

ESE

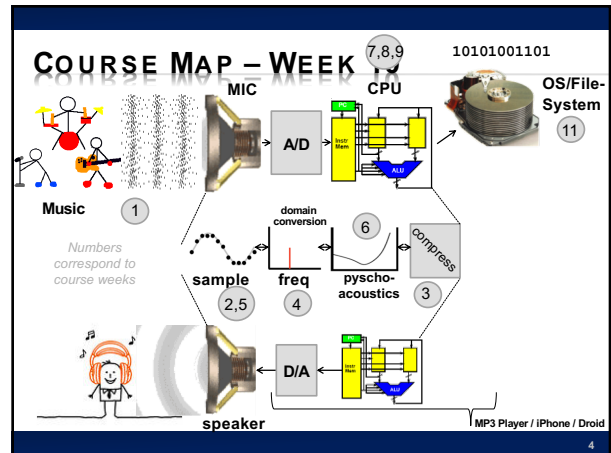
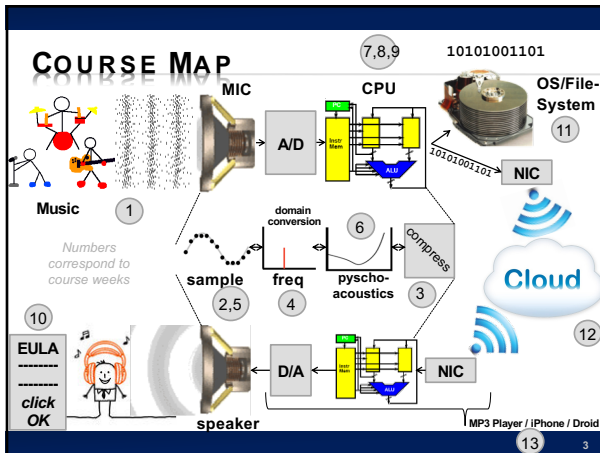
Lecture #11 – Storage/Filesystems

ESE 150 – DIGITAL AUDIO BASICS

Based on slides ©2009–2019 DeHon
Additional Material © 2014 Farmer

LECTURE TOPICS

- ✗ Where are we on course map?
- ✗ Overview of Today's Lecture
- ✗ How/Where do we store our digital music?
 - + Persistent Storage Technology
 - + Filesystems
 - ✗ Abstraction 1: files/directories
 - ✗ Abstraction 2: b-nodes/i-nodes
 - ✗ Abstraction 3: filesystems
- ✗ Next Lab



WHAT WE'LL COVER TODAY...

10101001101

The mighty File System!

- ✗ Last Lecture...
 - + We discussed the idea of an OS
 - ✗ Virtualizes Hardware
 - + Key part of any OS is its filesystem...we'll talk about that today
 - + From floppy disk/hard disk/compact disc/flash drive... "virtually" the same!

STORE AND FIND DATA

- ✗ MP3 encoded Songs on Smart Phone
- ✗ Assignments on your laptop
- ✗ Programs on eniac

PERSISTENT STORAGE TECHNOLOGY

Flash-drives / Hard-drives

7

FLASH MEMORY

- ✗ **A little like memory circuits we have learned about...**
 - + Except it is non-volatile or simply...persistent storage
 - + Data won't go away when power is turned off
 - + Based on the "floating gate" transistor
- ✗ **Today's Examples**
 - + Persistent storage in your MP3 player, cell phone
 - ✗ FYI: first iPod had a hard disk...
 - + USB Flash drive
 - + Solid-State Disk (SSD)

8

FLASH MEMORY

- ✗ **Two ways to configure FG transistors in Flash Memory**
 - + NOR/NAND
- ✗ **NOR -- Read like other memories**
 - + Fast, but not very dense...used when speed is a must
- ✗ **NAND -- Sequential read within "page"**
 - + Denser than NOR, but slower, use when area is a must
- ✗ **Can only "erase" in blocks**
 - + 4KB, 64KB→256KB
- ✗ **Once erased can write byte (page) at a time**
 - + Write time variable
 - + Typically need feedback to sense when written

9

SAMSUNG 256Mx8 NAND FLASH

Parameter	Symbol	Min	Typ	Max	Unit
Program Time	tPROG ⁽¹⁾	-	200	500	µs
Dummy Busy Time for Multi Plane Program	tbusy	-	1	10	µs
Number of Partial Program Cycles in the Same Page	Main Array	-	-	1	cycle
	Spare Array	-	-	2	cycle
Block Erase Time	tBERS	-	2	3	ms
RE Pulse Width	tRP	25	-	-	ns
WE High to Busy	tWB	-	-	100	ns
Read Cycle Time	tRC	50	-	-	ns
CE Access Time	tCSA	-	-	45	ns
RE Access Time	tRESA	-	-	30	ns
RE High to Output Hi-Z	tRHZ	-	-	30	ns
CE High to Output Hi-Z	tCHZ	-	-	20	ns
RE or CE High to Output hold	tOH	15	-	-	ns
RE High Hold Time	tREH	15	-	-	ns
Output Hi-Z to RE Low	tRL	0	-	-	ns
WE High to RE Low	tWRL	60	-	-	ns
Device Resetting Time(tReset)	tRST	-	-	5/10/500 ⁽²⁾	µs
Last RE High to Busy(at sequential read)	tRB	-	-	100	ns
CE High to Ready(in case of interception by CE at read)	tCRV	-	-	50 + tr(RB) ⁽³⁾	ns
CE High Hold Time(at the last serial read) ⁽²⁾	tCBH	100	-	-	ns

10

INTEL SOLID-STATE DRIVE (SSD)

Table 3. Maximum Sustained Read and Write Bandwidth

Access Type	MB/s
Sequential Read	up to 250
Sequential Write	up to 170

Table 4. Random Read and Write Input/Output Operations per Second (IOPS)

Access Type	IOPS
4K Read	35,000
4K Write	3,300

35,000/s x 4KB = 140MB/s

Table 5. Latency Specifications

Type	Average Latency
Read	75 µs (TYP)
Write	85 µs (TYP)
Power On to Ready	1 s

SSDhttp://download.intel.com/design/flash/nand/extreme/extreme-sata-ssd-datashet.p

11

FLASH MEMORY

- ✗ **Basic access time model**
 - + $T \sim A + B \times N$
 - + Large fixed expense A
 - ✗ Erase block for flash ~ 3ms
 - ✗ (less for read, but for simplicity we'll keep that)
 - ✗ (Move in R and Θ for disk drives ~ 10ms)
 - + High bandwidth B for sequential data
 - ✗ ~100s of MB/s
 - ✗ B=10 ns

12

PRECLASS 1

- × Read N bytes: $T=3\text{ms} + N \cdot 10\text{ns}$
- × **Throughput when reading random 32b words from memory?**
 - + Randomly select an address
 - + Read 4B
 - + repeat
- × **Throughput when reading 4KB blocks from memory?**
 - + Randomly select an address
 - + Read 4KB
 - + repeat

13

PRECLASS 1

- × Read N bytes: $T=3\text{ms} + N \cdot 10\text{ns}$
- × **Throughput when reading 1MB blocks from memory?**
 - + Randomly select an address
 - + Read 4KB
 - + repeat

14

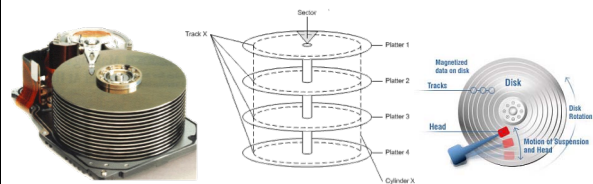
THROUGHPUT AND IMPLICATIONS

- × **Flash throughput and access time**
 - + 3ms latency
 - + 100MB/s throughput ($\sim 1\text{B}/10\text{ns}$)
- × **Observations?**
 - + **Throughput** faster than **access time**
 - + 3ms seek \rightarrow Random bit access
 - + Sequential access 1000MB/s
- × **Conclude:**
 - + **Want to exploit sequential access!**
 - × Read blocks of data

15

HARD DRIVES / HARD DISKS (HD)

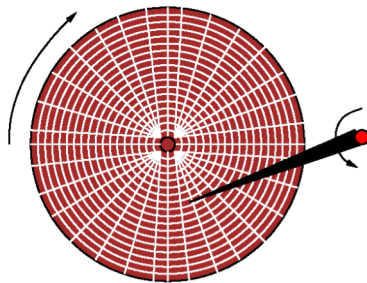
- × A collection of metallic “platters”
- × Each platter covered with magnetic material
- × Magnetic charge ‘stores’ 1 bit of information
- × Can vary charge across platter!



16

HARD DISK

- × Each bit located at a position (R, θ)
- × Head arm moves
 - + Varies R
- × Disk spins
 - + Vary θ

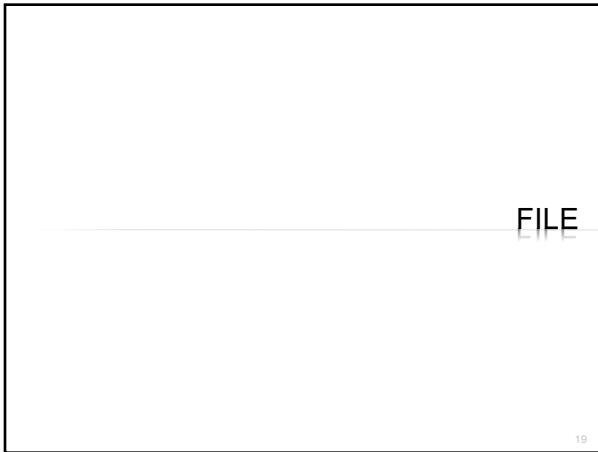


17

THROUGHPUT AND IMPLICATIONS

- × **Disk throughput and access time**
 - + 10ms latency (move head in R, disk spin θ)
 - + 280MB/s throughput ($\sim 1\text{B}/4\text{ns}$)
- × **Observations?**
 - + **Throughput** faster than **access time**
 - + 10ms seek \rightarrow Random bit access 100b/s
 - + Sequential access 280MB/s
- × **Conclude:**
 - + **Want to exploit sequential access!**
 - × Read blocks of data

18



HOW ORGANIZE DATA

- ✗ **Have technology to store bits**
 - + GB, TB of data
- ✗ **How do we find our data?**
 - + Remembering one 40b address could be hard
- ✗ **How do we distinguish used/unused storage?**
 - + Make sure someone doesn't overwrite our data

FILE

Example File: my_file.txt (14 characters)

I LOVE ESE150!

X49 x20 x4C x4F x56 x45
x20 x45 x53 x45 x31 x35
x30 x21

01001001 00100000 01001100
01001111 01010110 01000101
...

- ✗ **A file is simply a collection of bits**
 - + Whether it's an ASCII file or a binary file (.docx, pdf, etc)
 - + A program gives the bits meaning
 - + Sequential access to bits efficient → useful to group together so can read all at once

FILES → BLOCKS → PHYSICAL DISK

Mapping physical storage media to "bit/bytes/blocks"

HOW IS A FILE STORED?

- ✗ **A file is an abstraction of the physical storage media**
 - + Media "independent"
 - + Don't care how technology is implemented, just read/write file!
- ✗ **Recall a file consists of:**
 - + A bunch of bits
 - + Logically related
 - + Ex: my_report.docx, etc
- ✗ **A file system is responsible for providing this abstraction of the disk**
 - + On storage media, at lowest level, file is in "blocks" of bits

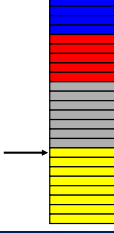
HOW WE STORE FILES ON PHYSICAL MEDIA

- ✗ **As we write data to a disk, we do it sequentially...**
 - + Data in File 1 = blue, Data in File 2 = red, Data in file 3 = grey...
- ✗ **Sequentially written files have huge advantage in terms of access time...**
 - + more so when we were dealing with hard disks, but some advantage for flash memory, as well.

HOW WE STORE FILES ON PHYSICAL MEDIA

First Model

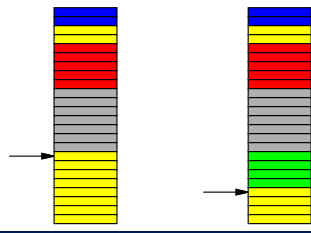
- × File
 - + start address
 - + Length
- × Allocate space sequentially
 - + Keep track of first free address
- × But what happens when...
 - + Delete file?
 - + Append to a file?



25

DELETING FILES

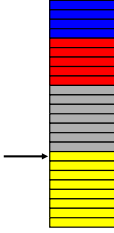
- × Let's say we shorten file1 (blue file)...
 - + We now have 2 open spaces (blocks) on disk
 - + What happens when write a 4th file of 4 blocks?



26

HOW WE STORE FILES ON PHYSICAL MEDIA

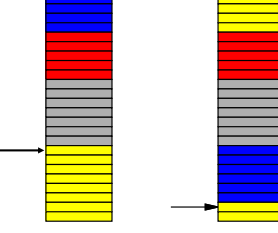
- × We could also lengthen file1 (blue file)...
 - + Ex. Now need six (blocks) to store file
 - + How accommodate?



27

HOW WE STORE FILES ON PHYSICAL MEDIA

- × We could also lengthen file1 (blue file)...
 - + Ex. Now need six (blocks) to store file
 - + How accommodate?

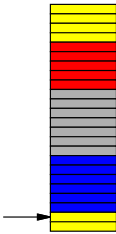


28

HOW WE STORE FILES ON PHYSICAL MEDIA

First Model

- × File
 - + start address
 - + Length
- × Allocate space sequentially
 - + Keep track of first free address
- × Problem:
 - + Can never reclaim space when freed up

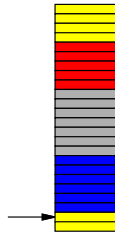


29

HOW WE STORE FILES ON PHYSICAL MEDIA

Second Model


- × File
 - + start address
 - + Length
- × Allocate space where will fit
 - + Keep track of list of free regions
 - + Search for free region that has enough space to hold file



30

FRAGMENTATION

- × Want to add 4 block file
- × What happens here?

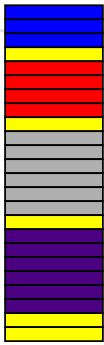


31

FRAGMENTATION

Second Model

- × File
 - + start address
 - + Length
- × Allocate space where will fit
 - + Keep track of list of free regions
 - + Search for free region that has enough space to hold file
- × **Problem:**
 - + Space becomes fragmented
 - + May have enough space, but not contiguous




32

HOW WE STORE FILES ON PHYSICAL MEDIA

Third Model


- × File
 - + Collection of blocks
 - + Not necessarily contiguous
- × Allocate unused set of blocks
- × Use pointers/links to connect together



33

PHYSICAL MEDIA DEFECTS

- × Physical media often has errors
 - + Bits that cannot be written or won't hold their values
- × Also prevents making all files contiguous




34

HOW WE STORE FILES ON PHYSICAL MEDIA

Third Model


- × File
 - + Collection of blocks
 - + Not necessarily contiguous
- × Allocate unused set of blocks
- × Use pointers/links to connect together
- × How does this model accommodate defects?



35

FRAGMENTATION

- × Fragmentation is a fact of life
 - + But who is keeping track of all this fragmentation?
 - × **Filesystem!**



× Note: de-fragmentation software built into OSs now

36

WHAT IS A FILESYSTEM?

FORMATION OF A FILE / FILESYSTEM

- × **Represent File**
 - + Not always just a sequence of bits on the media
- × **When files become fragmented...**
 - + How account for and manage fragmentation?
- × **How do we know where freespace is?**
- × **For that matter...**
 - + How are all the files kept track of, or even found on the disk?
- × **The file system**
 - + Part of the operating system that organizes/keeps track of the disk
 - + To understand what its keeping track of, we need to see what a file is...

FILE REPRESENTATION

- × **File is not just a sequence of bits**
- × **Contains some data about it**
 - + Length
 - + Type
 - + Timestamp,
- × **Set of pointers to the data (when large)**
 - + Allowing the data to be non-sequential

FILE AS LINKED BLOCKS

- × **File is a linked set of blocks**
- × **Reserve last 6 Bytes of file to point to next file in block (or indicate end-of-file)**

FILES AS LINKED BLOCK

- × **What if want to start at particular place in file?**
 - + 4KB blocks
 - + Start at byte 10,000 ?

FILES AS LIST OF BLOCKS

- × **File is a block that has pointers to all the files in the block**
- × **Limiting?**
- × **When inefficient?**

FILES AS TREE OF BLOCKS

- File is tree overlaying blocks
- Leaf nodes hold data
- Non-leaf nodes hold pointers to
 - More non-leaf nodes
 - Leaf data blocks

43

PRECLASS 2

- 10^9 data items
- Assume at bottom of balanced tree
- Each tree-node has c leaves
- Directory i-node to hold c leaves needs
 - $32 \times c$ Bytes
- How many tree nodes must visit?
- How long to read a tree node?
- Time to lookup item (traverse tree?)
 - $c=2, c=10^3, c=10^9$

44

PRECLASS 2 FULL SWEEP

levels	read tree node	total time
2	3000440.01	0.0000162003
4	3001280.01	0.04501920015
8	3002560.01	0.0300256001
16	3005120.01	0.02404960008
32	3010240.01	0.01801440006
64	3020480.01	0.01510240005
128	3040960.01	0.01520480005
256	3081920.01	0.01232768004
512	3163840.01	0.01266304004
1024	3327680.01	0.00998304003
2048	3655360.01	0.01096608003
4096	4310720.01	0.01232168003
8192	5621440.01	0.01686432003
16384	8242880.01	0.02472864003
32768	13485760.01	0.02697152002
65536	23971520.01	0.04794304002
131072	44843040.01	0.08988080002
262144	8688080.01	0.17377216
524288	170772160	0.34154432
1048576	335544320	0.67708864
2097152	674088640	1.34817728
4194304	1345177280	2.69635456
8388608	2687354560	5.37470912
16777216	5371709120	10.74341824
33554432	10740418240	21.48063648
67108864	21477836480	42.95567296
134217728	42955672960	85.90334592
268435456	85903345920	171.80469184
536870912	171806691840	343.6033837

45

WHAT DOES UNIX DO?

46

FILES → I-NODES → B-NODES/BLOCKS

In the Linux OS

- Popular filesystem named: ext3
- Derived from original Unix File System (UFS)
- Uses i-nodes

What is an i-node?

- A data-structure that represents a file on the storage media
- Consists of file information:
 - Owner, size, timestamp, etc.
- Also 15 pointers
 - 12 pointers that point directly to "blocks"
 - 3 additional pointers that point to other pointers! (indirect blocks)

47

FILES → I-NODES → B-NODES/BLOCKS

How does it work?

- If a file only needs 12 blocks on disk...
 - 12 direct block pointers are used
 - Each of them point to a "bnode" or "block" of data on the disk
- If a file requires 13 blocks...
 - 12 direct blocks are set/allocated
 - 1 indirect block is also needed

48

FILES → I-NODES → B-NODES/BLOCKS

i-node

Mode
Owner info
Size
Timestamps
Direct Blocks (12 pointers)
Indirect blocks
Double Indirect
Triple Indirect

What exactly is a b-node?

- Smallest unit of storage allocation
- Fixed-size of data on the disk itself
- Typical sizes:
 - 512 bytes for hard drives
 - 2048 bytes for CDs/DVDs
 - 4096 bytes (4kB) for todays drives
- Smallest "addressable" unit on disk
 - Example: bnode address 76
 - Would get $76 \times 4096 = 311,296$ byte on flash
- Address actually maps to physical address on (flash, disk)
 - Example: bnode address 76
 - $R=1.012in, \theta=32.07 \text{ degrees}$
 - Actual location on disk
 - One level above media itself

49

FILES → I-NODES → B-NODES/BLOCKS

i-node

Mode
Owner info
Size
Timestamps
Direct Blocks (12 pointers)
Indirect blocks
Double Indirect
Triple Indirect

- At 4KB / Bnode
- How large a file can represent using only Direct Blocks?
 - (assuming get full 4KB block for data)

50

FILES → I-NODES → B-NODES/BLOCKS

Bnode's structure:

- Bnode contains metadata (like a header)
 - Description of data to follow
 - Block type, File type, length
- Bnode contains data itself (contents)
 - This is actual data for the file in question
 - Example shows only a 3172 block file (could use all 4072 bytes)

Example:
4096 Byte
bnode

Type/length

24 Bytes metadata

3172 Bytes
Block contents

900 Bytes unused

51

FILES → I-NODES → B-NODES/BLOCKS

Bnode's structure:

- bnode contains metadata (like a header)
 - Description of data to follow
 - Block type, File type, length
- Alternatively... can be table of pointers (indirect block)
 - Can be a multi-level tree if necessary (doubly/triply indirect)

24 Bytes metadata

3172 Bytes
Block contents

900 Bytes unused

76: obj, 15.791

77: []

78: []

79: []

80: []

mp3, 12MB

1024

52

FILES → I-NODES → B-NODES/BLOCKS

How help with shortening or appending to file?

Example:
4096 Byte
bnode

Type/length

24 Bytes metadata

3172 Bytes
Block contents

900 Bytes unused

53

SIZE OF B-NODE/BLOCK REPRESENTATION

Why not make block sizes smaller? (say 1-bit)

- How about 1 bit at a time? Addressing each bit wouldn't be easy...
 - 1 TB disk ~ 9 trillion bits...9 trillion addresses!
- Most times files are larger than just a few bits

Why not make block sizes larger? (512, 2048, 4096 bytes)

- Usually, we transfer "blocks" of data to/from media at a time
- Why not go larger?
 - Remember, block size is, smallest unit of addressable space

Example:
4096 Byte
bnode

Type/length

24 Bytes metadata

3172 Bytes
Block contents

900 Bytes unused

In this 4kB block, 900 bytes are wasted
"Internally fragmented"
No way to 'un-fragment' unless file itself grows

54

B-NODE/BLOCK SIZE

- ✘ **Performance: Balance**
 - + Number of blocks need to read
 - + Efficiency of reading large blocks
- ✘ **Space efficiency: Balance**
 - + Internal fragmentation
 - + Overhead for metadata and links

55

LEVELS OF ABSTRACTION

i-node

Mode
Owner info
Size
Timestamps
Direct blocks (12 pointers)
Indirect blocks
Double Indirect
Triple Indirect

Physical Location on disk
 $R=1.012in, \theta=32.07 \text{ degrees}$
Block of data
 4096 continuous bytes on disk
B-node
 Datastructure representing block of data
i-node / file
 Datastructure representing many blocks of data
Programmers work here (file level) saves us from knowing about physical details which may change across storage technologies

57

HOW KEEP TRACK OF FILES?

- ✘ **Add another file that tells us where the files are**
 - + Directory
- ✘ **On ext3 filesystem...**
 - + Directories are just files themselves!
 - + Pairs of (Name, i-node)
 - + Contain pointers to other nodes
- ✘ **...and since files can be directories**
 - + Directories can contain directory files ...which can contain directory files...
- ✘ **Leading to a directory hierarchy**

58

UNIX SYSTEM FILE STRUCTURE

This is called the "root" directory

Below "home" is up to eniac admins

ESE150 "Home directory" is `/home1/e/e/e150` or simply: `~`
 Because the structure under "home" is typically different on all unix systems, `~` is a convention, and is always points to *your* home directory

59

SUPERBLOCK

- ✘ **For bootstrapping and file system management**
 - + Each file system has a master block in a canonical location (first block on device)
 - + Describes file-system type
 - + Root bnode
 - + Keeps track of free lists ...at least the head pointers to (bnodes, blocks)
- ✘ **Corruption on superblock makes file system unreadable**
 - + → Store backup copies on disk

61

FORMAT DISK

- ✘ **Identify all non-defective bnodes**
 - + Defective blocks skipped
 - + → those addresses not assigned to bnodes
- ✘ **Create free bnode data structure**
- ✘ **Create superblock**

62

DISK DATA SECURITY

- × **How is security enforced?**
 - + OS demands credentials for login
 - + User doesn't get direct access to hardware
 - + OS intermediates

63

SECURITY CAVEATS

- × **On standard Unix/Windows setups**
 - + Without the OS to provide protection, all the data is accessible
 - × Sometimes good for recovery
 - + On standard Unix/Windows setups
 - × rm/del doesn't make the data go away
 - × Also sometimes useful for recovery
 - + Even format does not guarantee data overwritten
- × **See:** Remembrance of Data Passed: A Study of Disk Sanitization Practices
- × **What about iPhone?**

64

OS'S AND THEIR FILESYSTEMS

- × **Typically an OS has a native filesystem it supports:**
 - × Early PC OS: DOS (Disk Operating System)
 - File system: FAT (File Allocation Table)
 - × Early Apple OS: Macintosh's "System"
 - File system: MFS (Machintosh File System)
 - × Today's PC OS: Windows
 - File System: NTFS (New Technology File System)
 - × Today's Apple OS: Mac OSx
 - File System: HFS+ (Hierarchical File System)
 - × Today's Linux OS: Ubuntu, Debian, Fedora, Red Hat, SUSE, etc)
 - File System: EXT (Extended File System)
- × **Now a days, many OS include support for more than one file system**
 - × Example...CDs/DVDs have their own file system:
 - CDs: CDFS (Compact Disc File System): ISO 9660
 - DVDs: UDF (Universal Disk Format)

68

BIG IDEAS...

- × **Often have fixed+per-item cost: $T=A+B \times N$**
 - + For data access (data transmission)
 - + Also for size of structures
 - + → Benefit to dealing with blocks of data as a group
- × **Need organization to**
 - + Hide physical media details
 - + Self-describe data layout
- × **When must delete and resize**
 - + Helps to build from data structure of blocks rather than demand one contiguous unit

69

THIS WEEK IN LAB

- × **Lab 10: Design multi-view file system**
 - + As might want for MP3 player
- × **Lab posted**

70

LEARN MORE AT PENN!

- × **Online reading/pointers**
 - + Unix File System Tutorial
 - + Flash, SSD, Hard drive data sheets
 - + Data found on hard drive articles
- × **Courses**
 - + CIS121 – efficient data structures
 - + CIS380 – operating systems

71