CIS 5512 - Operating Systems
File Systems

Professor Qiang Zeng

Fall 2017

Previous class…

- I/O subsystem: hardware aspect
  - Terms: controller, bus, port
  - Addressing: port-mapped IO and memory-mapped IO
- I/O subsystem: software aspect
  - Device drivers: polling, interrupt-driven, and DMA
  - API styles: blocking, non-blocking, & asynchronous
- I/O subsystem: performance aspect
  - Caching, Buffering and Spooling
  - I/O scheduling

Some slides are courtesy of Dr. Abraham Silberschatz and Dr. Thomas Anderson

Previous class…

Compare “buffering” and “caching”

- Data in cache is repetitively accessed, while data in buffer is consumed like a stream
- Buffering is mainly used to cope with the speed mismatch between producers and consumers (and other purposes), while caching is used to speed up the access

Previous class…

When to use blocking I/O and when to use non-blocking I/O?

- If the current I/O is the only thing the current thread/ process should do, then use blocking I/O; it is easier
- If the current thread/process needs to receive whatever data it can receive and consume it ASAP, or send whatever data it can send out and get back to continue producing (e.g., voice call), or the process has to handle multiple I/O streams simultaneously, then use non-blocking I/O; it is a little more complex

Previous class…

Advantage and disadvantage of DMA

- The data passing does not go through CPU, so it saves CPU cycles when passing a large amount of data
- But it is not worthwhile if you only need to pass a small amount of data each time, because setting up a DMA operation is costly

Previous class…

What is the benefit of having the drivers to expose the same set of interfaces to the kernel?

Modularized and layered design, so that the core part of the kernel can be coded according to those small set of interfaces, no matter how diverse and complex the devices are

Outline

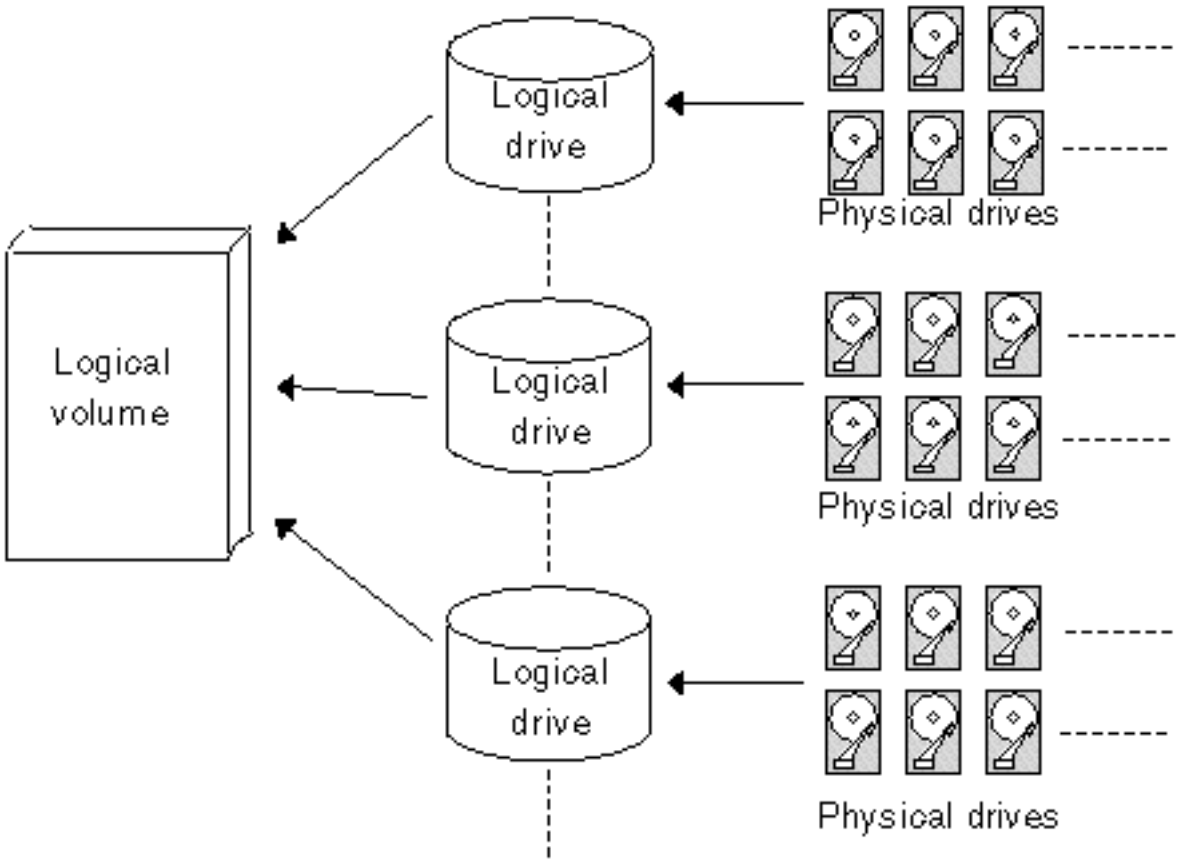
- File System concepts
  - Mounting
  - File and directory
  - Hard link and soft link
- File system design
  - FAT32
  - UFS

Some slides are courtesy of Dr. Thomas Anderson

Volume and Partition

- Volume: a collection of physical storage resources that form a logical storage device
  - One volume contains one file system

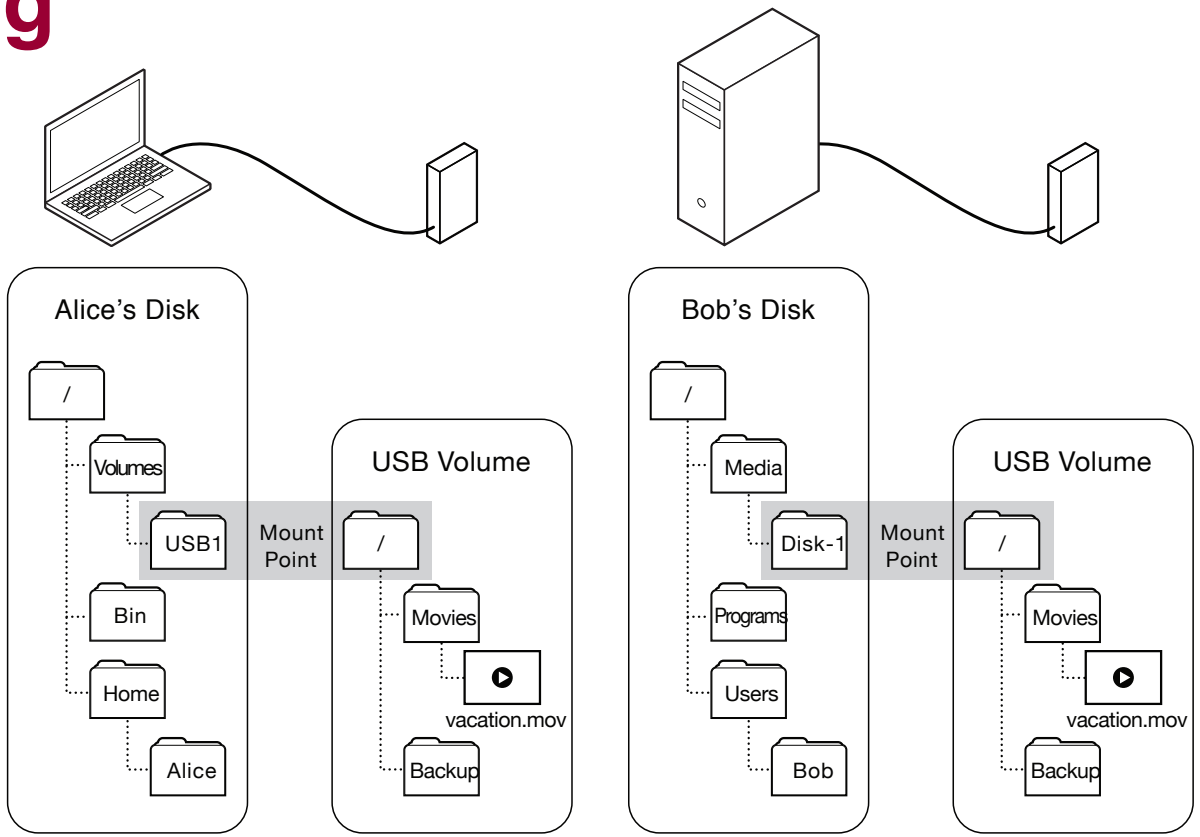
Volume


Logical volume
Logical drive
Logical drive
Logical drive
Physical drives
Physical drives
Physical drives

Mount

- Each volume contains a file system
- Mounting volumes is to organize the multiple file systems into a single logical hierarchy
  - E.g., assuming your usb disk contains one volume, when you insert your usb stick, the system mounts the volume to the existing file system hierarchy
- Mount point: the path where a volume is mounted to
  - "`mount device dir`" tells the kernel to attach the file system found on device at the directory *dir*

Mounting

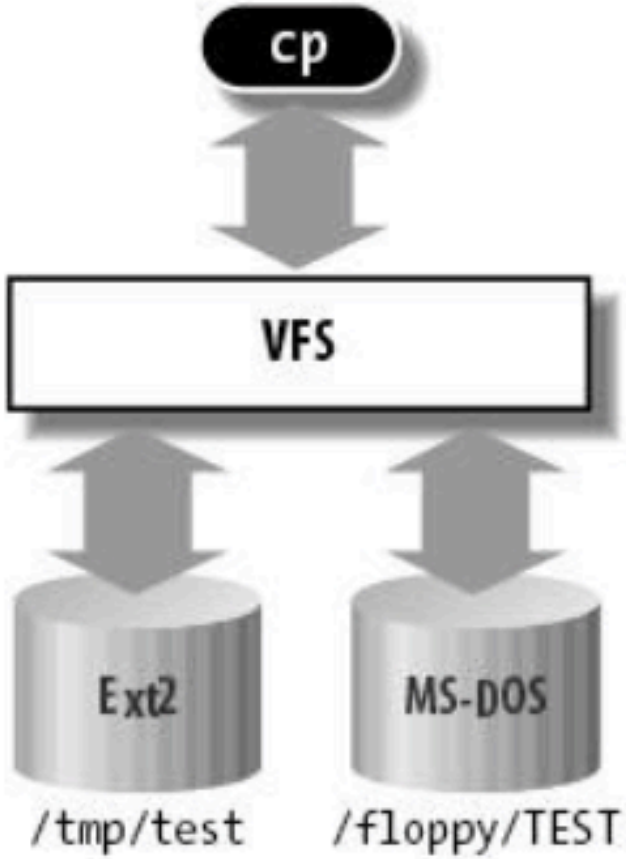


- How to access the movie?
  - Alice: /Volumes/usb1/Movies/vacation.mov
  - Bob: /Media/disk-1/Movies/vacation.mov
- Where does the original content in /Volume/usb1 and /Media/Disk-1 go?
  - Temporarily hidden

Mounting associates file system with drivers --- very important idea

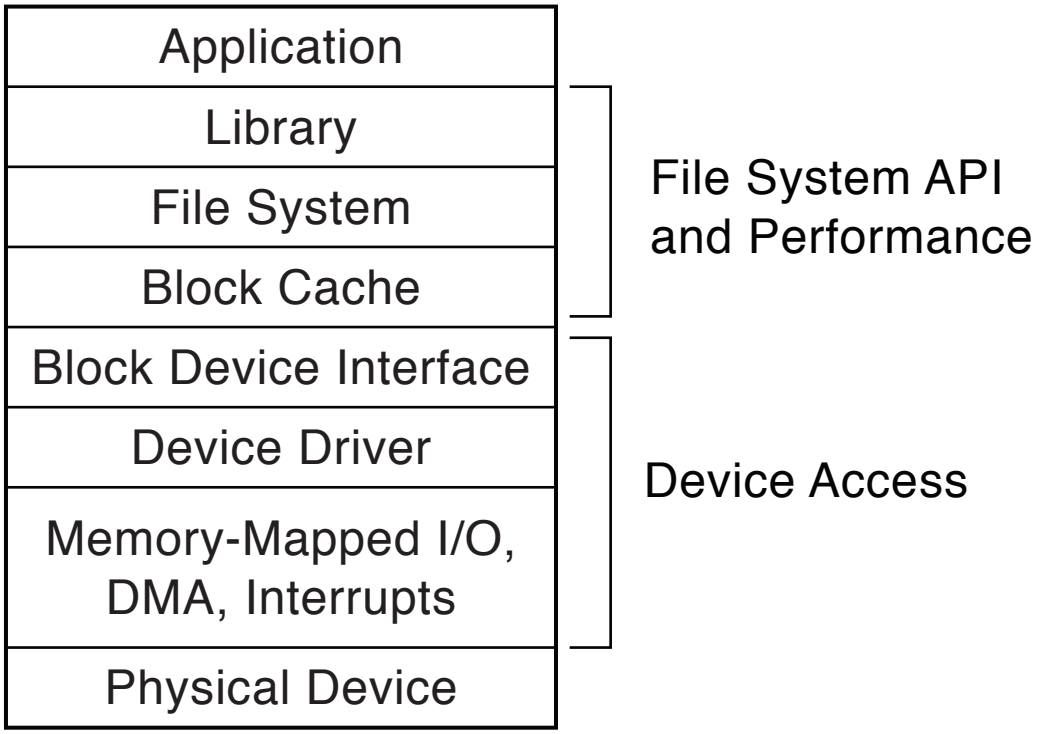
- Once a file system, which is on some device, is mounted, the kernel associates the file system *F* with the driver *D* for the device
- So that when you read/write/seek a file in *F*, it will route the device operation to functions in *D*

Users usually don't need to know the file system types and device drivers




```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
       O_WRONLY|O_CREAT|O_TRUNC, 0600);
do {
    i = read(inf, buf, 4096);
    write(outf, buf, i);
} while (i);
close(outf);
close(inf);
```

The transparency benefits from layering



open() vs fopen()

- `open()`, `read()`, `write()`, and `close()` are system calls
- `fopen()`, `fread()`, `fwrite()`, and `fclose()` are stdio lib calls
  - The library includes buffers to aggregate a program's small reads and writes into system calls that access large blocks
  - If a program uses the library function `fread()` to read 1 byte of data, the library may use read() system call to read in large block of data, e.g., 4KB, into a user space buffer, so that when the process calls `fread()` again to read another byte, it reads from the buffer instead of issuing another system call

Block

- In the view of file systems, disk space is divided into blocks
- A disk block consists of a fixed number of consecutive disk sectors
  - E.g., sector size = 512B, block size = 4KB
- A file is stored in blocks
  - E.g., a file may be stored in blocks 0x0A35, 0xB98C, 0xD532

File

- File: a named collection of data
  - E.g., /home/qiang/file-systems.pptx
- A file consists of two parts
  - Metadata
  - Data
- Metadata: data that describes other data
  - File type
  - File size
  - Creation/access/modification time
  - Owner and group a file belongs to
  - Access permissions

Metadata of a Linux/Unix file

- `ls -l`

```
-rwxrw-r--    10    root   root 2048    Jan 13 07:11 afile.exe
?UUUGGGOOOS   00  UUUUUU GGGGGG ####    ^-- date stamp and file name are obvious ;-)
^ ^  ^  ^ ^    ^       ^      ^    ^
| |  |  | |    |       |      |    \--- File Size
| |  |  | |    |       |      \-------- Group Name (for example, Users, Administrators, etc)
| |  |  | |    |       \--------------- Owner Acct
| |  |  | |    \----------------------- Link count (what constitutes a "link" here varies)
| |  |  | \--------------------------- Alternative Access (blank means none defined, anythir
| \--\--\----------------------------- Read, Write and Special access modes for [U]ser, [G]r
\------------------------------------- File type flag
```

File type in linux/Unix (the first column in `ls -l`)

```
`-'
      regular file

`b'
      block special file

`c'
      character special file

`C'
      high performance ("contiguous data") file

`d'
      directory

`D'
      door (Solaris 2.5 and up)

`l'
      symbolic link

`M'
      off-line ("migrated") file (Cray DMF)

`n'
      network special file (HP-UX)

`p'
      FIFO (named pipe)

`P'
      port (Solaris 10 and up)

`s'
      socket
```

Metadata is a generic concept

- Name the possible metadata of a song
  - Singer
  - Author
  - Style
  - Creation time
  - …
- How does “Pandora” infer which songs you like?
  - Use the metadata (and other info) as “features”
  - The features will be used in a set of machine learning and data mining algorithms to infer your likes

Directory

- Directory maps the names of files the directory contains to their locations
- Directory is a special type of "file"; like a file, a directory also consists of
  - Metadata:
    - Size, owner, date, etc.; same as ordinary files
  - Data:
    - An array of entries, each of which is a mapping between file name and the index of the first block of the file
    - Logically, a file is a linked list of blocks; once you locate the first block, you have located the file
    - Let's call "the index of the first block of a file" as a "file number", which describes the location of a file

Question

- When you rename a file, do you modify the file?
  - No, you actually modify the containing directory
- File *a.txt* is in directory *A*; assume you don't have write permission with *a.txt* but do have write permission with directory *A*, can you rename *a.txt*?
  - Yes
  - Because you are modifying the containing directory

Hard link and soft link

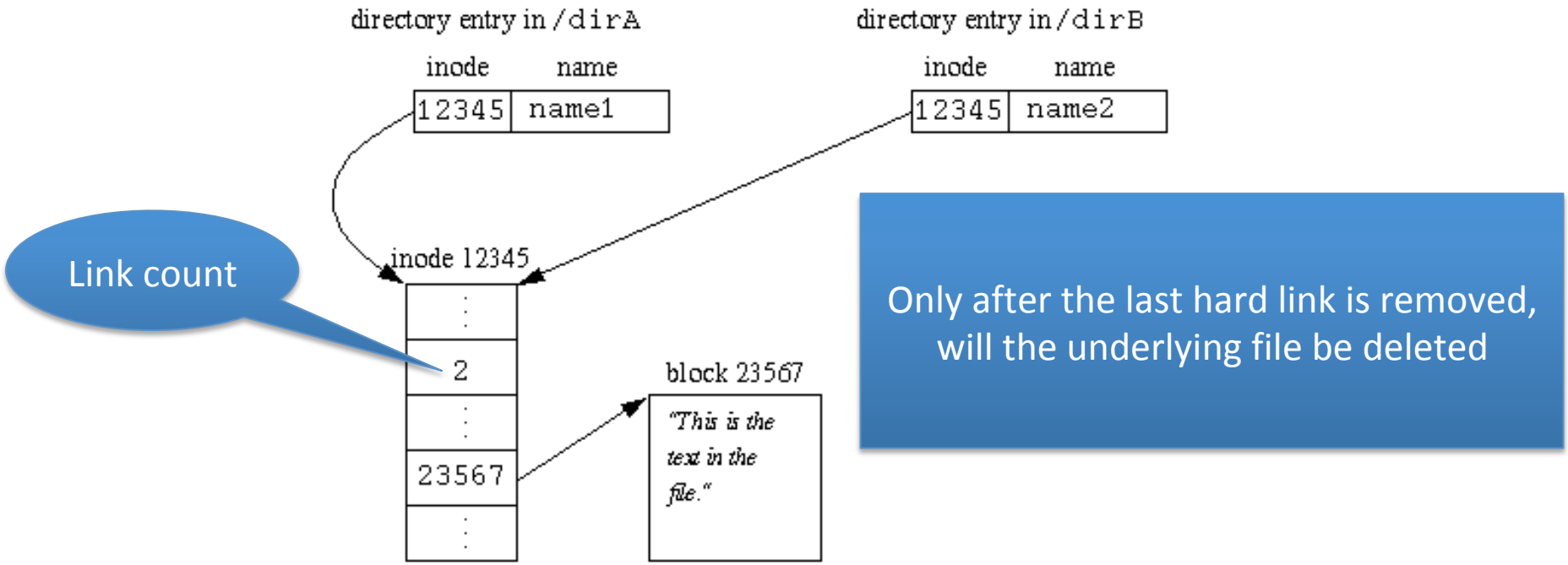
- A hard link: a mapping from a file name to some file's data
  - `ln original.txt hard.txt`
- A soft/symbolic link: a mapping from a file name to another file name; whenever you refer to a symbolic link, the system will automatically translate it to the target name
  - `ln -s original.txt soft.txt`
- Question: if a program execution opens a file with a fixed name, but you want to the program execution to refer to your another file, what should you do?

Hard link and soft (symbolic) link

```
qiang@ubuntu:~/tmp$ touch original.txt
qiang@ubuntu:~/tmp$ ln original.txt hard.txt
qiang@ubuntu:~/tmp$ ln -s original.txt soft.txt
qiang@ubuntu:~/tmp$ ls -al original.txt soft.txt hard.txt
-rw-r--r-- 2 qiang qiang  0 2015-11-19 13:48 hard.txt
-rw-r--r-- 2 qiang qiang  0 2015-11-19 13:48 original.txt
lrwxrwxrwx 1 qiang qiang 12 2015-11-19 13:48 soft.txt -> original.txt
```

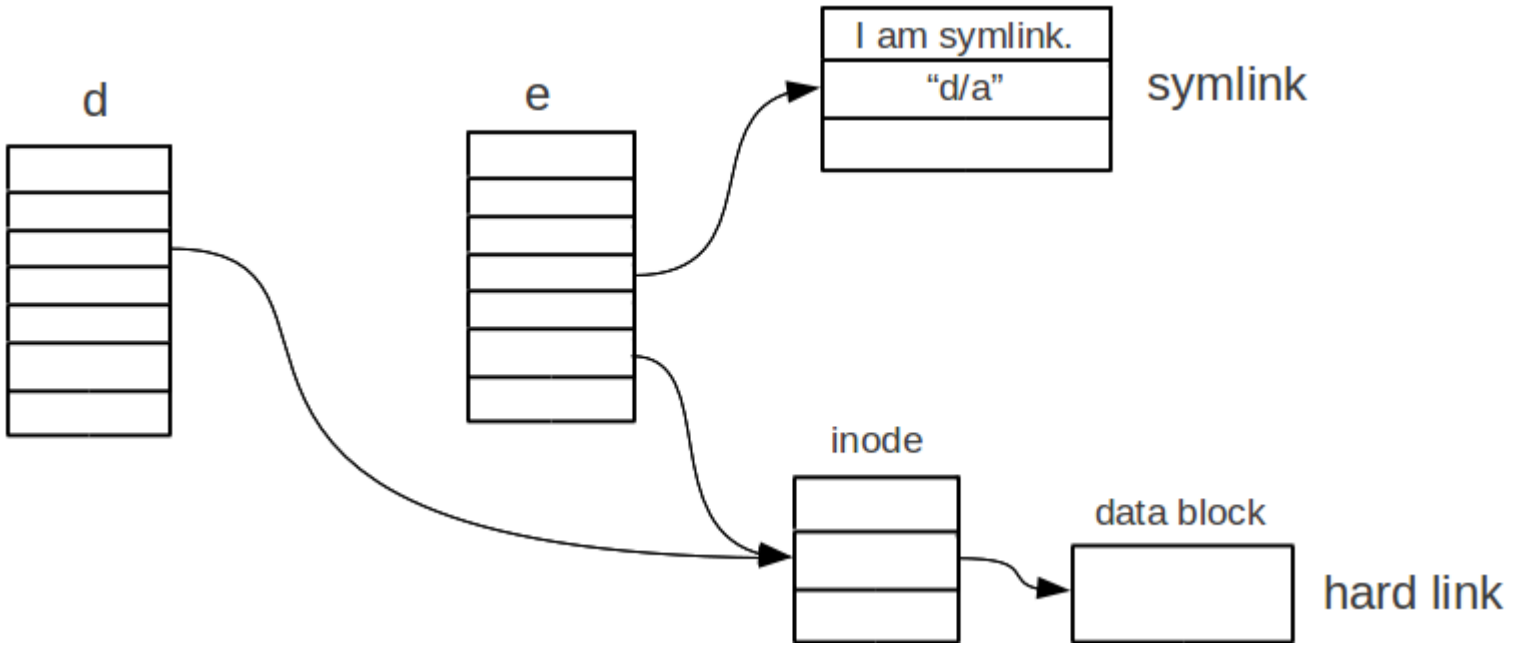
- We have created a file *original.txt*, and a hard link named *hard.txt*, and a symbolic link named *soft.txt*
- Can you distinguish *original.txt* and *soft.txt*?
  - Certainly
- Can you distinguish *original.txt* and *hard.txt*?
  - Hmmm…

Truth about hard link



- inode block: this is the *formal* name used to refer to "the first block of a file" in Linux;
- *"Hey, you! Don't call me a hard link; we are equal"*
- Hard link: another directory entry for a given file

Symbolic link vs Hard link



- The inode of a symbolic file contains:
  - A flag saying that "I am symbolic link"
  - A file name of the target file
- Very important for software upgrade
  - After upgrade, you just redirect the symbolic link to the new version

Question

- If you modify a file through a hard link, will the modification time of another hard link of the same file be updated as well?
  - Yes
  - They point to the same inode block, which stores the modification time (and other metadata)
  - Hard links of a file share the same piece of “metadata and data” of the file
  - The only difference is the names

Question

- Most systems do not allow creating a hard link for a **directory**. Why?
  - Avoiding it ensures that the directory structures form a directed acyclic graph (DAG). The benefits:
  - It simplifies the directory traversal
    - No circles during traversal
    - Consider the implementation of “du” (disk use by a directory)
  - It simplifies the deletion logic
    - If the link count of a file reaches zero, you can delete the file and reclaim the space
    - If the directories form a circle, even when the hard link count of files do not reach zero, they should be garbage collected; but now you need a dedicated garbage collector to detect such circles

File System Design

- Need to consider data structures for
  - Directories: file name -> location of file
  - File: locations of the disk blocks used for each file
  - Tracking the list of free disk blocks
- How to design and organize these data structures is the task of file system design

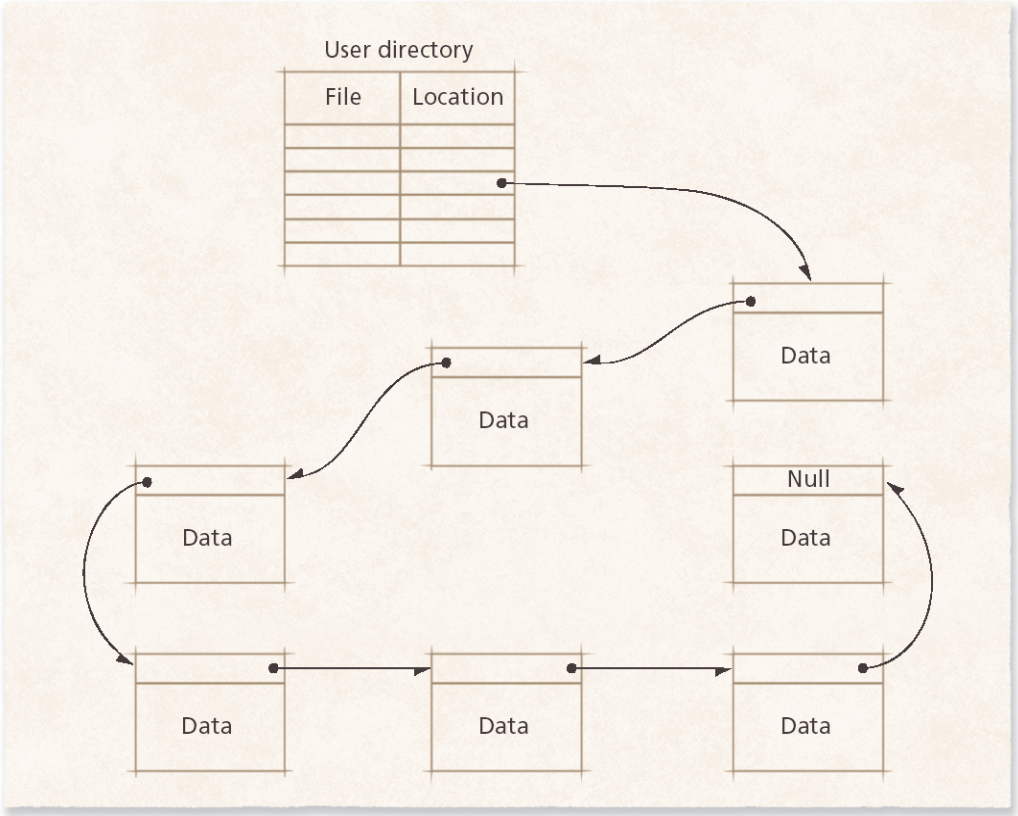
Extra considerations

- Efficiency
  - How to quickly map the offset of a file (e.g., the 1000th byte) to the location of the containing data block?
  - How to quickly insert a byte into a file?
  - What block size do we use?
  - How to decrease fragmentation?
  - How do we preserve spatial locality?
- Reliability
  - What if machine crashes in middle of a file system op?

File System Design

- Contiguous file allocation
  - Each file occupies a contiguous area in disk
  - Advantage: very good locality and high performance
  - What are the disadvantages?
    - Bad performance when files grow and shrink
    - Severe external fragmentation when files are created and deleted
- Noncontiguous file allocation

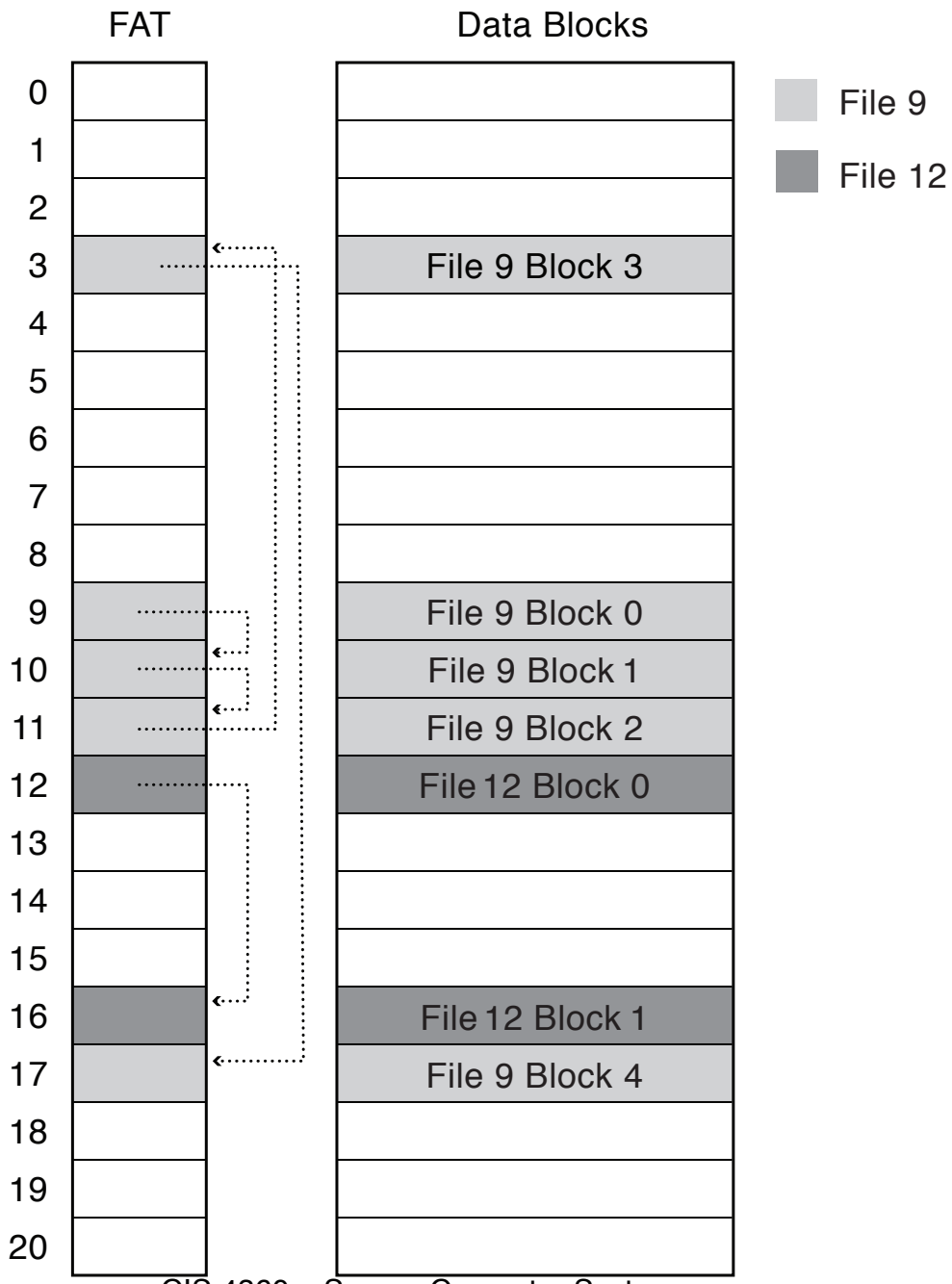
A simple design: linked list of blocks



- Now consider a seek operation targeting the fifth blocks
- The seek operation has to search from the first hop by hop, leading to frequent disk seek operations

Tabular design (FAT32 as an example)

- File Allocation Table (FAT), an array of 32-bit entries in a reserved area of the volume
  - *volume space = space for the table + space for real data*
  - Each FAT entry corresponds to a block in the data area
    - That is, one-to-one mapping
  - If a file has entry *i* in the FAT, it owns block *i* in the data blocks
- Each file in the system corresponds to a linked list of FAT entries


FAT
Data Blocks
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
File 9 Block 3
File 9 Block 0
File 9 Block 1
File 9 Block 2
File 12 Block 0
File 12 Block 1
File 9 Block 4
File 9
File 12

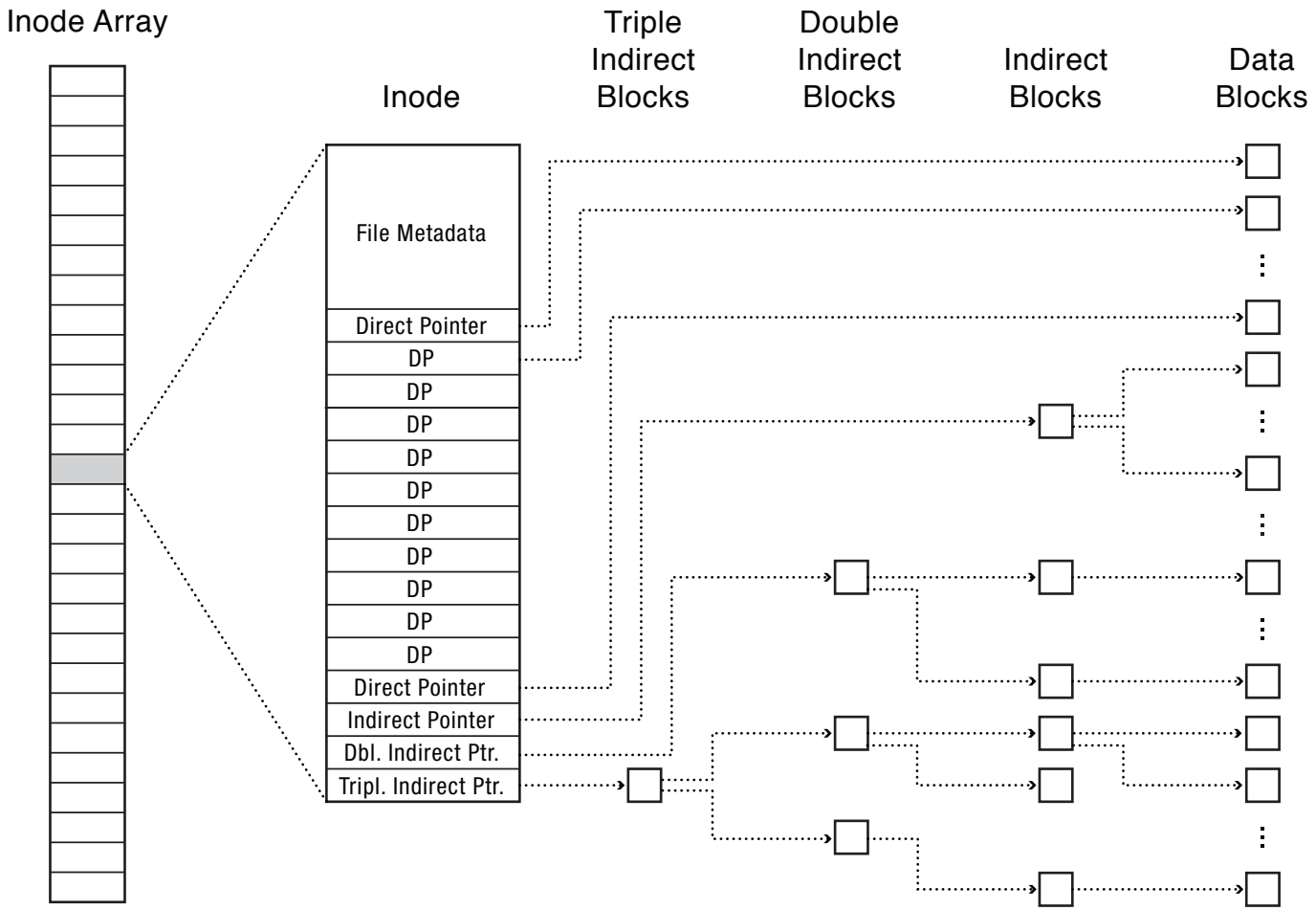
Analysis on tabular design

- Advantage:
  - Pointers that locate file data are stored in a central location, that is, the FAT
  - The table can be cached in memory so that the chain of blocks that compose a file can be traversed quickly
  - Improves the seek performance
- Disadvantage:
  - the locality of data itself is poor, as data blocks may spread across the disk => more disk seeking operations
  - No support for hard links
  - Limitations on volume and file size: take the file size as an example; the system uses a 32-bit field to describe the number of bytes in a file; so the file size limit is 4GB

Multilevel Index design

- The metadata of a file is stored at an inode, which contains
  - An array of direct pointers pointing to data blocks
  - One indirect pointer
  - One double-indirect pointer
  - One triple-indirect pointer
- Each file is a *tree* of blocks rooted at its inode
- Volume space = inodes + indirect blocks + file data
- The design is widely used in
  - Berkeley Fast File System (FFS), also known as Unix File System (UFS)
  - Linux ext2 and ext3

Multilevel Index design



Analysis

- Advantages
  - Each indirect block is 4KB, and each pointer takes 4 bytes, so a block contains 1024 pointers. That is, the tree has a very high fan-out degree, which implies that the search tree can be very shallow, and hence, efficient
  - Scalable design: for small files all the data blocks can be reached through the direct pointers, while the multi-level tree supports very large files (4TB max)

Writing assignments

- Why does not the file system design use a uniform, say, two-level index design, just as the design of the two-level page table?
- What are the difference and relation between “read” and “fread”
- What does mounting do under the hood?