

File Systems

Operating Systems – EDA093/DIT401

Vincenzo Gulisano

vincenzo.gulisano@chalmers.se



UNIVERSITY OF
GOTHENBURG

Reading instructions

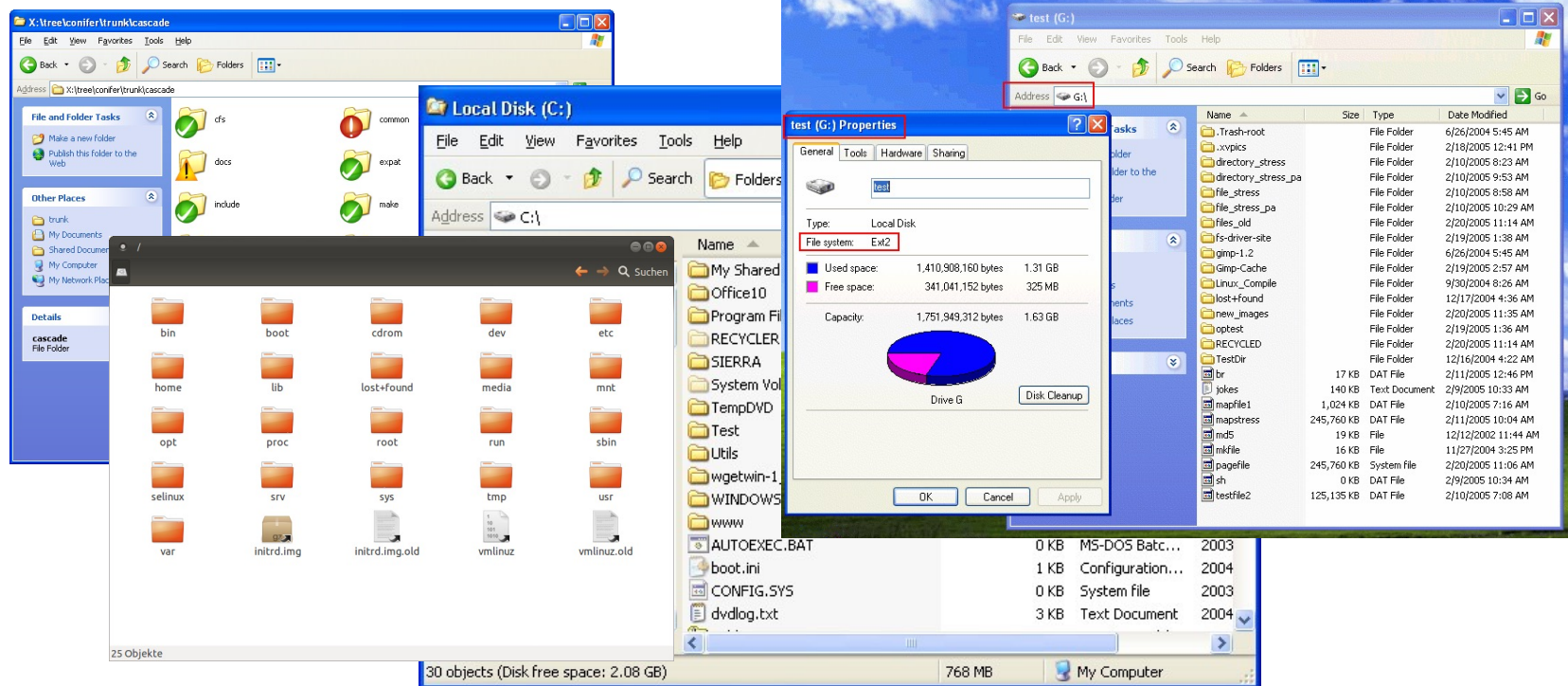
- Chapter 4.1 to 4.4

(extra facultative reading: 10.1-10.3.6,10.4-10.6,11.1-11.2,11.4-11.5 from Silberschatz Operating System Concepts)

Which are the most visible aspects of a File System?

Files

Directories



Objectives

- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection
- To discuss block allocation and free-block algorithms and trade-offs

AGENDA

- File concept



What extra information / overheads / interfaces does the OS need to provide file and directory abstractions to users?

- Access methods



How to we access files and which is the minimum unit we can read/write from secondary storage?

- Disk structure
(Self reading)



How is the physical disk organized in order to maintain 1 (or more) file systems?

- Directory structure



How can we organize directories (and files) in a file system?

- File-system structure



How is a file system organized in layers?

- Allocation methods



How do we know where each file is?

- Free-space management



How do we know what portions of a file systems are free?

- File sharing and protection



How do we share and protect files among different users?

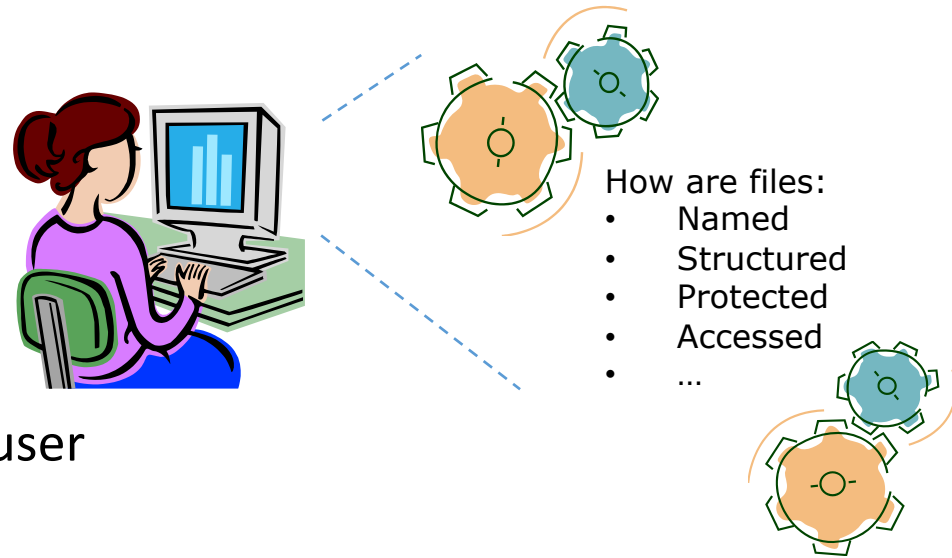
AGENDA

- File concept
- Access methods
- Disk structure
- Directory structure
(Self reading)
- File-system structure
- Allocation methods
- Free-space management
- File sharing and protection



What extra information / overheads / interfaces does the OS needs to provide file and directory abstractions to users?

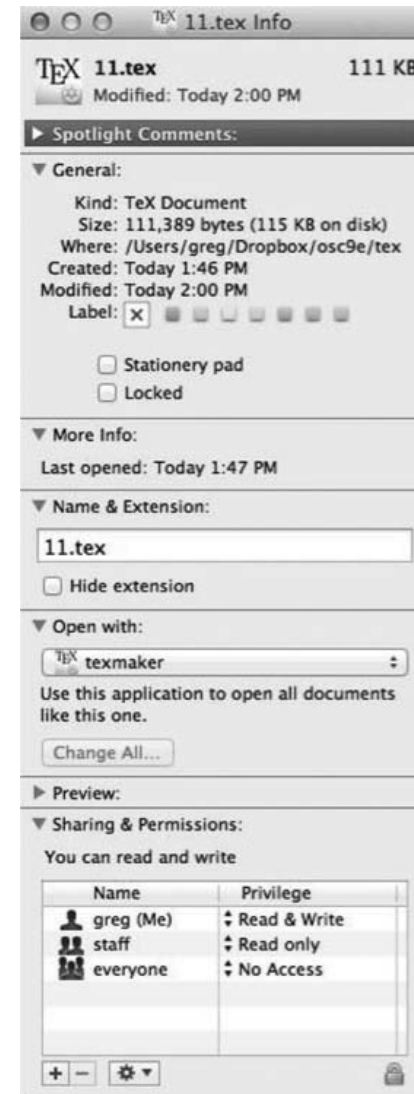
File Concept



- Minimum allocation unit for the user
- File system abstracts how they are mapped into secondary storage
- Similarly to processes, files have contiguous logical address space (but not physical)
- Different types: Data files (numeric, character, binary) and Program files
- Contents defined by file's creator! (Consider **text file**, **source file**, **executable file**)

File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring



File Types – Name, Extension

- Operating System, Processes and File System are dependent on each other
- Extensions provide hints to the Operating System about which program to load in order to present the content to the user

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Open Files

- Providing a File System implies that the OS needs to maintain information. For instance, information about the open files:
 - **Open-file table**: tracks open files
 - File pointer: pointer to last read/write location, per process that has the file open
 - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
 - Disk location of the file: cache of data access information
 - Access rights: per-process access mode information

Some information
is specific to a
process

Some information
is global,
system-wide

File Operations

- File is an **abstract data type**, we need ways to operate on them:
 - **Create**
 - **Write** – at **write pointer** location
 - **Read** – at **read pointer** location
 - **Reposition within file** - **seek**
 - **Delete**
 - **Truncate**
- We can combine these basic operations into complex operations
- Example:
 - ***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

System calls - Example program – 1/2

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);    /* ANSI prototype */

#define BUF_SIZE 4096                /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700             /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);           /* syntax error if argc is not 3 */

    /* Open the input file and create the output file */
    in_fd = open(argv[1], O_RDONLY);  /* open the source file */
    if (in_fd < 0) exit(2);           /* if it cannot be opened, exit */
    out_fd = creat(argv[2], OUTPUT_MODE); /* create the destination file */
    if (out_fd < 0) exit(3);          /* if it cannot be created, exit */
}
```

Fig. Tanenbaum, Modern Operating Systems

System calls - Example program – 2/2

```
/* Copy loop */
while (TRUE) {
    rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
    if (rd_count <= 0) break; /* if end of file or error, exit loop */
    wt_count = write(out_fd, buffer, rd_count); /* write data */
    if (wt_count <= 0) exit(4); /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0) /* no error on last read */
    exit(0);
else
    exit(5); /* error on last read */
}
```

Fig. Tanenbaum, Modern Operating Systems

AGENDA

- File concept
- Access methods
- Disk structure
- Directory structure
(Self reading)
- File-system structure
- Allocation methods
- Free-space management
- File sharing and protection

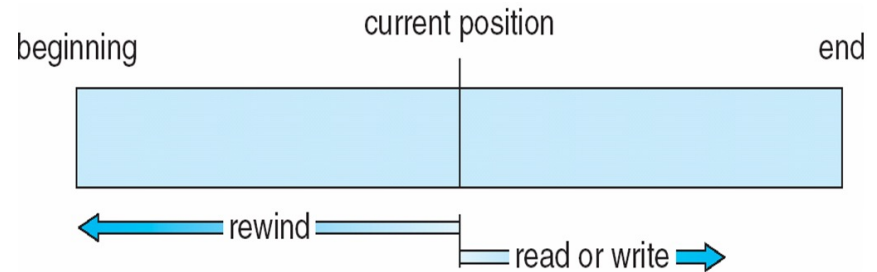


How to we access files and which is the minimum unit we can read/write from secondary storage?

Access Methods

- **Sequential Access**

```
read next  
write next  
reset
```



- **Direct Access** – file is fixed length **logical records**

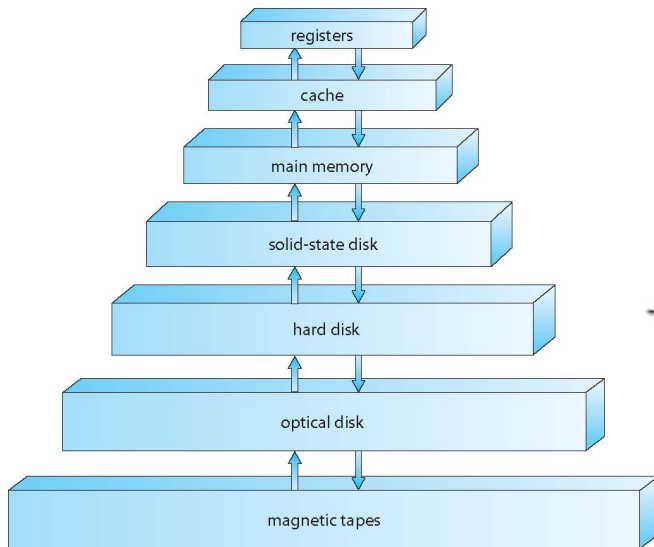
```
read n  
write n  
position to n  
  read next  
  write next  
rewrite n
```

n = **relative block number**
(0 being the beginning of the file)



Why do we read blocks and not just bytes?

Reading / Writing blocks



I need the byte at position X
from the disk!

System call...
Context switch...
Moving to waiting queue...
I/O disk...
Interrupts...
More context switch...

If we pay this price for each
bytes it does not pay off...

Block Size 1/2

- Which is the appropriate size of a block?
- Large block
 - + : Less blocks to read (→ faster reading)
 - : More wasted space if files are small
- Small block
 - + : Less space wasted
 - : More blocks to read (→ slower reading)

Block Size 2/2

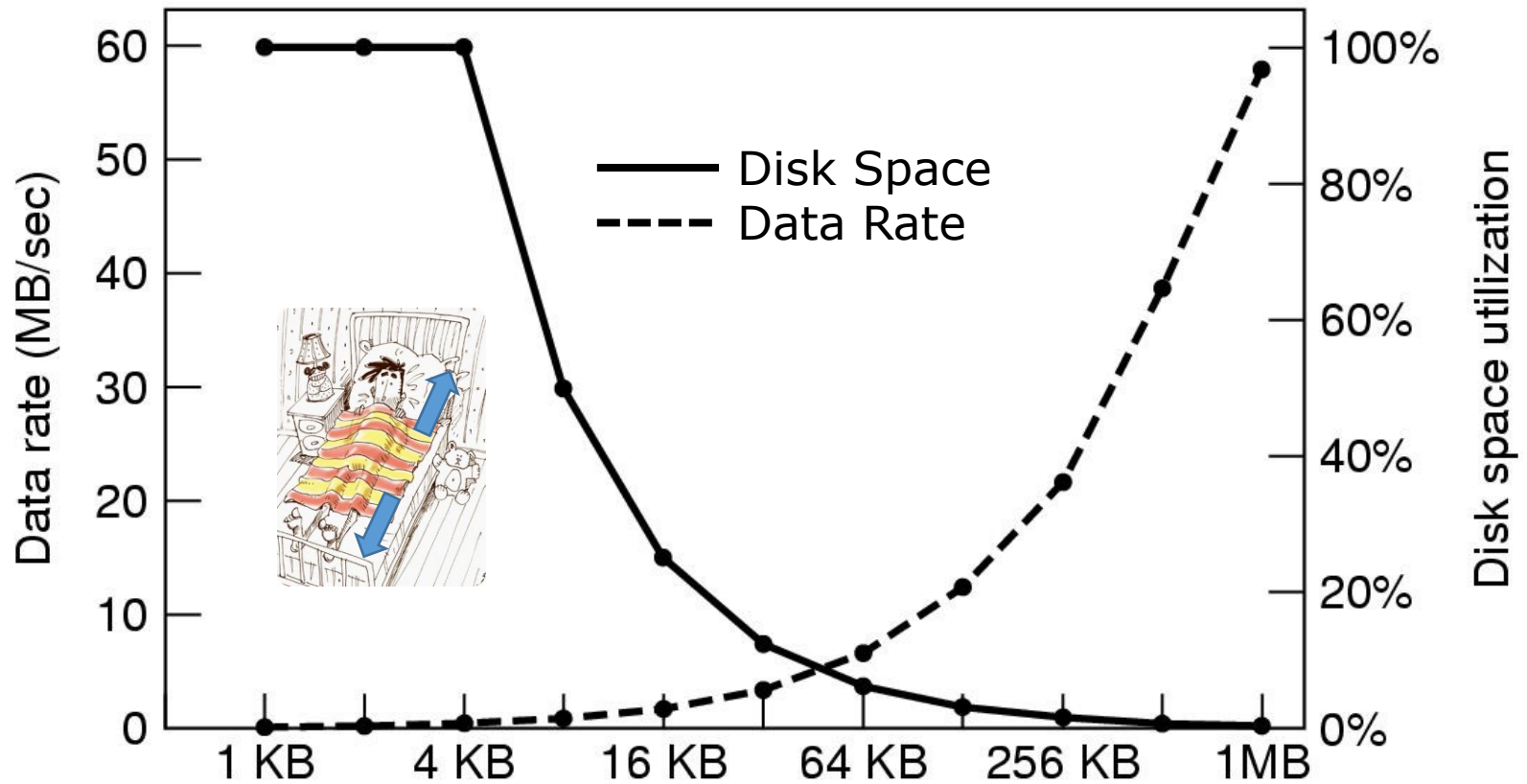


Fig. Tanenbaum, Modern Operating Systems



Why is it so challenging to chose the right block size?

AGENDA

- File concept
- Access methods
- Disk structure
- Directory structure
(Self reading)
- File-system structure
- Allocation methods
- Free-space management
- File sharing and protection



How is the physical disk organized in order to maintain 1 (or more) file systems?

File system layout

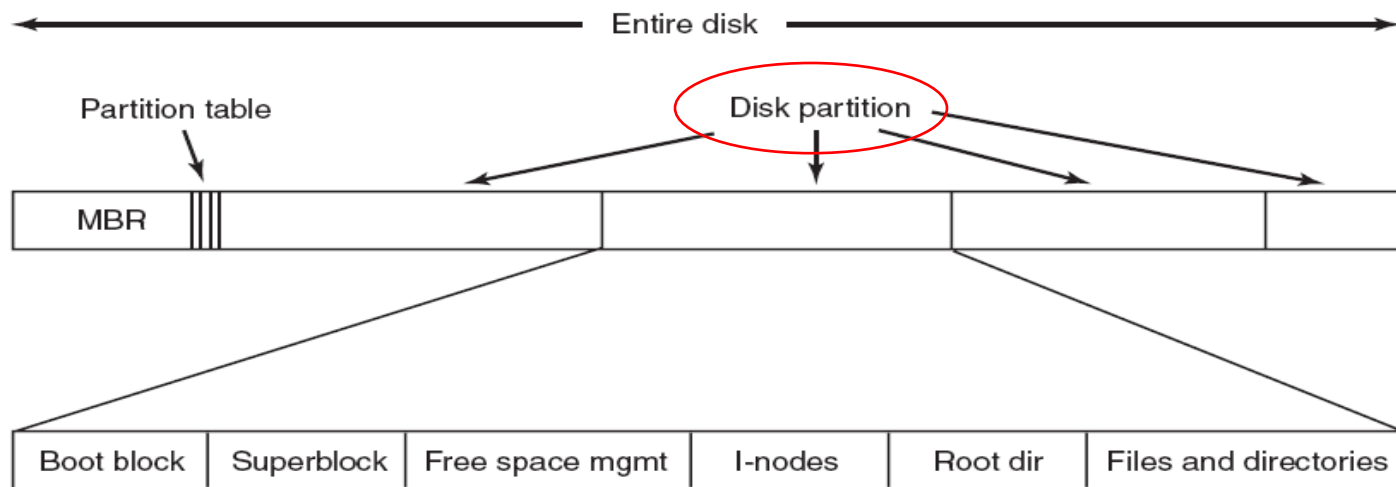


Fig. Tanenbaum, Modern Operating Systems

Disk partitions: 1 (or more) independent file systems

File system layout

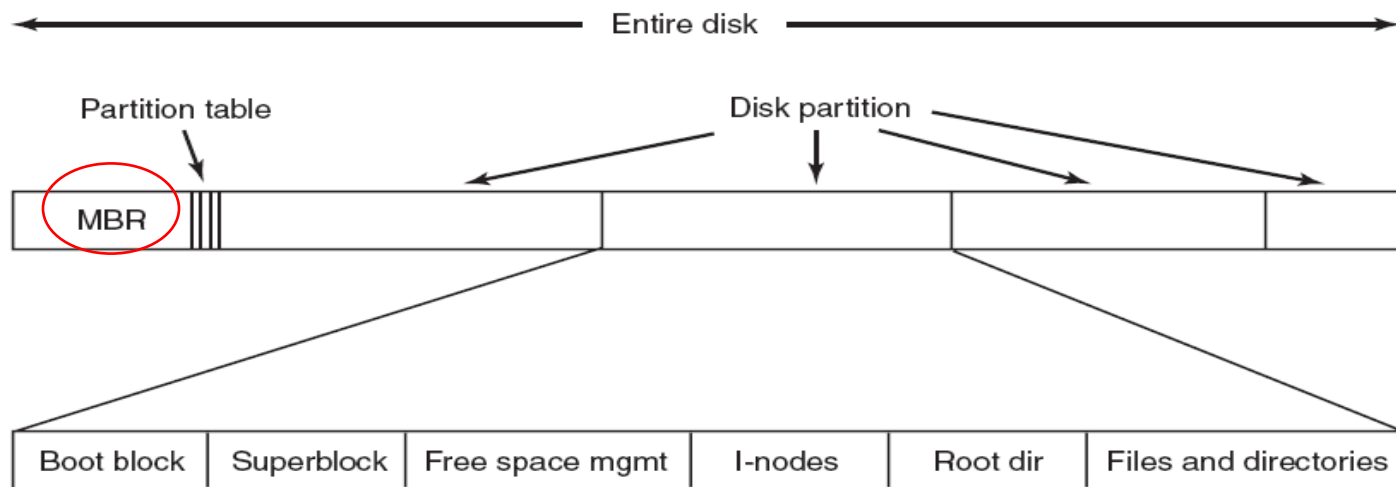


Fig. Tanenbaum, Modern Operating Systems

MBR: Master Boot Record. Starts at sector 0 and is used to boot the computer. At the end, it contains the partition table.

File system layout

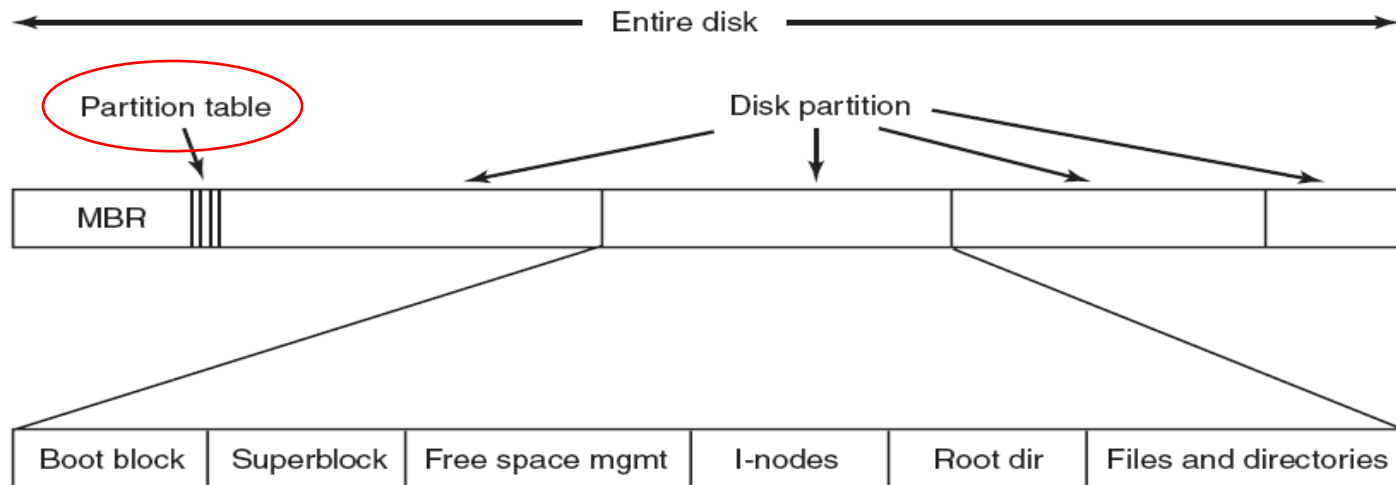


Fig. Tanenbaum, Modern Operating Systems

Partition table: starting and ending addresses of each partition.
One of the partitions is marked as active. Boot sequence:

1. Read the MBR
2. Locate the active partition
3. Read Boot block

File system layout

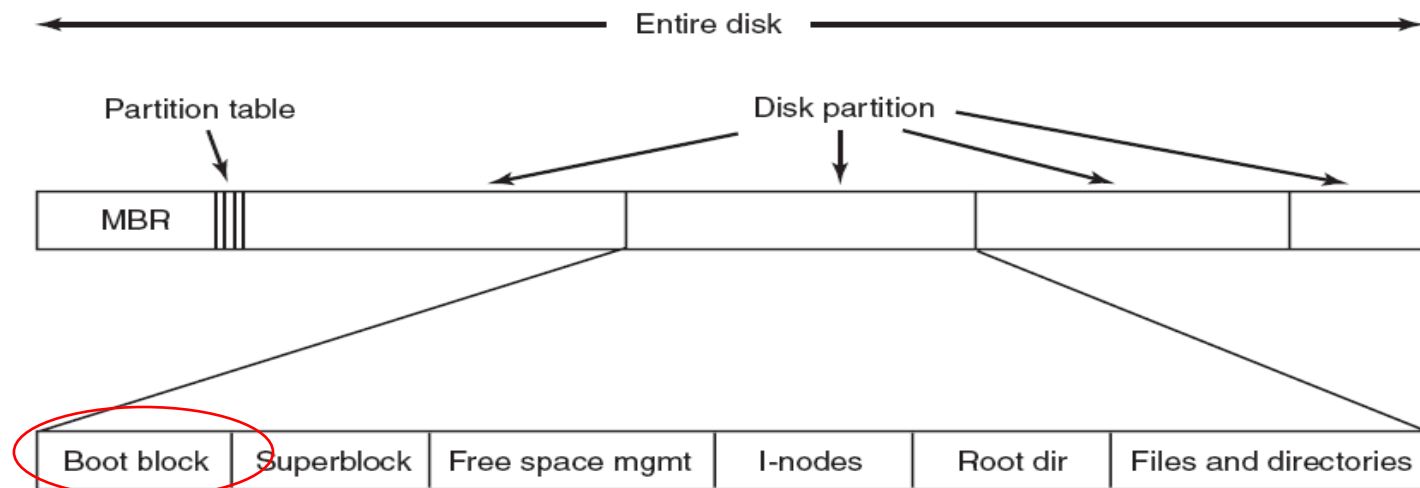


Fig. Tanenbaum, Modern Operating Systems

The Boot block is used to load the operating system of the partition

File system layout

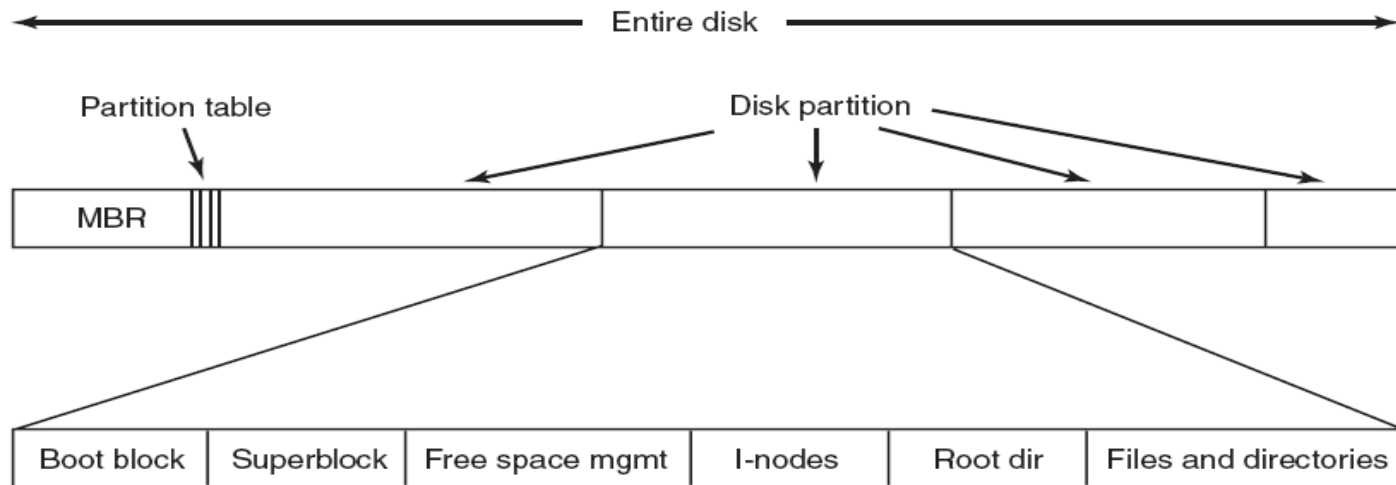


Fig. Tanenbaum, Modern Operating Systems

What there is after the boot block depends on the operating system. Superblock usually contain key information specific to each operating system.

AGENDA

- File concept
- Access methods
- Disk structure
- Directory structure
(Self reading)
- File-system structure
- Allocation methods
- Free-space management
- File sharing and protection



*How can we organize directories
(and files) in a file system?*



What should directories allow us to do?

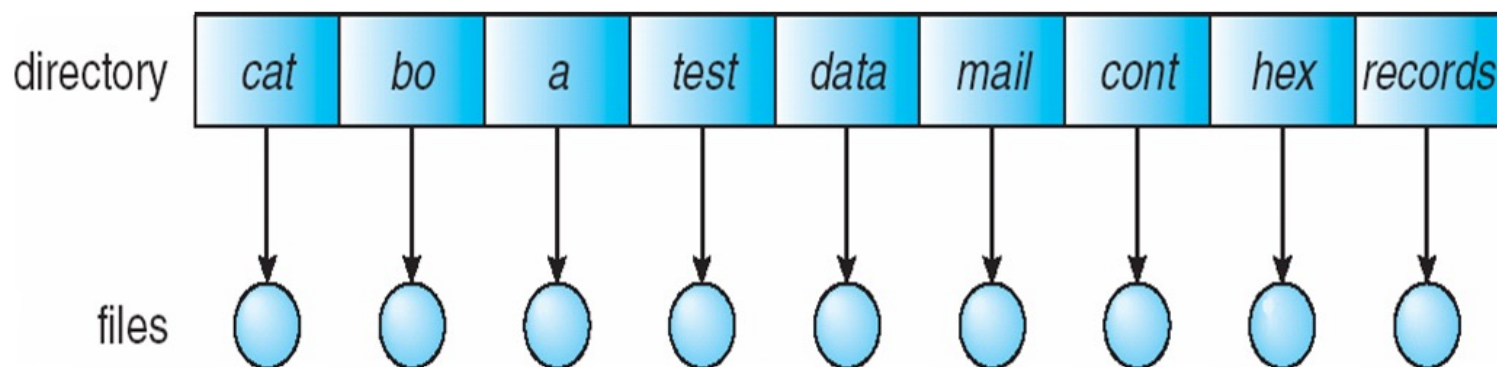
- Search for a file (or search all the files with a given extension)
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system
- Locate a file quickly
- Provide convenient naming (Two users can have same name for different files)
- logical grouping of files by properties, (e.g., all Java programs, all games, ...)



The following is a list of common schemes for the logical structure of a directory

Single-Level Directory

- A single directory for all users



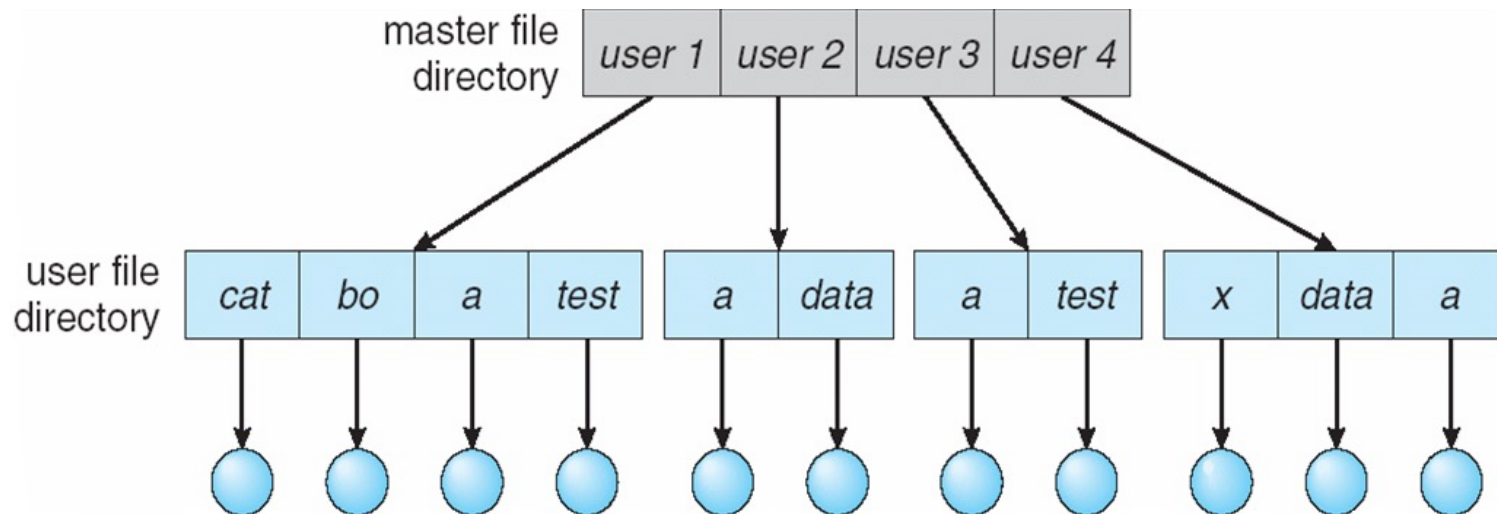
- Simple (good for small devices)
- Easy to locate files



- Different users / same file name?

Two-Level Directory

- Separate directory for each user

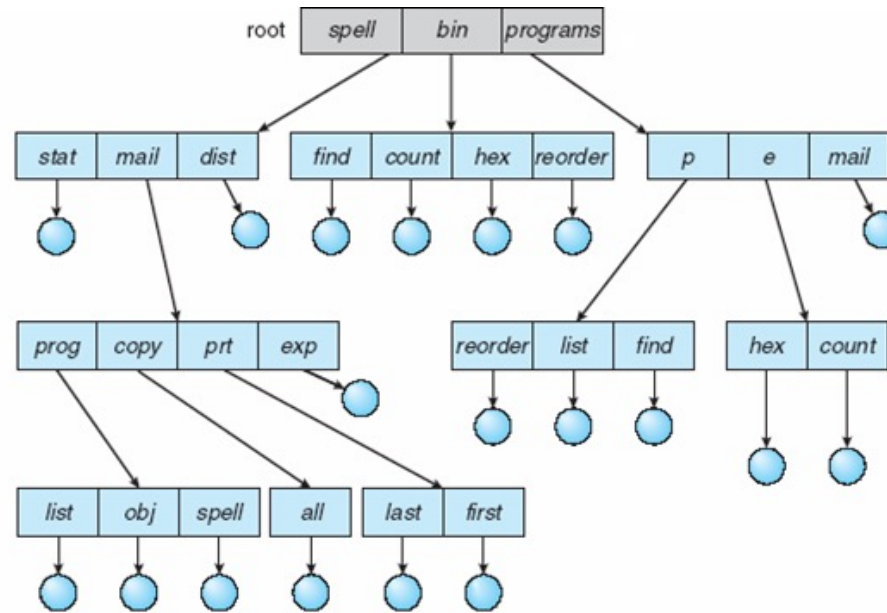


- Same file for different users
- Better searching



- Will require a sort of login
- Not convenient for large number of files
- Will require a path name (the file name is not enough to access it)

Tree-Structured Directories



- Users decide how to organize their data
- Even better searching



- Arbitrary numbers of directories (in practice, each directory becomes a file...)
- Introduces **current directory**
- Introduces more operations on directories

Path names

- Absolute path name
- Relative path name (given working directory)

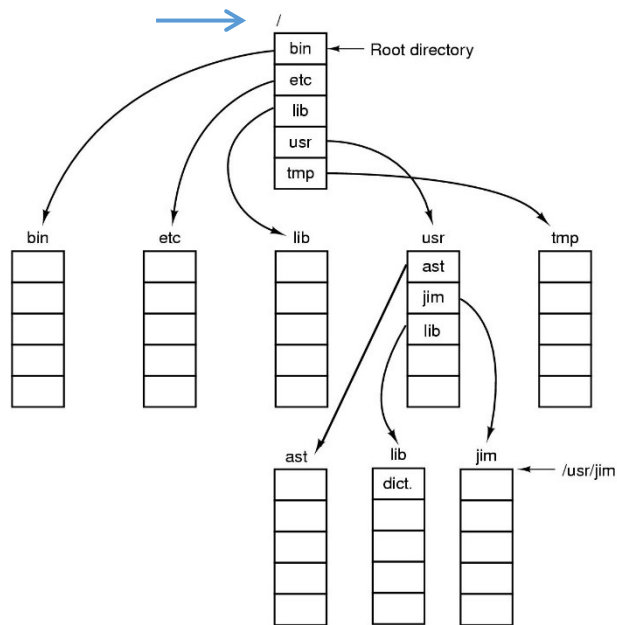


Fig. Tanenbaum, Modern Operating Systems

`cd /bin` ← **absolute path name**
change working directory

`ls .` ← **list files of this folder (.)**

`ls ..` ← **list files of parent folder (.)**

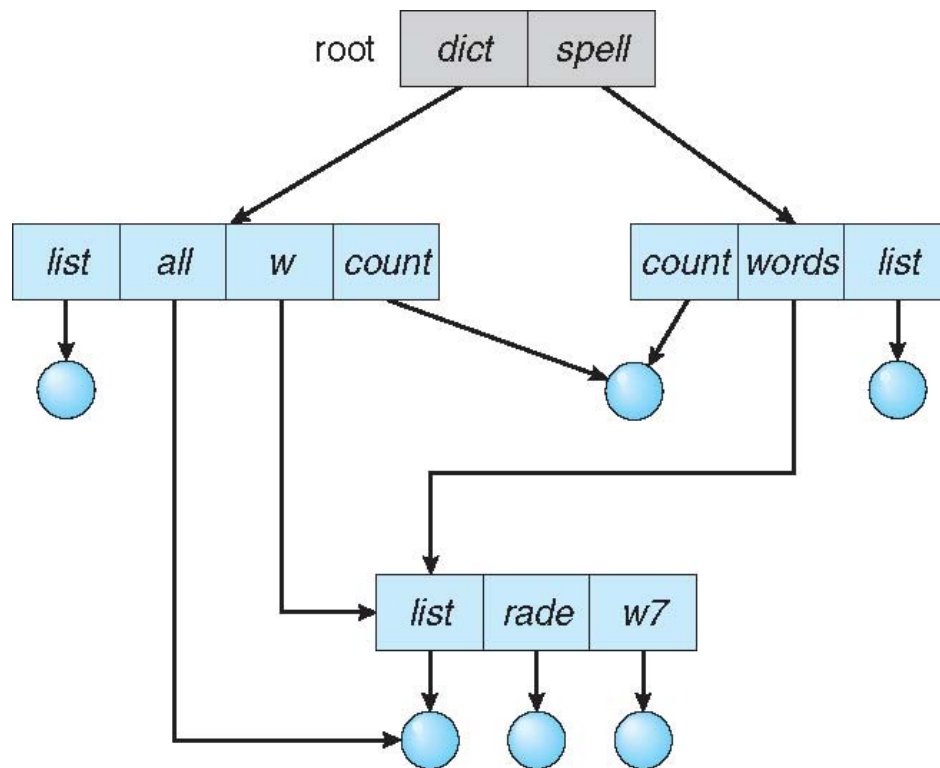
`cd ../lib` ← **relative path name**
change working directory

`cp ../usr/lib/hello.txt /tmp/hello.txt`

← **relative path name**
absolute path name

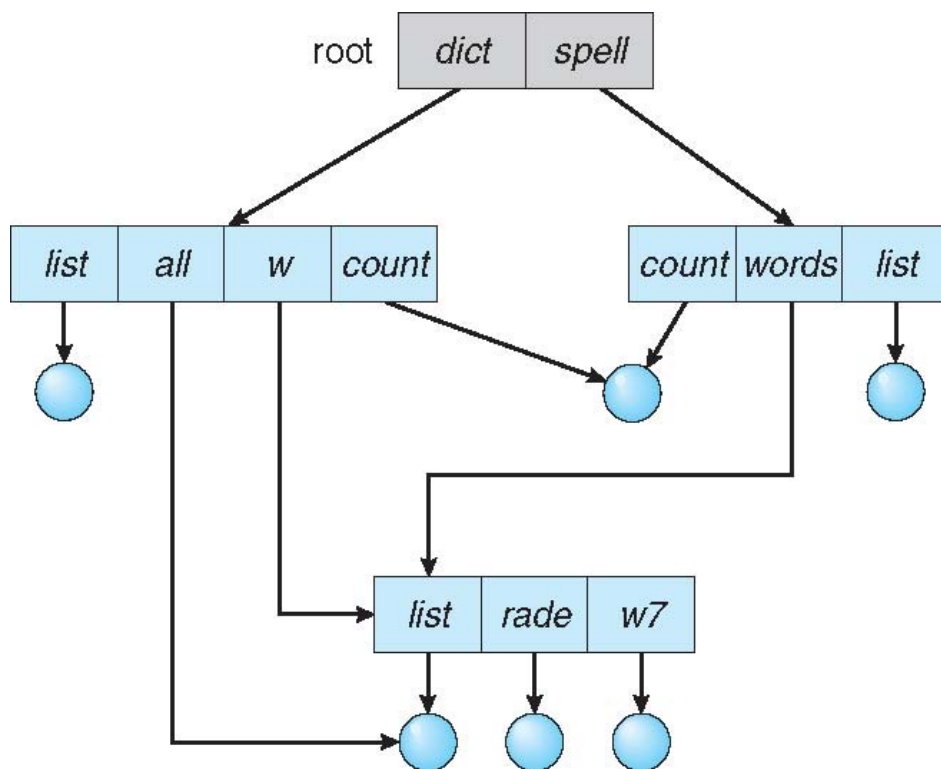
Acyclic-Graph Directories

- Have shared subdirectories and files



In this case, we have a new type of file, the **link** (a pointer to an existing file)

Acyclic-Graph Directories bring trade-offs...



Suppose we are in spell/words and we delete list. Should we delete it from dict/w too?



Let's say we remove it, should we update the links of other references or not?



Let's say we remove it and update the links of other references, how do we know how many references we have?

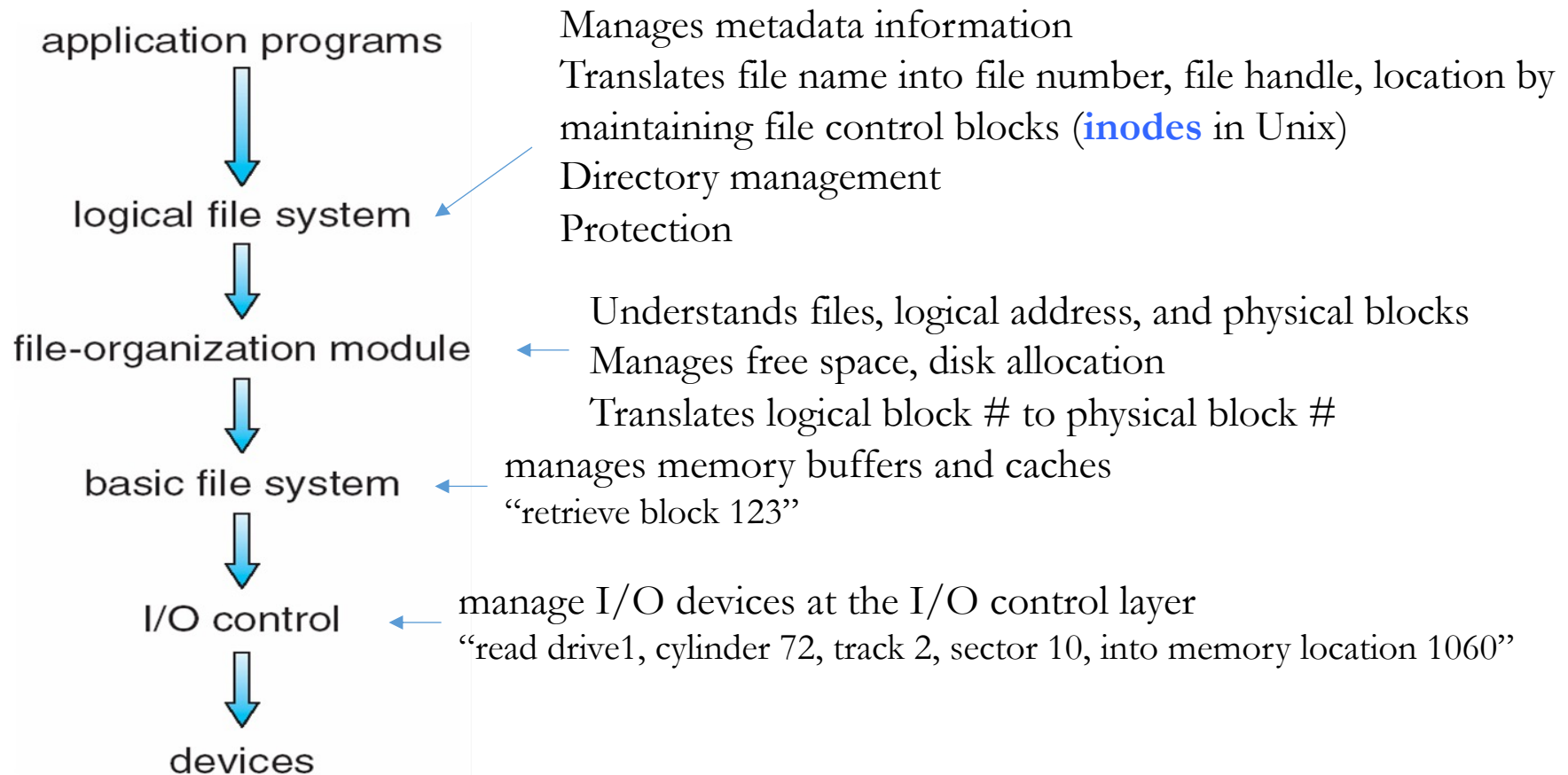
AGENDA

- File concept
- Access methods
- Disk structure
- Directory structure
(Self reading)
- File-system structure
- Allocation methods
- Free-space management
- File sharing and protection



*How is a file system
organized in layers?*

Layered File System



Virtual File Systems

- Suppose a system has multiple partitions with different file systems, how can you integrate all of them?

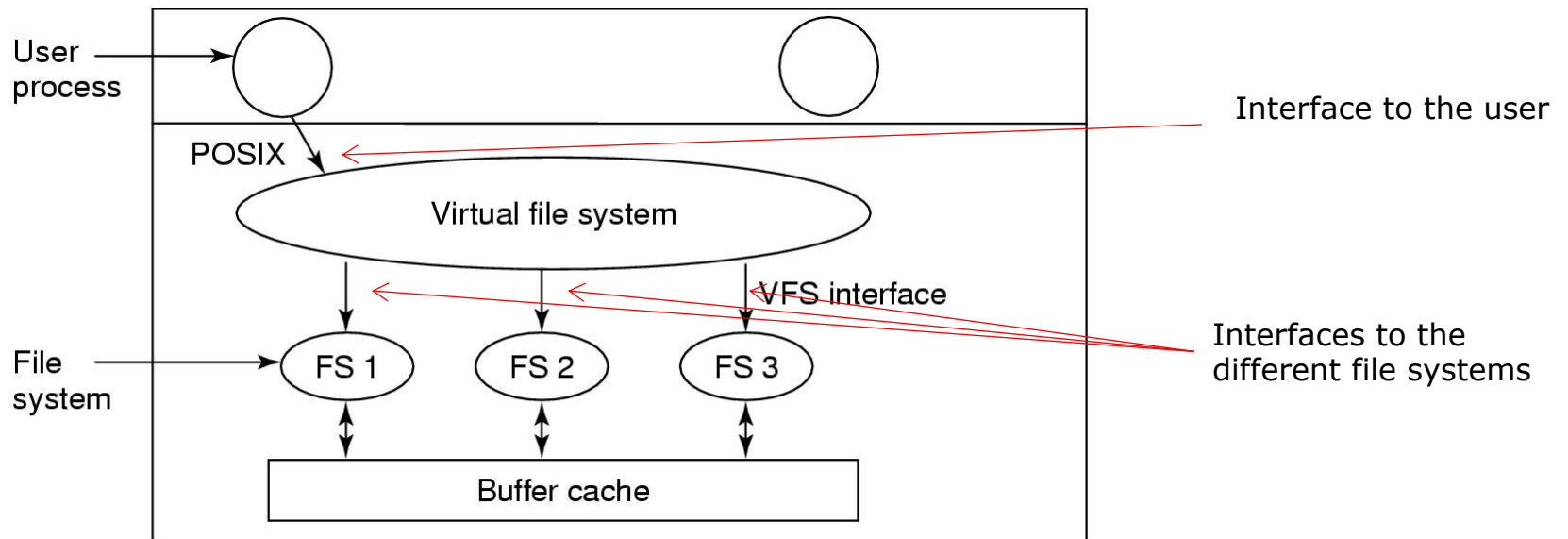
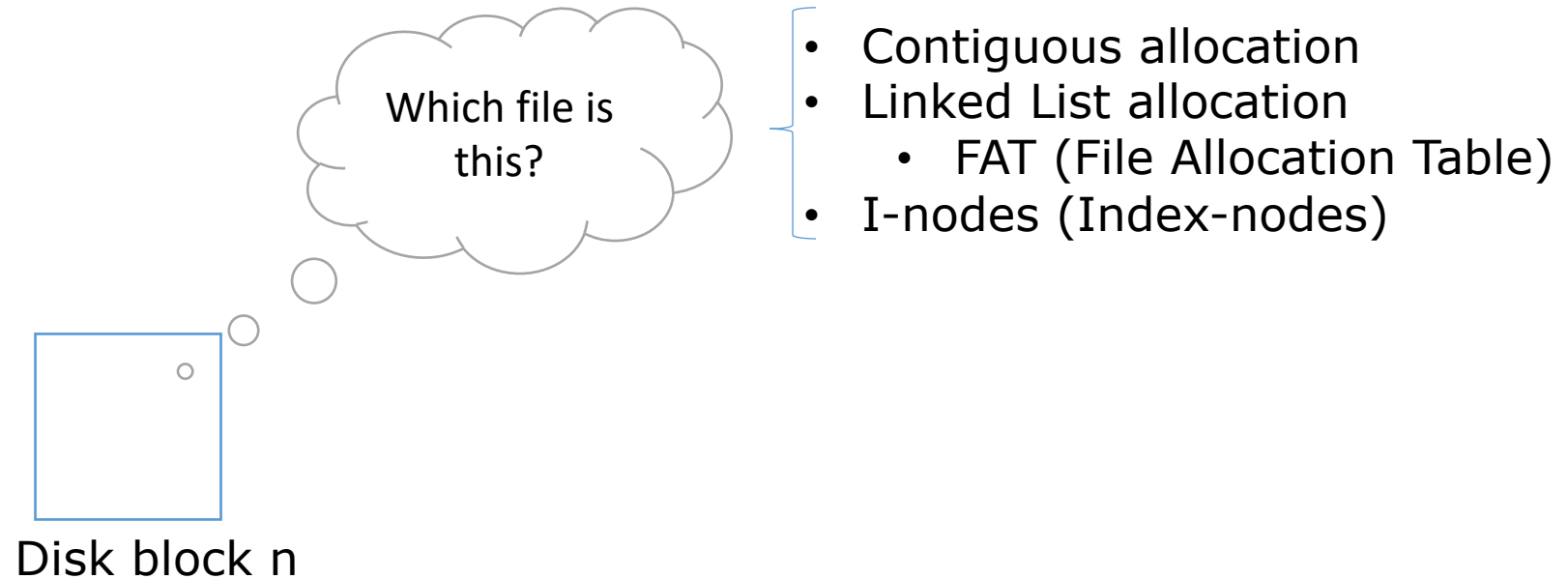


Fig. Tanenbaum, Modern Operating Systems

Implementing Files



AGENDA

- File concept
- Access methods
- Disk structure
- Directory structure
(Self reading)
- File-system structure
- Allocation methods
- Free-space management
- File sharing and protection



How do we know where each file is?

Contiguous allocation

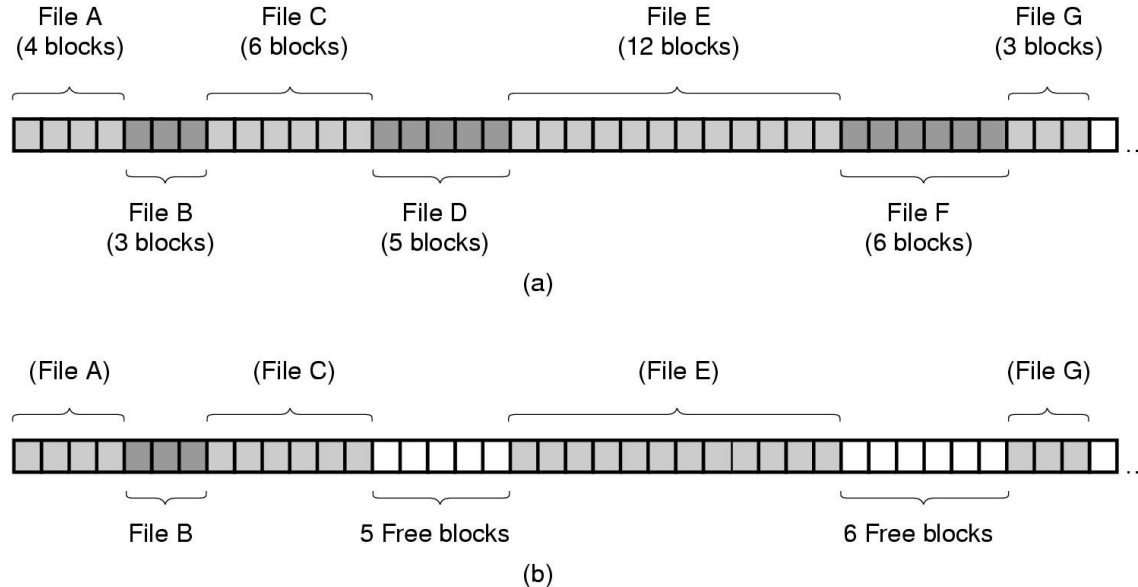


Fig. Tanenbaum, Modern Operating Systems



- Need to maintain 2 numbers per file (first address, number of blocks).
- Read performance (single scan).



- Fragmentation.
- New file size required in advance.

In which support is contiguous allocation actually a good idea?

Linked List allocation

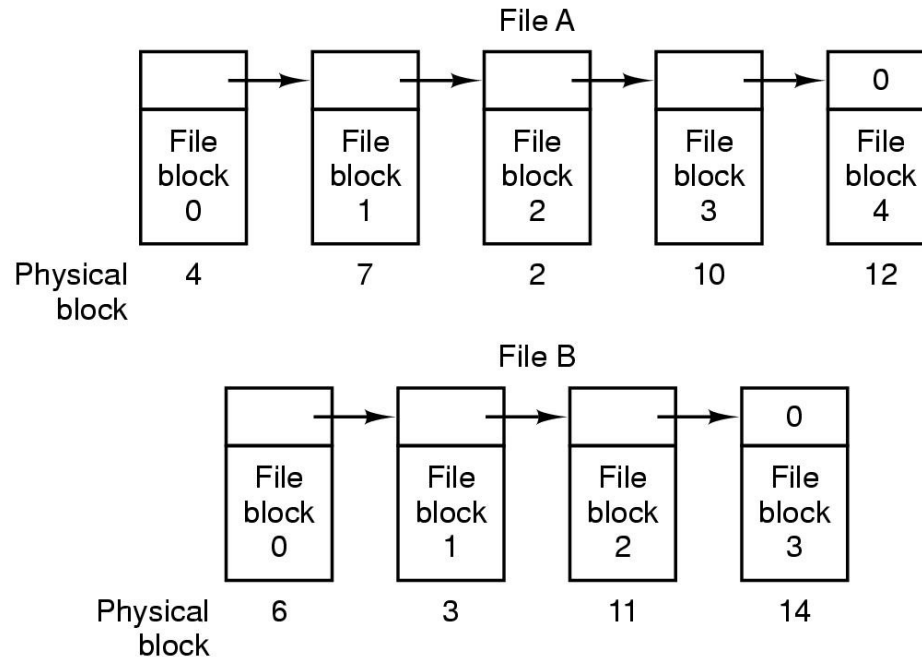


Fig. Tanenbaum, Modern Operating Systems

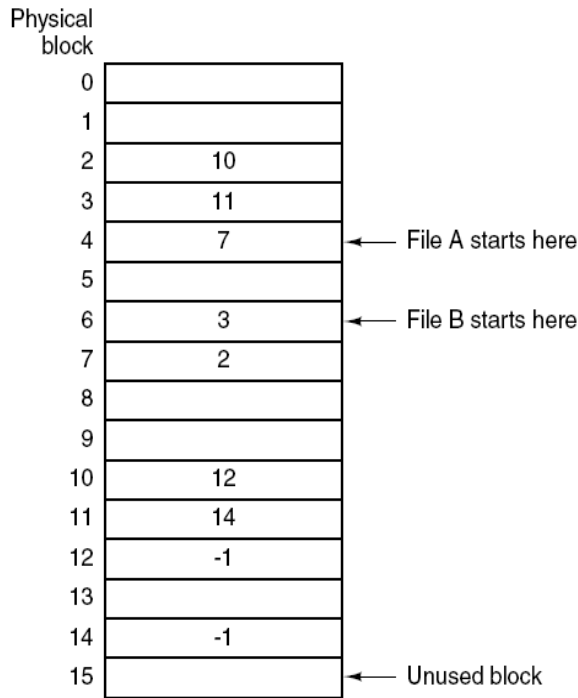


- Need to maintain 1 number per file (first address).



- Get block n?
- Less space available per block.

Linked List allocation using FAT



Disk: 40 GB

Block size: 1 KB

Table entry: 3 B

How much memory?

$40 \times 1024 \times 1024 \text{ blocks} \times 3 \text{ bytes/block}$
120 MB

Fig. Tanenbaum, Modern Operating Systems



- When loaded in memory, disk blocks used only for data.
- Faster random access.



- Need to keep the table in memory.

I-nodes

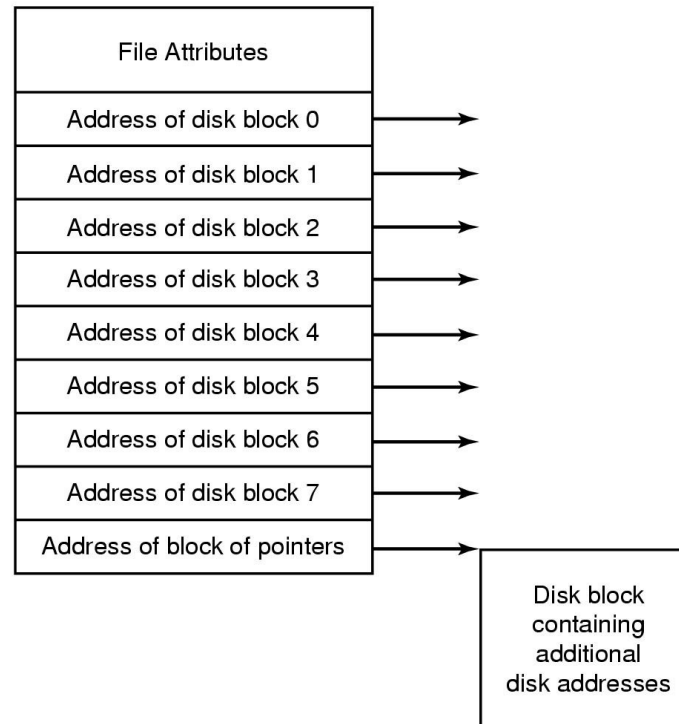


Fig. Tanenbaum, Modern Operating Systems



- If loading I-nodes only for opened files → Less memory needed.
- Memory needed proportional to the maximum number of open files.



- Complex if file grows beyond number of addresses.

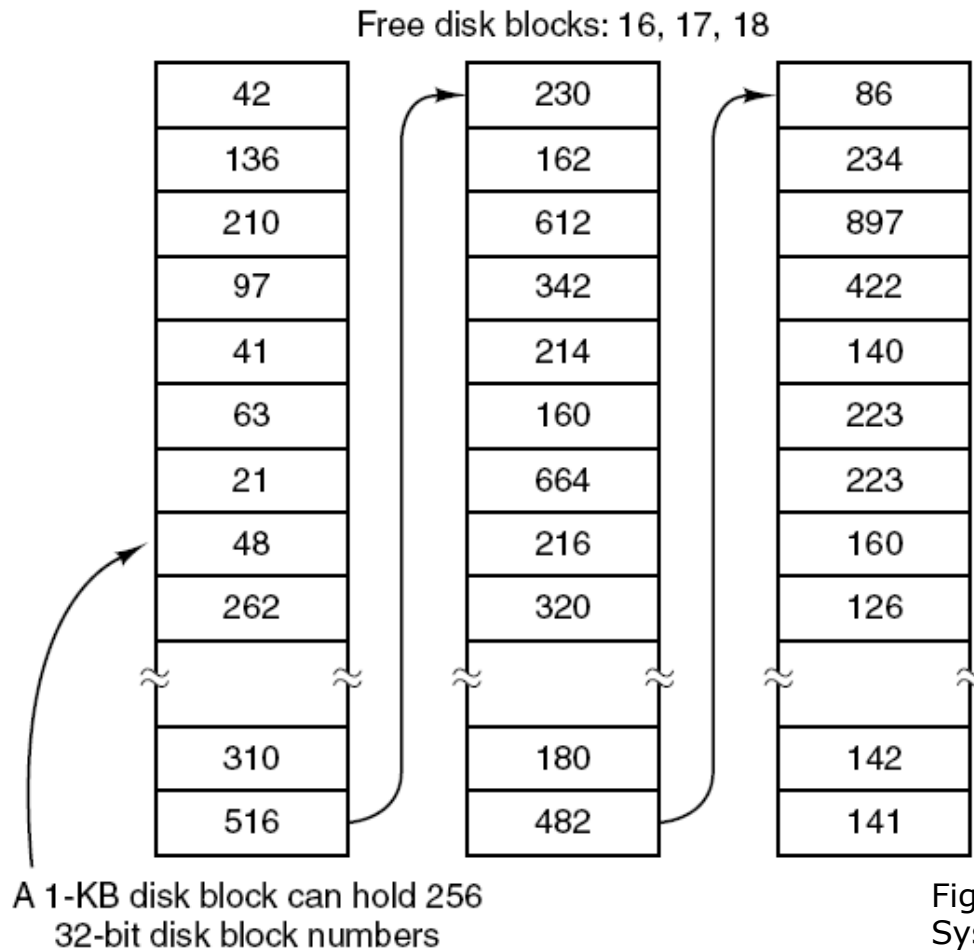
AGENDA

- File concept
- Access methods
- Disk structure
- Directory structure
(Self reading)
- File-system structure
- Allocation methods
- Free-space management
- File sharing and protection



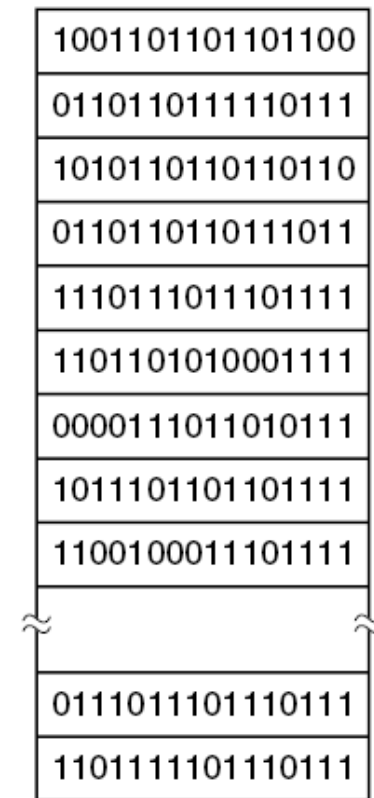
*How do we know what portions
of a file systems are free?*

Keeping track of free blocks



Linked List

Fig. Tanenbaum, Modern Operating Systems



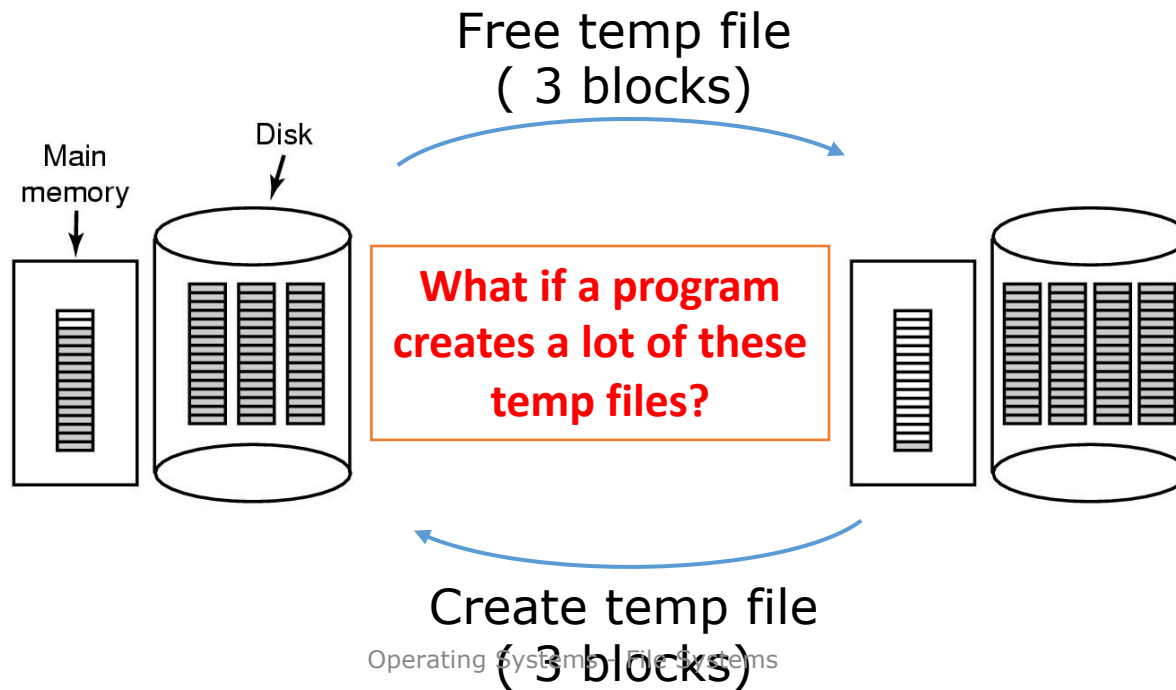
bitmap

Keeping track of free blocks

- Designing the right way of keeping free blocks is not trivial...

Linked List. Keep only 1 block of pointers to free blocks in memory, others are stored

- Run out of free blocks? Read the following one from disk
- New free blocks? Add them to block in memory, if full → store it.



AGENDA

- File concept
- Access methods
- Disk structure
- Directory structure
(Self reading)
- File-system structure
- Allocation methods
- Free-space management
- File sharing and protection



How do we share and protect files among different users?

Protection

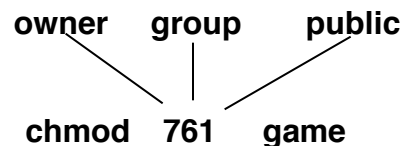
- File owner/creator should be able to control:
 - what can be done
 - by whom
- Types of access
 - **Read**
 - **Write**
 - **Execute**
 - **Append**
 - **Delete**
 - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

a) owner access	7	⇒	RWX 1 1 1
b) group access	6	⇒	RWX 1 1 0
c) public access	1	⇒	RWX 0 0 1

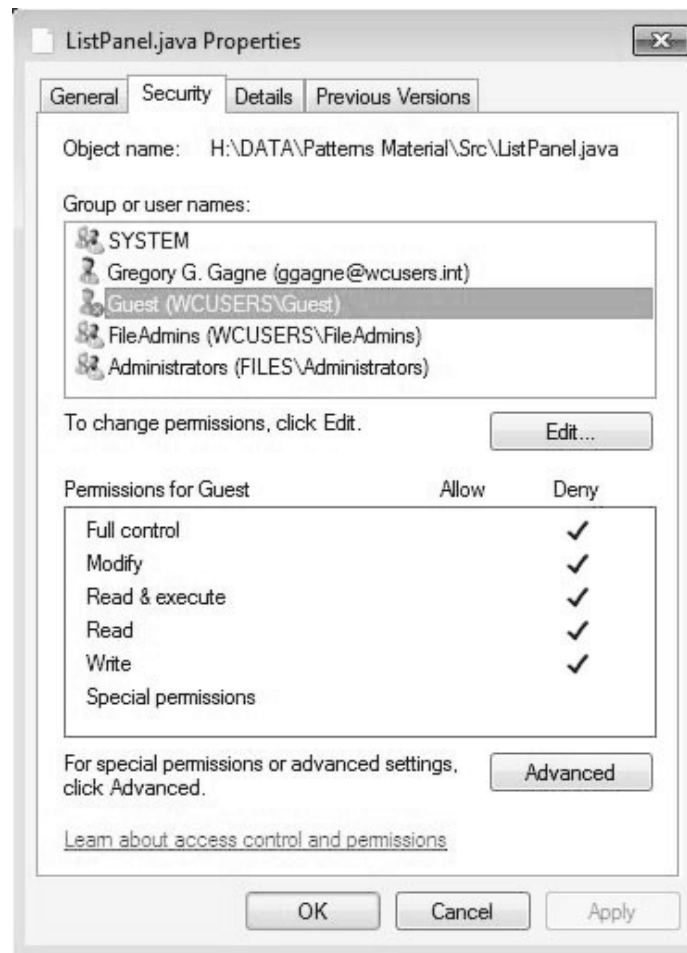
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

chgrp **G** **game**

Windows 7 Access-Control List Management



A Sample UNIX Directory Listing

-rw-rw-r--	1	pbg	staff	31200	Sep 3 08:30	intro.ps
drwx-----	5	pbg	staff	512	Jul 8 09:33	private/
drwxrwxr-x	2	pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2	pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1	pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1	pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4	pbg	faculty	512	Jul 31 10:31	lib/
drwx-----	3	pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3	pbg	staff	512	Jul 8 09:35	test/