DATE / /

Module-4

Input Output and File Management

4.2

I/O Devices

1) Because there is a wide variety of devices and applications used for I/O, it is difficult to develop a general and consistent solution.

2) There are 3 types of I/O devices:

a) Human Readable: Suitable for communi-cation with user, Eg: Printers, keyboard, mouse, etc.

b) Machine Readable: Suitable for communicating with electronic equip-ment. Eg: Disk drives, USB keys, etc.

c) Communication: Suitable for communication with remote devices. Eg: Modems

3) The key difference between each type of I/O device is:

a) Data Rate: Difference in data transfer

rates.

b) Application: The type of application use of the device influences the type of support it gets from OS.
Eg: A disk used for file files requires the support of file management software.

It might get different privileges and priorities by the OS depending on the application.

c) Complexity of Control: Some devices require a more complex control interface (interface that is used to manage that I/O device).
Eg: A printer's complexity is lesser than disk's

* Data Rate of different I/O devices:

{{{ Figure 11.1 }}}

d) Unit of Transfer: Data transfer may happen in any unit of transfer: bytes, characters, blocks, etc.

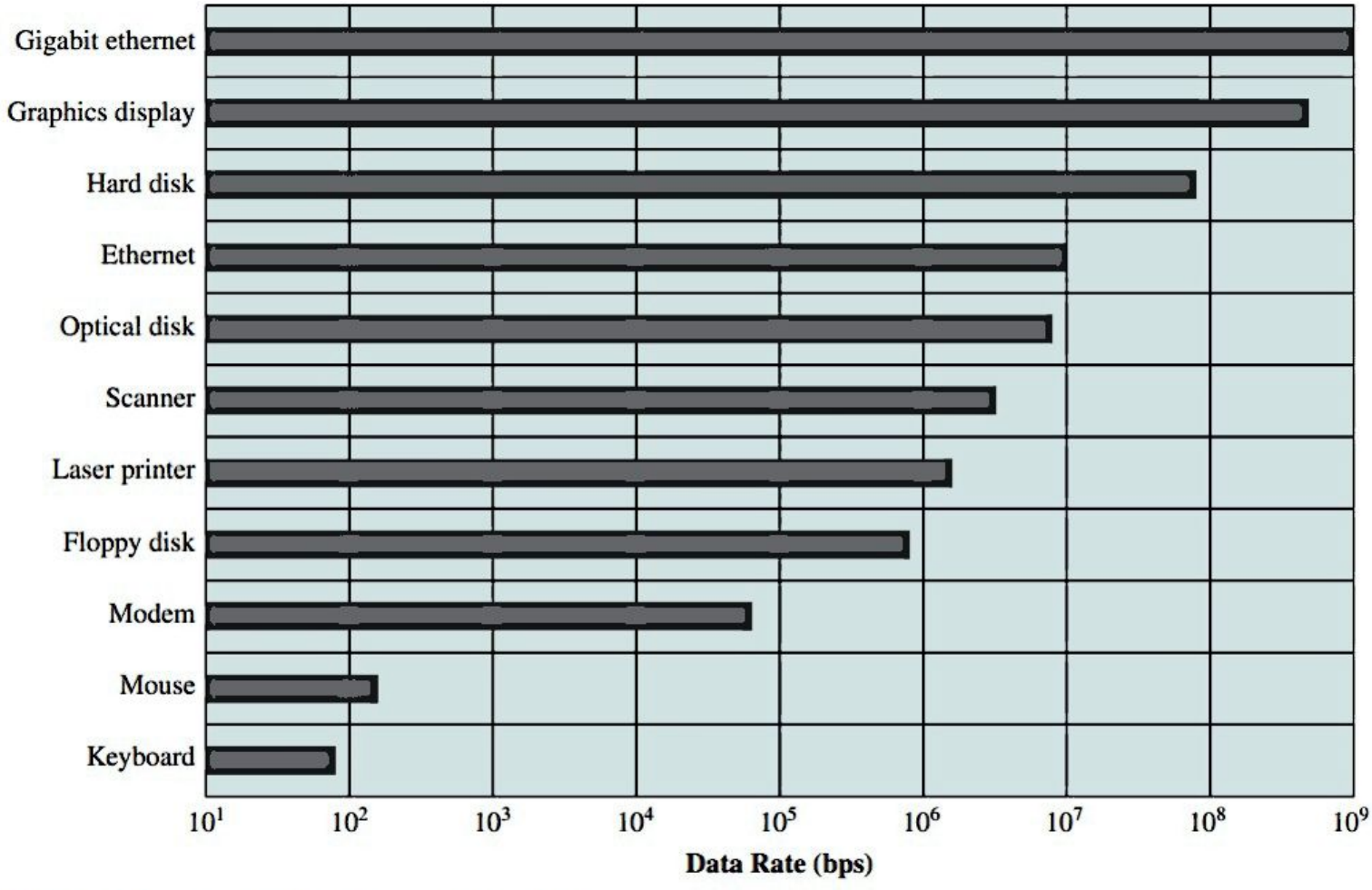


Figure 11.1 **Typical I/O Device Data Rates**

e) Data Representation: Different data encoding is used by different devices.

f) Error Conditions: The different types of errors, the ~~repo~~ way of reporting these errors, their consequences, & the responses to these errors.

Above factors are to be considered by the OS before designing an approach to I/O devices. This diversity makes it difficult for achieving a consistent solution.

Organization of the I/O function

1) There are 3 techniques to perform I/O:

a) Programmed I/O: The process issues an I/O command to an I/O module on behalf of the process. The process waits for I/O to complete before proceeding.

b) Interrupt Driven I/O: After the processor issues an I/O command for the I/O process, the I/O instru-ction is either blocking or

non-blocking to the process.

If blocking I/O, then the control is taken from this process & given to the OS, which blocks this process & schedules another process.

If non-blocking, the other instructions in the process are continued by the processor to execute.

c) Direct Memory Access (DMA): The DMA module controls the exchange of data between the I/O module and the main memory. The processor requests data transfer to the DMA module and is interrupted only after the entire data is transfered, PS: The process that requests I/O is in the main memory.

DMA is most used by the OS.

* Note: Programmed I/O - The processor waits for I/O instruction to complete along with the process.

Interrupt I/O - The processor leaves the

process execution in case of blocking interrupt I/O instruction. After this instruction is done execution, the process sends an interrupt to the processor.

{{Table 11.1}}

2) The evolution of the I/O function:
The evolution steps are as followed:

a) The processor directly controls the peripheral devices (I/O devices). Used in simple microprocessor-controlled device.

b) An I/O module is added. Processor uses the programmed I/O.

c) Same as b), but interrupts are introduced, thus processor can do some other task while waiting.

d) The I/O module is given direct access to the main memory using DMA, thus data transfer to main memory can happen without involving the processor.

Table 11.1 I/O Techniques

	No Interrupts	Use of Interrupts
I/O-to-Memory Transfer through Processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-Memory Transfer		Direct memory access (DMA)

e) The I/O module is enhanced to become a separate processor. The I/O processor fetches
f) and executes instructions from the I/O program without processor intervention.
Thus the processor need not be interrupted before the sequence of instructions are performed.

g) The I/O ~~memory~~ module has a local memory of its own, further reducing process intervention requirement.

With each step in evolution, the processor is getting more & more relieved of the I/O related tasks.

- I/O channel: The I/O module with a separate processor is called I/O channel.

- I/O processor: The I/O module with its own memory is called I/O processor.

3) Direct Memory Access:
The DMA is capable of mimicking the

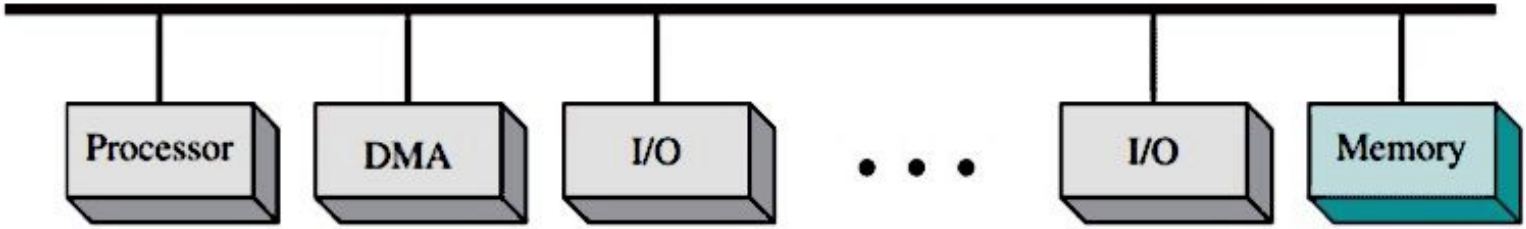
processor and deal with system buses.

When the processor wishes to read or write a block of data, it commands the DMA module by sending it the following information:

a) Whether a read or write is requested
b) The address of the I/O device.
c) The starting location in memory to read from & write to. DMA stores this in the address register it has.
d) The number of words to be read or written. DMA stores this in its data count register.

- The processor gives its work to the DMA module & leaves. The DMA module performs the I/O data transfer and interrupts the processor after it is done.

Thus, processor only involved in the beginning and the end of data transfer.



(a) Single-bus, detached DMA

In the above figure, 1 system bus is used. The # DMA module uses programmed I/O for data transfer between I/O device & the memory. This method is ~~inex~~ cheap but inefficient as 2 by bus cycles required for transfer of each data word.

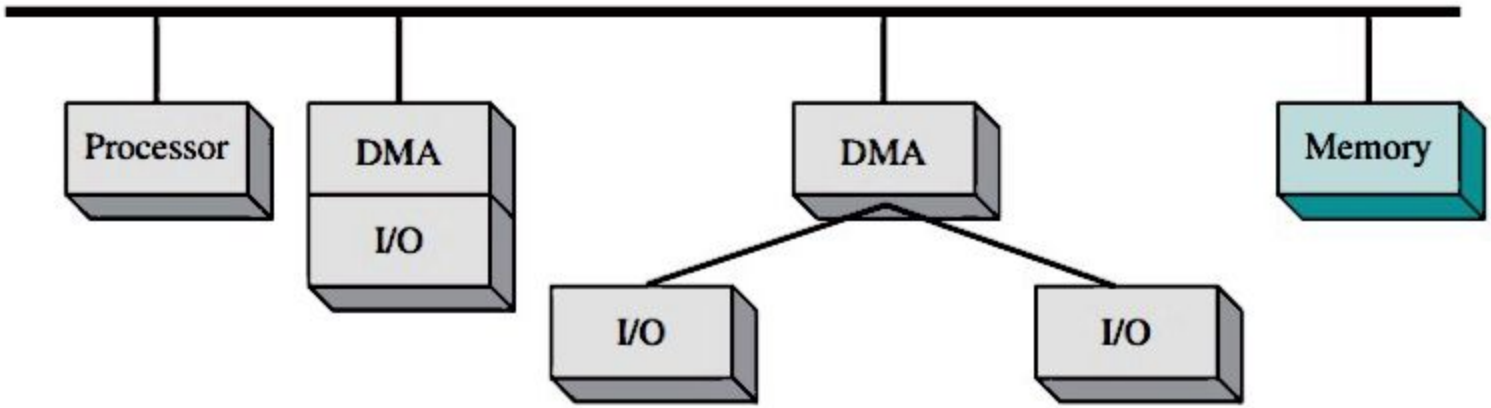
{{{ Figure 1.13 (b) }}}

In the above method, the DMA module is integrated with the I/O functions and no system buses are required for this data transfer. The DMA may control 1 or more I/O.

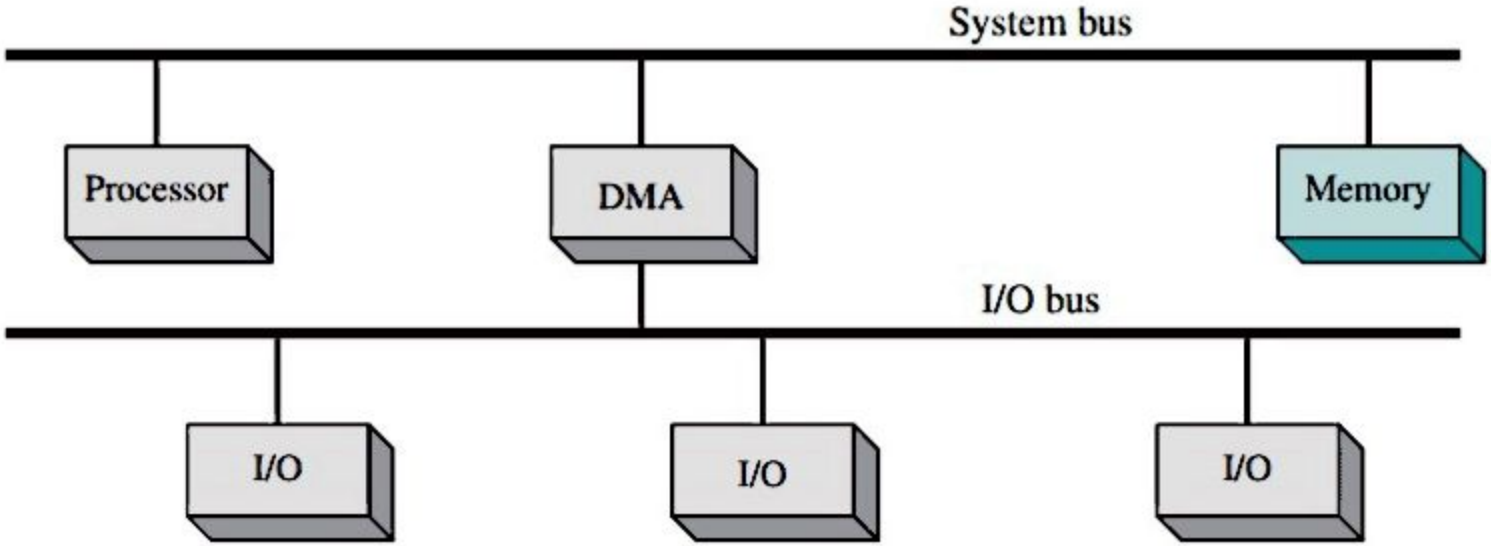
{{{ Figure 11.3 (c) }}}

In the above structure, the I/O bus is used to connected I/O modules with the DMA module. The DMA module now only needs 1 I/O interface.

* Note that the system bus is always required by the DMA module to transfer data to the main memory.



(b) Single-bus, integrated DMA-I/O



(c) I/O bus

DATE / /

Operating System Design Issues:

1) Design Objectives:

There are 2 main objectives in designing the I/O facility:

- Efficiency
- Generality.

a) Efficiency: As we know, I/O buffers are degrees slower than the processor, and can slow it down even with a great deal of multiprogramming support (which allows the process to be in blocked state during I/O).

A major effort is thus given to I/O design to improve the efficiency of I/O, most importantly for disk I/O, which is most often used.

b) Generality: For simplicity & freedom from errors, it is desirable to handle all devices in uniform manner. But this is difficult to do because of the diversity of I/O types.

Thus we can use a modular approach for the device functions design. This design hides details of the I/O device and the device I/O can be accessed using functions like read, write, open, close, lock & unlock.

2) Logical Structure of the I/O function:

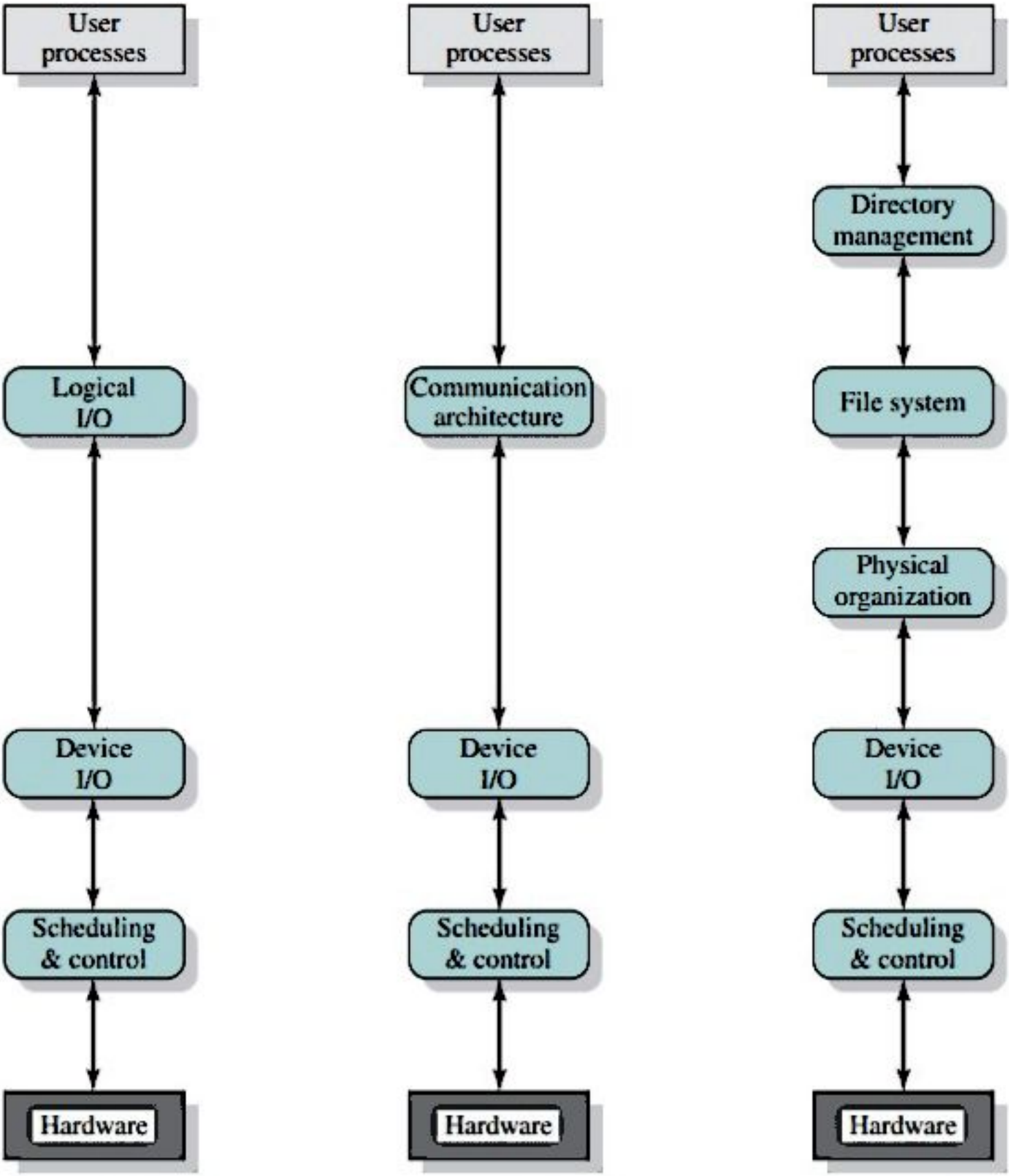
a) The OS design uses a hierarchical structure, in which there are layers and the function of the upper layers use the functions in the lower levels which are more primitive than the upper layer function.

Thus a big problem is decomposed to smaller subproblems.

b) The functions in the lower levels that interact with the hardware take 1/billion seconds to complete.

While, the functions in user level can take many seconds.

The layered structure does well in such environment.



(a) Local peripheral device (b) Communications port (c) File system

Figure 11.4 A Model of I/O Organization

DATE / /

c) Same layered structure philosophy can be used for I/O devices, as shown above.

The logical layers are:

→ Local Peripheral Device

i) Logical I/O: Manages general I/O functions on behalf of the user processes, allowing them to access device in terms of simple commands like open, close, read & write.

ii) Device I/O: The requested operations are converted to appropriate sequence of I/O instructions, channel commands & orders. Buffering techniques may be used here.

iii) Scheduling and Controlling: The scheduling & queuing of I/O operations occur at this layer. Interrupts are also handled at this layer, the I/O status is collected and reported. This layer interacts with the I/O module, and the device hardware.

→ Communications Port:

Similar to local the peripheral device but the logical I/O is replaced by communications architecture.

→ File System:

i) Directory Mangement: The symbolic file names are converted to identifiers that are used to reference the files.
Also concerned with user operation like add, delete, etc.

ii) File system: Deals with logical structure of the file, operations such as open, close, read, write and access rights.

iii) Physical Organization: logical references to files are converted physical storage addresses at this layer (like how logical addresses need to be mapped to physical addresses in memory).

I/O Buffering

1) If an I/O operation is to be done at

to 1500

location 1000 of main memory, then the data & processes in this memory cannot be swapped out by the OS, because that would result in loss of data. Thus I/O interferes with swapping decisions by the OS.

Thus, the user memory involved with the I/O request should be locked in main memory.

Some considerations are required for outgoing operation.

~~a) a) Block oriented device:~~

2) There are 2 types of I/O devices:

a) Block Oriented Device: Stores information in blocks of usually fixed sizes, transfers are done 1 block at a time. Data can be referenced based on block nos. Eg: Disks and USB keys

b) Stream Oriented Device: Transfers data as a stream of bytes, no blocks are used. Eg: Terminals, printers, etc.

3) To avoid the inefficiencies caused by the difference in I/O speed and processor speed, buffering is used. There are 3 types of buffering:

a) ~~Simple buffers~~ :

a) Single Buffer :

When a user process requests I/O, a buffer is assigned to the user process, in main memory.

→ For block oriented devices, the input transfers are made to the buffer. When a block is transferred to the buffer, the block is moved to the user space and another block is requested.

Thus, the user process process that 1 block of data while the next block is being read in.

The OS can also swap out process without the worry of data loss, because the I/O will put data in the buffer

that exists in the system, not in the user process memory like before.

Note: The buffer exists in main memory seperate from the user space, but it is associated with that process still.

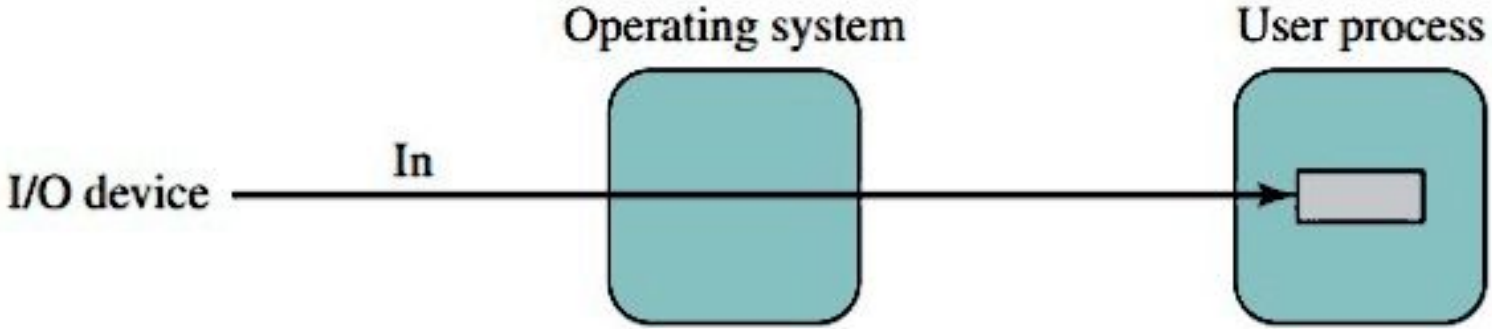
However, the OS must now keep track of the assignment of user buffers to the user processes.

Similarly, outgoing data from user process to I/O modules also happen using system buffer.

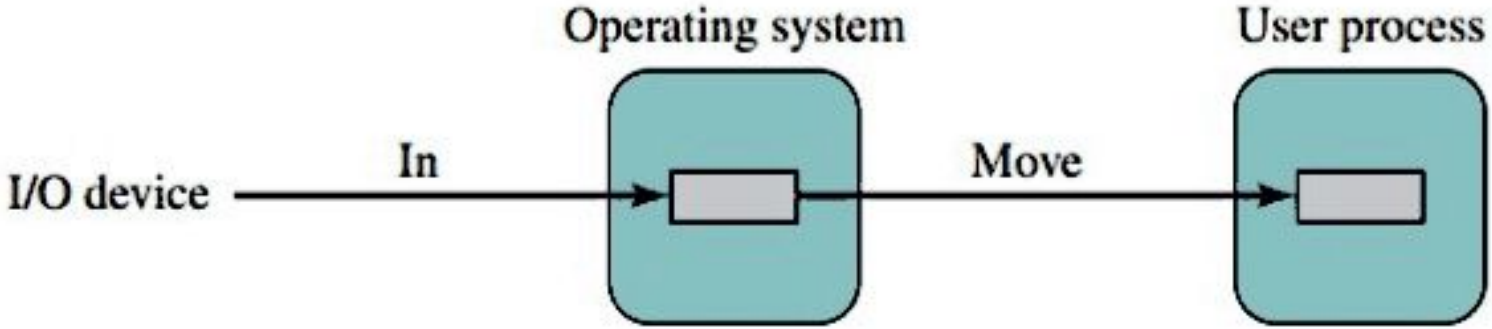
{{ figure 11.5 (a) and (b) }}

→ for Stream oriented I/O, buffering is either in byte-at-a-time or line-at-a-time fashion.

In line-at-a-time, the user input is one line at a time & the buffer holds a ~~sing~~ single line & sends it to the user program when 1 line is received. Eg: Scroll-mode terminals (like CMD), line printer.



(a) No buffering



(b) Single buffering

Byte-at-a-time operation is used when each keystroke is important & in peripherals like sensors.
The interaction between OS and user process follows the producer-consumer probl- model.

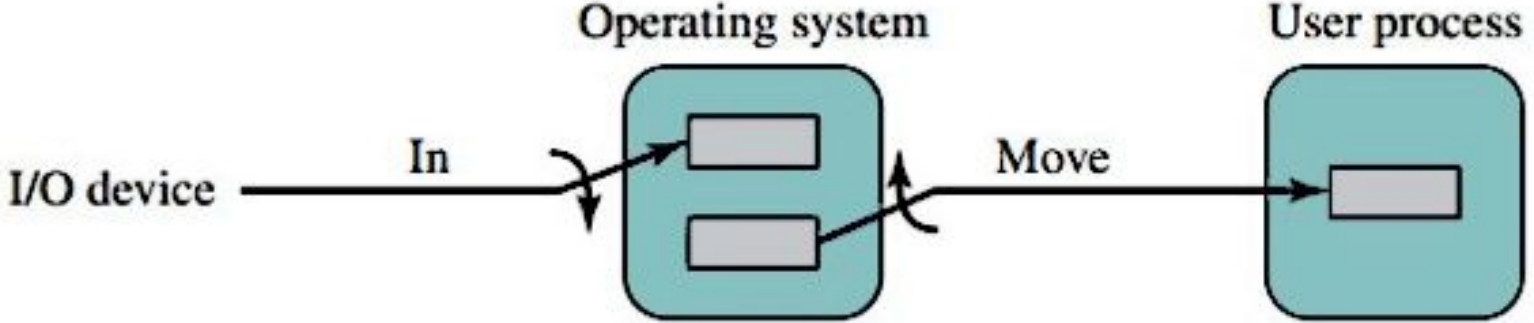
b) Double Buffer

Now 2 system buffers are assigned. ~~Data is inputted to a buffer &~~
I/O sends data to 1 buffer, user program takes data from the other buffer.
Then, the buffers are switched.

For stream oriented there are again 2 modes, line-at-a-time and byte-at-a-time.

#Note: The buffers in single buffering cannot be accessed by the I/O module & the user program simultaneously, that's why double buffering is useful.

{{{ Figure 11.5 (c) }}}



(c) Double buffering

c) Circular Buffer

When more than 2 buffers are called, these multiple buffer systems are called circular buffer. Each buffer being 1 unit of the circular buffer.
This is the bouded-buffer producer / consumer model.

4) Utility of Buffering:

In multiprogramming environment, buffering can increase efficiency of the os & performance of individual processes.

Disk Scheduling

The speed of processors has increased substantially, but the speed of disks has not.
This gap hinders performance.

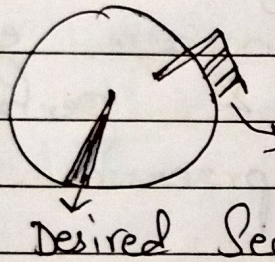
Thus, for efficient disk access, many algorithms are created.

1) Disk Performance Parameters:

The disk is rotating at constant speed. To read or write, the head is positioned at the desired track & a desired sector is scanned.

a) Seek Time: The time it takes to position the head at the track is known as seek time.

b) Rotational delay: The time it takes for the desired sector to be reached.



c) Access Time = seek time + Rotational delay + Transfer time

d) Transfer time: Once the header reaches the start of the sector, the time it takes to perform the reading is transfer time.

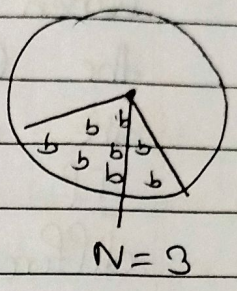
$$T = \frac{b}{rN}$$

T = Transfer Time

b = no. of bytes to be transferred

N = no. of bytes on a track

r = rotations speed, in revolution per second.



The disk scheduling Policies are better studied in lab.

Disk Management

1) Disk Formatting:

a) Low-Level formatting / Physical formatting:
Before a new disk can store data, it must be divided into sectors that the disk controller ~~and~~ ~~*~~ can read or write.

Each sector is given a data structure each. This data structure consists of a header, data area and a trailer.

The header contains information used by the ~~device~~ disk controller: Sector number and an error-correcting code (ECC).

b) ECC: When the disk controller writes

data on a disk, the ECC is calculated & stored.
When a read is later performed, the ECC is calculated again with the data present in the data area. If this new calculated ECC is different from the stored ECC, then an error has happened in the data area.

It contains enough ~~en~~ information to identify which bits have changed in the data area, thus it can be used to correct error ~~code~~. that was found. Given only few bits were corrupted.

The controller automatically does ECC processing whenever a sector is read or written.

c) Low level formatting is done at the factory as a manufacturing process.

The size of the data area is also decided at this stage.

d) Partitioning of disks The OS # partitions the disk into a few so that ~~it can store~~ one partition can store

a copy of the operating system's executable code, while another stores user files.

c) Logical formatting: The OS stores the initial file system data structures onto the disk. Eg: initial empty directory.

2) Boot Block

a) Bootstrap: The program that runs first when the system is booted. This program locates the kernel code from the disk and loads it onto the main memory.

This program is present in ROM, which ~~not~~ makes it secure from viruses & doesn't require initialization.

But this makes changing the bootloader version difficult. Thus, many systems use a tiny bootstrap loader program which brings the main bootstrap program from disk. Thus, for updating versions, only that part of the disk ~~can~~ need be updated.

b) Boot Block: This full bootstrap program is loaded in "boot blocks" at a fixed location on the disk.

The disk with boot partition is called boot disk or system disk.

3) Bad Blocks

Disks are prone to failure.
Sometimes the complete disk is failed, in which case it needs to be replaced & restored.
More frequently, sectors or blocks become defective - Bad Blocks.

a) One strategy to deal with bad blocks is to find them while the disk is being formatted & flag them as unusable.

b) If blocks go bad during execution, a special program like the badblocks command in Linux needs to be manually run to search & lock away the bad blocks.

b) On more sophisticated disks, a list of bad blocks is maintained. This list is initialized ~~it~~ in the factory during low level formatting & update through the disk's lifetime.

Some spare sectors are stored. When a block goes bad, the OS is told to replace the bad block with one of the spare sectors.
This scheme is called sector sparing or forwarding.

{{ Page 481, points 1, 2, 3, 4 }}

c) Sector Slipping: An alternative to sector sparing.

If ~~7th block~~ 17th block is defective and the spare block location is 202, then all blocks from 17 to 201 are shifted 1 block to the right.

17 → 18 (happens last)
20 → 21
100 → 101
201 → 201 (happens first)

- The operating system tries to read logical block 87.
- The controller calculates the ECC and finds that the sector is bad. It reports this finding to the operating system.
- The next time the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.
- After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.

replacing a bad sector while the system is running could potentially introduce errors or instability. By waiting until a reboot, the system can perform the replacement in a controlled and stable environment.

The dispatcher dispatches from the sorted queues, & with each dispatched request the ~~read or write~~ same request is removed from the read queue or write queue (wherever it was present).

If the expiration is reached for the header of any 1 of the 2 queues, that request is prioritized & dispatched and the corresponding entry in the sorted queue is also removed.

This scheduler overcomes the starvation problem & read vs write problem.

*All queues are FIFO.

d) Anticipatory I/O Scheduler

The philosophy behind this scheduler is that the process that issues a read request may issue another read request after the first request is ~~over~~ served & this request will be for a block close to the previous request block. Thus ~~there~~ the scheduler delays by the next request for 6ms, anticipating another request from the same process ~~in~~ a block closeby.

for

I/O Scheduler and Kernel	Test 1	Test 2
Linux elevator on 2.4	45 seconds	30 minutes, 28 seconds
Deadline I/O scheduler on 2.6	40 seconds	3 minutes, 30 seconds
Anticipatory I/O scheduler on 2.6	4.6 seconds	15 seconds

2) Linux Page Cache

Maintaining a page cache has 2 benefits:

a) When a collection of pages are to be written out to disk, the pages can be ordered so that the writing to disk can be efficient.

b) Pages in Page cache are likely to be referenced again in main memory. Thus it reducing disk I/O time.

Linux 2.4 uses a page cache that is involved in all traffic between disk and main memory.

Dirty (modified) pages are written to disk in 2 situations:

- When free memory falls below a certain threshold.
- When dirty pages grow older.

4.1

Overview

1) For the user the most important aspect of an operating system is its file system. The file manager permits users to create data collections with desirable properties like:

i) Long term existence
ii) Sharable between processes
iii) Structure

The file system maintains a set of attributes associated with each file: owner, creation time, time last modified, access privileges.

2) File Structure:

4 important terms while discussing files:

a) Field: A field is the smallest element of data in an entity.

Eg: An employee database contains name, ID, salary, age etc. A field would be the "name".

b) record: A collection of fields.
Eg: A database of employees.

c) file: Collection of similar records.
File is treated as a single entity by users and applications.

In some cases, a file only contain fields, not records. In those cases, a file is a collection of fields.

~~d) database:~~

3) Typical file operations:

- Retrieve_all: Retrieves all the ~~files~~ records in the ~~record~~ file.
- Retrieve_One: for retrieval of a single record from the file.
- Retrieve_next: for retrieving the record that comes next after the recently retrieved record.
- Retrieve_previous: Similar to retrieve_next, but for retrieving previous record instead.

- Insert_one: To insert a record in the file.
- Delete_one: To delete a record in the file.
- Update_one: To retrieve the record, & update the fields of the record, then putting the updated record back into its place.
- Retrieve_few: Retrieve a number of records.

4) File Management Systems:

A file management system provides services to the user/application in the use of files.

→ Objectives of the file management system:

- To meet the needs of the user by having the ability to perform all operations.
- To guarantee that the data in the file are valid.

- To optimize performance.
- To provide I/O support.
- To minimize the loss/corruption of data.
- To provide I/O support for multiple users, in case of a multi-user system.

→ User requirements:

- To be able to create, delete, read, write, modify files.
- Access control for different files
- To be able to restructure the files.
- To be able to move data between files
- To backup & recover files in case of damage.

5) File System Architecture:

- Level 1: Device Drivers
  To communicate directly with peripheral devices or their controllers.

The device driver is responsible for starting I/O operations. It is a part of the operating system.

- Level 2: Basic file System or Physical I/O

Deals with blocks of data that are exchanged with the disks.

Eg: Placement of data blocks on disk and buffering / loading of that data block from disk to main memory.

Considered part of the operating system.

- Level 3: Basic I/O supervisor

Deals with scheduling of disk access, file status. Considered part of the operating system.

- Level 4: Logical I/O

Enables User & applications to access file records, thus allowing access to data inside the data blocks from physical I/O.

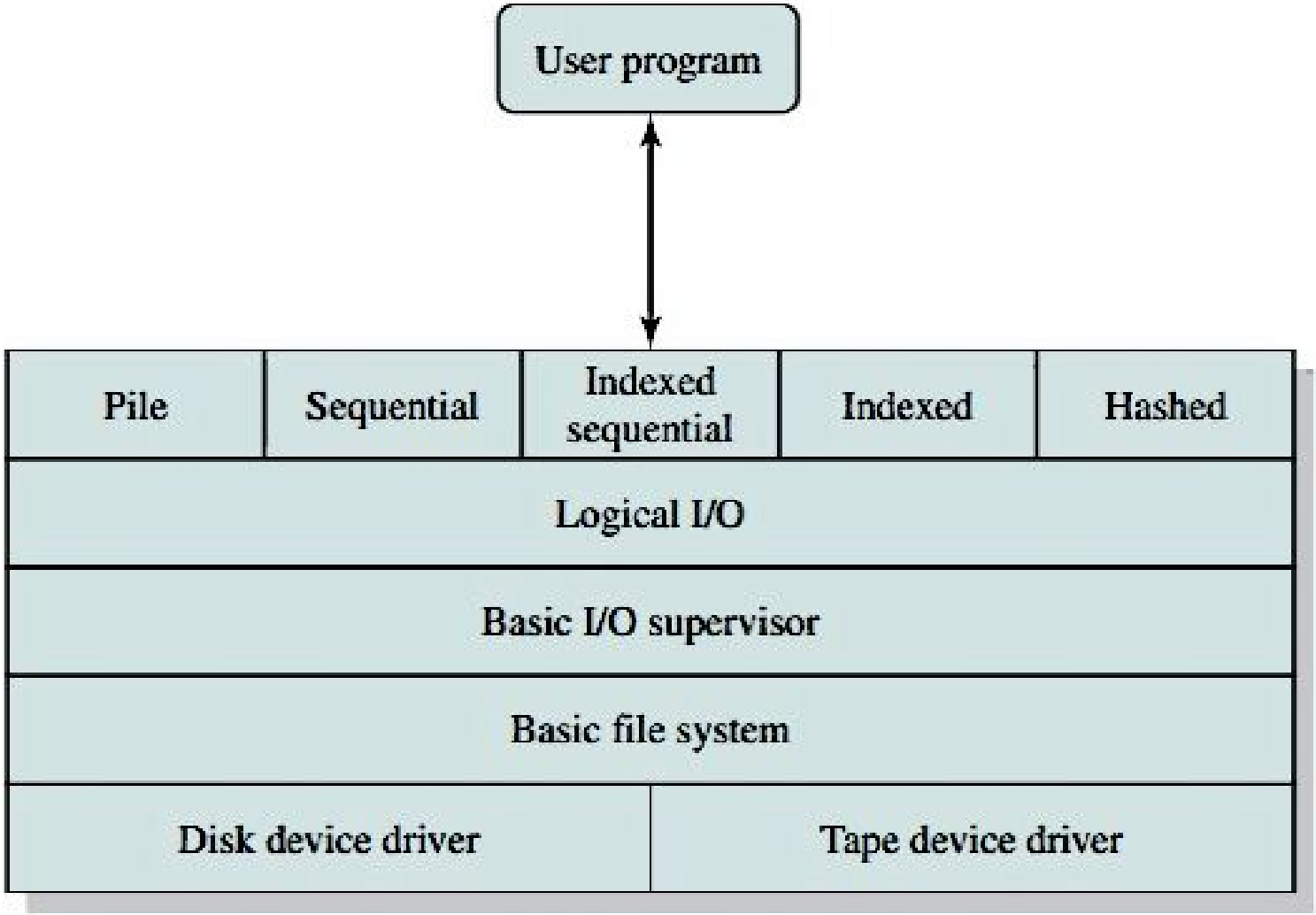


Figure 12.1 File System Software Architecture

- Level 5: Access Method

Provides an interface between applications & the file system.

6) File Management Functions:

- User Side: Users interact with the file systems using commands

→ Directory Management: Used for locating the file that the user is trying to access.

→ User Access Control: Most files have access specifications.

→ The user views the file system as a structure.

The user or application ~~sh~~ deal with records.

- System Side: The system deals with data blocks instead of records.

→ Blocking: Done to convert records to blocks of data.

→ Disk Scheduling: Each block must scheduled for optimized performance. (Same for file ~~scheduling~~ alloc)

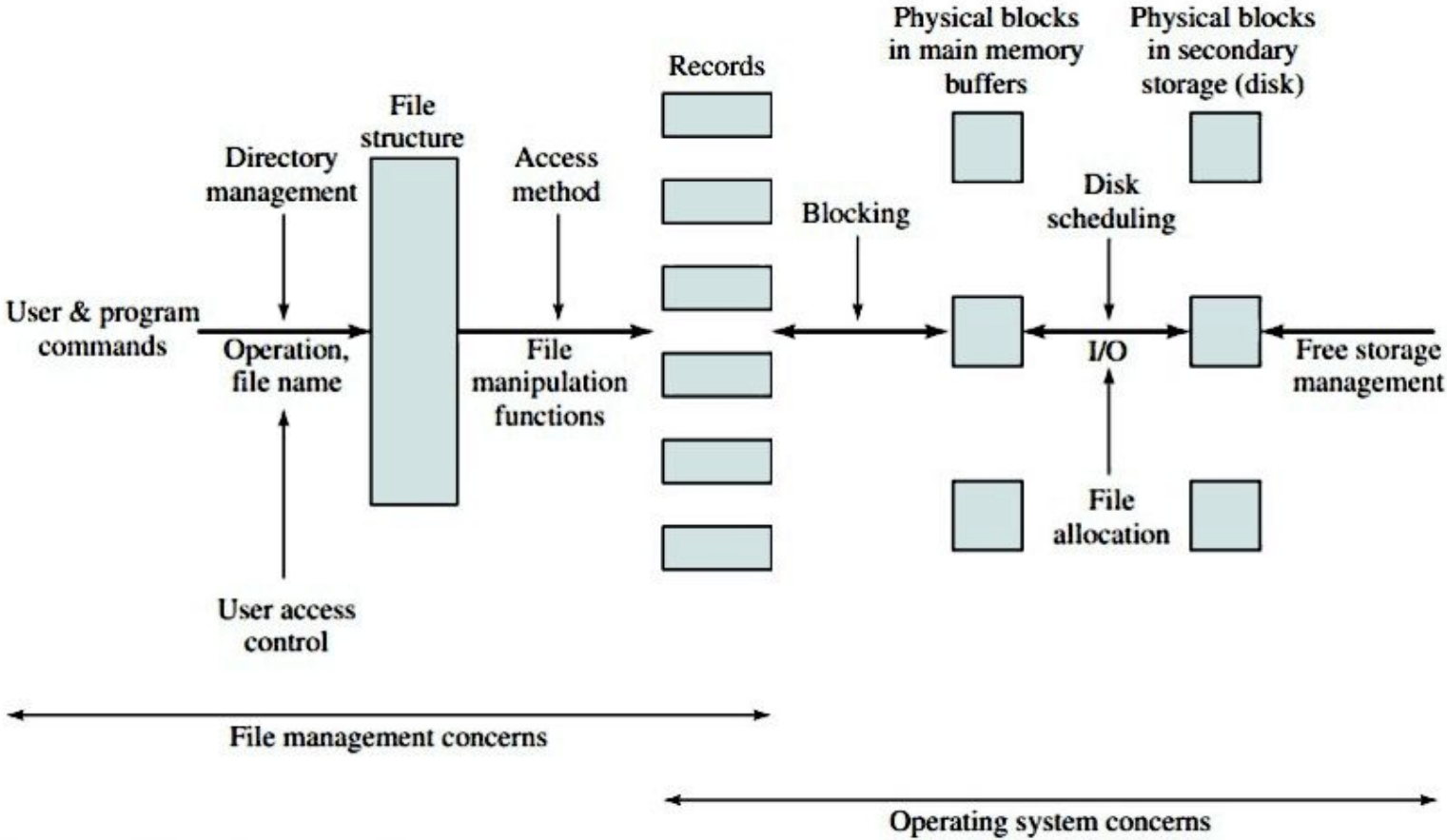


Figure 12.2 Elements of File Management

File Organization and ~~architecture~~ access

1) File organization refers to the ~~way~~ in which logical structuring of records. (how users access it)

In choosing a file organization, following ~~ch~~ criteria are important:

- Short Access time
- Ease of Updatt
- Economy of Storage
- Simple Maitenance
- Reliability

There are 5 major file organization:

a) The Pile
b) The sequential file
c) The indexed sequential file
d) The indexed file
e) The direct, or hashed, file

a) The Pile

Simplest ~~form~~ form of file organization. Data are collected in the order in which they arrive.

Each record thus contains random fields, thus each field should be self describing.

Searching of a field require exhaustive searching. Meaning, to find one field, every single record needs to be examined.

Each field in the record should have a name, value and length.

The File structure supports variable length fields.

a

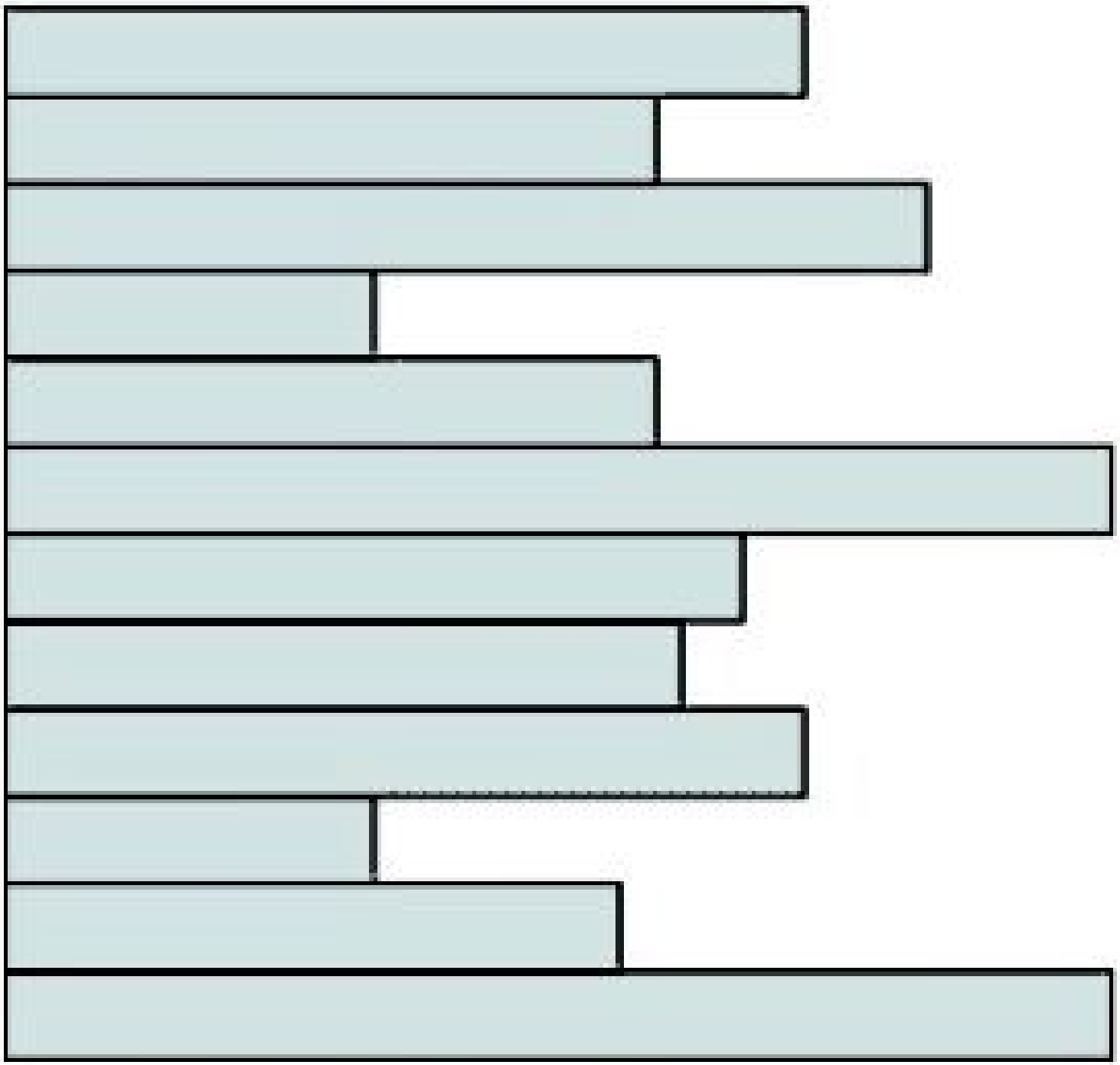
{{{figure 12.3 (B)}}}

b) The sequential file

All records in the sequential file structure are of the same length, containing fixed size fields.

The first field in the record is known as the key field, which uniquely identifies the record.

Records are stored alphabetically if text key is used. And stored



Variable-length records
Variable set of fields
Chronological order

(a) Pile file

numerical ordered if numerical key is used.

Thus, the field only needs to have the value defined. The field name & length are described by the file structure.

I assume that the name of the file would be given according to the key value.

Retrieval of record is difficult because the record still needs to be searched sequentially. When a key value matches with the desired key value, the record is found.

Adding a record also requires ordering it according to the key value.

(Even though the records are ordered sequentially, the search can't jump directly to desired recording using the key value as offset because there could be missing key values in between).

Since updating (adding) records can be costly,

this can be done in batch.
The updates are stored in a log file instead of the original file, and the two are merged after a batch of updates are accumulated.

Thus, the update is done in batch using the log file.

c) The indexed sequential file

The records are still organized on the basis of the key value but two features are added.

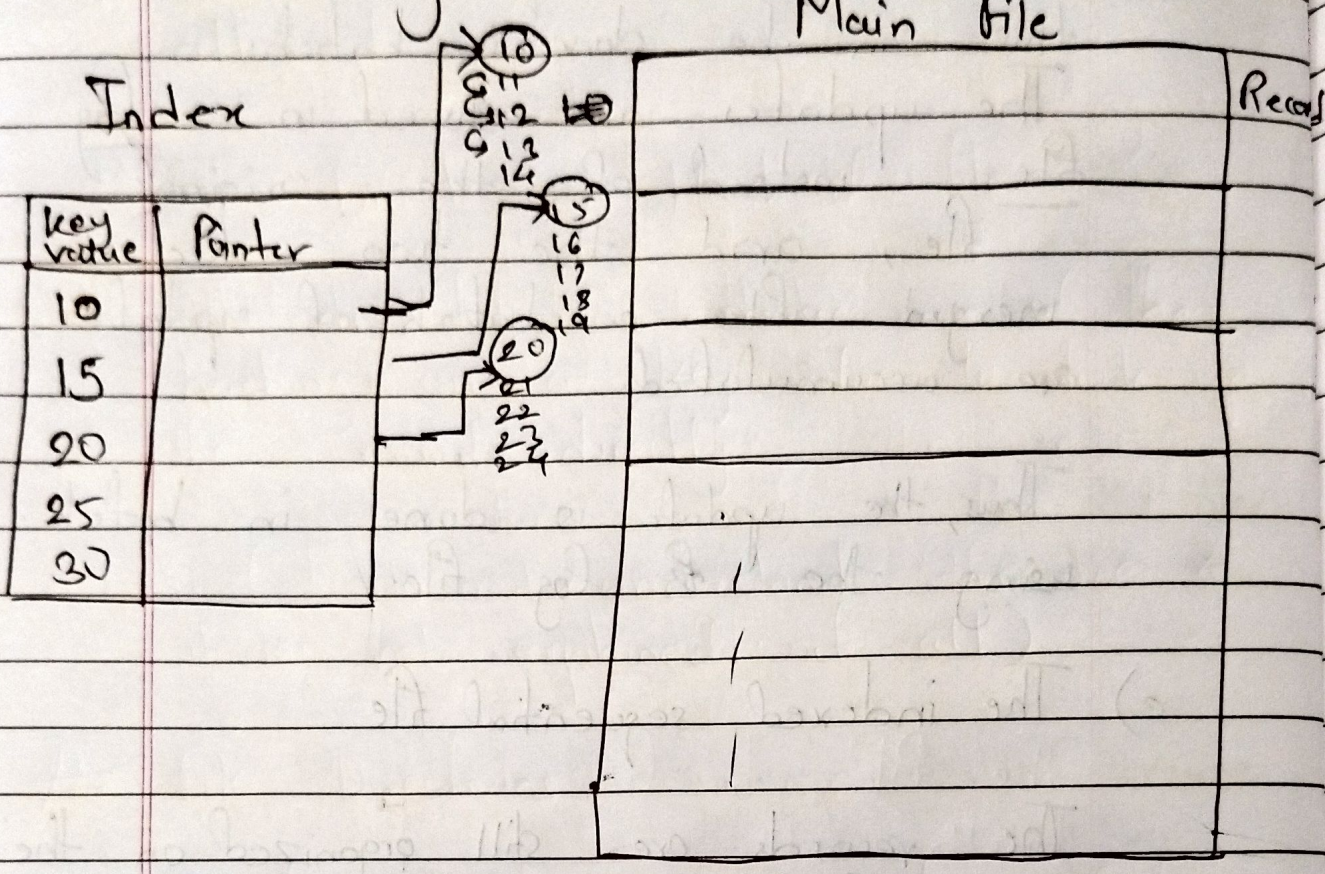
- Index: To improve lookup capability.

The index contains entries that help find a block of the main file that may contain the desired field. It is much like the index in a textbook, which tells us the block of pages in which we will find our desired page.

The entries contain $\frac{b}{2}$ → A key field, which is same as a key value.

→ A pointer to the record with

that key value.

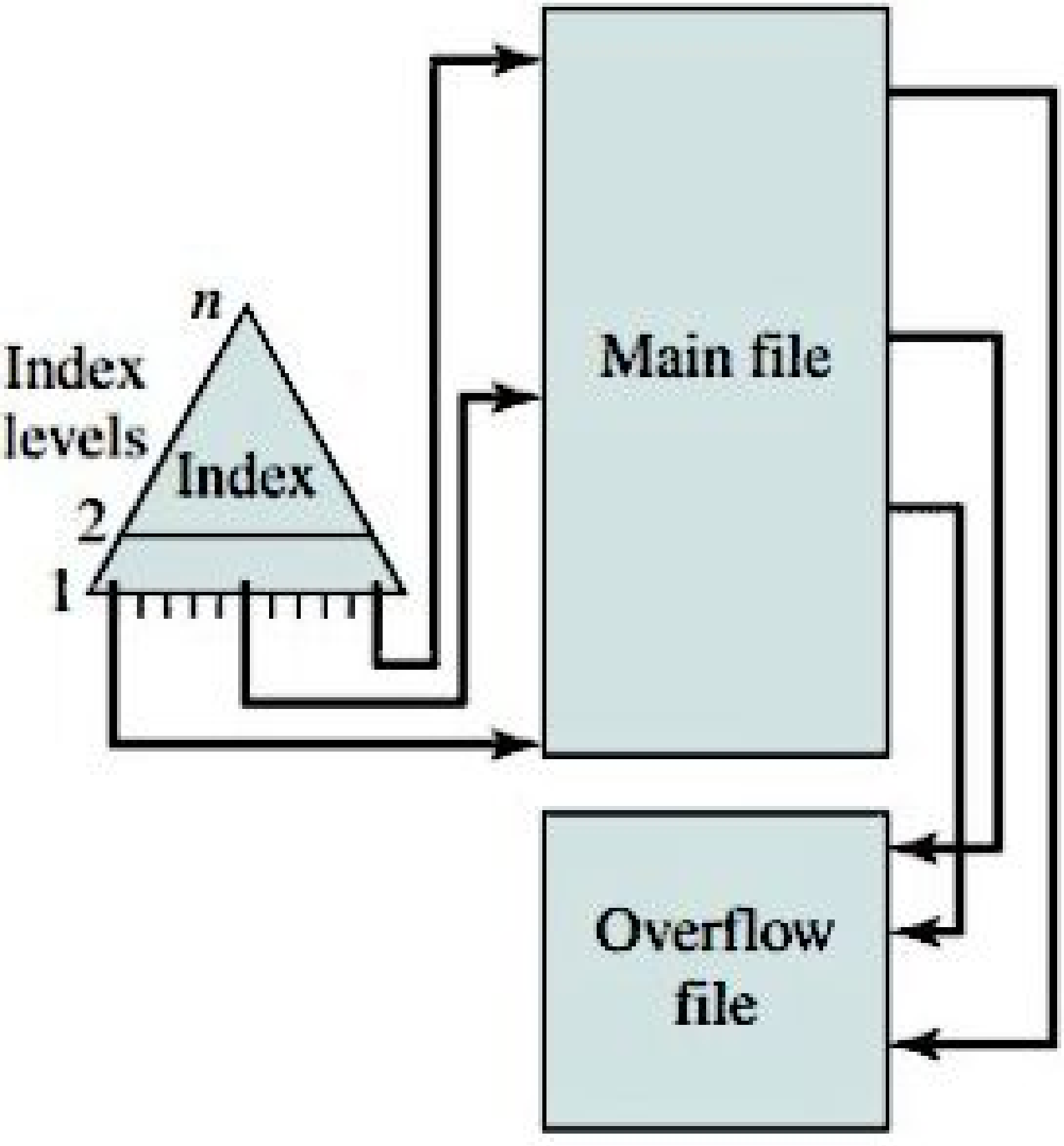


The lookup is done by finding the key value that is $\leq$ the desired key value in the index.

Then going # to the block pointed at by the index and traversing it sequentially till the desired key value is found.

- Overflow file: Same as the log file but the record in the overflow file is pointed at by the previous record.

When a new record is to be added, it is added in the overflow



(c) Indexed sequential file

file and the record that precedes it updates a field (invisible to the application, every record has this field) that points to the next record.
(like a linked list)

Multiple level of indexing can also be used for improved efficiency.

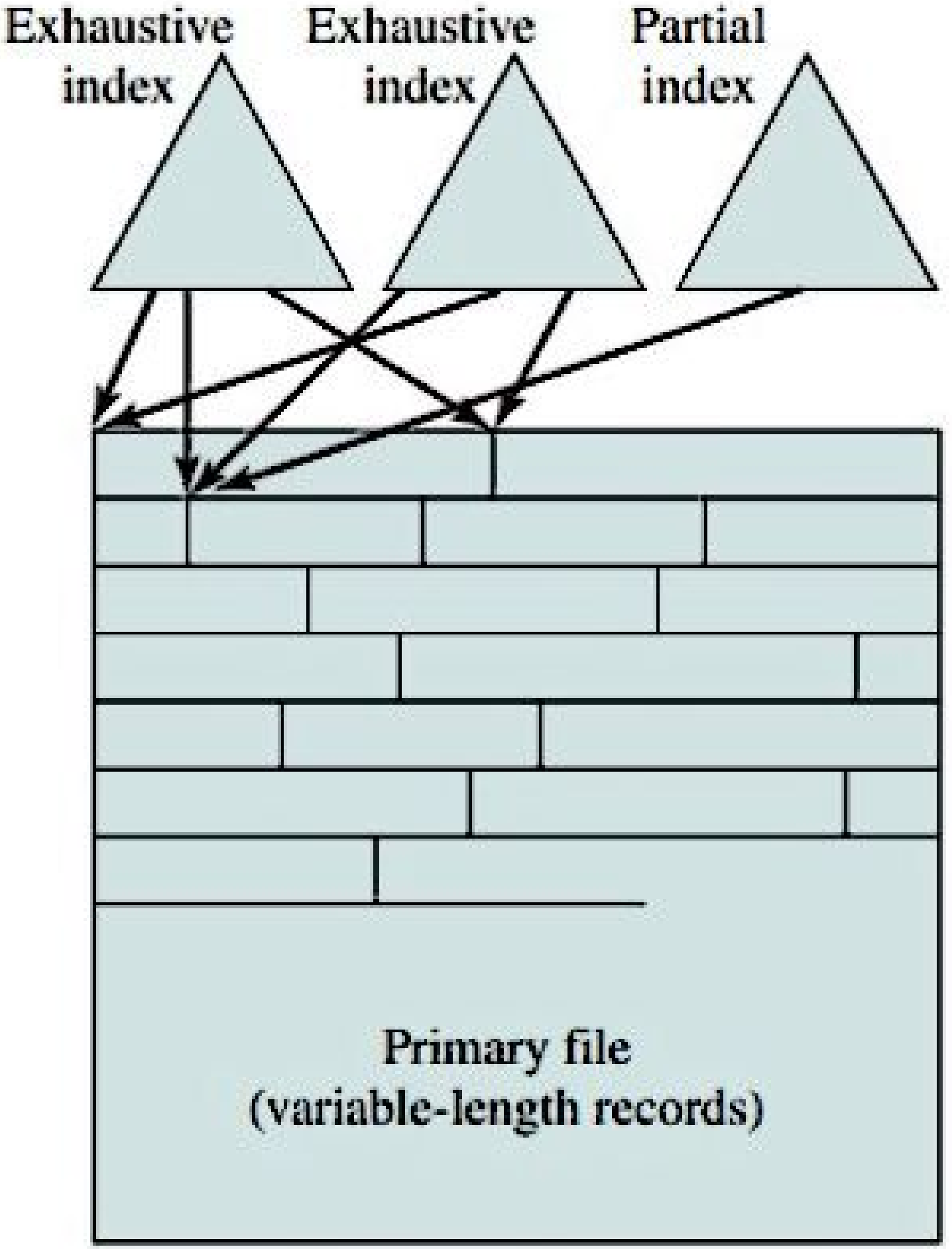
d) The Indexed File

Both the sequential structures lack efficiency for lookup.
In the indexed file, the concepts sequentiality & key value are discarded.

Thus, in indexed files, the index contains direct pointer to the records in the field.

- Exhaustive Indexing: Each record has an index entry that points to it.

- Partial Indexing: The lookup is based on the "field of interest".
The records containing information about a particular field are pointed.



(d) Indexed file

Variable length records & fields are allowed.

c) The Direct or Hashed file

The direct files make use of the hashing on a key value to look up the record directly.

(I do not understand this one & it is not explained at all).

File Directories

A file directory is a collection of files. The directory contains information about the files, ~~at~~ its attributes, location & ownership.

The directory itself is a file managed by file management routines.

The user views the file directories as a mapping between different files.

1) Structure

The file directory needs to store a lot of data about the files it contains, like the file name, file type,

file organization, volume, size used, owner, etc.

To reduce the size of the directory, it can store common data in a header.

→ Requirements of a file structure:

- Search

→ Operations on a file directory:

- Search: directory has to be searched for desired file
- Create file: an entry must be added in the directory.
- Delete file: Entry must be removed
- List Directory: When all or some of the files in the directory are to be requested (eg: display all files that a user can access)
- Upgrade Directory: When file update one of the fields that is stored in the directory, the directory must update.

a) Simple form: Create 1 entry for each file in the directory.
Unsuitable for performing all required operations.

b) Master directory: Another way is to have user directories & a master directory. The master directory points to the different user directories for enforcing access control.

2 level scheme.

c) Hierachical / Tree Structure: There are directories inside directories, and can contain any number of directories.
Most widely used (our PCs have).

{{ Figure 12.6 }}

2) Naming

Each directory should contain unique names for each file.
Pathname: The path take down the tree to reach a particular file.
eg: ~~local~~ desktop\subjects\OS

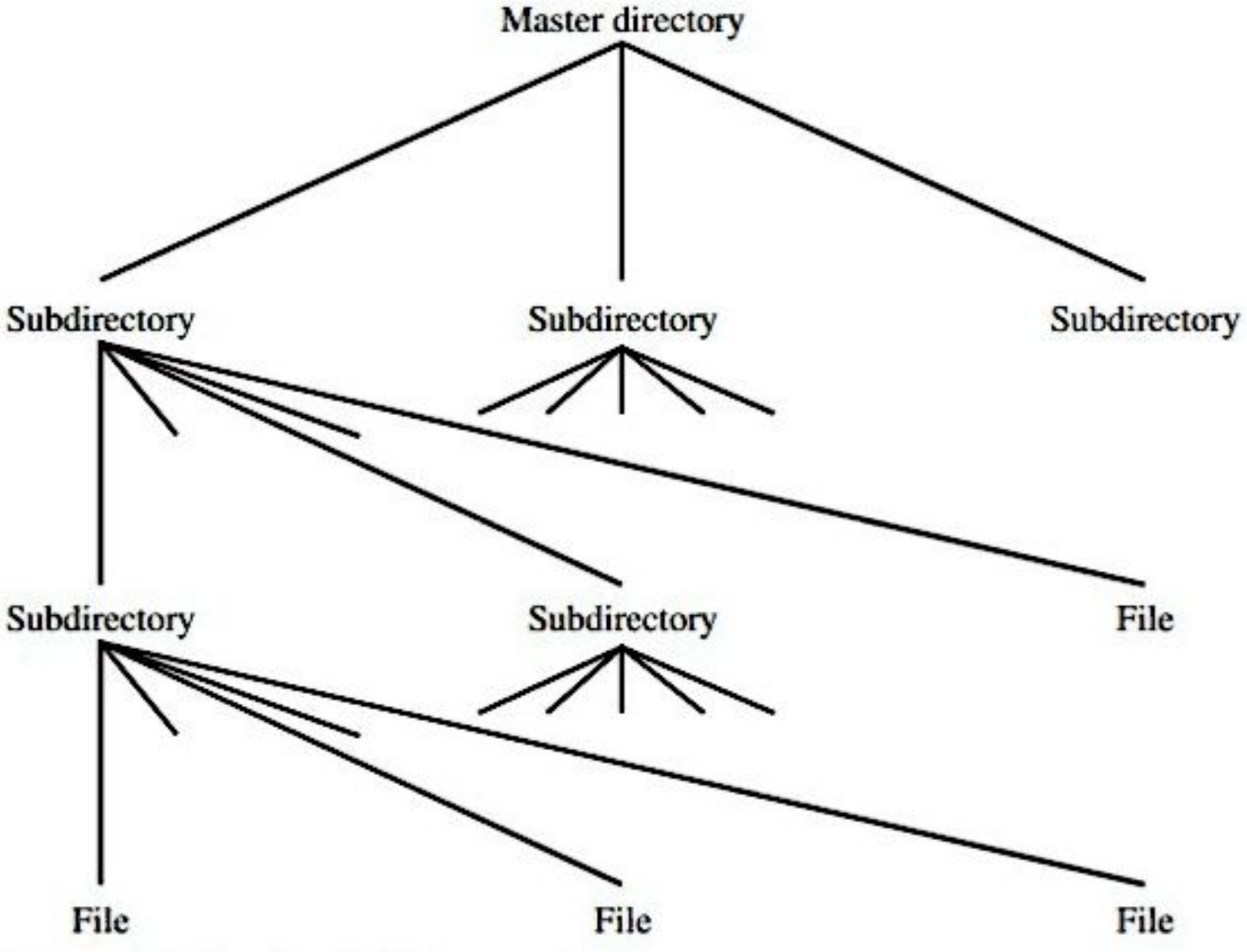


Figure 12.6 Tree-Structured Directory

Two files can have the same name but not if when the pathname for the 2 are same.

File Sharing

1) Access Rights

In a shared system, the ability to give different access rights to different users is important.

Some of the types of access rights are:

- None: The user won't know about the existence of this file. To enforce this restriction, the directory containing the file won't be accessible.

- Knowledge: The user ~~knows who the~~ can see the file & its owner, & can request access from owner.

- Execution: The user can load & execute a program but can't copy it.

- Reading: The user can read the file for any purpose.
- Appending: The user can append data to the file, but cannot delete or modify any content.
- Updating: The user can modify, delete or add to the file's data.
- Changing Protection: The user can change the access rights granted to other users. Often only the owner.
- Deletion: The owner can delete the file.

Each of the above rights are implying the rights that precede it.

The owner can provide access to different class of users:

- Specific User: Users designated by user ID.
- User Groups: Set of users that are not individually defined.

- All : All users who have access to this system.

2) Simultaneous Access:

When more than 1 users are simultaneous updating a file, the OS must enforce rules to avoid loss of data (readers-writers problem).

→ One way is to fully lock the file when one user is updating it (brute force).

→ Another way is to lock individual records of that file that are being updated.

Secondary Storage Management

As seen before, the file contains a collection of blocks in the secondary storage.
The OS or file management system is responsible for allocating blocks to the files.
Two management issues rise:
a) space in secondary management must be allocated to files.

b) necessary to keep track of the unused blocks in secondary storage.

1) file Allocation

The several issues to be addressed for file allocation are:

a) When a new file is created, is the maximum amount of space required by a file, allocated at once?

b) ~~Space~~ Blocks allocated to a single file contiguously are together called one portion.
What is the ideal size of the portion?

c) What data structure can be used to keep track of the portions used by a file?

2) Preallocation

The preallocation policy requires that the maximum size of the file be allocated to it at the time of file creation. Can be used for program compilations. It is possible to estimate the max size.

3) Dynamic Allocation:

Dynamic allocation of space to a file. Used when accurate estimation of file size max cannot be determined.

4) Portion Size:

The portion size has to be chosen considering the following factors:

a) Contiguity of space increases performance, especially for retrieve_next operations.

b) Having large number of small portions increases the size of tables needed to manage allocation information.

c) Having fixed size portions simplifies reallocation of space.

d) Having variable sized or small fixed size portions minimizes waste of unused storage due to overallocation.

There are 2 major alternatives:

a) Variable, large contiguous portions: Provides

better performance

b) Blocks : Provides better flexibility

In variable - size portions, fragmentation is a problem.
It is similar to the problem discussed in module 5.2.

Strategies to avoid fragmentation in variable-size portions:

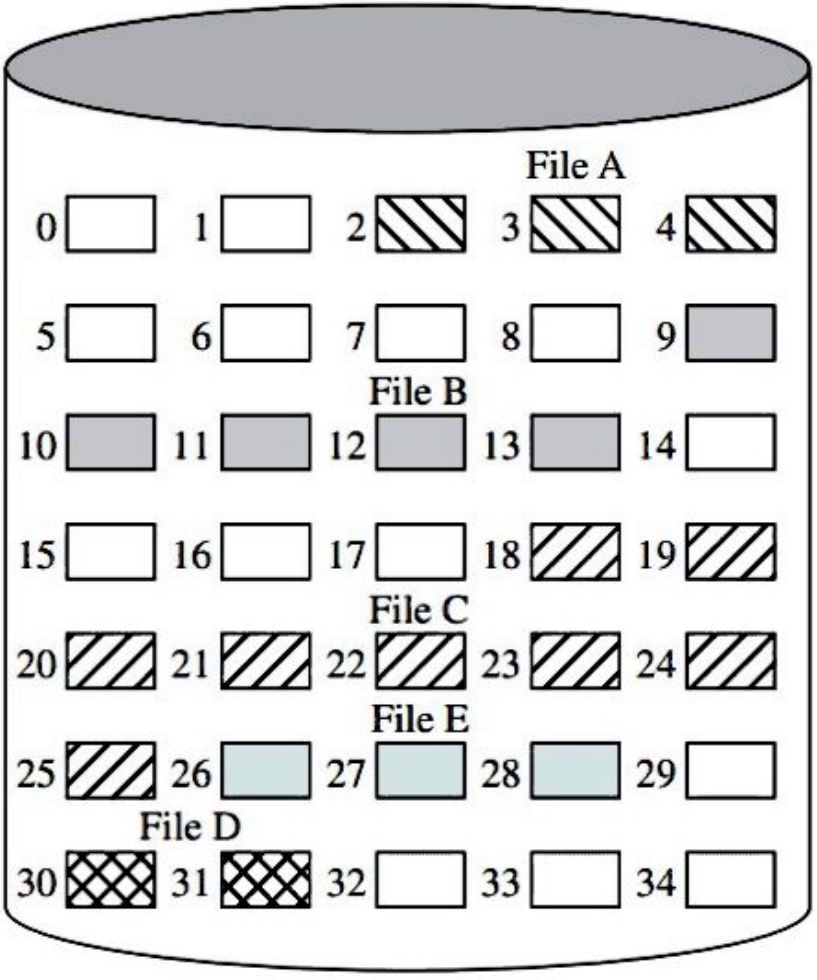
- **First fit:** Choose the first unused contiguous group of blocks of sufficient size from a free block list.
- **Best fit:** Choose the smallest unused group that is of sufficient size.
- **Nearest fit:** Choose the unused group of sufficient size that is closest to the previous allocation for the file to increase locality.

5) File Allocation Methods:

a) Contiguous Allocation:

A contiguous set of blocks is allocated to a file at the time of creation. This is thus, preallocation strategy.

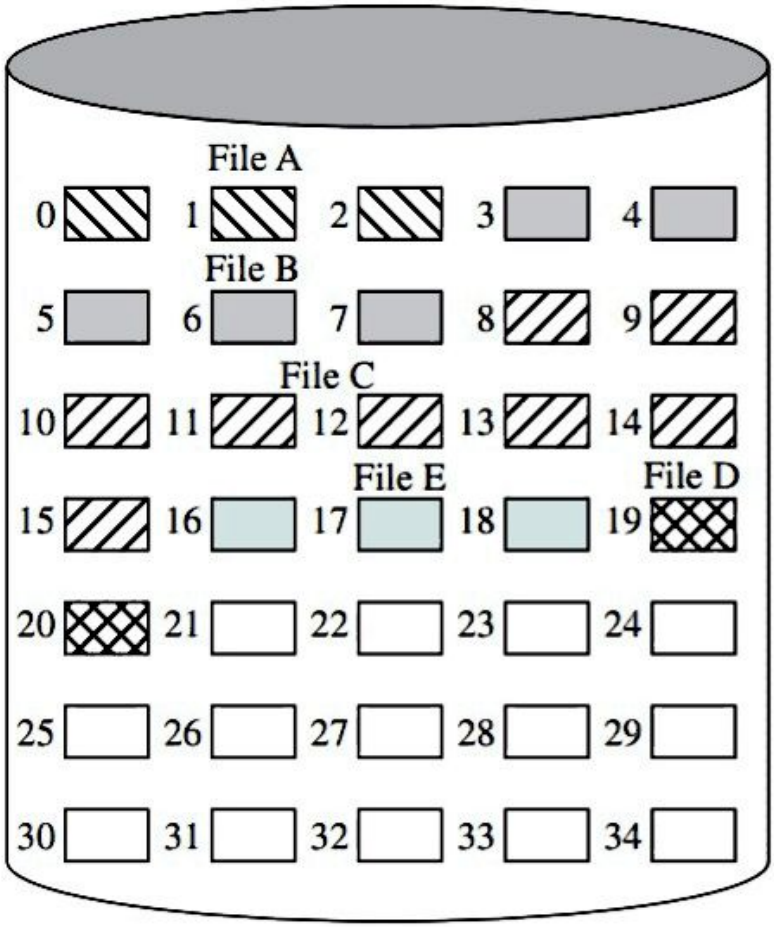
The retrieval of data is fast & the size of the allocation table is small, allowing as the starting block & length is enough information for telling which blocks belong to a file



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 12.9 Contiguous File Allocation



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 12.10 **Contiguous File Allocation (After Compaction)**

The size of the file needs to be declared first.

Can cause external fragmentation, thus compaction algorithm need to be run time to time.

b) Chained Allocation

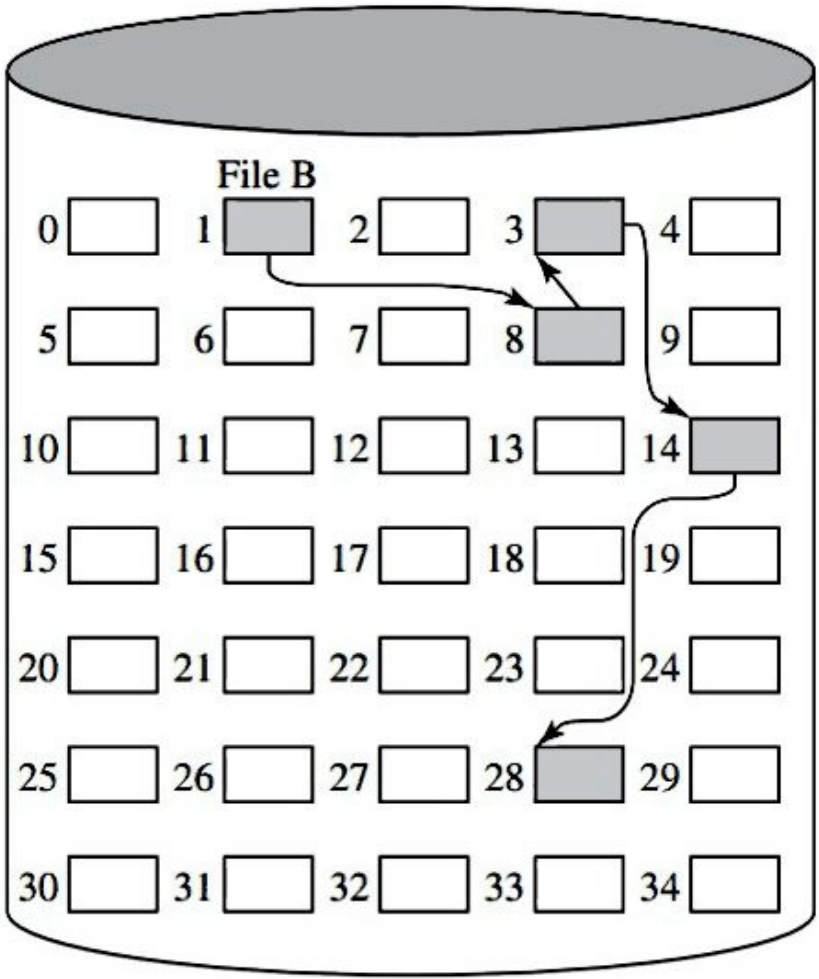
Allocation is on a ~~sim~~ single block basis. The file allocation table still needs only a ~~sim~~ single entry for each file, the starting block & the length of the file.

Preallocation is possible, but more commonly used with dynamic allocation.

Uses linked list like structure. Any free block can be ~~allocated~~ added to the chain as needed.

No external fragmentation.

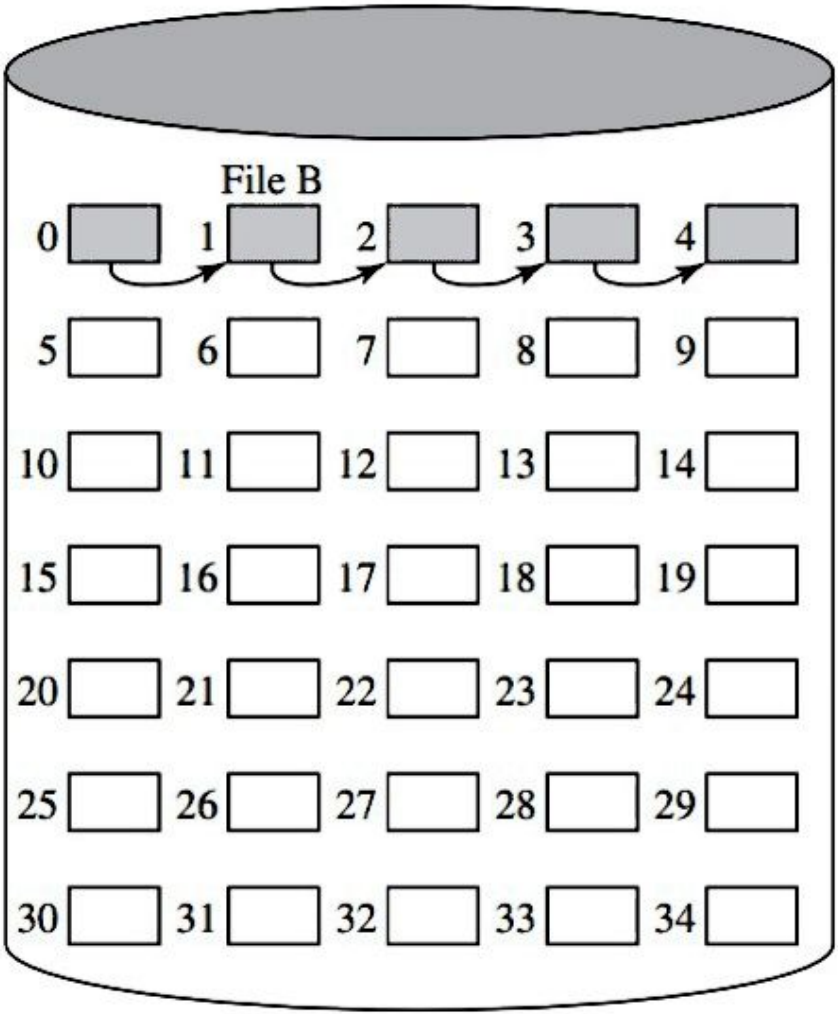
However, if several blocks of a file are to be brought at the same time, Then a series of accesses to the different parts of a disk are required. To overcome this, systems periodically consolidate files.



File allocation table

File name	Start block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Figure 12.11 Chained Allocation



File allocation table

File name	Start block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Figure 12.12 Chained Allocation (After Consolidation)

c) Indexed Allocation

Addresses many problems of the contiguous & chained allocation.

The file allocation table for each file points to a block in the disk.

This block contains an index table that points to each block of the particular file.

Can be used with both fixed-size block & variable size portions.

{{{ figure 12.13 & 12.14 }}}

* FAT → File Allocation Table

6) Free Space Management

The free space in the disk also need to be managed. This is done using the disk allocation table (DAT).

Here are some methods to implement DAT:

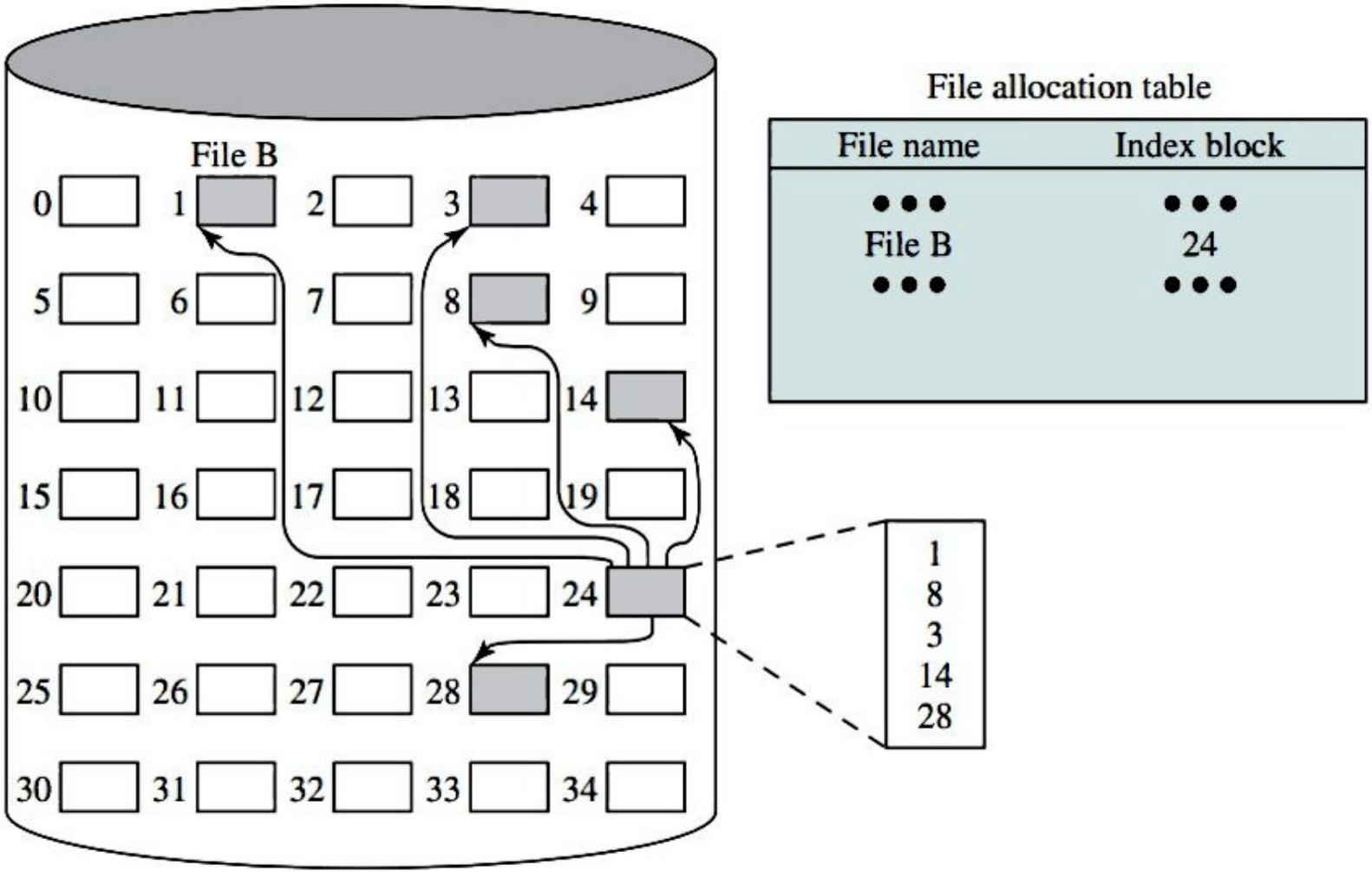


Figure 12.13 Indexed Allocation with Block Portions

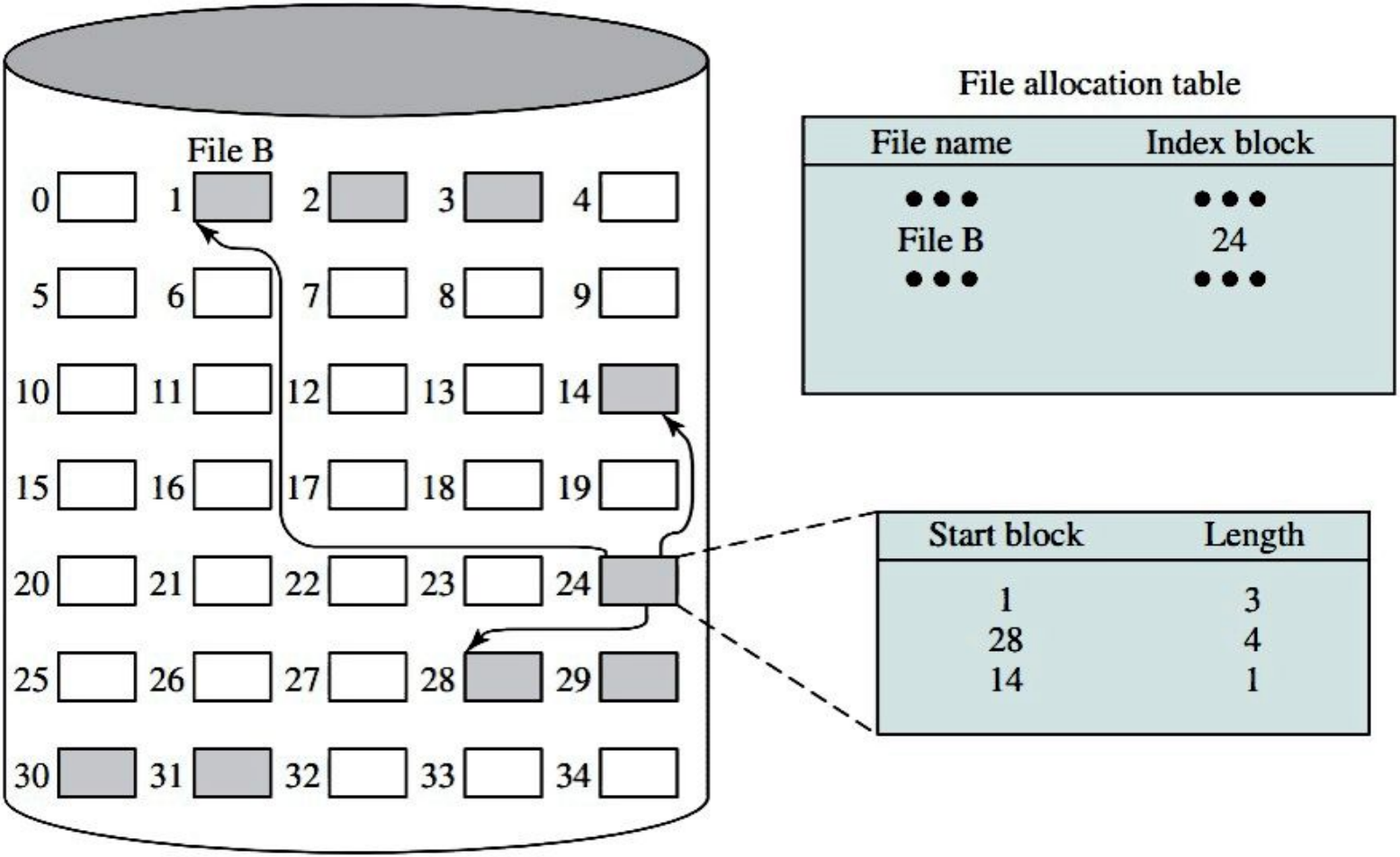


Figure 12.14 **Indexed Allocation with Variable-Length Portions**

a) Bit Tables

This method contains a vector containing 1 bit for each block.

If that block is used, the bit value is 1, if not used, the bit value is zero.

Eg: 0011011110001010011111

b) This method is small but can still be relatively large in ~~system~~ systems that have bigger disks (stop laughing).

This can be make it difficult for searching for free blocks.

b) Chained Free Portions:

In this method, each free block points (linked to) the next free block.

This method does not require DAT.

Can slow down file creation as the free blocks' pointers need to be arranged. Similarly for deleting files.

c) Indexing:

Similarly to the indexing used for blocks allocated to files. But only supports variable size portions rather than blocks.

d) Free Block List:

Each free block is assigned a number & these numbers are stored in the DAT.
These numbers are 24-32 bits long.

The size of DAT will be (24-32) × Size of the disk, thus DAT cannot be stored in main memory, only on disk.

Advantages:

- The space of this disk devoted to DAT is less than 1% of the total disk space.
- There are 2 ways to store a small part of the DAT in the main memory:

i) The first few 1000 blocks can be stored in main memory, in the form of stack. When a file is to be allocated, the first entry from that stack can be popped & the corresponding block can be used.

When block is ~~del.~~ deallocated, it may be pushed onto the stack.

~~ii) A few th 1000~~

ii) The blocks can be treated as a FIFO queue, & few ~~=~~ 1000 of the first blocks can be kept in main memory.

7) Volume : A collection of addressable locations on ~~disk~~ secondary memory that the OS can use for data storage.

These collections need not be contiguously placed in the disk, but all the free spaces can combine to form a volume that appears to be free contiguously.

In simple cases, 1 whole disk is a volume, but it reduces as the blocks are allocated.

8) Reliability:

The DAT & FAT are partially stored in main memory & the updates are done there, then these updates are written back to the disk.

If the system crashes before an update is written back to the disk, it leads to loss of data.

To prevent this, following actions can be taken:

a) Lock the disk allocation table on disk. This will slow down the allocation.

b) Write an update to the disk immediately after an update on DAT or FAT in the main memory.