Operating system

Part X: File System (Interface & Implementation)

By KONG LingBo（孔令波）

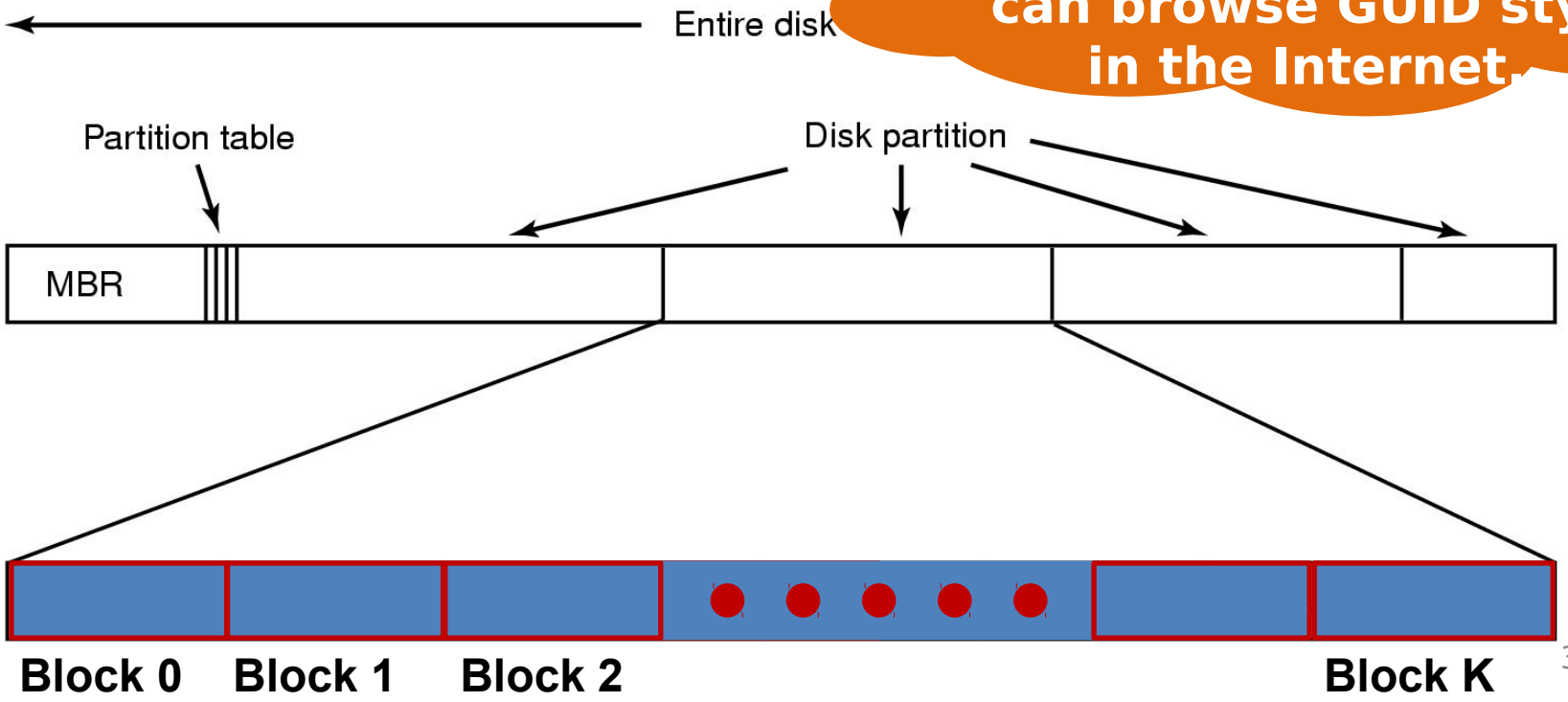
Goals

- Know the basic concepts related to file system
  - File, directory,
- Know the implementation techniques
  - File organization
  - Directory implementation
- Samples of File systems
  - NFS, NTFS …

We’ ve learned - Disk Space Organization

- Disk can be **PARTITION**ed
  - Each partition can have a different OS and/or different file system
    - One partition can be swap sp
- Each partition has

MBR style is the traditional way to partition the disk. You can browse GUID style in the Internet.



We’ ve known - linear addressed blo ck space for files

Processes

File System Calls

User’s (process’s) view of files

File 1

File 2

Manage files based on blocks? – Need data structures

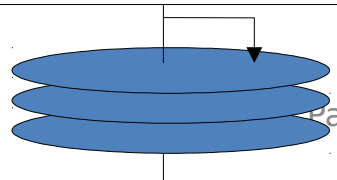
0	1	2	3	4	5

file system’s view of the disk

Disk Driver

Disk Controller

Disk

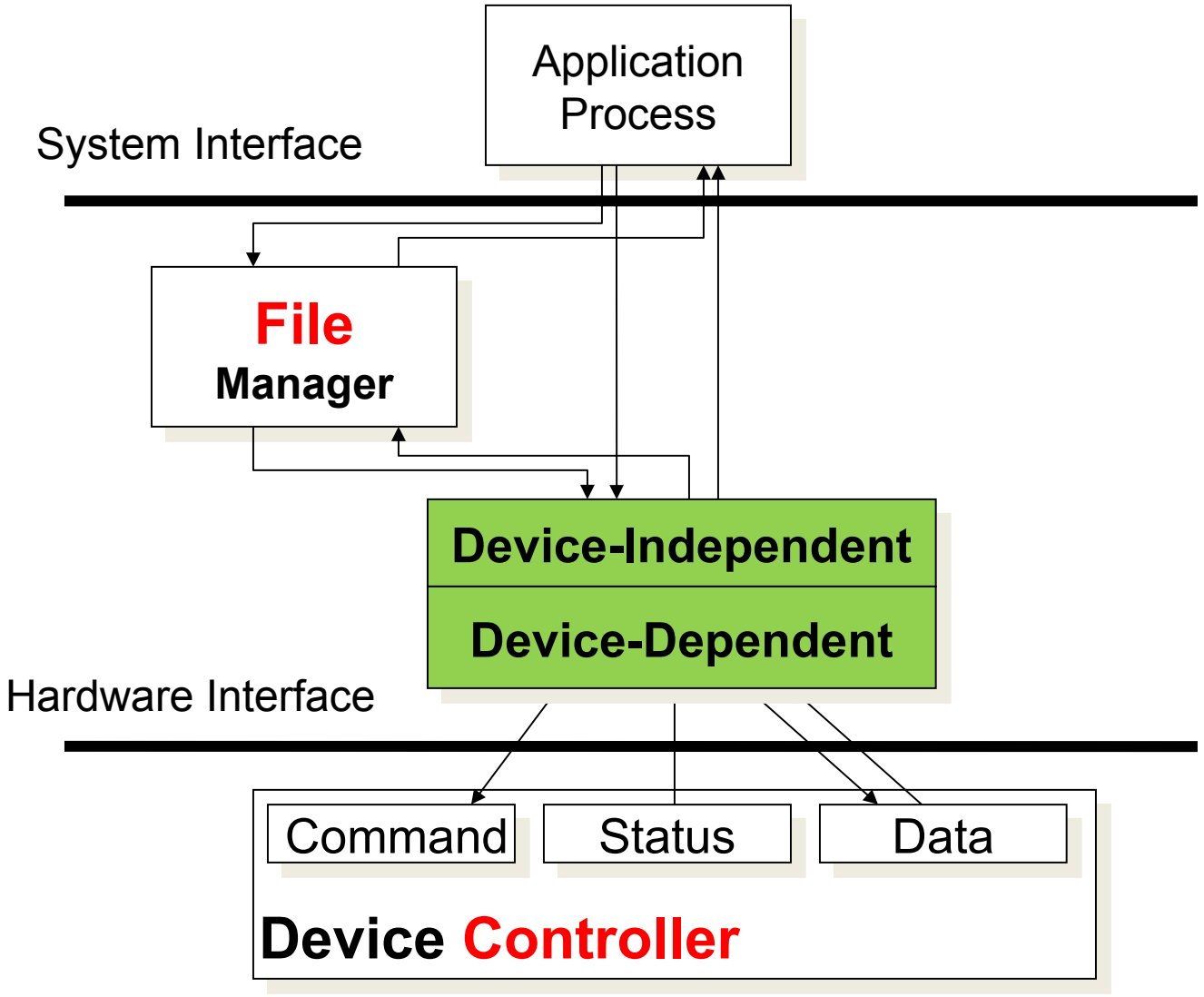


Cylinders, Header/tracks, Sectors

0	1	2	3	4	5
6	7	8	9	10	11

Sector space (like storage units in MM)

File & Device Management



Problems?

- Like Mapping 1, we need following information – data structures & Related Operations
  - Organize blocks into semantic regions
    - Boot block, superblock, directory
  - Free space
    - You need know where available blocks are.
  - Map a file (collection of bits) into blocks
    - Needed blocks, and how to indicate them?
  - How to organize so many files?
    - Directory

File System

- Mapping 2: File to HDisk (Linear addressed block Space)
  - Organize blocks into semantic regions
    - Boot block, superblock, directory
  - Free space
    - You need know where available blocks are.
  - Map a file (collection of bits) into blocks
    - Needed blocks, and how to indicate them?
  - How to organize so many files?
    - Directory

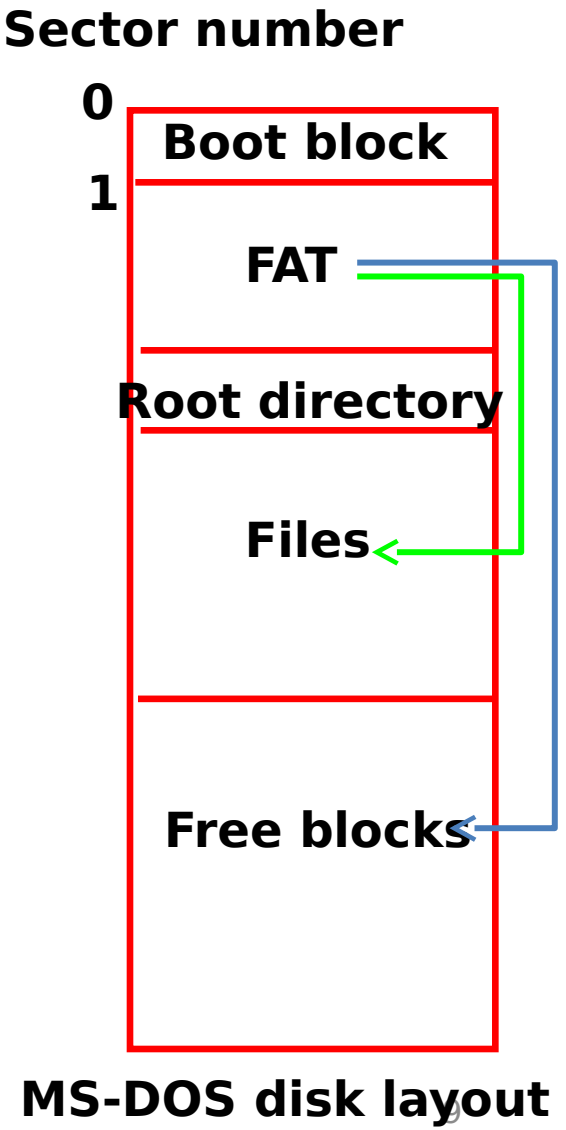
Boot block (or sector)

some old system uses only one sector

- Like the MBR for the hard disk, **each partition usually uses the 1st block for special usage**, called "**boot block**"
  - to record some critical information of how those blocks are used, such as regions for
    - Free space, files, directory, …
- If the partition contains OS, the boot block also records the information to locate the "bootstrap loader"

We have many partition layouts

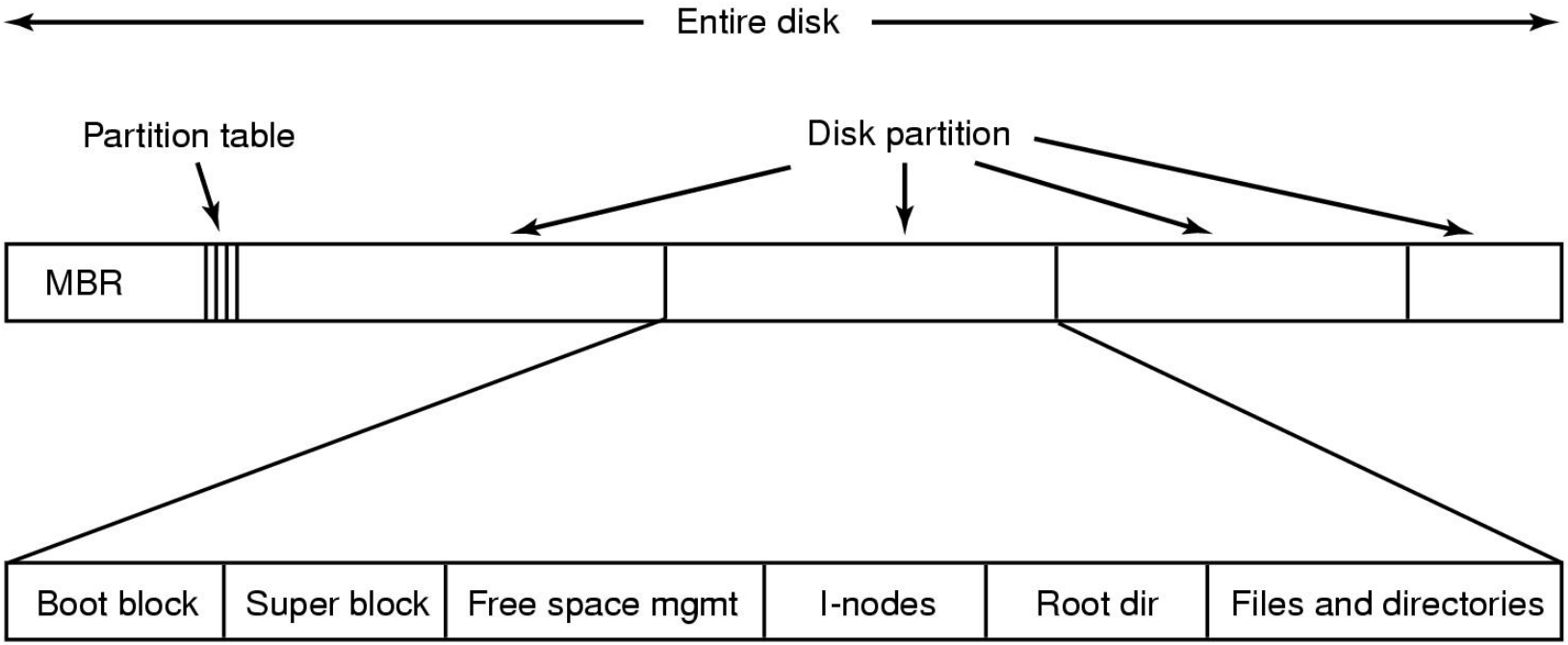
- MS-DOS layout
  - System disk contains boot block in first block of each partition
    - Boot block has bootstrap executable
  - Bootstrap loader copies bootstrap program into memory
    - Bootstrap program initializes all registers, finds OS kernel on disk, loads OS, and jumps to the initial address of OS



MS-DOS disk layout

Cont'

- UNIX like



A possible file system layout w*ith a **Unix** partition*

File System

- Mapping 2: File to HDisk (Linear addressed block Space)
  - Organize blocks into semantic regions
    - Boot block, superblock, directory
  - Free space
    - You need know where available blocks are.
  - Map a file (collection of bits) into blocks
    - Needed blocks, and how to indicate them?
  - How to organize so many files?
    - Directory

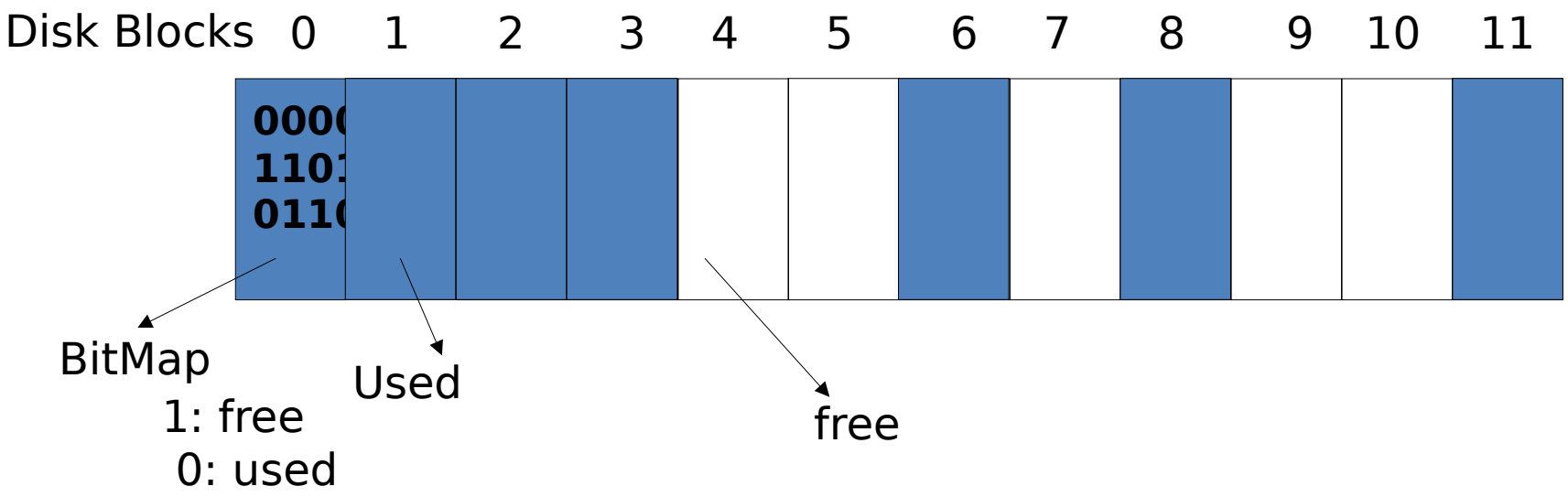
Free Space Management

- How can we keep track of free blocks of the disk?
  - Which blocks are free?
- We need this info when we want to allocate a ne w block to a file.
  - Allocate a block that is free.
- There are several methods to keep track of free b locks:
  - Bit vector (bitmap) method
  - Linked list method
  - Grouping
  - Counting

Free-Space Management:
Bit Vector (Bit map: 位图)

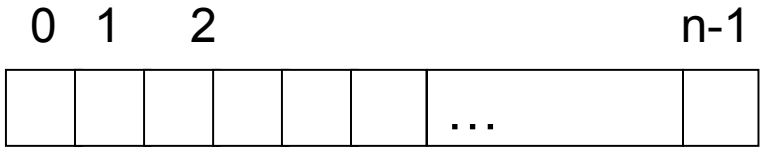
- We have a bit vector (bitmap) where we have one bit per block indicating if the block is used or free.
- If the block is free the corresponding bit can be 1 , else it can be 0 (or vice versa).

Example:



Cont'

- Bit vector (*n* blocks in disk)



bit[*i*] = 0 ⇒ block[*i*] used
1 ⇒ block[*i*] free
(or vice versa)

Finding a free block (i.e. its number)

Start searching from the beginning of the bitmap:
Search for the first 1

First Free Block Number =
(number of 0-value words) * (number of bits per word)
+ offset of first 1-valued-bit

0000000000000000
0000000000000000
0000000000000000
0000000010000000
0000000000000000
0000000000011000
0001100010000000
0000000011110000

3x16+8 = **56**

Simple check

- If the block size is 4KB, and the hard disk size (or a partition) is 500 GB, how many blocks are used to store the bitmap?
  - 1 GB = 2^{30} B
- You'll meet this kind of questions all the time
  - How many blocks are used to store the needed information?
  - Do you know in HDD, GB means 10^6 B?

File System

- Mapping 2: File to HDisk (Linear addressed block Space)
  - Organize blocks into semantic regions
    - Boot block, superblock, directory
  - Free space
    - You need know where available blocks are.
  - Map a file (collection of bits) into blocks
    - Needed blocks, and how to indicate them?
  - How to organize so many files?
    - Directory

File System Implementation: File Space Allocation

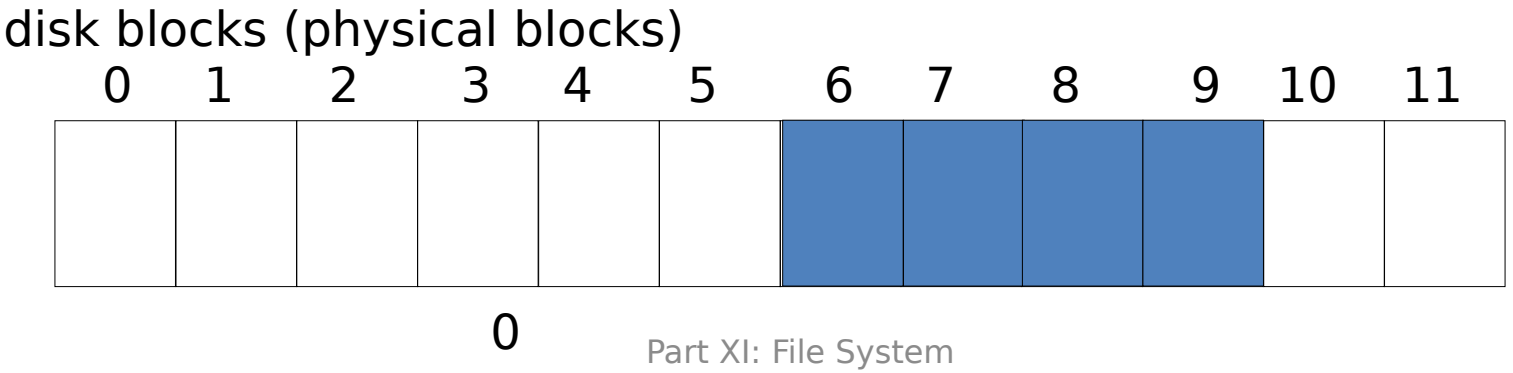
- Goals
  - Fast sequential access
  - Fast random access
  - Ability to dynamically grow
  - Minimum fragmentation
- Standard schemes
  - Contiguous allocation (fixed)
  - Linked list allocation
  - Linked list with file allocation table (FAT)
  - Linked list with Indexing (I-nodes)

Contiguous Allocation

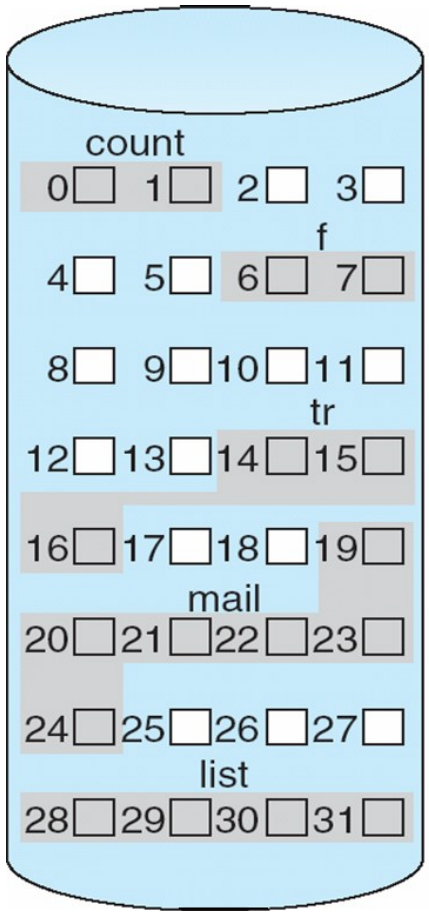
- Each file occupies a set of contiguous blocks on the dis k
- + Simple – only starting location (block #) and length (number of blocks) are required to find out the *disk da ta blocks of file*
- + Random access is fast
- - Wasteful of space (dynamic storage-allocation proble m)
- - Fi

file data

Start address = 6
Number of blocks = 4



Contiguous Allocation of Disk Space



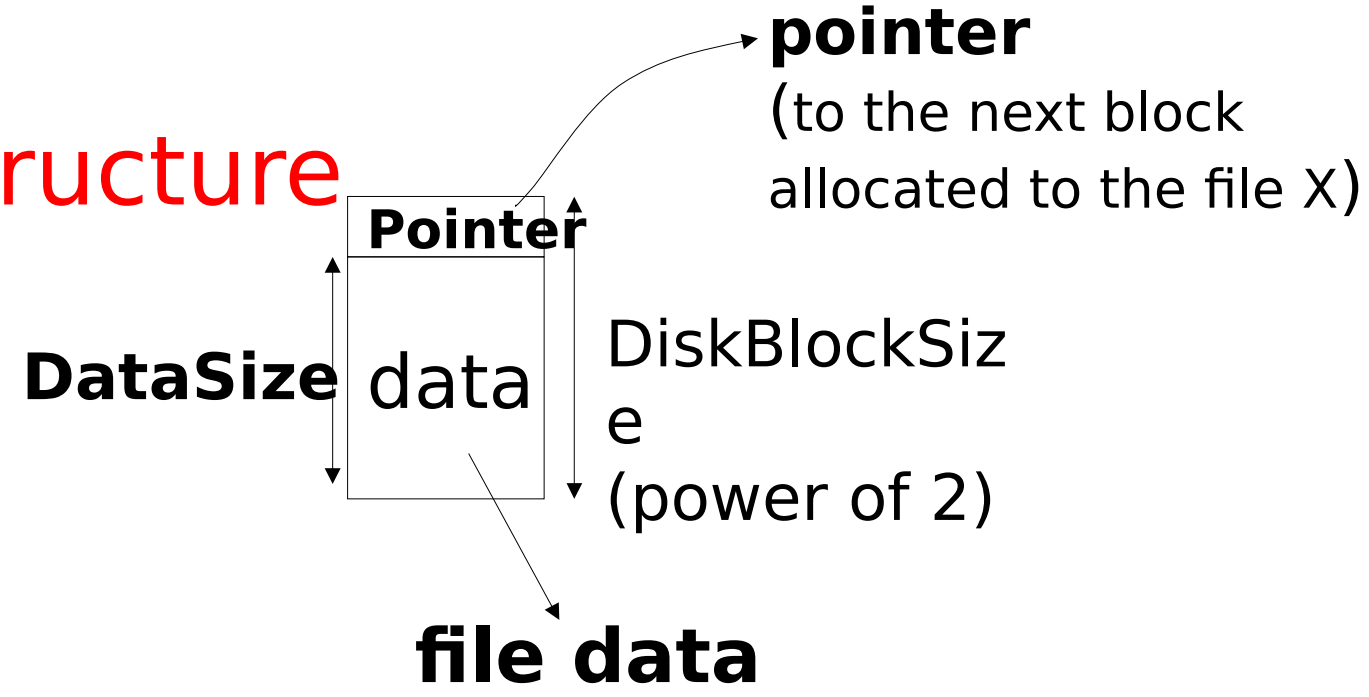
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked Allocation

- Each file is a linked list of disk blocks: bloc ks may be scattered anywhere on the disk .

block structure

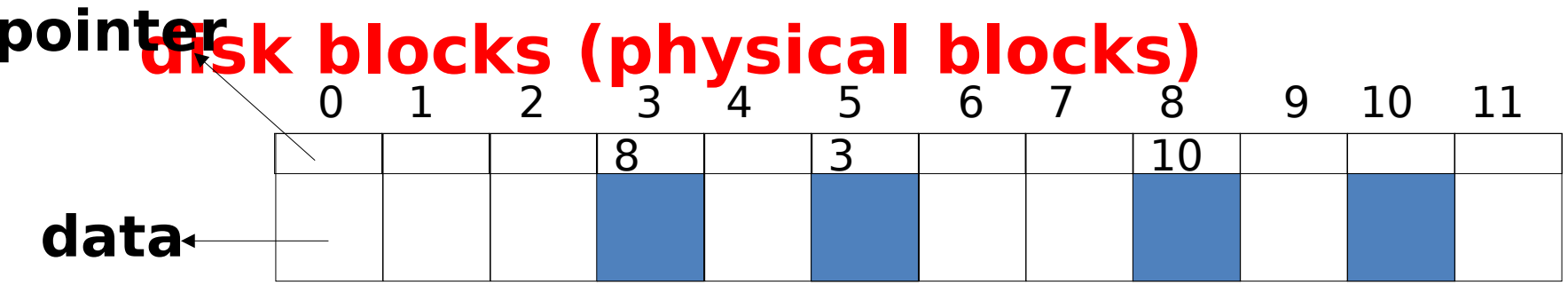


e data size in a disk block is no longer a powe

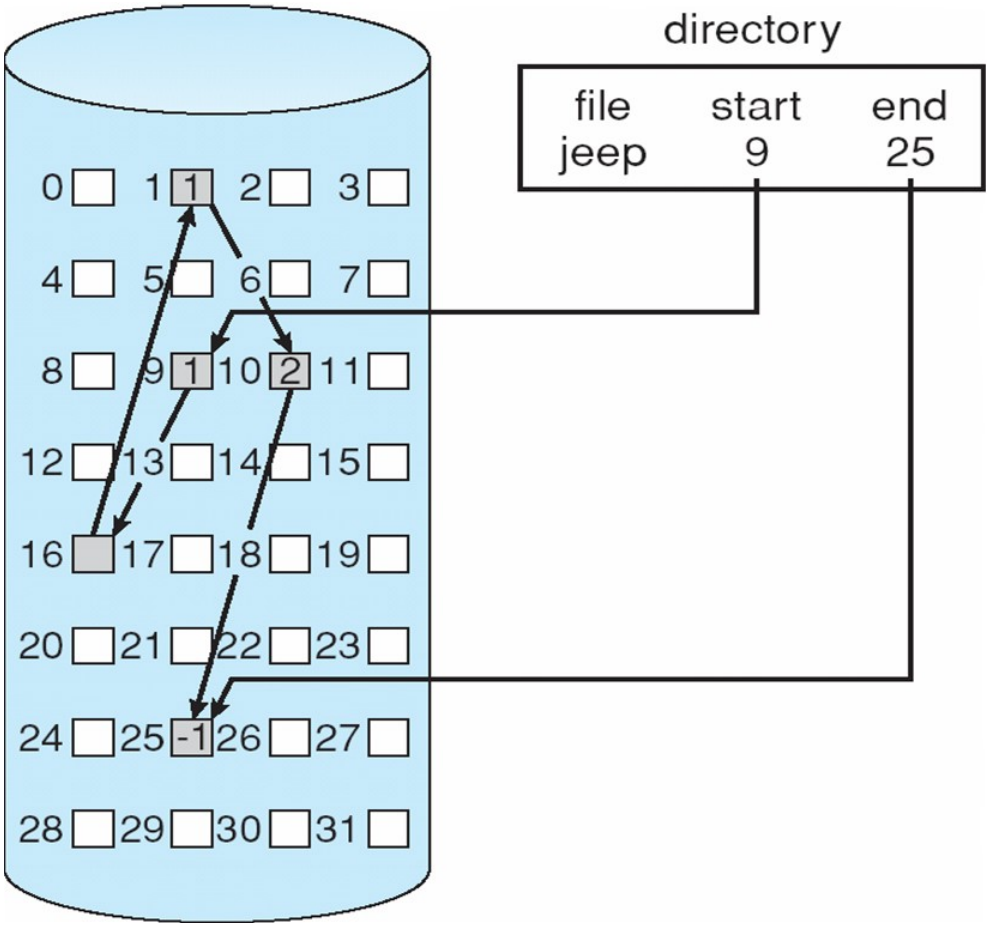
Linked Allocation (cont’)

File X

File starts at disk block 5



Linked Allocation



Linked Allocation (cont.)

- Slow - defies principle of l
  - Need to read through l
    lly to find the rec
- Not very reliable
  - System crashes can
    d.
- Important variation on linked allocation method
  - File-allocation table (FAT) - disk-space allocation used by MS-DOS and OS/2.

Just as those shortcomings of linked list you learned in DSA

File Allocation Table

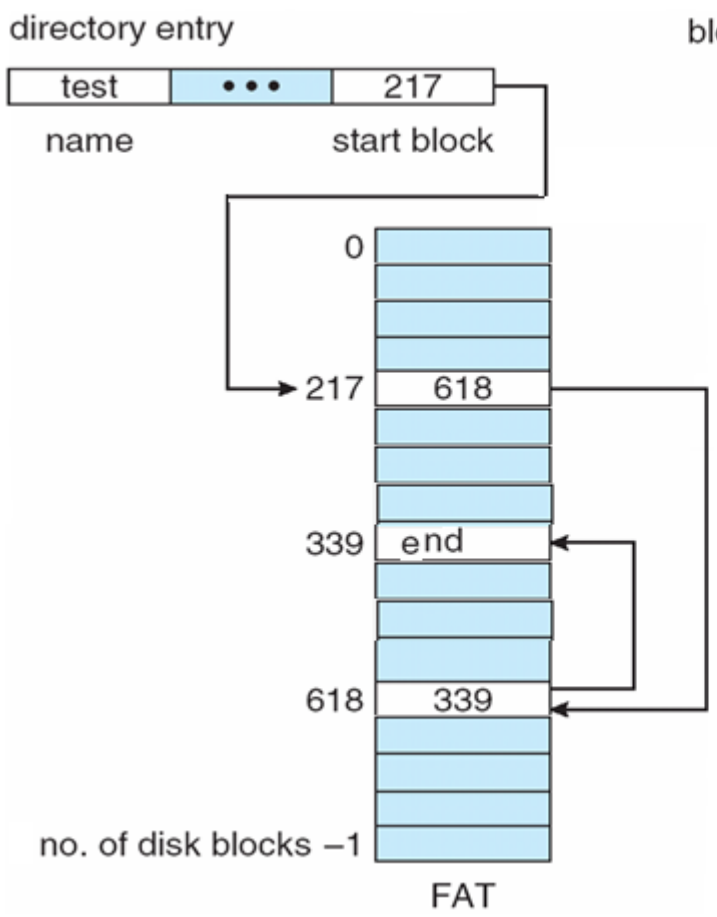
- The File Allocation Table (**FAT**) is **a variation to th e linked allocation** method used to support direct a ccess
  - disk-space allocation used by MS-DOS and OS/2.
- The FAT file system is a simple file system originally designed for small disks and simple folder structure s.
  - named for its method of organization, the **file allocation table**, whic h resides at the beginning of the volume.
- To protect the volume, two copies of the table are ke pt, in case one becomes damaged.
  - In addition, the file allocation tables and the root folder m ust be stored in a fixed location so that the files needed to start the system can be correctly located

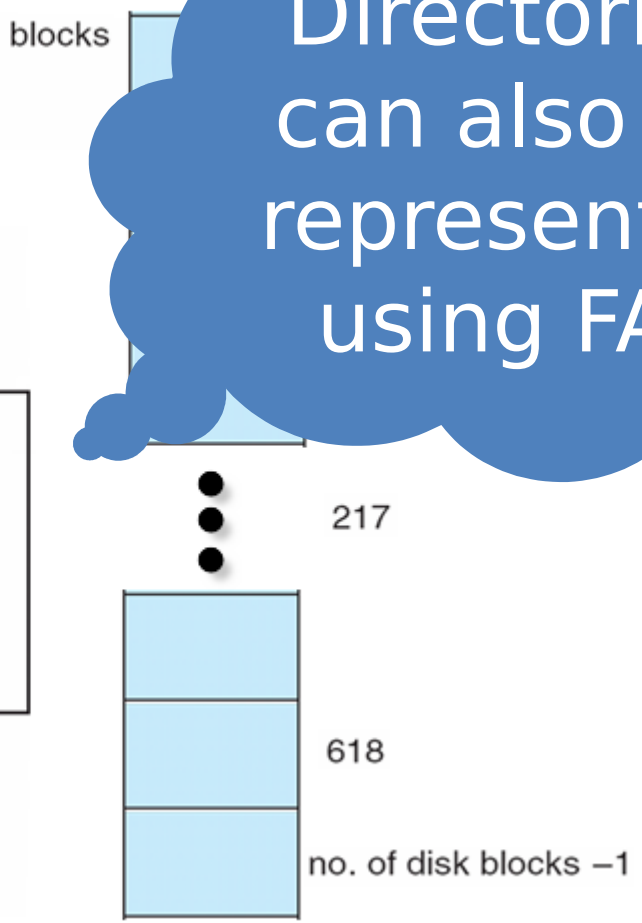
File Allocation Table

- The File Allocation Table (FAT) has many versions: FAT12, 16, 32 …
  - **FAT12** is only seen on floppy disks and very small storage media, while **FAT16** is the older version of FAT from the Windows 95 days, and **FAT32** is newer, from the Windows 98 days.
- NT, 2000, XP, Vista, and Windows 7 can use all the FAT file systems, plus the **NTFS** (New Technology File System)

Cont'

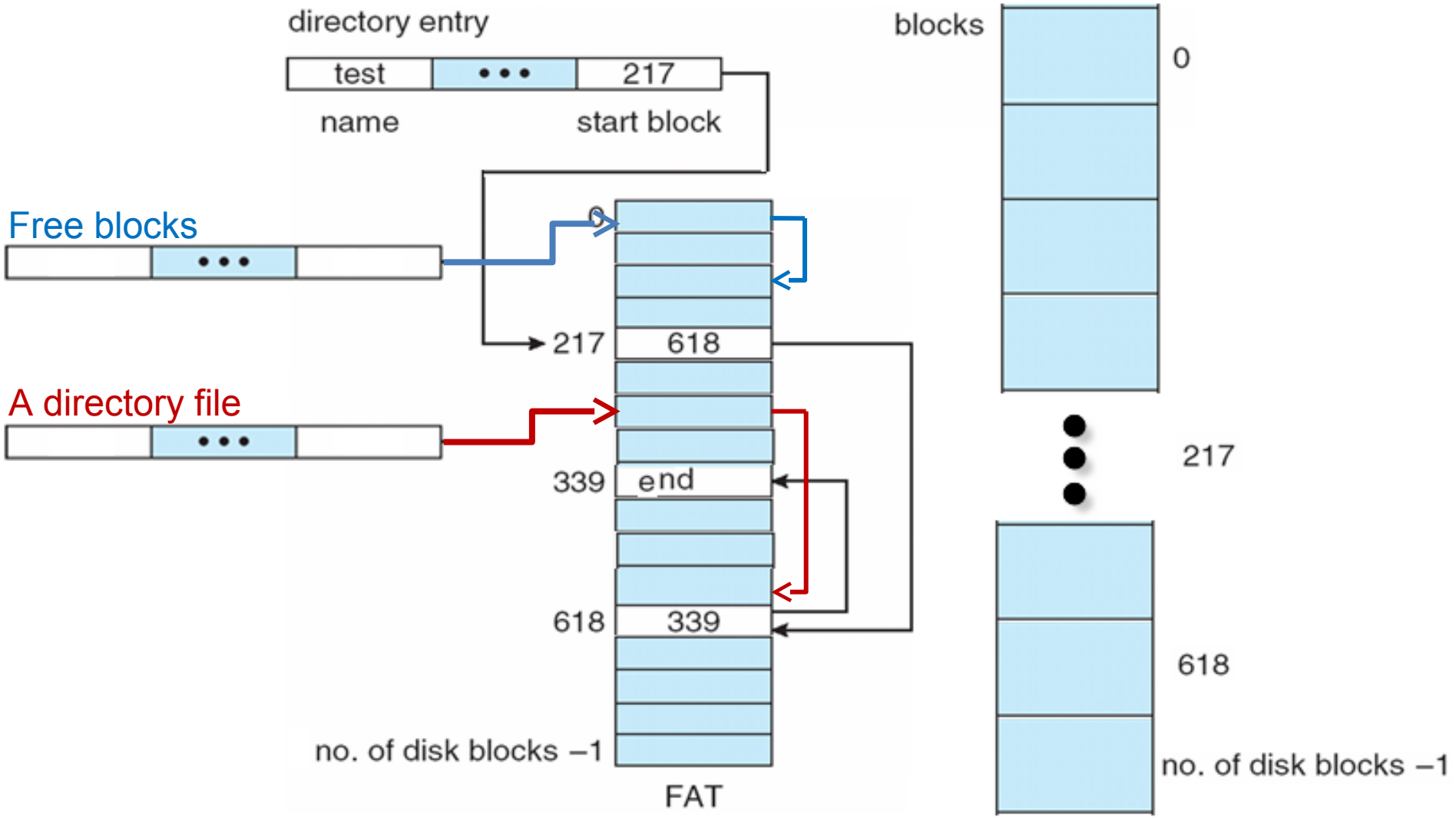
- **Pointers are kept in a table (FAT)**
- Data Block does not hold a pointer; hence data size in a block is a pow





Directories can also be represented using FAT

Cont'


directory entry
test
217
name
start block
Free blocks
A directory file
0
217
618
339
end
618
339
no. of disk blocks –1
FAT
blocks
0
217
618
no. of disk blocks –1

Cont'

- Of course the FAT is also stored in blocks
- So the size limit of a file is determined by
  - number of FAT entries
  - Size of block
- Given:
  - Block size = 4096 bytes
  - Length of a entry in FAT = 8 bytes

  Ⓟ one block can contain 4096/8 = 512 entries

  So the size of a file by using only one block could be: 512* 4KB = 2048KB = 2GB

Questions

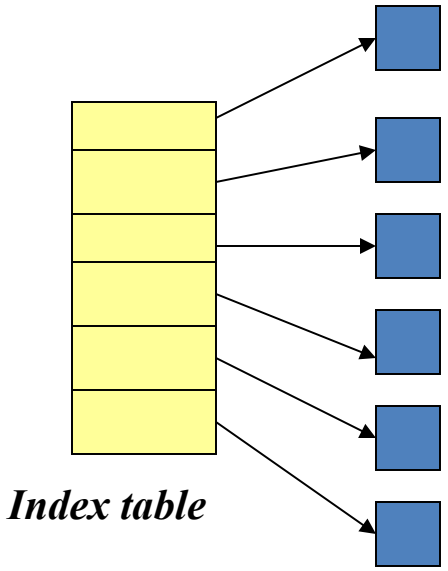
- A hard disk has 40G, its each block size is 1 K， and each table entry of FAT needs 20 b its， then Its FAT (File Allocation Table) ne ed （ ） memory space

  A） 100M　　B） 120M

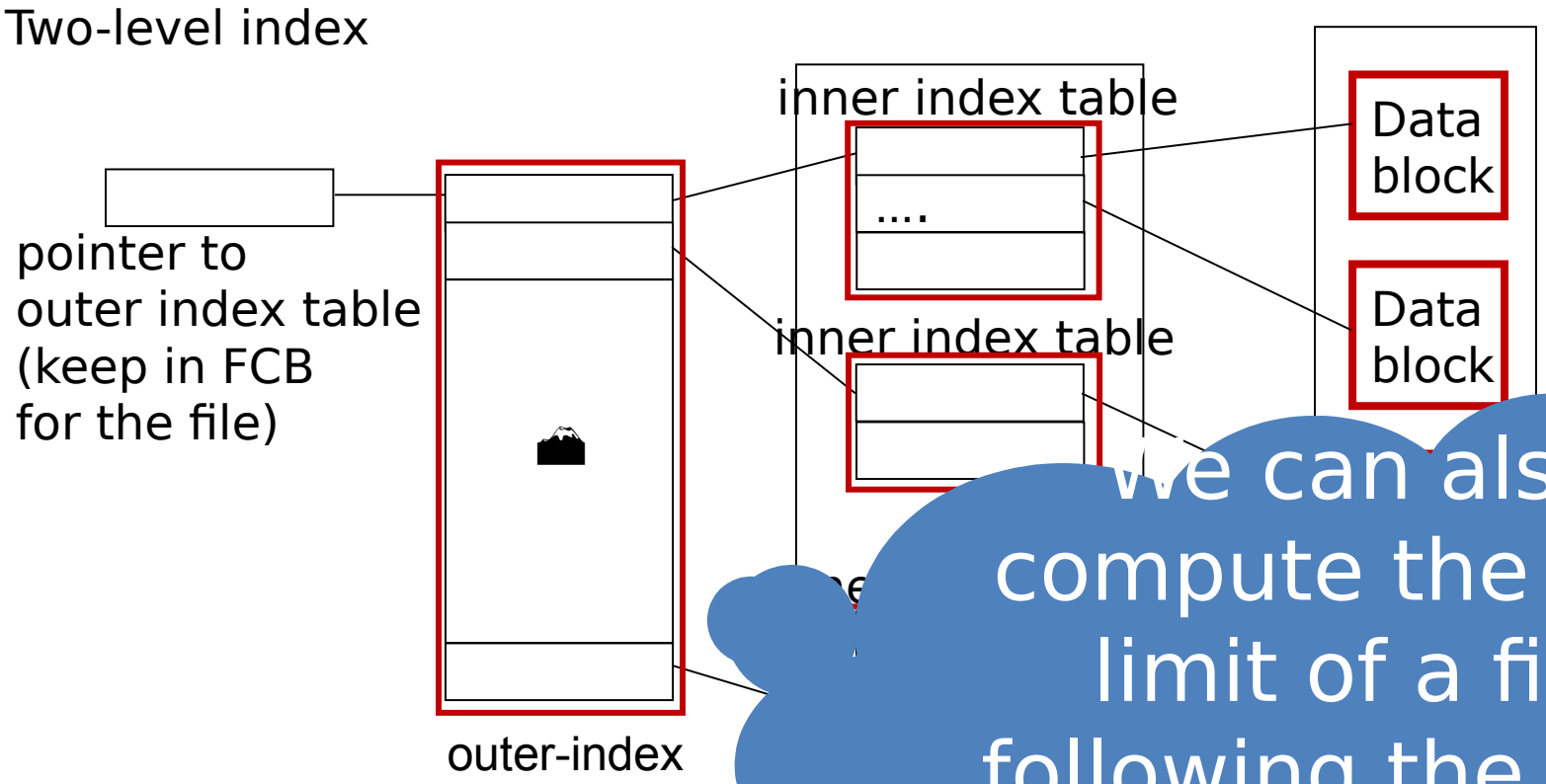
  C） 140M　　D） 160M

Indexed Allocation

- Brings all pointers together into the index block.
- Logical view



Indexed Allocation – Mapping (Cont.)

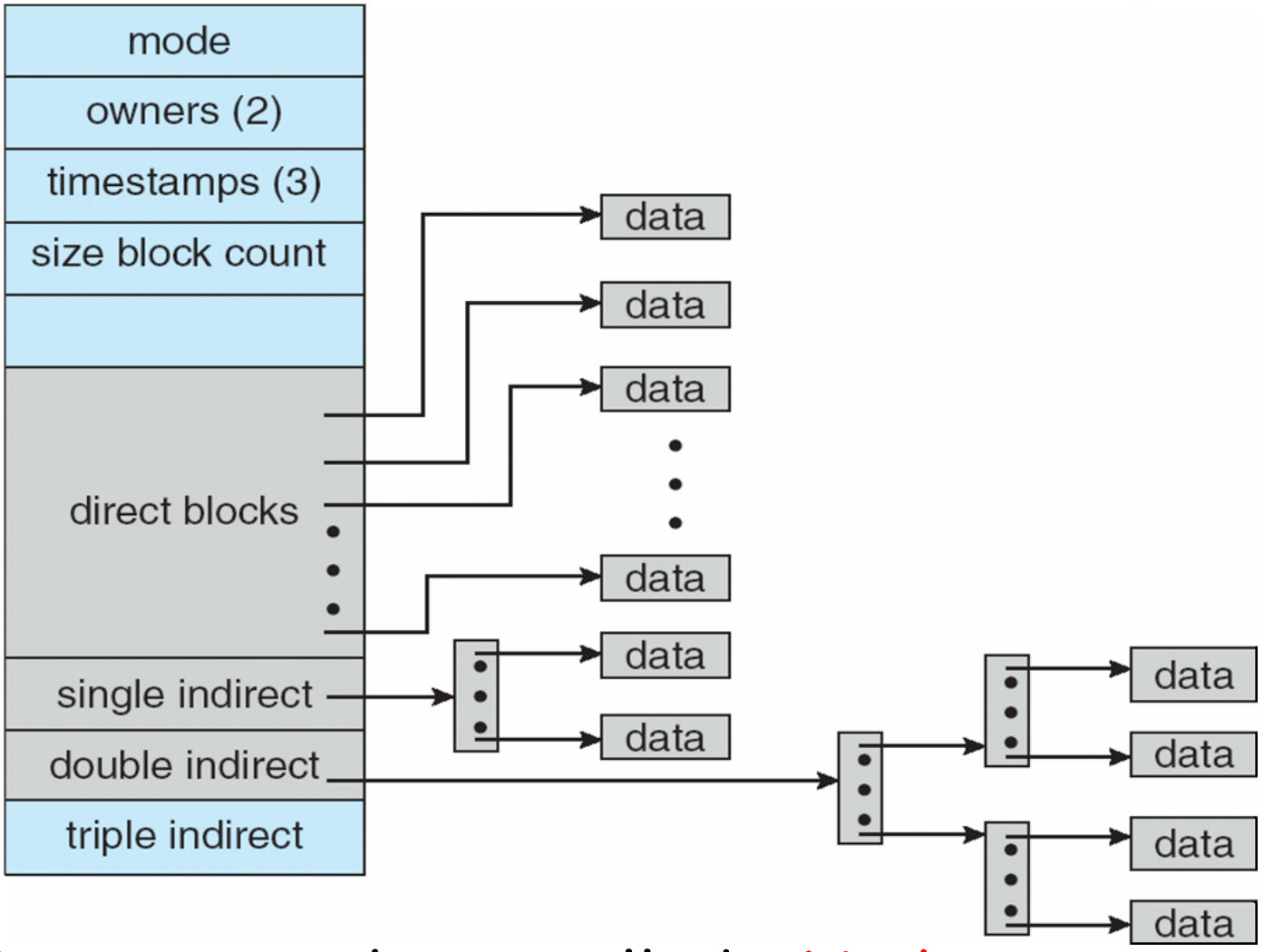


We can also compute the size limit of a file following the idea introduced in FAT. Attention to the levels

Indexed Allocation - Mapping

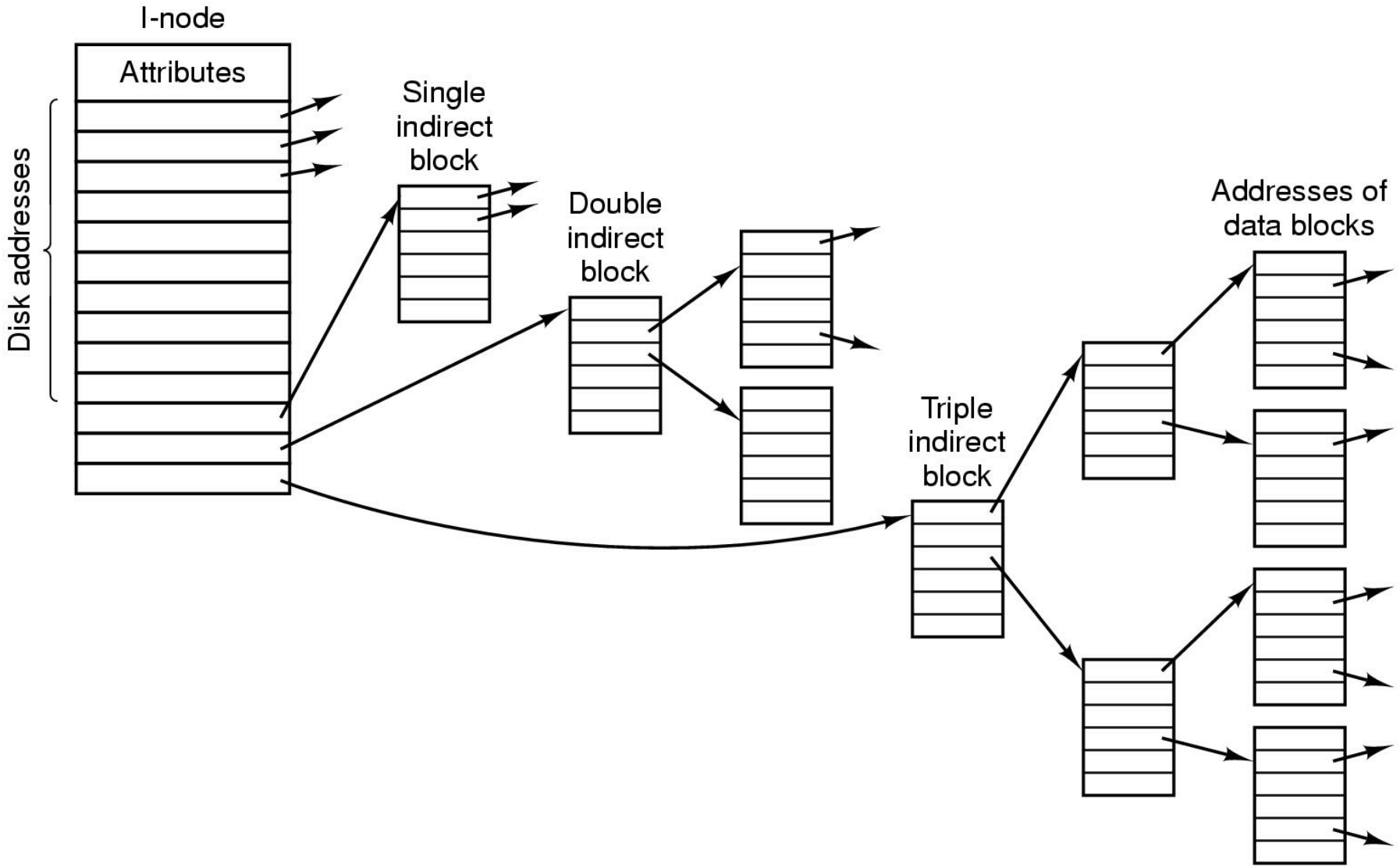
- Mapping from logical to physical in a file of unbounded length.
- Linked scheme -
  - Link blocks of index tables (no limit on size)
- Multilevel Index
  - E.g. Two Level Index - first level index block points to a set of second level index blocks, which in turn point to file blocks.
  - Increase number of levels based on maximum file size desired.
  - Maximum size of file is bounded.

Combined Scheme: UNIX (4K bytes per block)

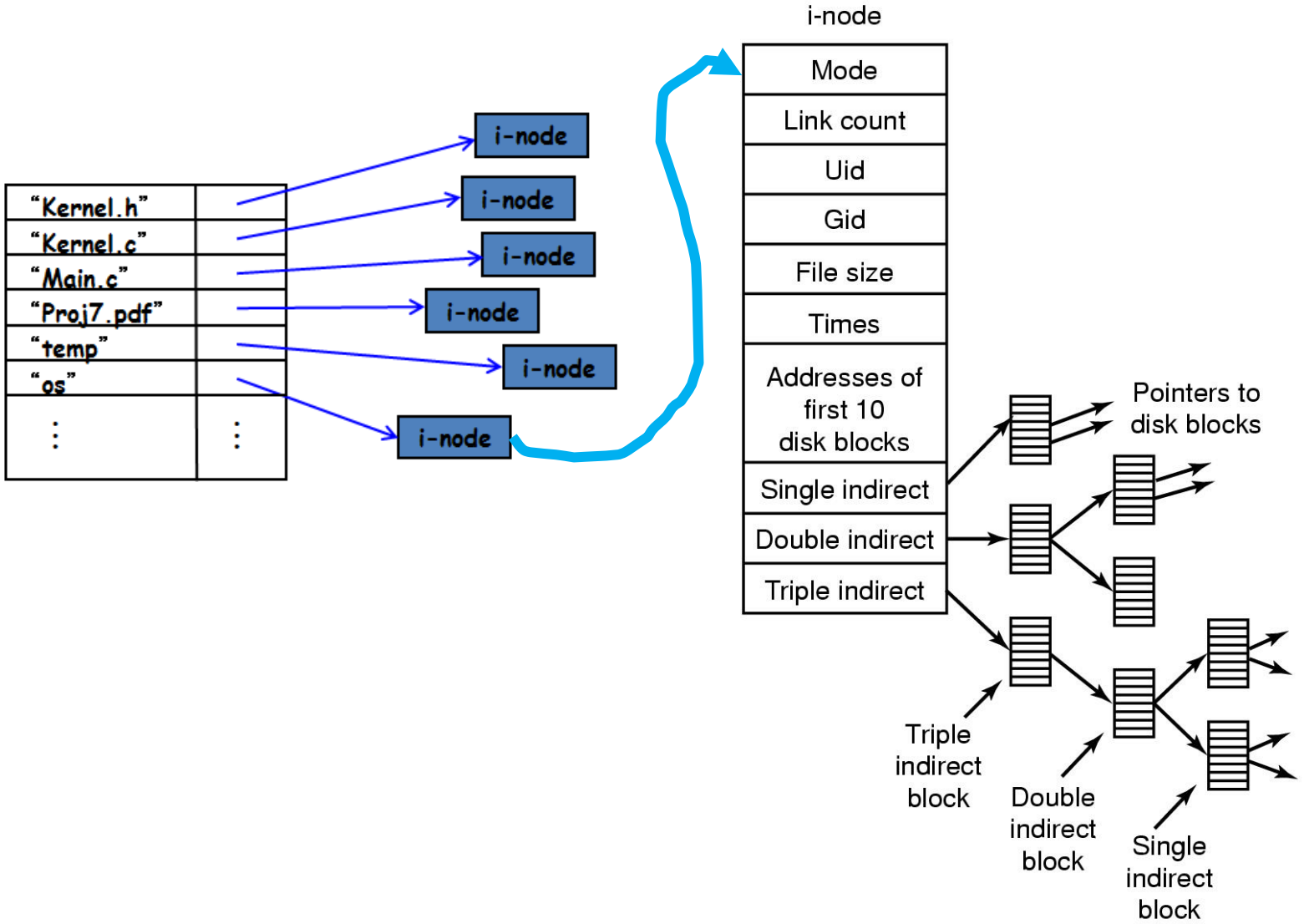


FCB - in Unix this is called i-Node

Unix i-node


I-node
Attributes
Disk addresses
Single indirect block
Double indirect block
Triple indirect block
Addresses of data blocks

The UNIX I-node



- Assume that the index-node of a file has 7 pointers, among which 4 are for data blocks, 2 pointers for “indirect block”, and 1 pointer for “doubly indirect block”.
- The size of both data block and index block are 256KB. The length of a pointer is 4 Bytes.
- What is the max file size this file system can support?

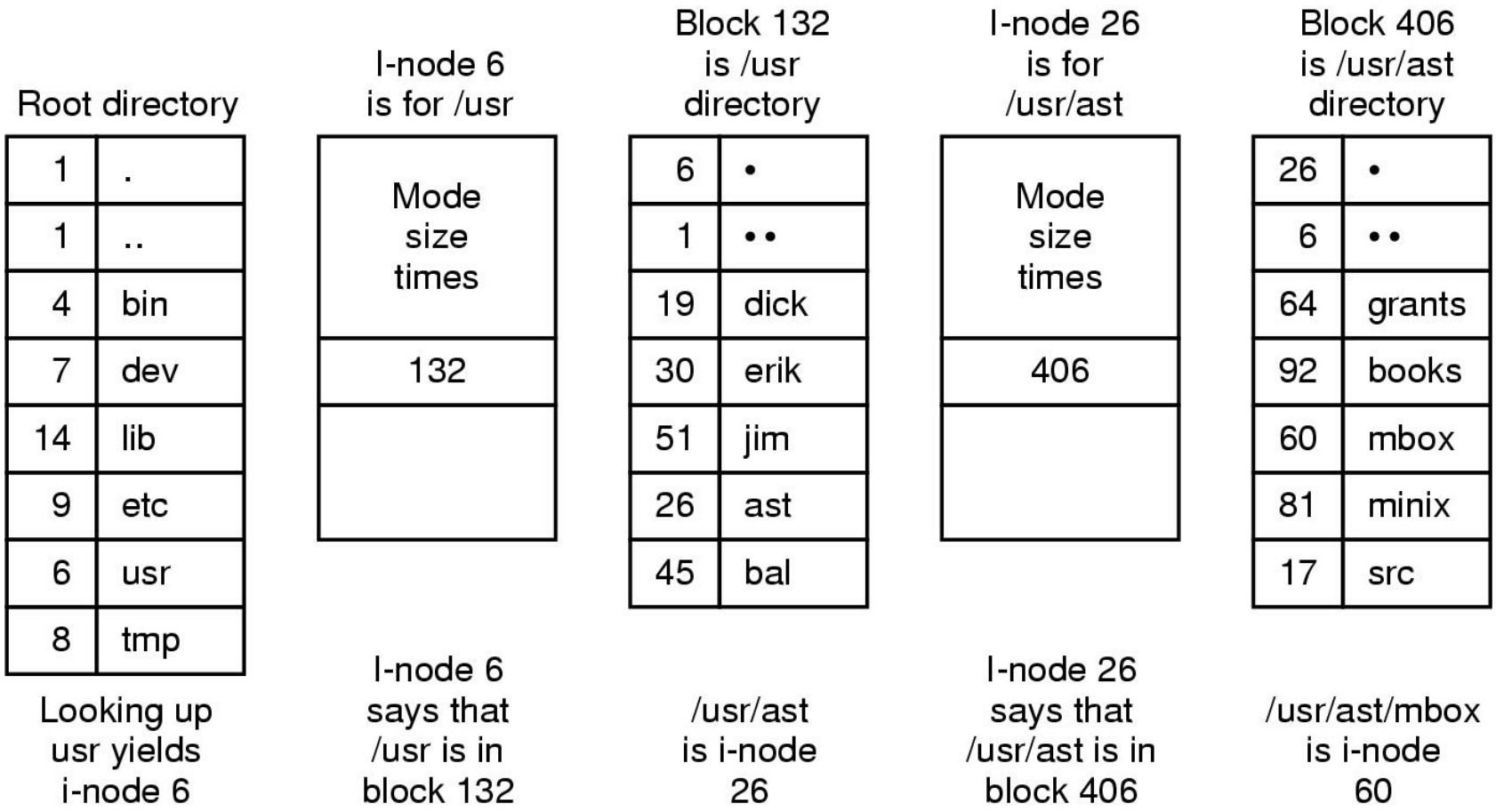
Entry Lookup

- In Unix systems
  - The superblock (among other things) has the location of the i-node which represents the root directory
- Once the root directory is located a search through the directory tree finds the desired directory entry
- The directory entry provides the information needed to find the disk blocks for the requested file

Entry Lookup

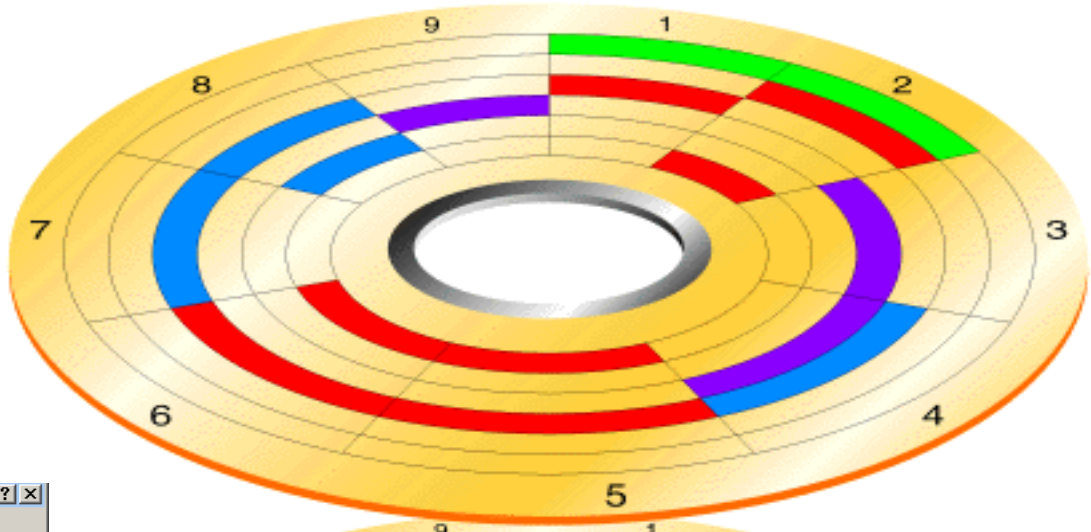
- This discussion focuses on Unix-related fil e systems
- When a file is opened, the file system must take the file name supplied and locate its di sk blocks
- Let's see how this is done for the path nam e */usr/ast/mbox*

Looking up for an entry

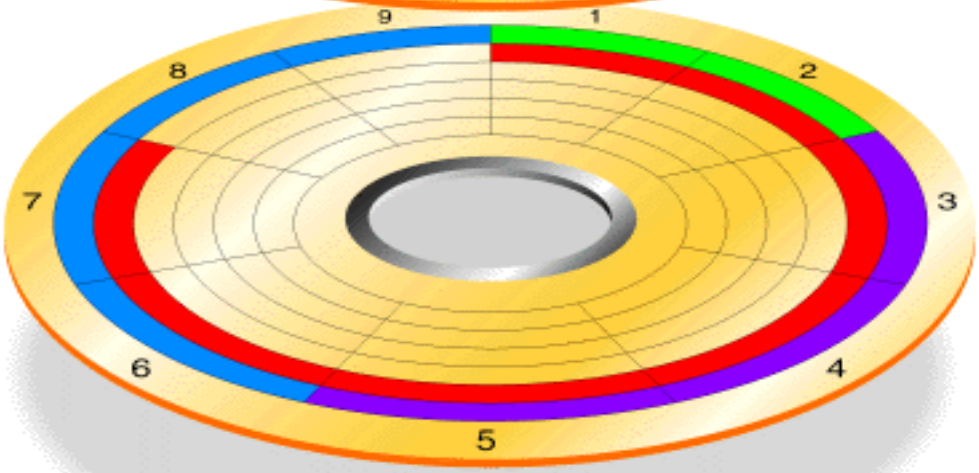


The steps in looking up /usr/ast/mbox

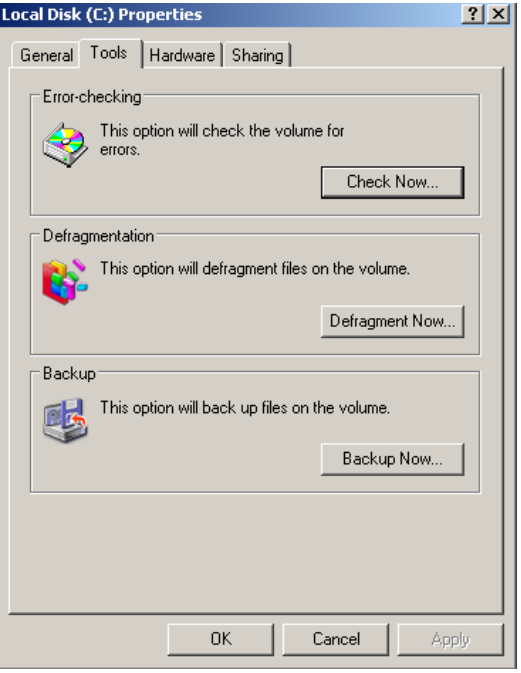
Fragmentation and Defragmenting for disks



fragmented disk



disk after defragmenting



File System

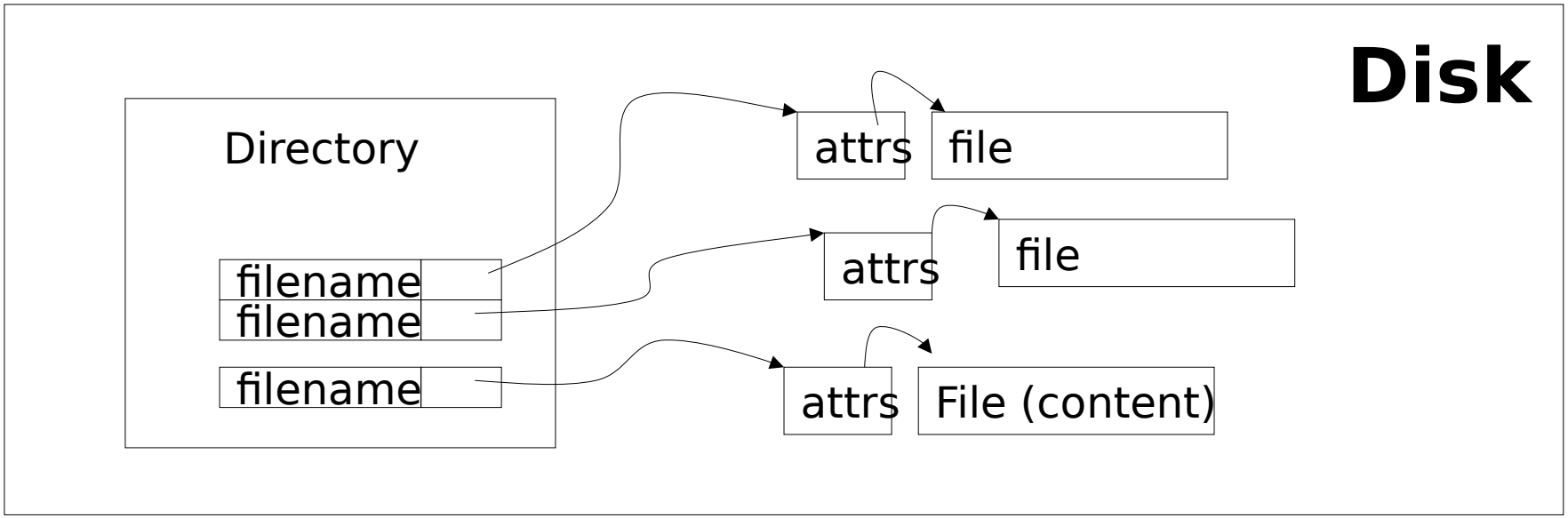
- Mapping 2: File to HDisk (Linear addressed block Space)
  - Organize blocks into semantic regions
    - Boot block, superblock, directory
  - Free space
    - You need know where available blocks are.
  - Map a file (collection of bits) into blocks
    - Needed blocks, and how to indicate them?
  - How to organize so many files?
    - Directory

Files and Directories

- There are two basic things that are stored on disk as part of the area controlled by the file system

  - files (store content)

  - directory information (can be a tree): keeps info about files, their attributes or locations

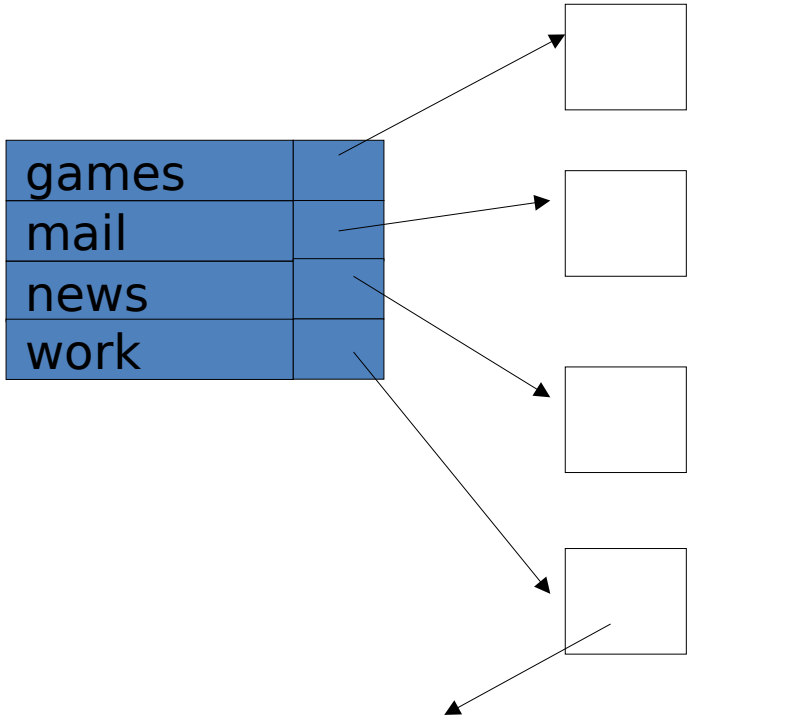


Directory Implementation: directory entries

games	attributes
mail	attributes
news	attributes
work	attributes

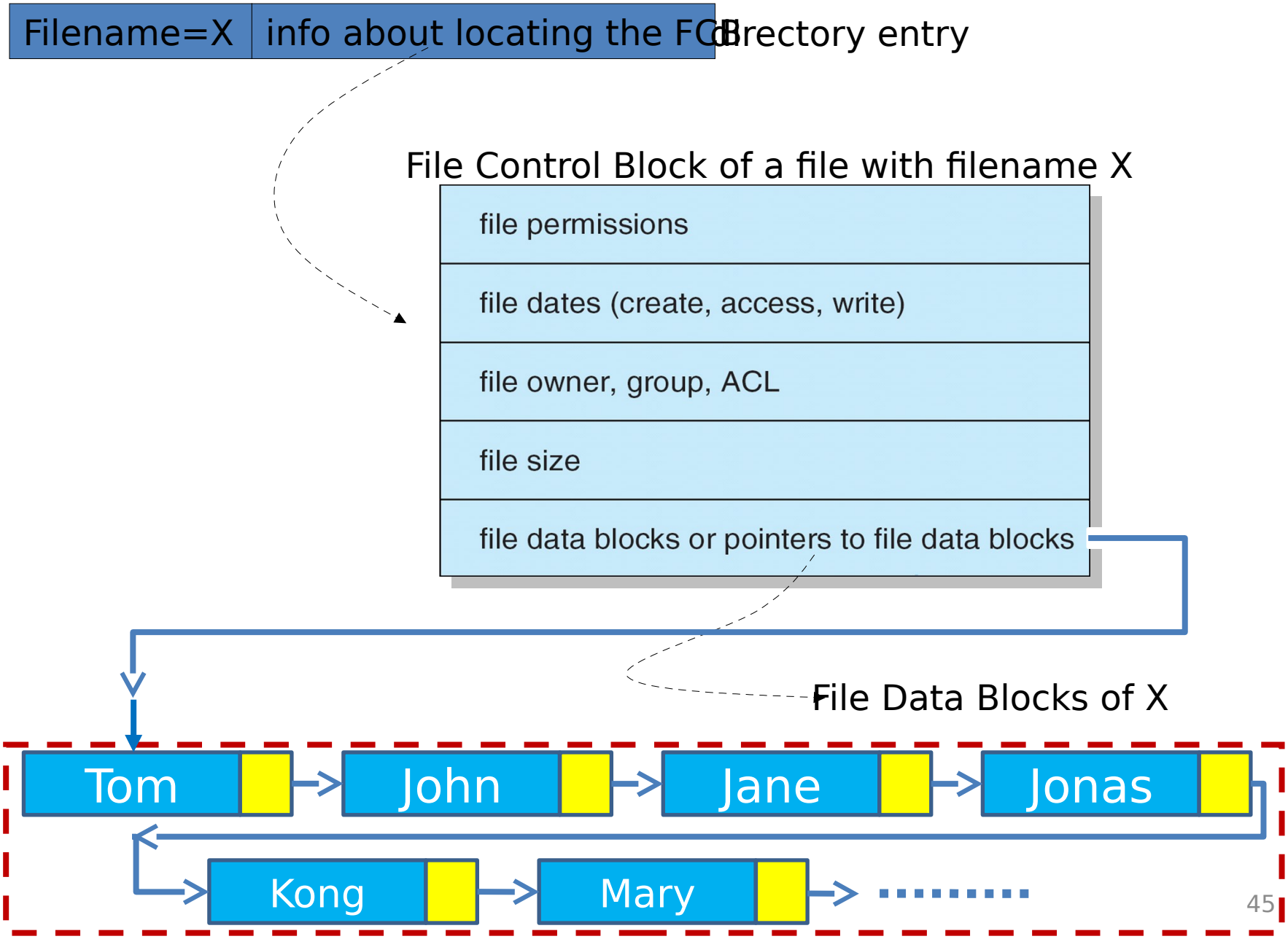
a directory with fixed sized entries

attributes include location info for data blocks of the file



Using fixed sized names

A Typical File Control Block



Cont'

- Linear list of file names with pointers to the data blocks
  - simple to program
  - time-consuming to execute - linear search to find ent ry.
  - Sorted list helps - allows binary search and decrease s search time.
- Hash Table - linear list with hash data structure
  - decreases directory search time
  - collisions - situations where two file names hash to t he same location.
  - Each hash entry can be a linked list - resolve collision s by adding new entry to linked list.

Of course, you should provide File Operations

- File is an **abstract data type**
- **Common Operations that are supported by the Operating System:**
  - Create
  - Write
  - Read
  - Reposition within file
  - Delete
  - Truncate

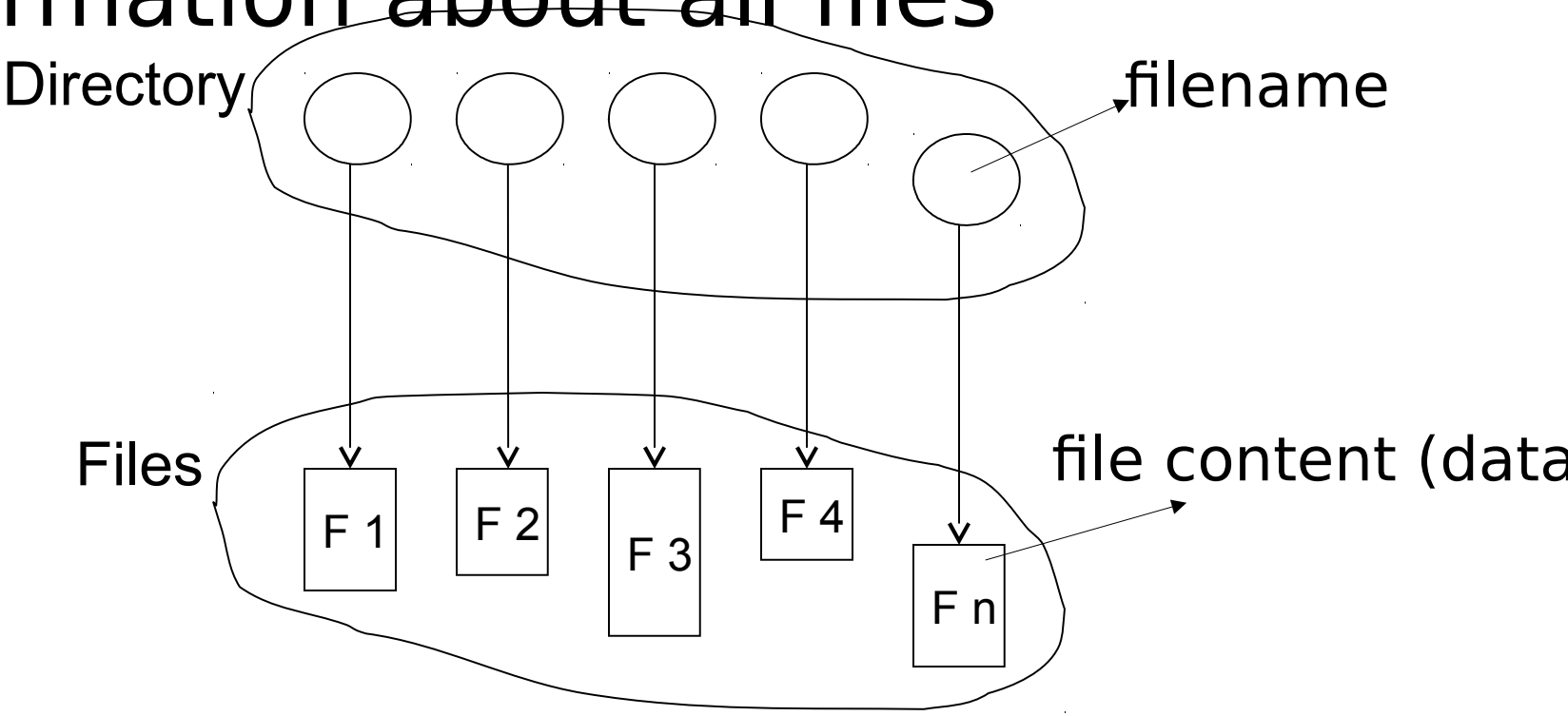
- **Open(F_i)** – search the directory structure on disk for entry F_i, and move the content of entry to memory
- **Close (F_i)** – move the content of entry F_i in memory to directory structure on disk

Directory Implementation

- A directory entry provides the info needed to find the disk data blocks of a file
  - disk address of first block and size
  - address of first block
  - number of associated i-node
- File attributes can be stored in the director y entry (Windows) or in its i-node (Unix)
- File name and support of variable length a nd long file names (255 chars)

Directory Structure

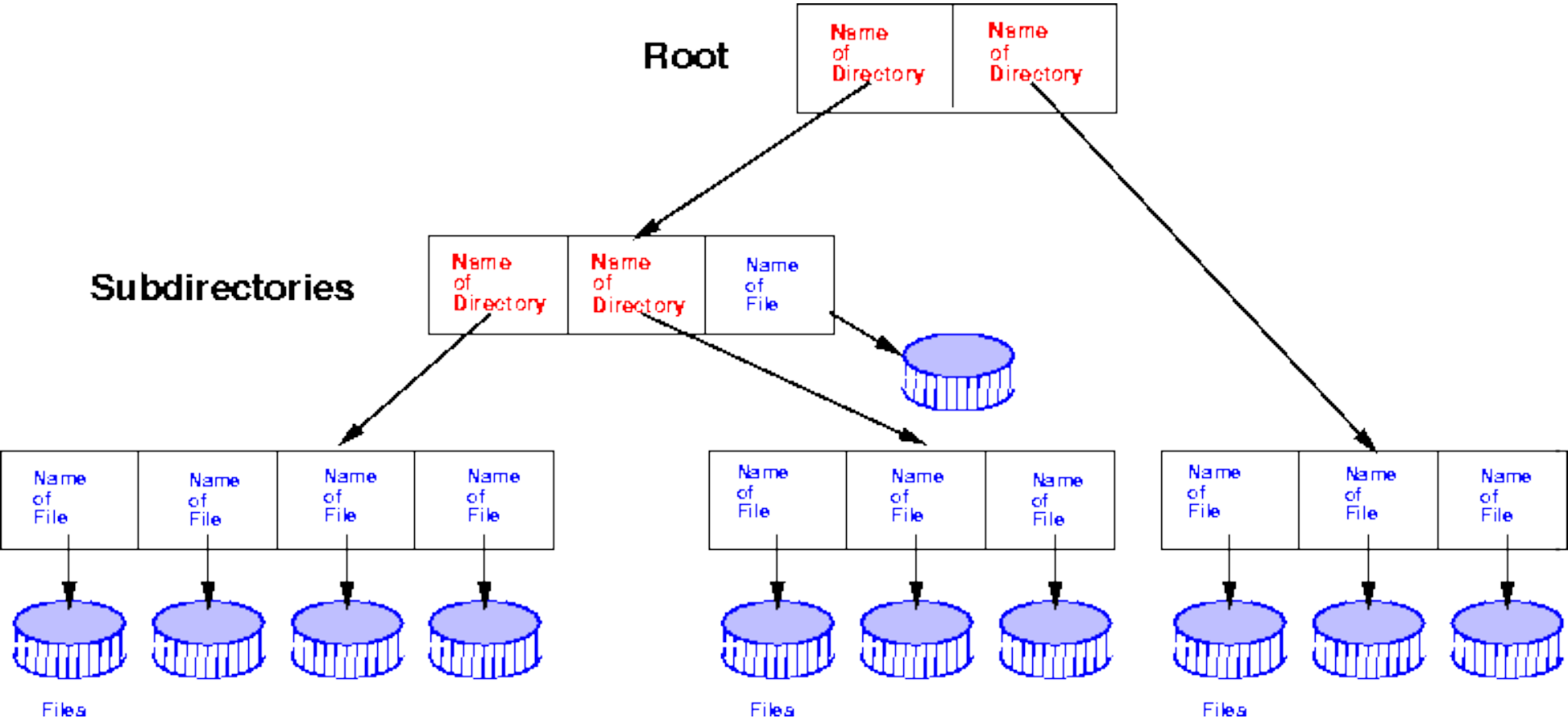
- A collection of nodes containing information about all files



- Both the directory structure and the files reside on disk
- Backups of these two structures could be kept on tapes

Directory Structure:

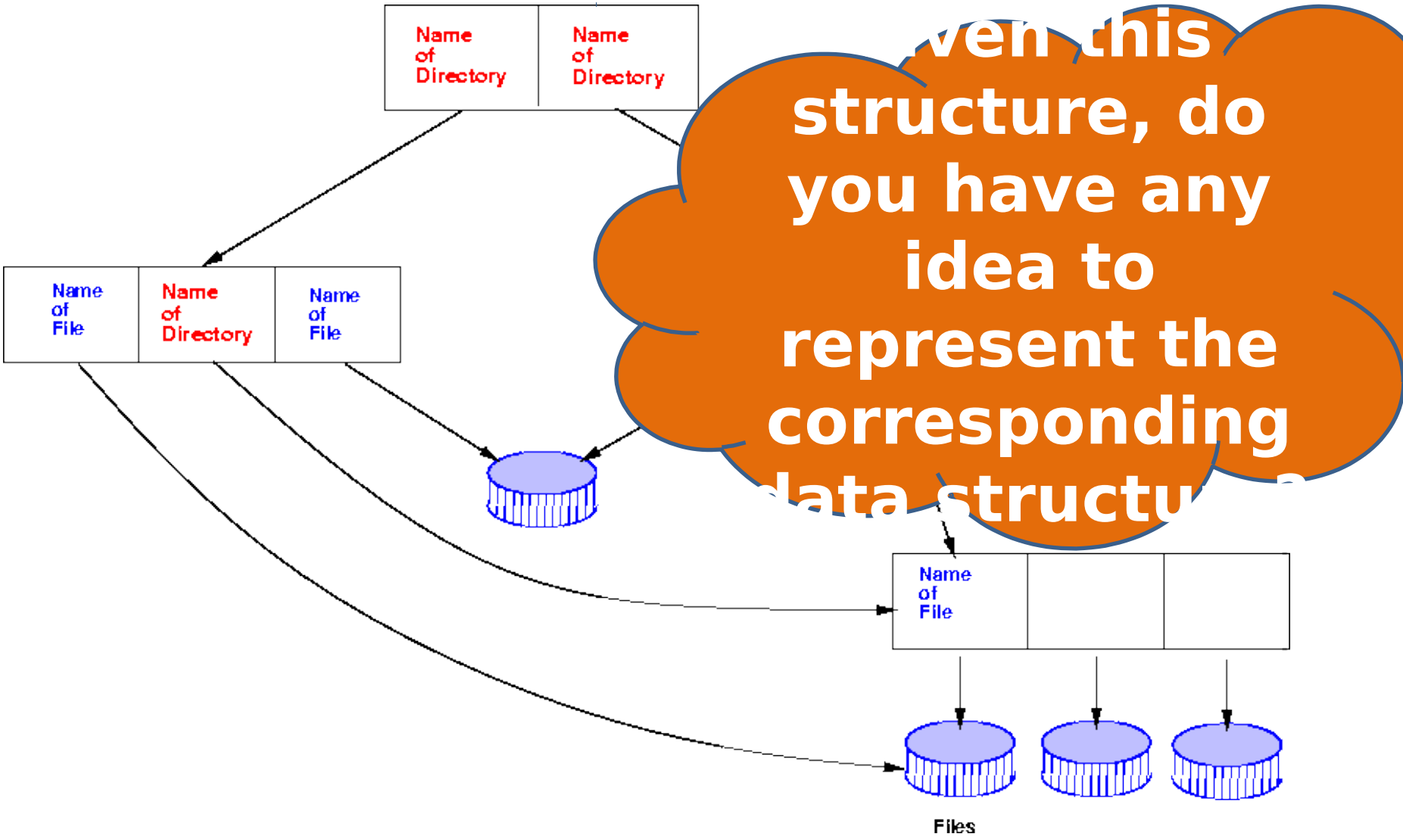
Tree structured Directories



Cont’

- Absolute or relative path name
  - Absolute from root
  - Relative paths from current working directory pointe r.
- Creating a new file is done in current directory
- Creating a new subdirectory is done in current directory, e.g. mkdir <dir-name>
- Delete a file , e.g. rm file-name
- Deletion of directory
  - Option 1 : Only delete if directory is empty
  - Option 2: delete all files and subdirectories under dir ectory

Directory Structure:
Acyclic Graph Directories [无环图目录]



...ven this structure, do you have any idea to represent the corresponding data structu...?

Cont'

- Acyclic graphs allow sharing
- Implementation by links
  - Links are pointers to other files or subdirectories
  - Symbolic links or relative path name
    - Directory entry is marked as a link and name of real fil e/directory is given. Need to resolve link to locate file.
- Implementation by shared files
  - Duplicate information in sharing directories
  - Original and copy indistinguishable.
  - Need to maintain consistency if one of them is m odified.

In summary

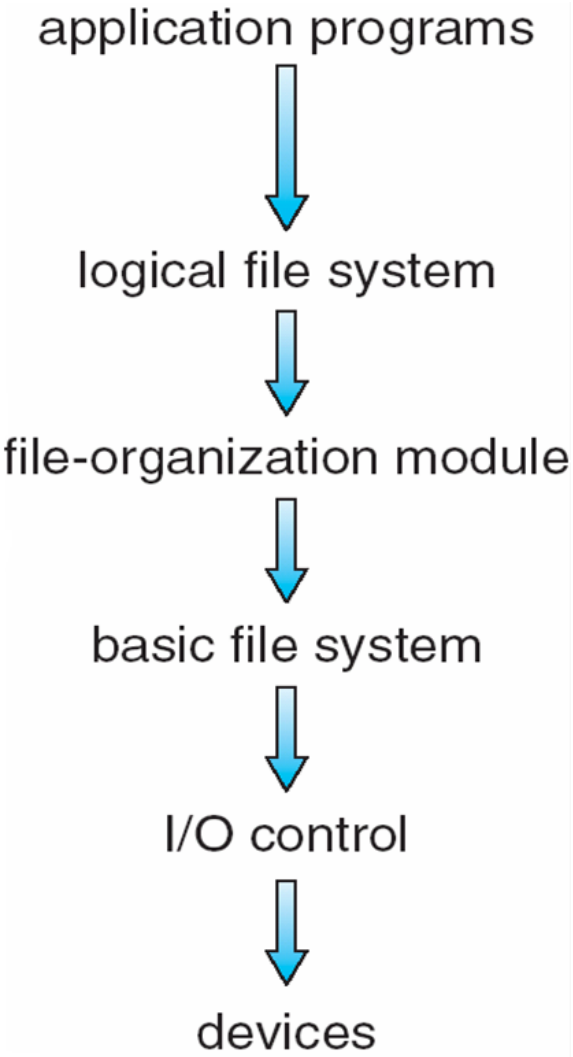
File system [文件系统]

- The collection of algorithms and data structures which perform the translation from **logical file operations** (system calls) to **actual physical storage of information**
  - Provide storage of data and manipulation
  - Guarantee consistency of data and minimize errors
  - Optimize performance (system and user)
  - Eliminate data loss (data destruction)
  - Support variety of I/O devices
  - Provide a standard user interface
  - Support multiple users

PPTs from others\www.ceng.metu.edu.tr_courses_ceng334\Ch_4_FS.ppt

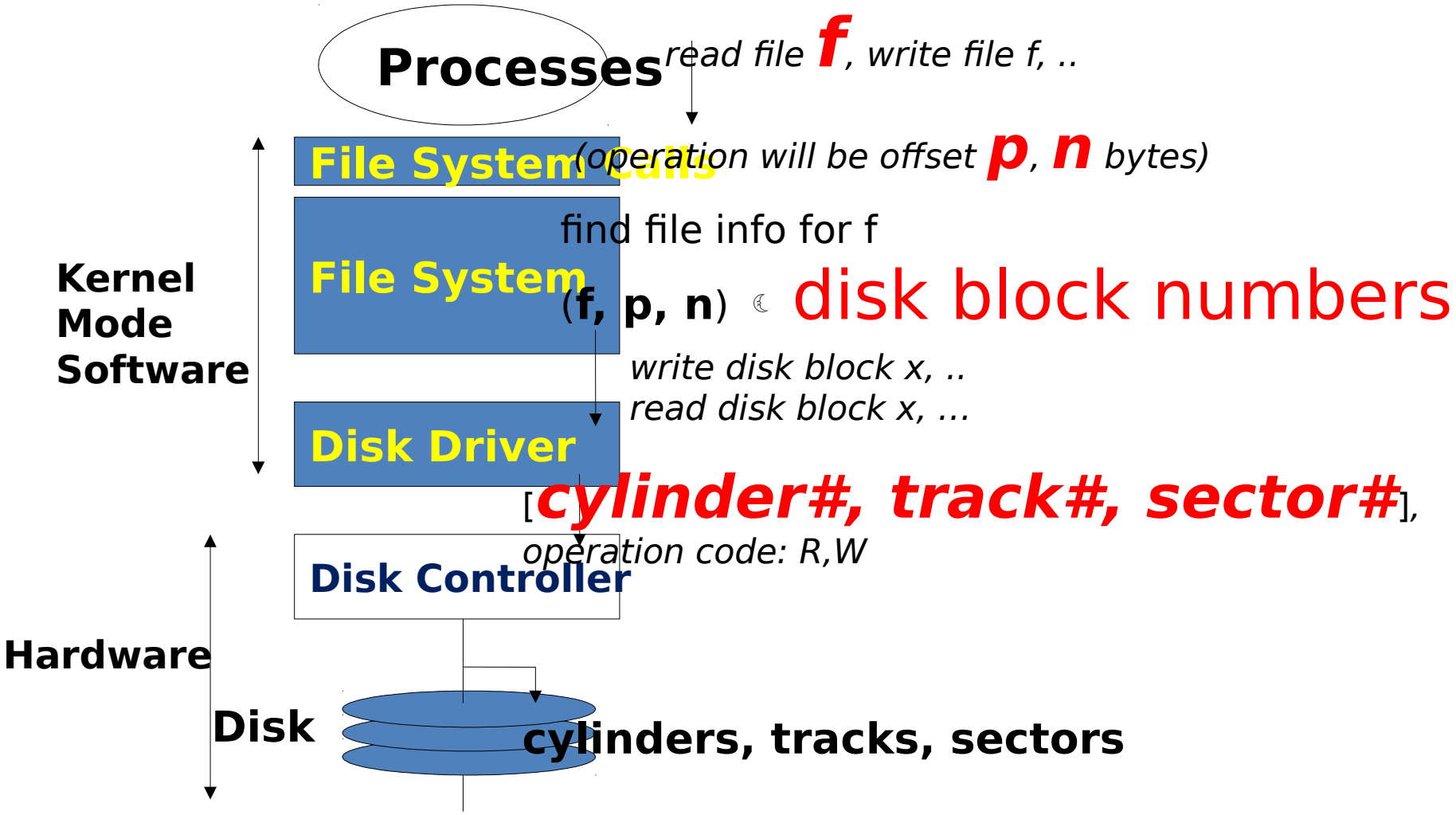
File System

- The file system should provide an efficient implementation of the interface
  - storing, locating, retrieving data
- The problem: define data structures and algorithms to map the **logical File System** onto the disk
  - some data structures live on disk
  - some data structures live (temporarily) in memory
- Typical layer organization:
  - Good for modularity and code re-use
  - Bad for overhead



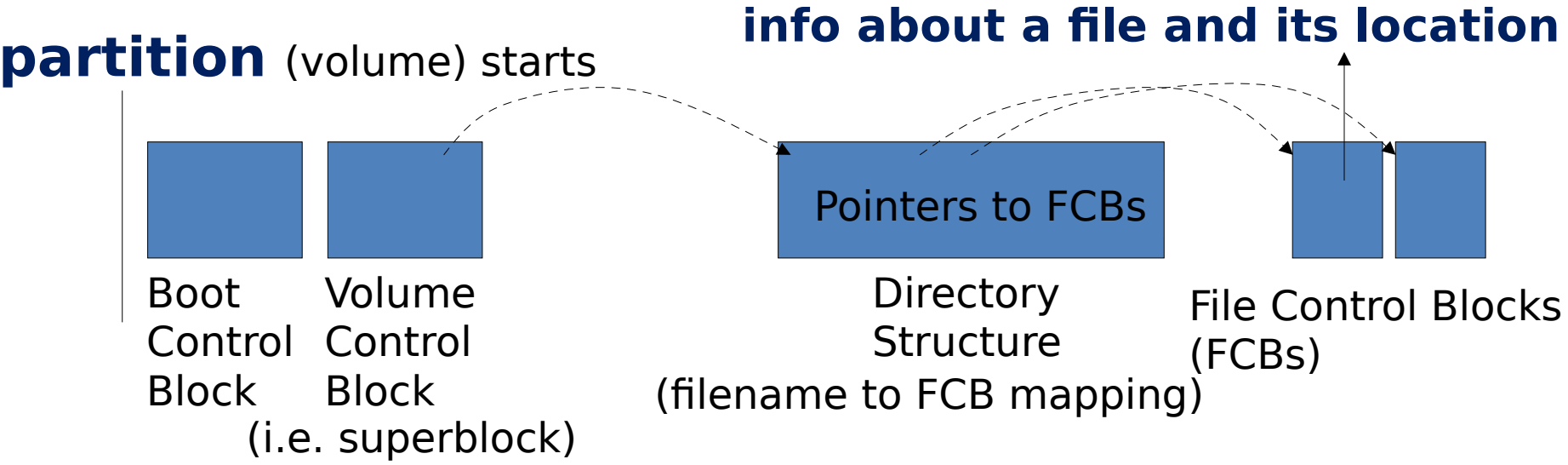
PPTs from others\navet.ics.hawaii.edu_~casanova_courses_ics412_fall09\ics412_fs2.ppt

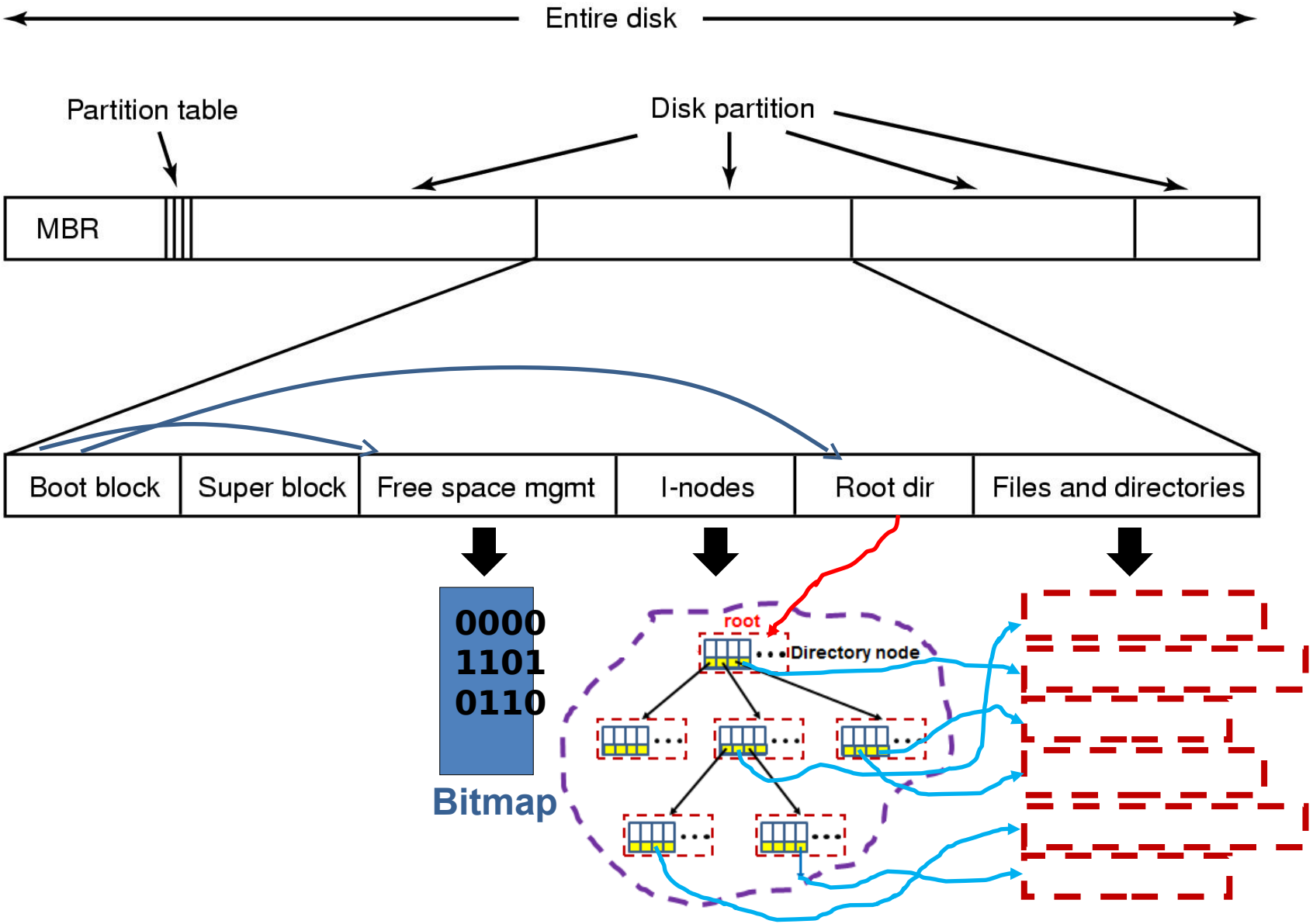
Layered Software



File System Implementation

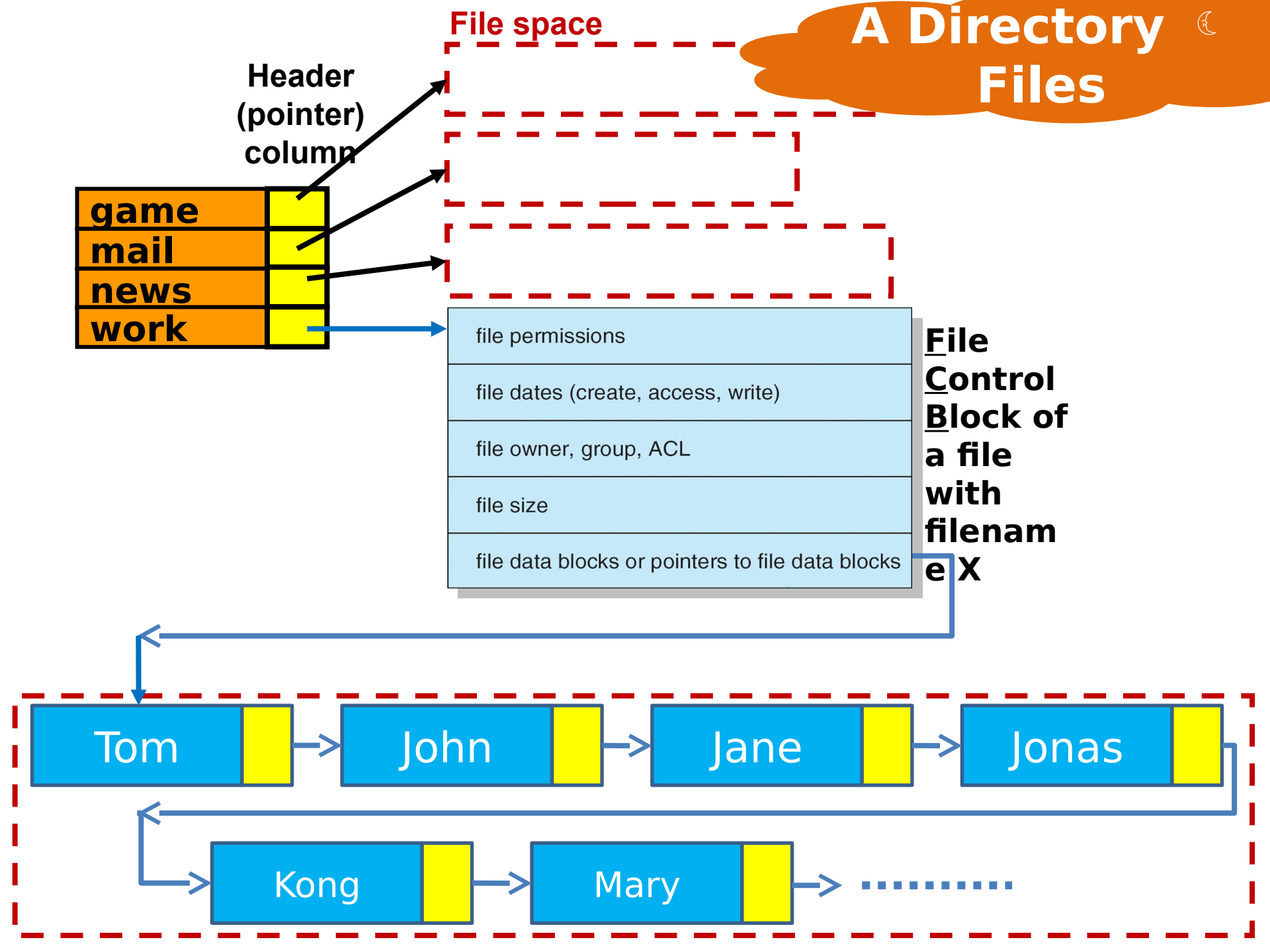
- Major On-disk Structures (information):
  - Boot control block contains info needed by system to boot O S from that volume
  - Volume control block contains volume details
  - Directory structure organizes the files
  - Per-file File Control Block (FCB) contains many details about t he file



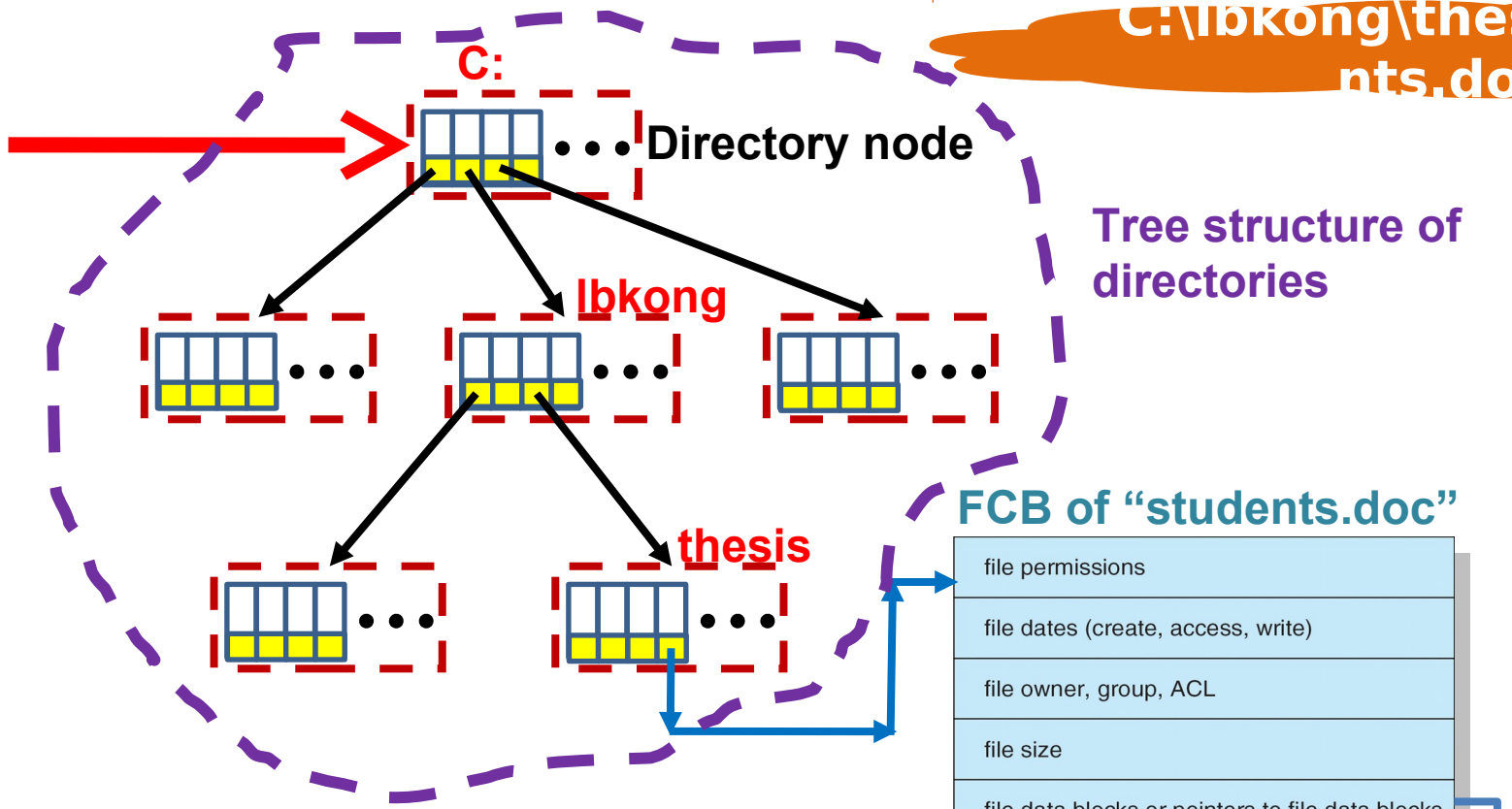


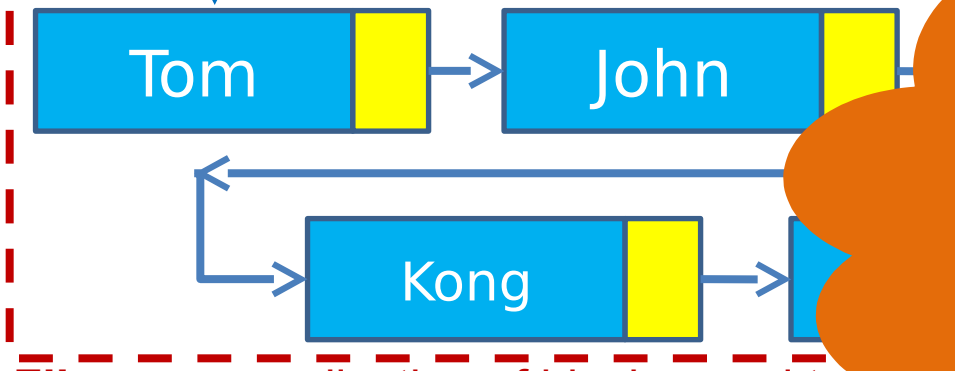
Disk ☾ Partitions ☾ Blocks ☾ FS

A Directory Files



C:\lbkong\thesis\stude
nts.doc





File space: collection of blocks used to st

ith your DSA experience, you should consider the related operations on this data structure. Could you imagine how to locate “students.doc”? And

Compared with MM

To run our program, the needed pages should be fetched into MM

(PCB maintains the reference for our program file)

(OS maintains the **mapping info** between program pages and blocks)

Our program exists first as file

which may contain many **pages**

(filename ☾ FCB ☾ Efficient data structure (like Hash, i-node,…) ☾ **Blocks**)

Addressed **frame/page** space (usually 4KB)

Addressed **block** space (usually 4KB)

Addressed **Storage unit** space (1 byte)

Addressed **Sector** space (512 bytes)

Suggest for deeper study

- You can try to implement a simulation progr
  am to support file + dir
  - Data structure + Ope
    - New file, Delete, Re
- Continue to consider
  ent access of a file by many user?
  - What should we consider ? – Data structure + Ope
    rators
- Have you ever considered how to store an arr
  ay of structs into a file? – Don’ t forget to rec
  onstruct their structure when reading into M
  M

helpful for your understanding about the implementation of DBMS