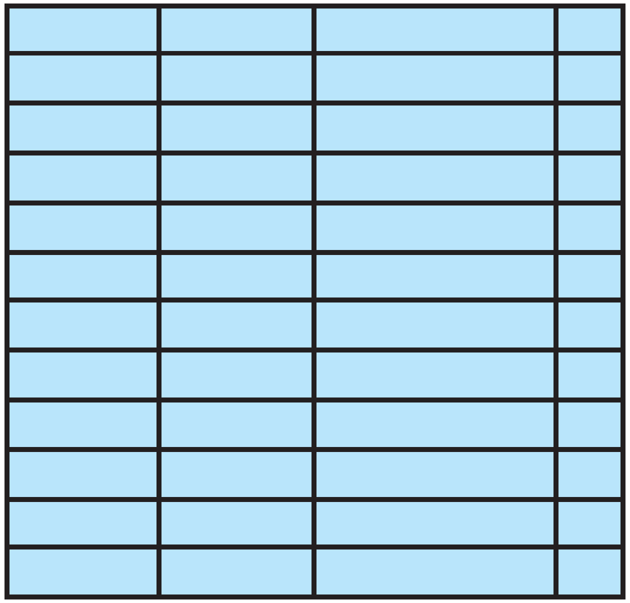
(a) Pile File

Name=Jim Raider, age=36,height=170, weight=248,…

A pile file record

Sequential file

- Most common form of file structure
- A fixed format is used for records
- Key field **uniquely** identifies the record
- Records are sorted in key sequence
- Only organization that is easily stored on tape as well as disk
- Suitable for batch applications
  - Poor performance for queries and/or update of individual records
- Addition to the file is problematic
  - Physical organization of the file should match its logical organization
  - Use a separate file called log file for new records
    - Periodically batch update is performed
  - Alternatively, physically organize the file as linked list with additional cost and overhead



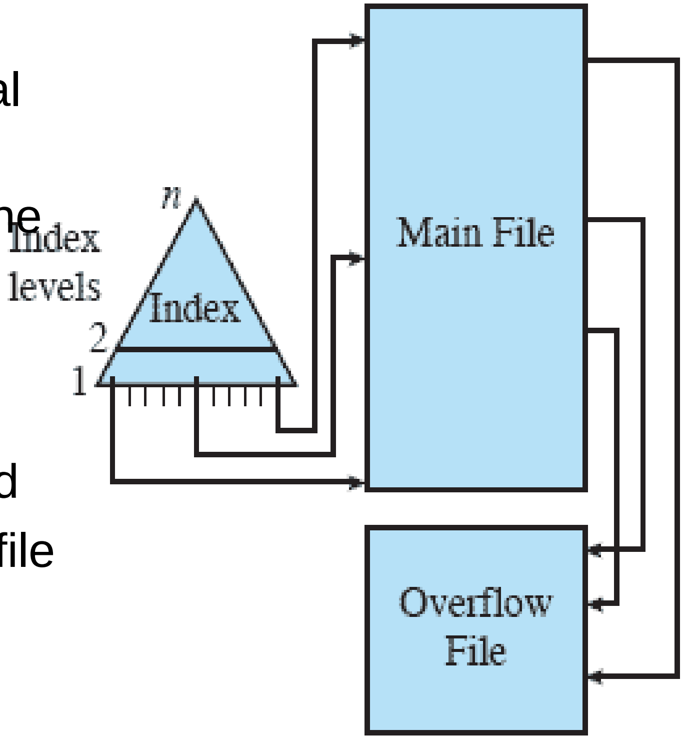
Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential File

Indexed sequential file

- Adds two additional features to the sequential file
  - An index to reach quickly the vicinity of the desired record
  - Adds an overflow file (similar to log file)

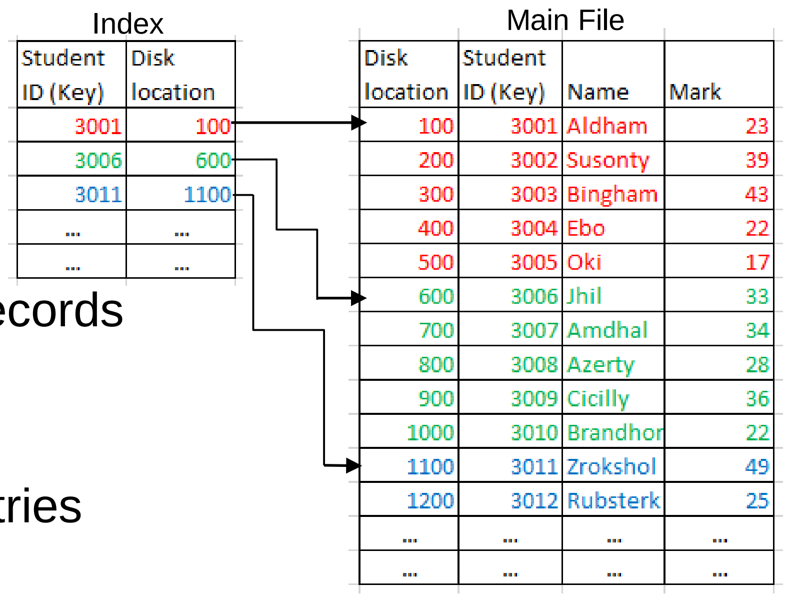
- Simplest case: a single level of indexing used
  - The index is itself is a simple sequential file
  - Two fields in record of the index:
    - (i) Key and (ii) pointer to main file



(c) Indexed Sequential File

Indexed sequential file

- Greatly reduces the time required to access a single record
  - Consider a sequential file of 1million records
    - Use an index of 1000 entries
- Multiple levels of indexing can be used to provide greater efficiency in access
  - Consider a sequential file of 1million records
    - Use an lower level index of 10,000 entries
    - Use a higher level index of 100 entries



Index

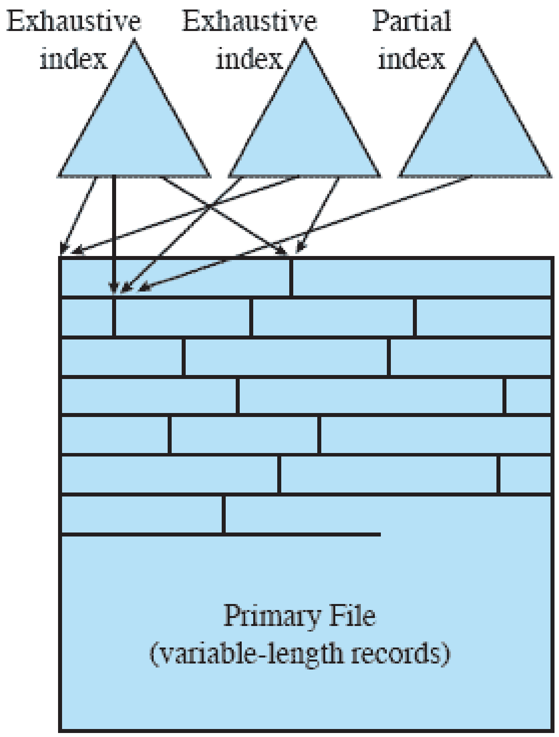
Student ID (Key)	Disk location
3001	100
3006	600
3011	1100
...	...
...	...

Main File

Disk location	Student ID (Key)	Name	Mark
100	3001	Aldham	23
200	3002	Susonty	39
300	3003	Bingham	43
400	3004	Ebo	22
500	3005	Oki	17
600	3006	Jhil	33
700	3007	Amdhal	34
800	3008	Azerty	28
900	3009	Cicilly	36
1000	3010	Brandhor	22
1100	3011	Zrokshol	49
1200	3012	Rubsterk	25
...	...	...	...
...	...	...	...

Indexed file

- Records are accessed only through their indexes
  - Sequentiality in the main file is abandoned
- Variable-length records can be employed
- Two types of indexing:
  - **Exhaustive index** contains one entry for every record in the main file
  - **Partial index** contains entries to records where the field of interest exists
- Used mostly in applications where timeliness of information is critical
  - Examples would be airline reservation systems and inventory control systems



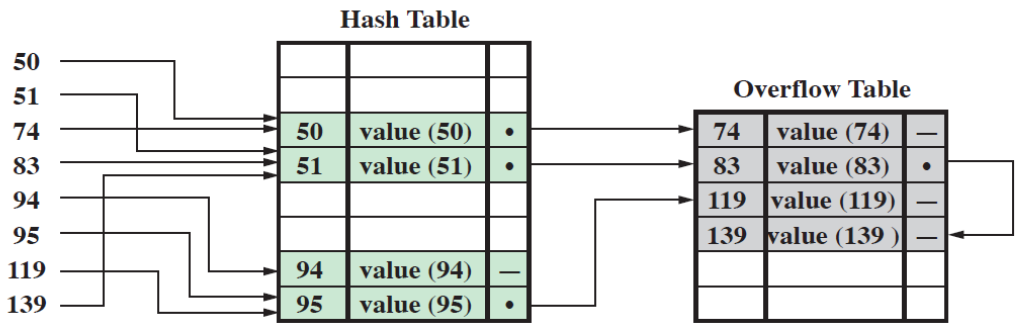
(d) Indexed File

Direct or hashed file

- Access directly any block of a known address
- Makes use of hashing on the key value
- Often used where:
  - very rapid access is required
  - fixed-length records are used
  - records are always accessed one at a time

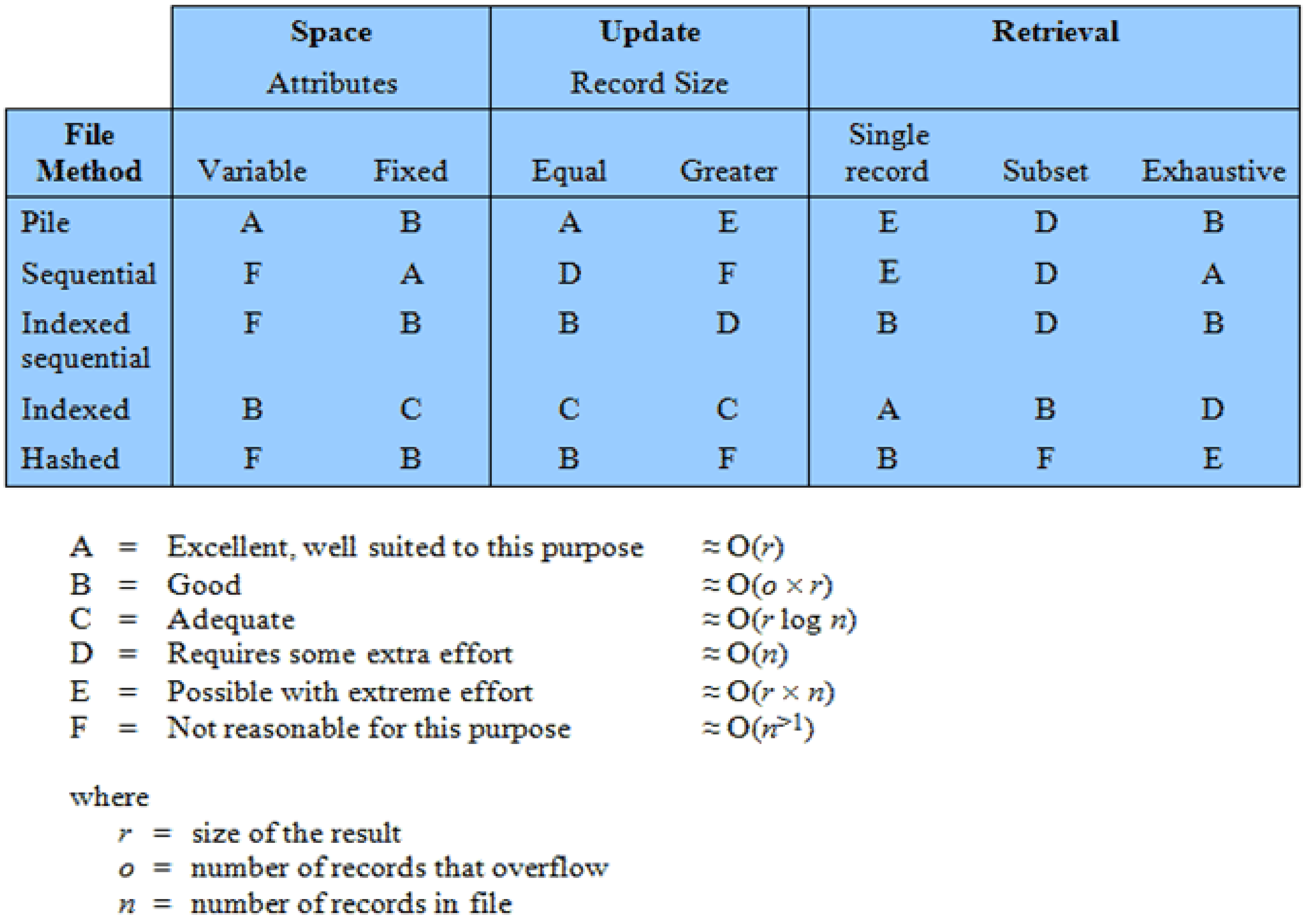
Examples are:

- directories
- pricing tables
- schedules
- name lists



Grades of performance

Table 12.1 Grades of Performance for Five Basic File Organizations [WIED87]



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

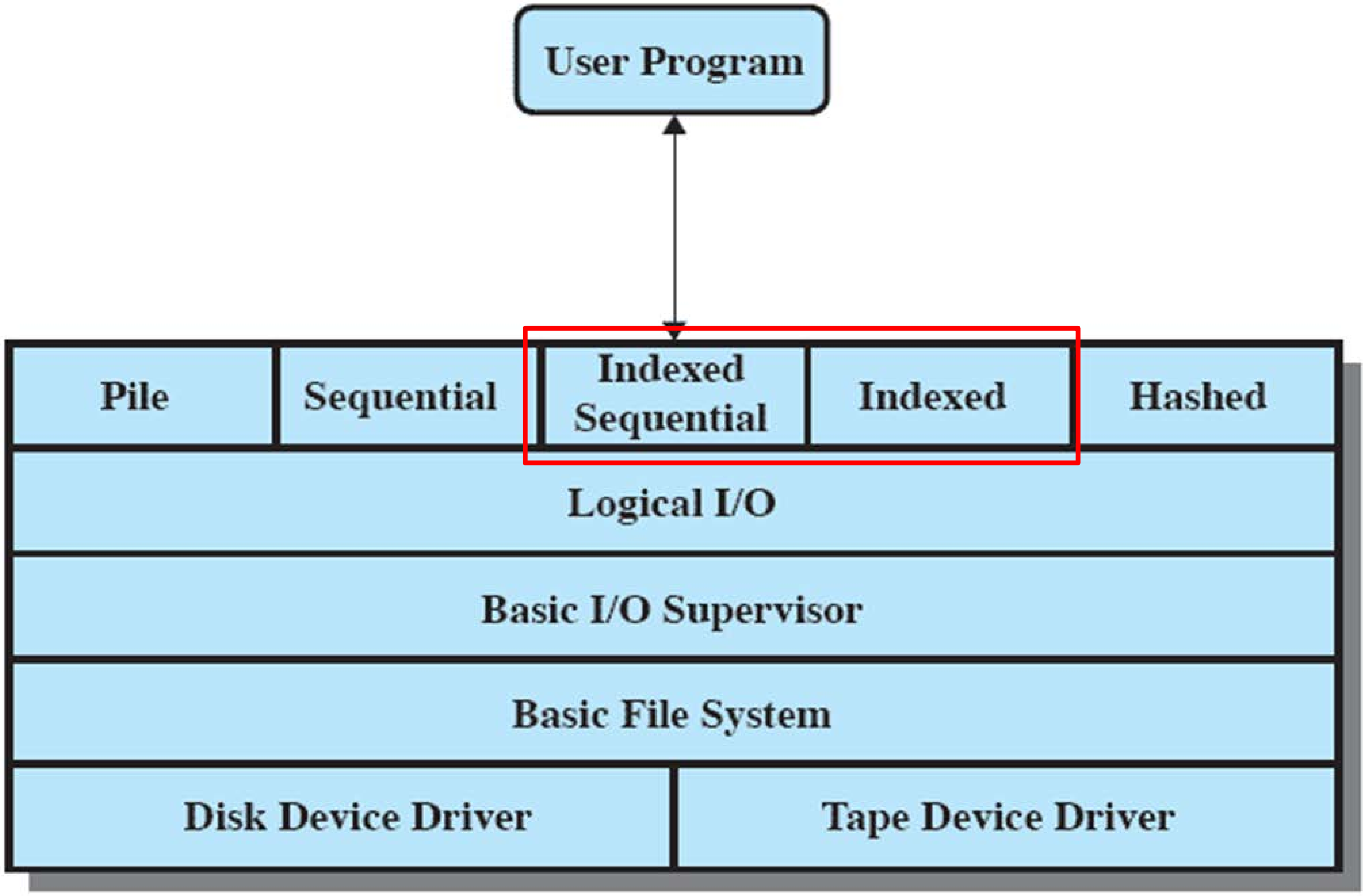
A	=	Excellent, well suited to this purpose	$\approx O(r)$
B	=	Good	$\approx O(o \times r)$
C	=	Adequate	$\approx O(r \log n)$
D	=	Requires some extra effort	$\approx O(n)$
E	=	Possible with extreme effort	$\approx O(r \times n)$
F	=	Not reasonable for this purpose	$\approx O(n^{>1})$

where

r = size of the result
o = number of records that overflow
n = number of records in file

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

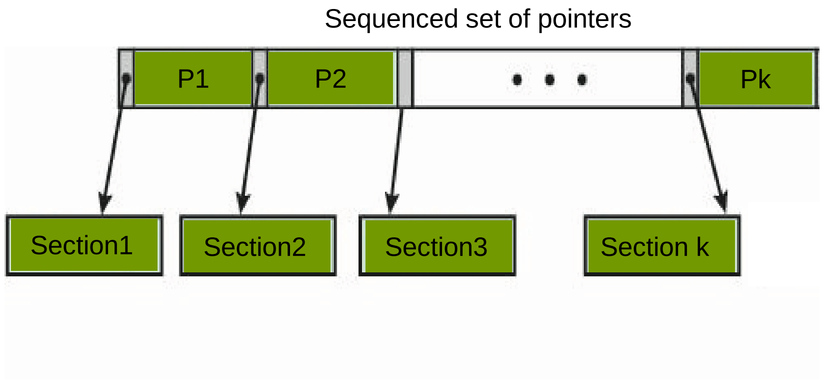
File system software architecture - revisited



Index structure

- There is common use of an index file to access individual records in a file or database.
  - For a large file or database, a single sequential file of indexes on the primary key does not provide for rapid access.

- To provide more efficient access, a structured index file is typically used.
  - The simplest such structure is a two-level organization in which the original file is broken into sections and the upper level consists of a sequenced set of pointers to the lower-level sections.
  - This structure can then be extended to more than two levels, resulting in a tree structure.



Index structure

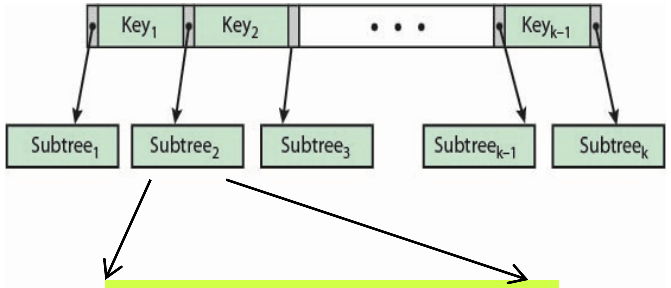
- Unless some discipline is imposed on the construction of the tree index, it is likely to end up with an uneven structure
  - with some short branches and some long branches, so that the time to search the index is uneven.
  - Therefore, a balanced tree structure, with all branches of equal length, would appear to give the best average performance.

- The B-tree has become the standard method of organizing indexes for databases and is commonly used in OS file systems
  - including Mac OS X, Windows, and several Linux file systems.

- B-tree structure provides for efficient searching, adding, and deleting of items.

B-Tree

- A B-tree is a tree structure (no closed loops)
  - The tree consists of a number of nodes and leaves.
  - Each node contains at least one key which uniquely identifies a file record, and more than one pointer to child nodes or leaves.
  - Each node is limited to the same number of maximum keys.
  - The keys in a node are stored in nondecreasing order. Each node has one more pointer than keys.



All nodes in $subtree_2$ has **Key** where
$Key_1 < Key \leq Key_2$

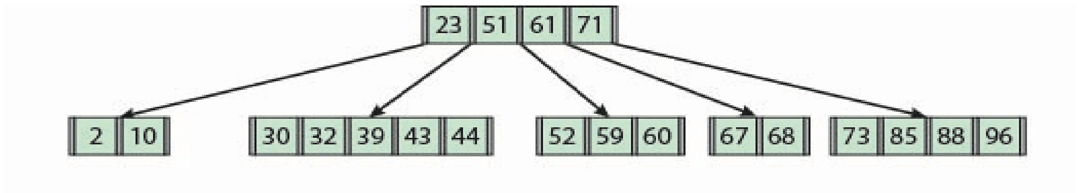
B-tree characteristics

- A B-tree is characterized by its minimum degree ***d*** and satisfies the following properties:
  - every node has **at most** 2*d* – 1 keys and 2*d* children or, equivalently, <u>2*d* pointers</u>
  - every node, <u>except for the root</u>, has **at least** *d* – 1 keys and *d* pointers, as a result, each internal node, except the root, is at least half full and has at least *d* children
  - the root has at least 1 key and 2 children
  - all leaves appear on the same level and contain no information. This is a logical construct to terminate the tree; the actual implementation may differ.
  - a nonleaf node with *k* pointers contains *k* – 1 keys

Inserting a new key in the B-tree structure

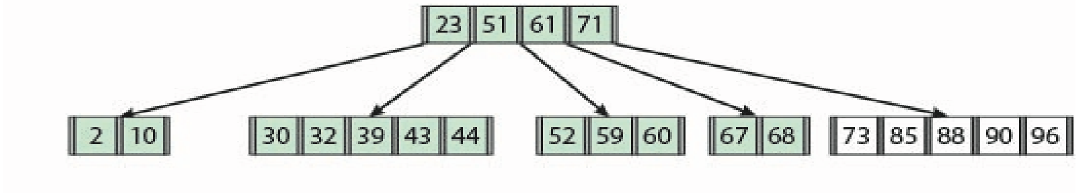
- After the insertion a B-tree must be a balanced tree:
  1. Search the tree for the key. If the new key is not in the tree, then you have reached a leaf (say L)
  2. If L has fewer than 2*d*-1 keys, insert in proper sequence
  3. If L is full (has 2*d*-1 keys), then split around its median key into two new nodes with *d*-1 keys in each and promote the median key to the next higher level. Place the new key in appropriate left or right node. The original node is now split into two nodes with *d*-1 keys and *d* keys.
  4. The promoted node is inserted into the parent node using the rules of Step 3.
  5. Apply the process of promotion if necessary until the root is reached. If the root is full then split using Step 3. This will increase the height of the tree by 1.

Inserting nodes into a B-Tree



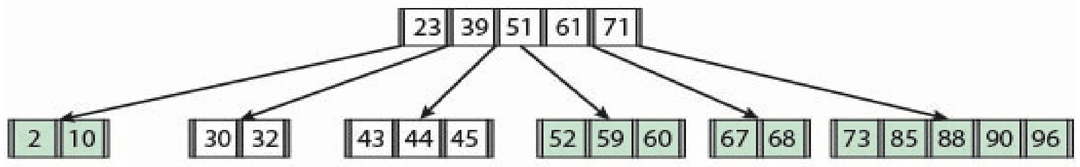
(a) B-tree of minimum degree d = 3.

Insert Key = 90



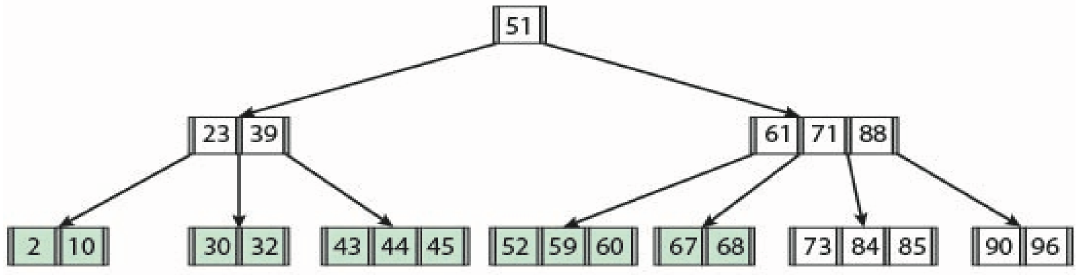
(b) Key = 90 inserted. This is a simple insertion into a node.

Insert Key = 45



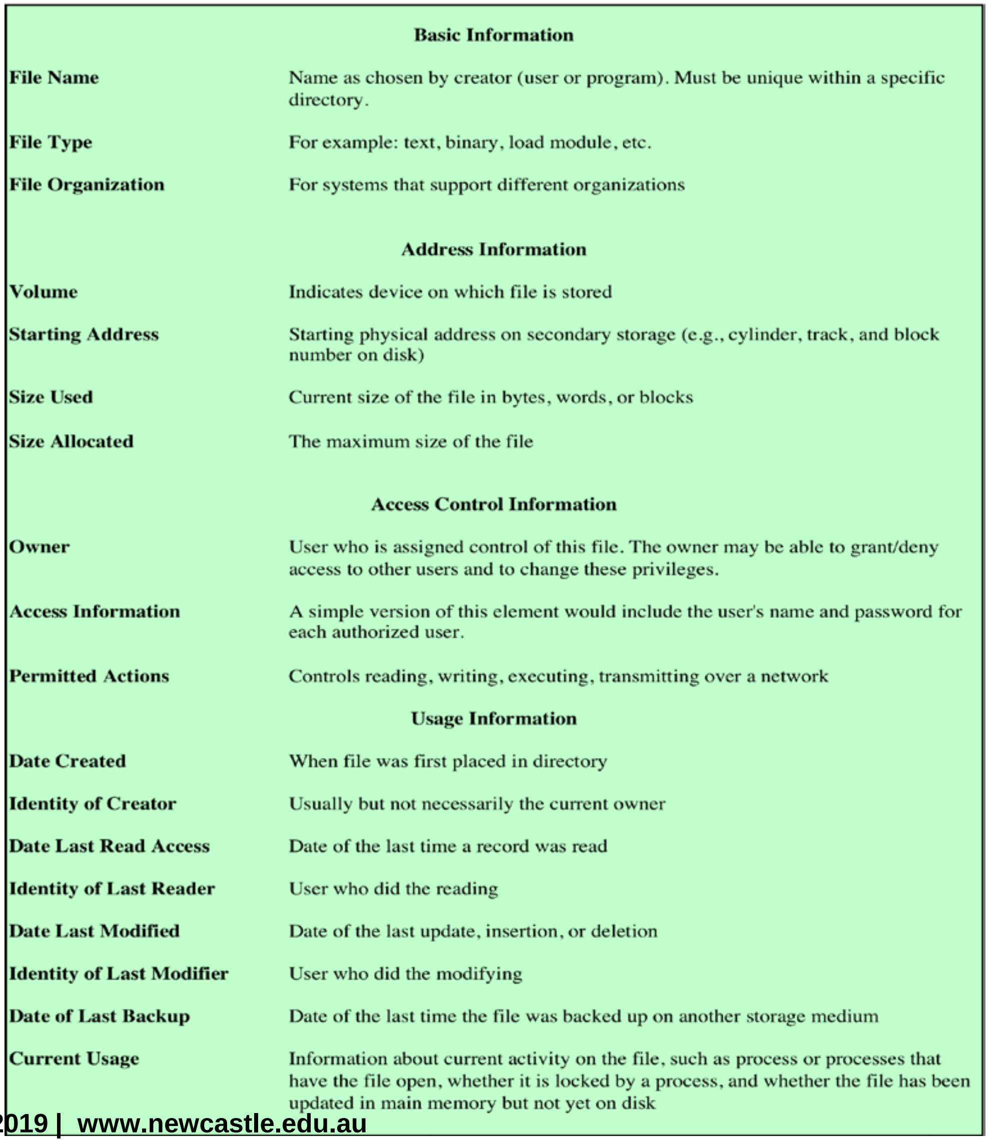
(c) Key = 45 inserted. This requires splitting a node into two parts and promoting one key to the root node.

Insert Key = 84



(d) Key = 84 inserted. This requires splitting a node into two parts and promoting one key to the root node
This then requires the root node to be split and a new root created.

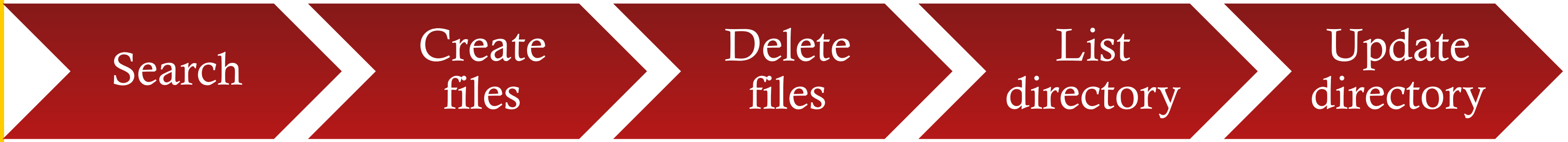
File directory information



	Basic Information
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File Type	For example: text, binary, load module, etc.
File Organization	For systems that support different organizations
	Address Information
Volume	Indicates device on which file is stored
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk)
Size Used	Current size of the file in bytes, words, or blocks
Size Allocated	The maximum size of the file
	Access Control Information
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges.
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network
	Usage Information
Date Created	When file was first placed in directory
Identity of Creator	Usually but not necessarily the current owner
Date Last Read Access	Date of the last time a record was read
Identity of Last Reader	User who did the reading
Date Last Modified	Date of the last update, insertion, or deletion
Identity of Last Modifier	User who did the modifying
Date of Last Backup	Date of the last time the file was backed up on another storage medium
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk

Operations performed on a directory

- To understand the requirements for a file structure, it is helpful to consider the types of operations that may be performed on the directory



- Use of a simple list for file names?

Two-level scheme

- There is one directory for each user and a master directory
- Master directory has an entry for each user directory providing address and access control information
- Each user directory is a simple list of the files of that user
- Names must be unique only within the collection of files of a single user
- File system can easily enforce access restriction on directories
- Does not help user in structuring collections of files

Tree-structured directories

- Master directory with user directories underneath it
- Each user directory may have subdirectories and files as entries
- Organization of directory:
  - Simplest approach is a sequential file
  - If directories may contain a very large number of entries then hashed structure may be preferred

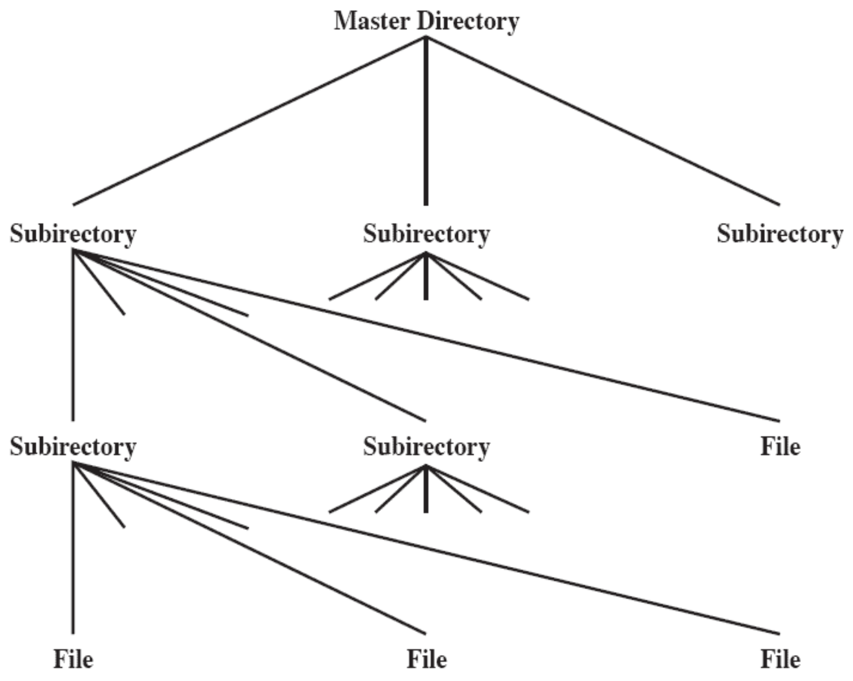


Figure 12.4 Tree-Structured Directory

Example Tree-structured directory

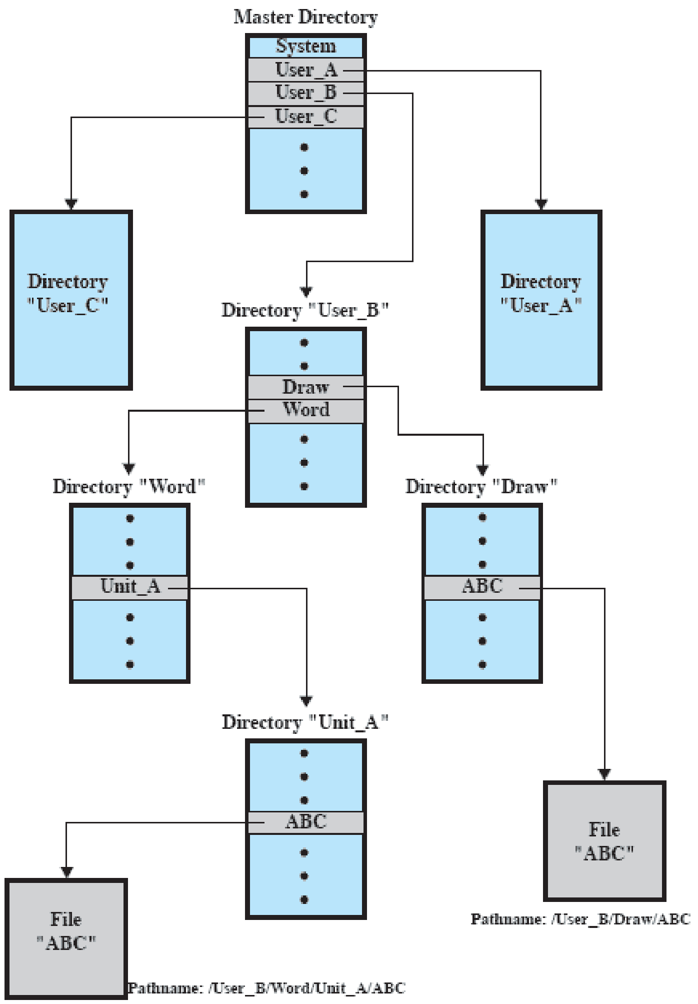


Figure 12.5 Example of Tree-Structured Directory

Example Tree-structured directory

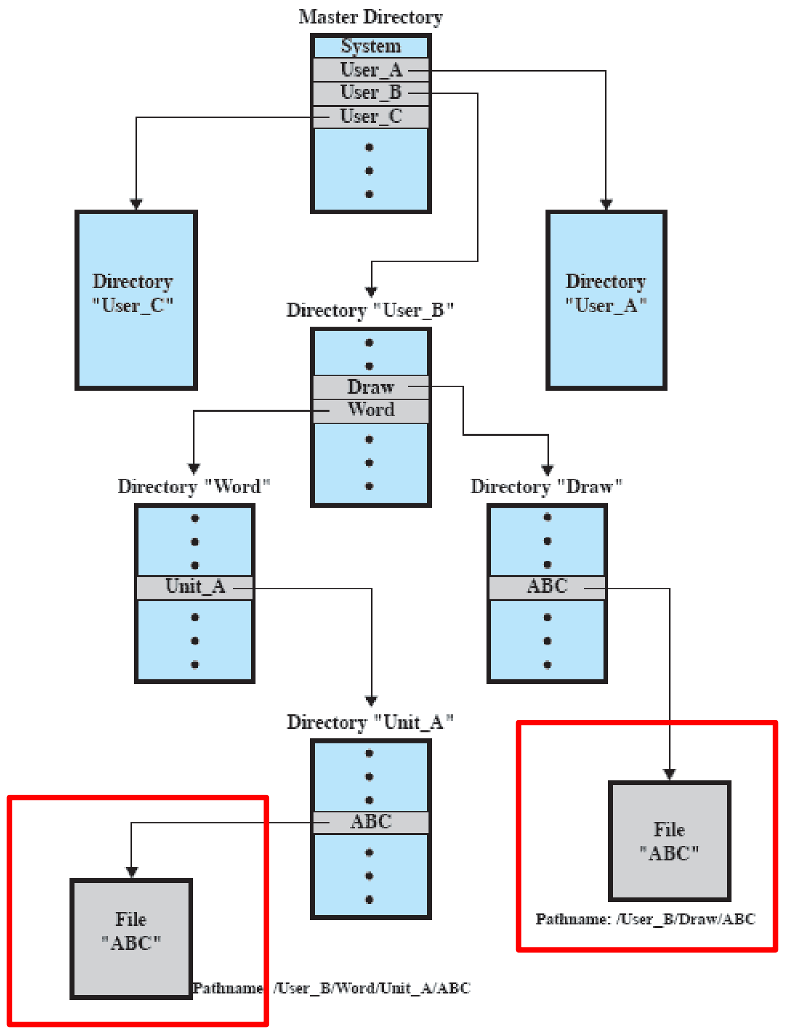
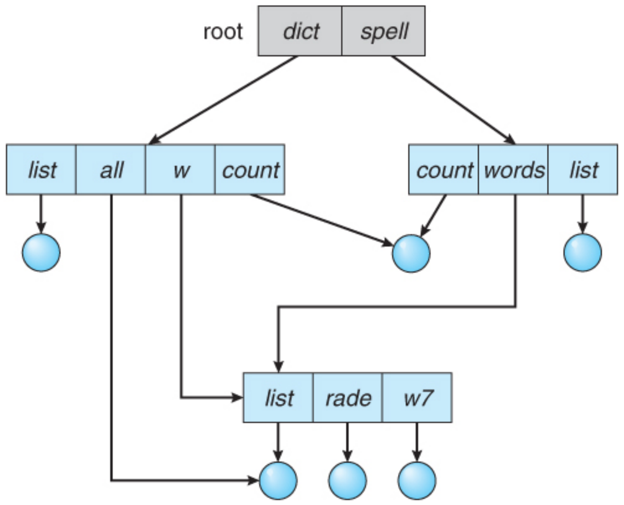


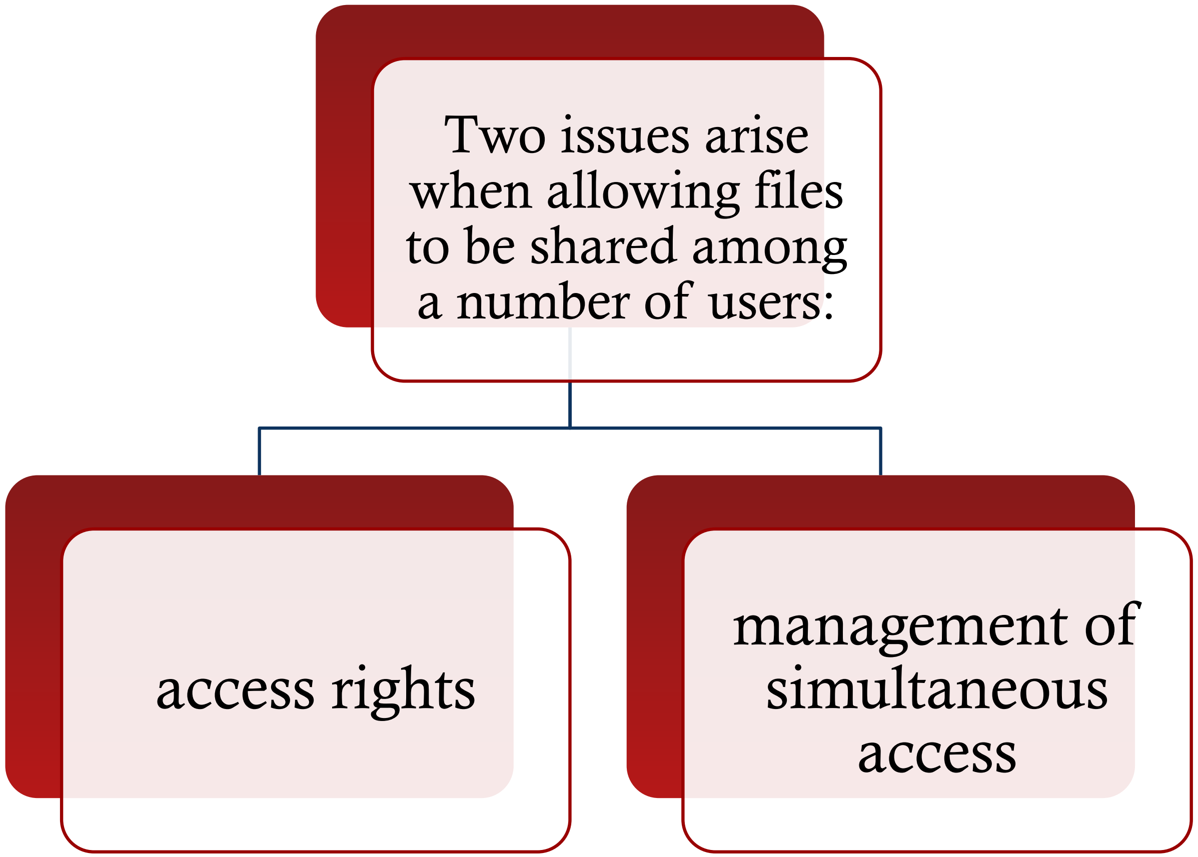
Figure 12.5 Example of Tree-Structured Directory

Acyclic-Graph directories

- The same file/subdirectory may belong to different directories
- Useful for sharing
- Common way to implement using **link**
  - A link is effectively a pointer
  - May be implemented as a pathname
- More complexities involved in accessing, deleting, renaming files and other operations.



File sharing



Two issues arise when allowing files to be shared among a number of users:

- access rights
- management of simultaneous access

Access rights

- File system should offer flexibility for allowing extensive file sharing
- Several options should be available to control access to a file
- Typically individual user or a group of users are granted access rights

- *None*
  - the user would not be allowed to read the user directory that includes the file
- *Knowledge*
  - the user can determine that the file exists and who its owner is and can then petition the owner for additional access rights
- *Execution*
  - the user can load and execute a program but cannot copy it
- *Reading*
  - the user can read the file for any purpose, including copying and execution
- *Appending*
  - the user can add data to the file but cannot modify or delete any of the file's contents
- *Updating*
  - the user can modify, delete, and add to the file's data
- *Changing protection*
  - the user can change the access rights granted to other users
- *Deletion*
  - the user can delete the file from the file system

User access rights

Owner

- usually the initial creator of the file
- has full rights
- may grant rights to others

Specific Users

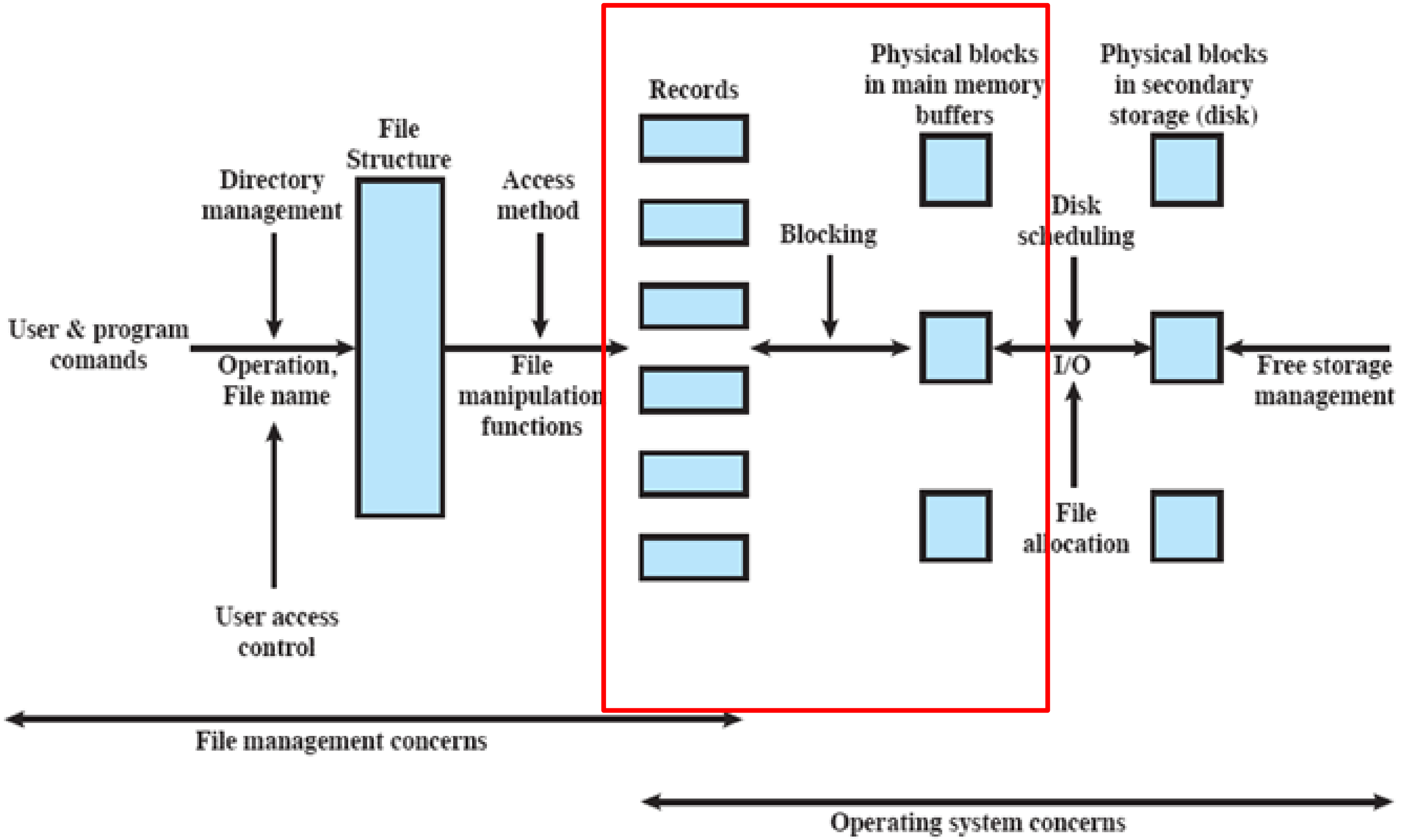
- individual users who are designated by user ID

User Groups

- a set of users who are not individually defined

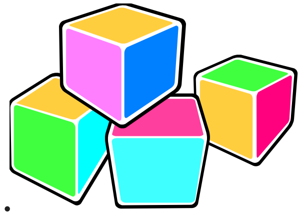
All

- all users who have access to this system
- these are public files


Records
Physical blocks in main memory buffers
Physical blocks in secondary storage (disk)
File Structure
Directory management
Access method
Blocking
Disk scheduling
User & program comands
Operation, File name
File manipulation functions
I/O
Free storage management
File allocation
User access control
File management concerns
Operating system concerns

Record blocking

- Blocks are the unit of I/O with secondary storage
    - for I/O to be performed records must be organized as blocks
- Most systems use fixed length block
    - Simplifies IO and buffer allocation in main memory
- Block size trade-off:
    - Larger blocks reduces I/O transfer time if records are accessed sequentially but needs larger buffer
    - Larger blocks transfer unused records if records are accesses randomly
- Given the size of a block, three methods of blocking can be used:

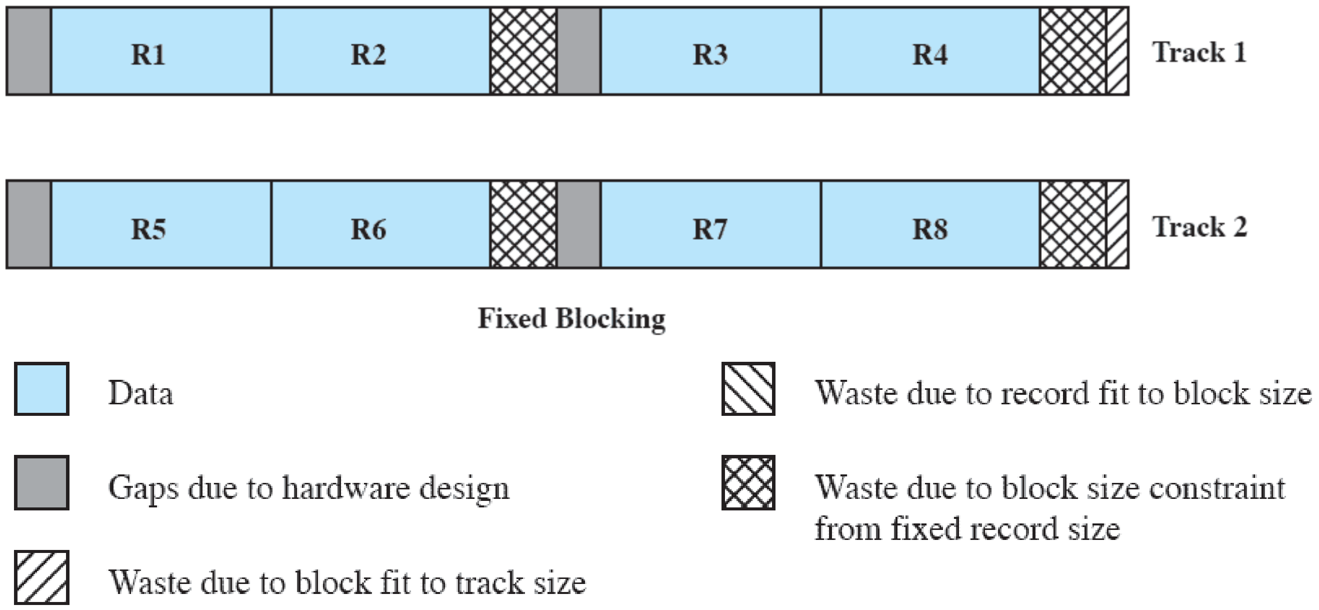


Fixed blocking

1) **Fixed Blocking** – fixed-length records are used, and an integral number of records are stored in a block

Internal fragmentation – unused space at the end of each block

Common mode for sequential files with fixed-length record.

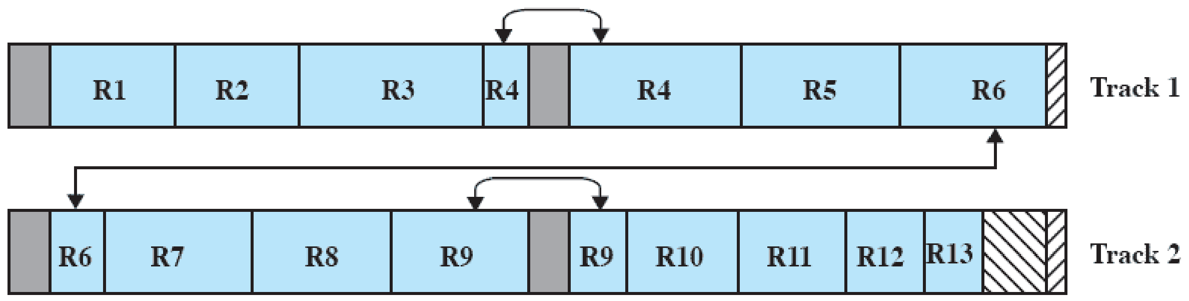

R1
R2
R3
R4
Track 1
R5
R6
R7
R8
Track 2
Data
Gaps due to hardware design
Waste due to block fit to track size
Waste due to record fit to block size
Waste due to block size constraint from fixed record size

Fixed Blocking

Variable blocking: spanned

2) **Variable-Length Spanned Blocking** – variable-length records are used and are packed into blocks with no unused space

Efficient of storage, not limited by the record size but difficult to implement



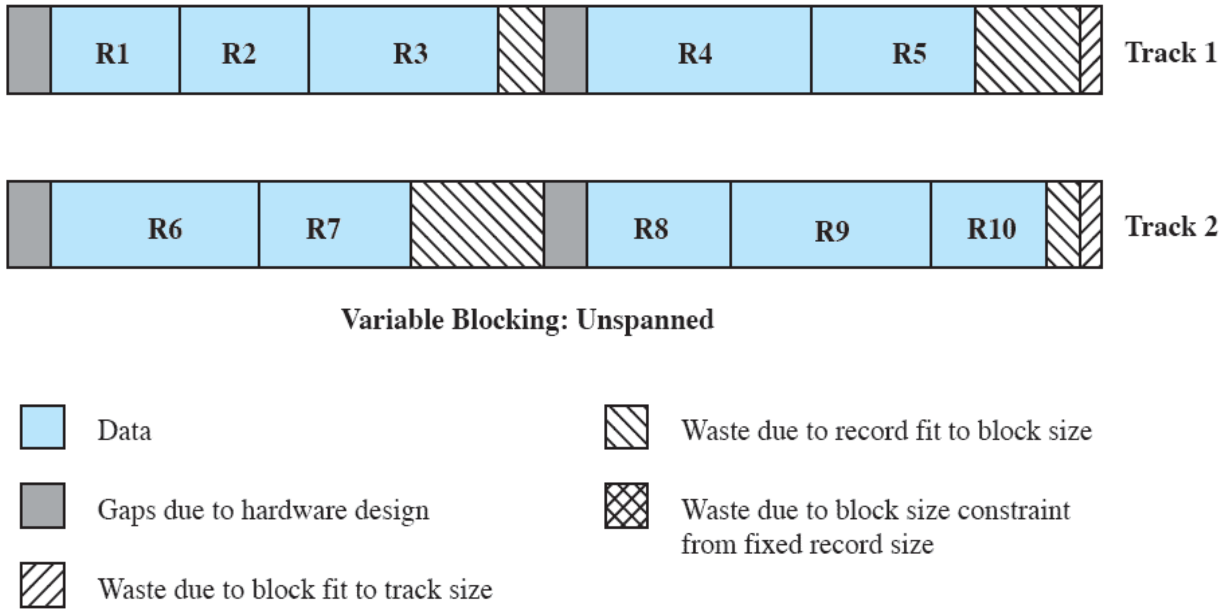
Variable Blocking: Spanned

🟦 Data	▧ Waste due to record fit to block size
⬛ Gaps due to hardware design	▦ Waste due to block size constraint from fixed record size
▨ Waste due to block fit to track size	

Variable blocking: unspanned

3) **Variable-Length Unspanned Blocking** – variable-length records are used, but spanning is not employed

Wastes space and limits record size to the size of a block


R1
R2
R3
R4
R5
Track 1
R6
R7
R8
R9
R10
Track 2
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 Blocking: Unspanned

File allocation

- On secondary storage, a file consists of a collection of blocks
- The operating system or file management system is responsible for **allocating blocks to files**
- The approach taken for file allocation may influence the approach taken for **free space management**
- These two tasks are related.

File allocation issues

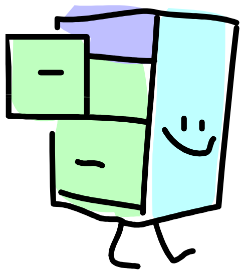
1. When a new file is created, is the maximum space required for the file allocated at once?

2. Space is allocated to a file as one or more contiguous units called ***portions*** (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?
   - An example of such a structure is a file allocation table (FAT) , found on DOS and some other systems.

Preallocation vs dynamic allocation

- A preallocation policy requires that the maximum size of a file be declared at the time of the file creation request
- For many applications it is difficult to estimate reliably the maximum potential size of the file
  - tends to be wasteful because users and application programmers tend to overestimate size
- Dynamic allocation allocates space to a file in portions as needed



Portion size

- In choosing a portion size there is a trade-off between efficiency from the point of view of a single file versus overall system efficiency

- Items to be considered:
  1. contiguity of space increases performance, especially for `Retrieve_Next` operations, and greatly for transactions running in a transaction-oriented operating system
  2. having a large number of small portions increases the size of tables needed to manage the allocation information
  3. having fixed-size portions simplifies the reallocation of space
  4. having variable-size or small fixed-size portions minimizes waste of unused storage due to overallocation

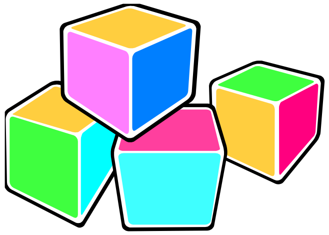
Alternatives

Variable, large contiguous portions

- provides better performance
- the variable size avoids waste
- the file allocation tables are small

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



Variable allocation

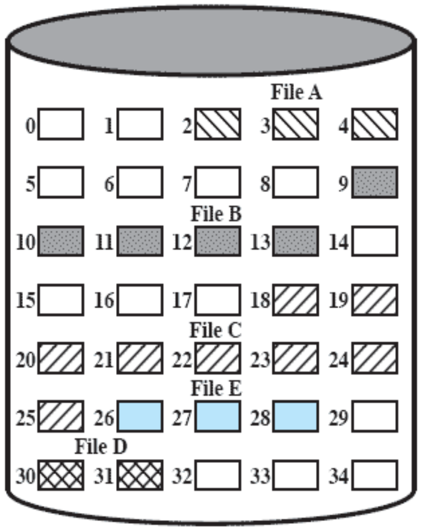
- **First fit**: Choose the first unused contiguous group of blocks of sufficient size from a free block list.
- **Best fit**: Choose the smallest unused group that is of sufficient size.
- **Nearest fit**: Choose the unused group of sufficient size that is closest to the previous allocation for the file to increase locality.
- It is not clear which strategy is best. The difficulty in modelling alternative strategies is that so many factors interact, including
  – types of files,
  – pattern of file access,
  – degree of multiprogramming,
  – other performance factors in the system,
  – disk caching, disk scheduling, and so on.

File allocation methods

1. Contiguous
2. Chained
3. Indexed

Contiguous file allocation

- A single contiguous set of blocks is allocated to a file at the time of file creation
- **Preallocation strategy** with **variable-size portions**
- A single entry for each file in file allocation table
- Is the best from the point of view of the individual sequential file
  - Multiple blocks can be read in at a time
  - Easy to retrieve a single block

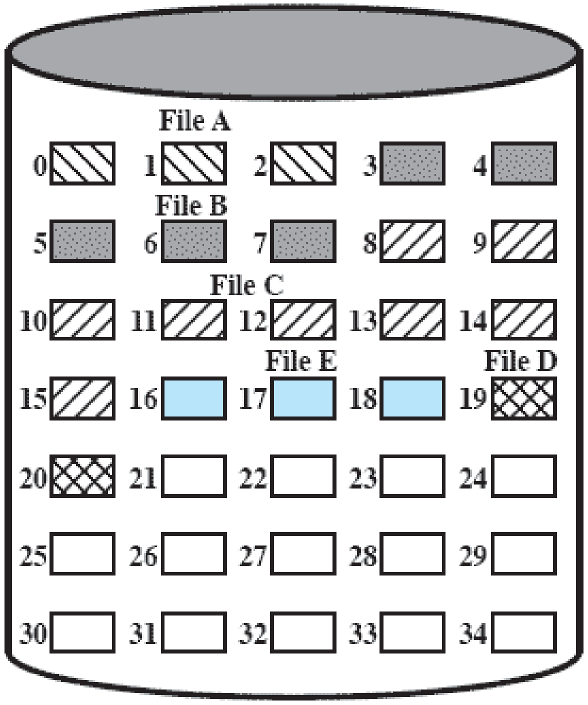


File Allocation Table

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

Figure 12.7 Contiguous File Allocation

After compaction



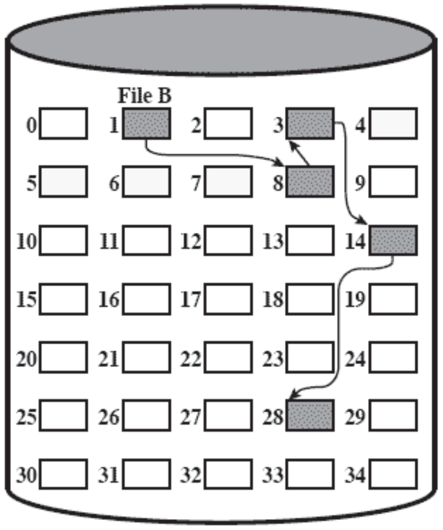
File Allocation Table

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

Figure 12.8 Contiguous File Allocation (After Compaction)

Chained allocation

- Allocation is on an individual block basis
- Each block contains a pointer to the next block in the chain
- The file allocation table needs just a single entry for each file
- Pre-allocation is possible but more common to allocate blocks as needed
- No external fragmentation
- Best for sequential files to be processed sequentially
- No accommodation of the principle of locality

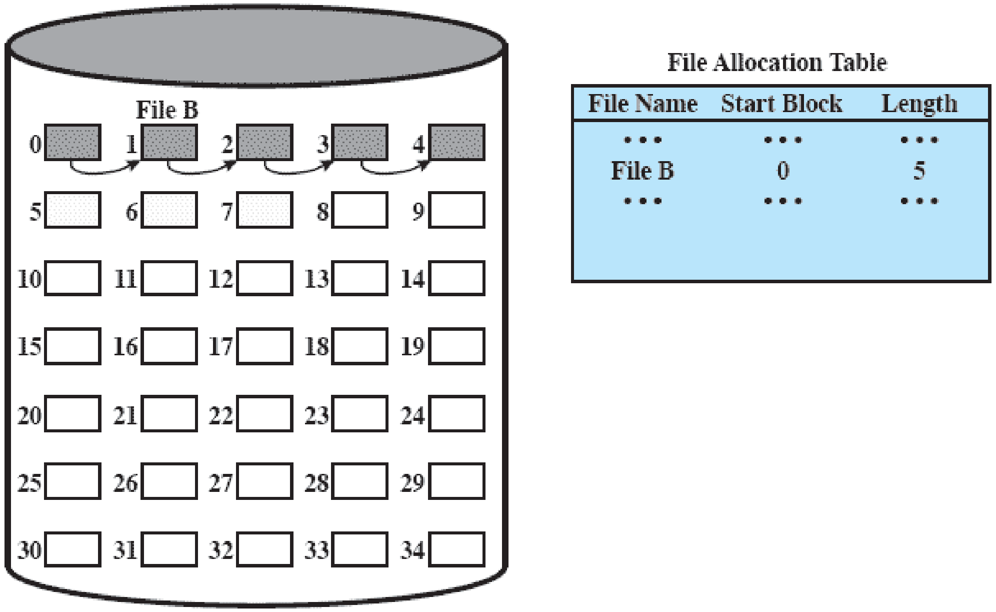


File Allocation Table

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

Figure 12.9 Chained Allocation

Chained allocation after consolidation



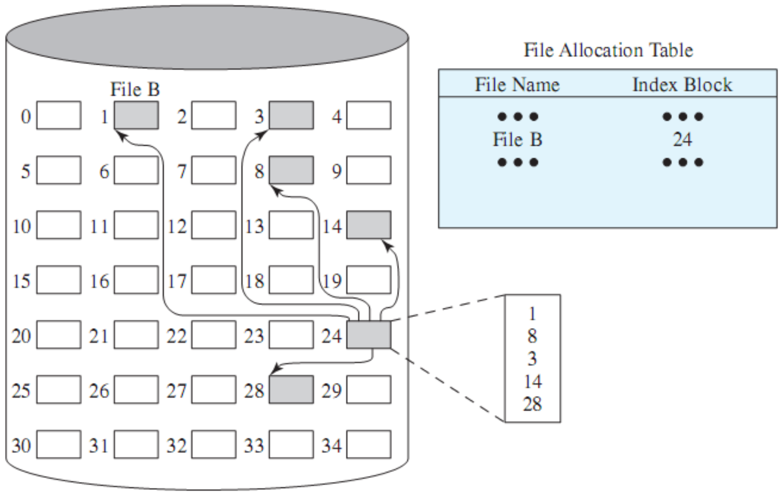
File Allocation Table

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

Figure 12.10 Chained Allocation (After Consolidation)

Indexed allocation

- Addresses many of the problems of the other two methods
- File allocation table contains a separate one-level index for each file.
  - Index has one entry for each portion allocated to the file
  - Typically index are not physically stored in the file allocation table rather file index is kept in a separate block.



File Allocation Table

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

Figure 12.11 **Indexed Allocation with Block Portions**

Indexed allocation

Block Portion VS Variable length portion

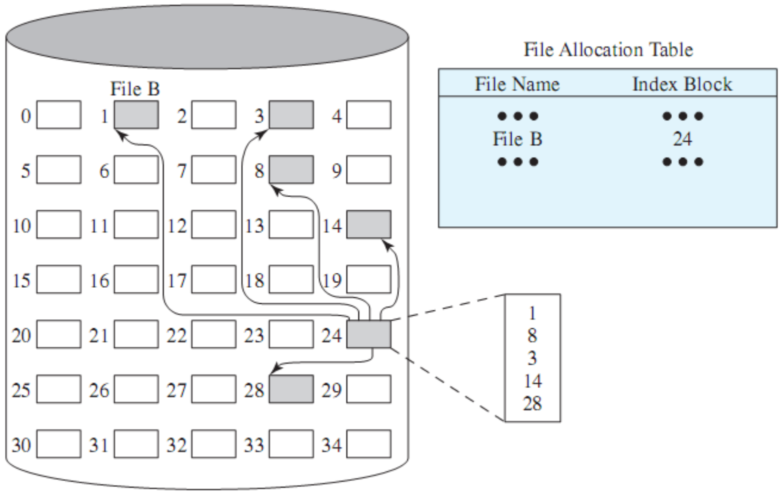


Figure 12.11 Indexed Allocation with Block Portions

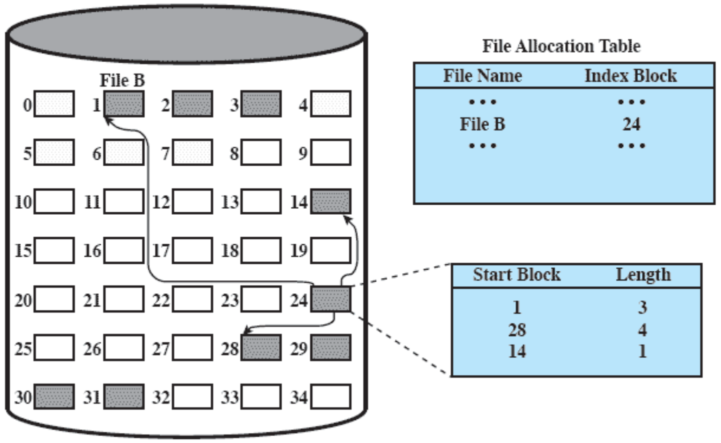


Figure 12.12 Indexed Allocation with Variable-Length Portions

File allocation methods

	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

Free Space Management

- Just as allocated space must be managed, so must the unallocated space
- To perform file allocation, it is necessary to know which blocks are available
- A ***disk allocation table*** is needed in addition to a file allocation table

Bit Tables

- This method uses a vector containing one bit for each block on the disk

  00111000011111000011111111111011000

- Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use

Advantages:

- easy to find one or a contiguous group of free blocks
- works well with any file allocation method
- it is as small as possible

Chained Free Portions

- The free portions may be chained together by using a pointer and length value in each free portion
- Negligible space overhead because there is no need for a disk allocation table
- Suited to all file allocation methods

Disadvantages:

- every time you allocate a block you need to read the block first to recover the pointer to the new first free block before writing data to that block

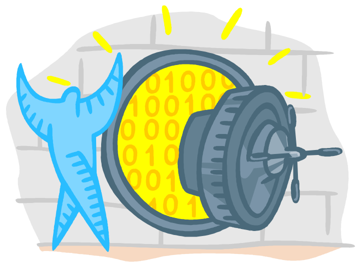
Indexing

- Treats free space as a file and uses an index table as it would for file allocation
- For efficiency, the index should be on the basis of variable-size portions rather than blocks
- One entry in the table for every free portion on the disk
- This approach provides efficient support for all of the file allocation methods

Free Block List

- Each block is assigned a number sequentially
  - the list of the numbers of all free blocks is maintained in a reserved portion of the disk
- Depending on the size of the disk, either 24 or 32 bits will be needed to store a single block number
  - the size of the free block list is 24 or 32 times the size of the corresponding bit table and must be stored on disk
- There are two effective techniques for storing a small part of the free block list in main memory:
  - the list can be treated as a push-down stack with the first few thousand elements of the stack kept in main memory
  - the list can be treated as a FIFO queue, with a few thousand entries from both the head and the tail of the queue in main memory

Volumes



- A collection of addressable sectors in secondary memory that an OS or application can use for data storage
- The sectors in a volume need not be consecutive on a physical storage device
  - they need only appear that way to the OS or application
  - i.e. a logical disk
- A volume may be the result of assembling and merging smaller volumes

Summary

- A file management system is a set of system software that provides services to users and applications in the use of files, including file access, directory maintenance and access control
- The file management system is typically viewed as a system service that itself is served by the operating system rather than being part of the operating system itself.
- However, in any system, at least part of the file management function is performed by the operating system.
- .

Summary

- If a file is primarily to be processed as a whole, a sequential file organization is the simplest and most appropriate
- If sequential access is needed but random access to individual file is also desired, an indexed sequential file may give the best performance
- If access to the file is principally at random, then an indexed file or hashed file may be the most appropriate
- Whatever file structure is chosen, a directory service is also needed. Directory service allows files to be organized in a hierarchical fashion
- File records generally do not conform to the size of a physical disk block. Therefore, some sort of blocking strategy is needed
- Key function of file management scheme is the management of disk space
  - strategy for allocating disk blocks to a file
  - maintaining a disk allocation table indicating which blocks are free

References

- **Operating Systems – Internal and Design Principles**
  - By William Stallings
- Chapter 12