University of Glasgow | School of Computing Science

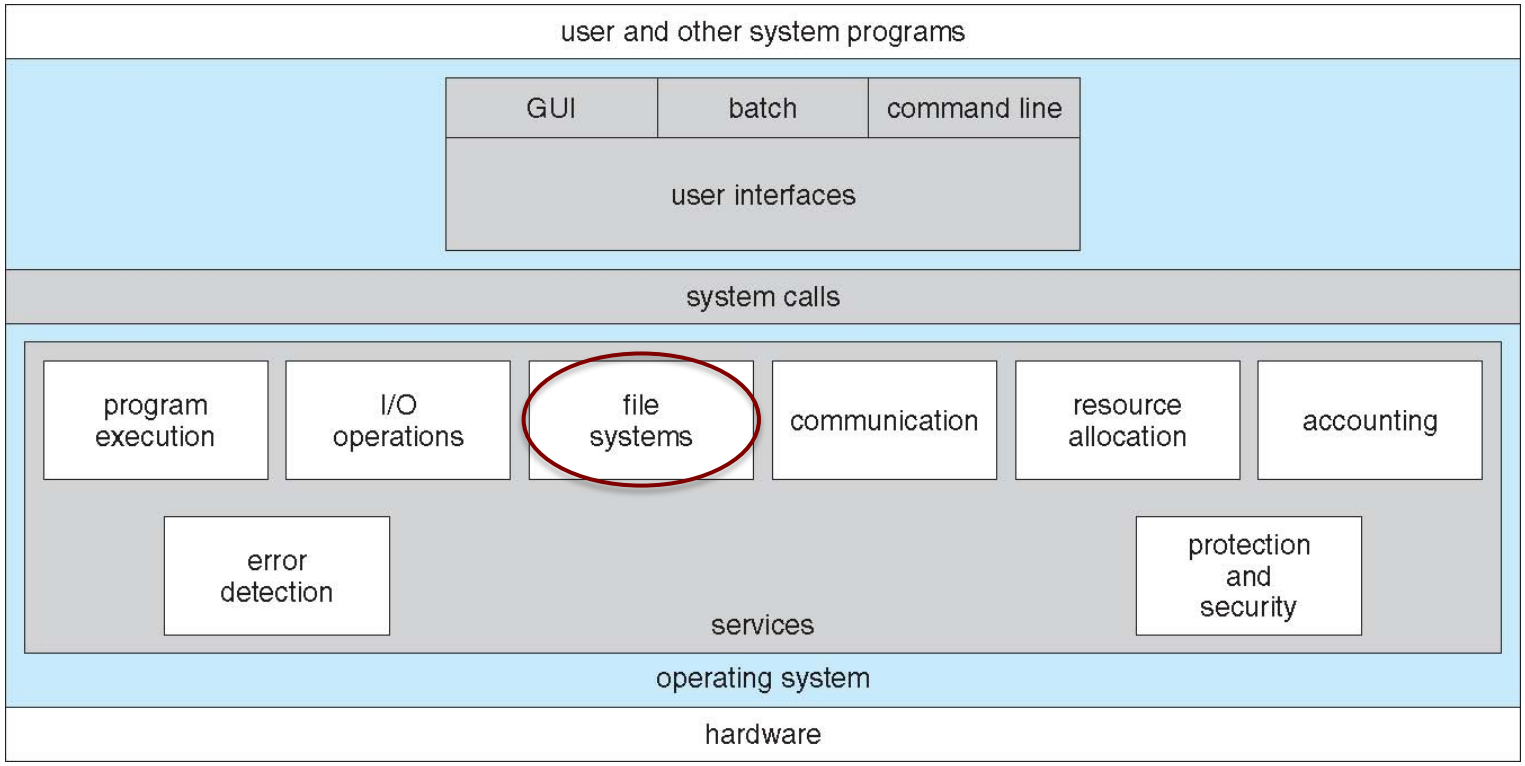
Networks & Operating Systems Essentials

Dr Angelos Marnerides

<angelos.marnerides@glasgow.ac.uk>

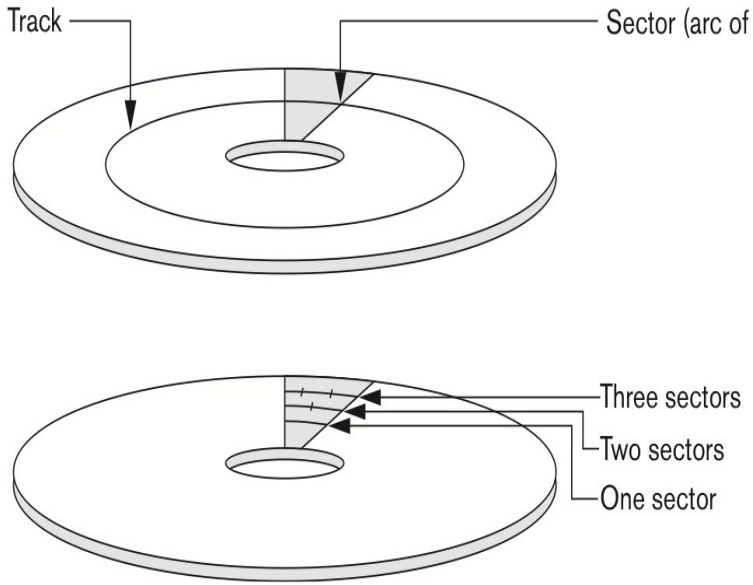
School of Computing Science

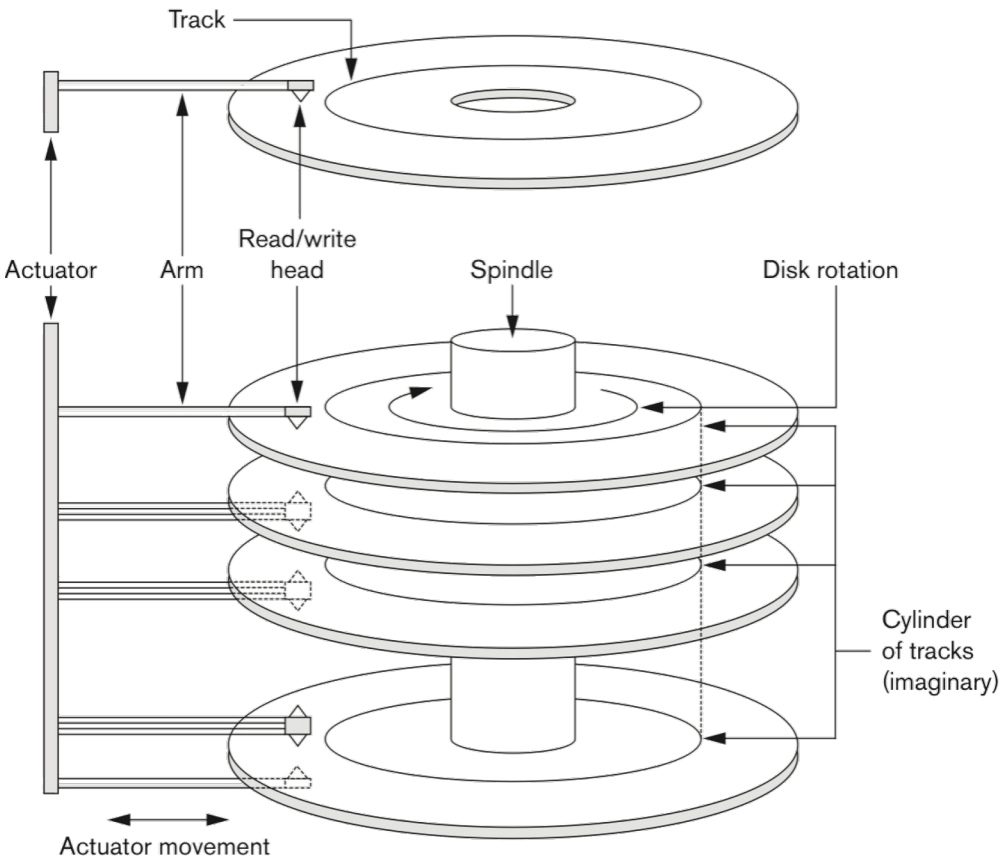
Operating System Services



Magnetic Hard Disks

Disk Anatomy & Data on Disk





Disk Anatomy & I/O Access

- Recipe:
  - Take a (magnetic) disk
  - Dig concentric tracks
  - Split tracks into circular sectors (~512B)
  - Logically group up consecutive sectors into fragments (~1KB)
  - Logically group up consecutive fragments into blocks (~1-10KBs)
  - Organise each file in blocks
  - Store each file block in a disk block
  - Done!

- Read/Write Access (I/O)
  - Position r/w heads:
    - **Seek delay**: ~3-6ms
  - Spin until selected sector under the head:
    - **Rotation delay**: ~3-5ms
  - Transfer data from *track* to *disk buffer* (extremely fast)
  - Transfer data from *disk buffer* to *RAM* via bus (using DMA -- also fast)

- **Challenge**: Spatially organise the file blocks so as to **minimize the expected seek & rotation time**

Addressing in hard disk drives (HDDs)

- Two main methods:
  - CHS (Cylinder-Head-Sector)
  - LBA (Logical Block Access)

- LBA mostly used nowadays (since 2002)
  - More simple
  - Device-independent

- Old CHS-based addressing can be mapped into LBA:
  - LBA address = (c x Number_of_heads + h) x Number_of_sectors + (s – 1)

Disk Anatomy -- Formatting/partitioning

- Disk platters are originally empty
  - Disk needs to be *low-level formatted* to create sectors
    - Usually done at the factory
    - Sectors usually also contain disk-related *metadata* (e.g., error correcting code, sector id/no, etc.) ➔ allows for detection of *bad blocks*
- Disks can also be *partitioned* so as to appear as several different "disks"
  - Usually on a cylinder boundary
    - I.e., cylinders 0 - X → partition 1, X+1 - Y→ partition 2, etc.
  - Nothing changes on the disk -- this is an OS abstraction
- Last, each partition needs to be initialized
  - *Logical formatting* or *file system formatting*
  - Computes and stores file-system specific metadata in blocks
    - Creates root directory
    - Initialises list of free/allocated blocks in the partition
  - Often also initializes the Master Boot Record (MBR)
    - First few blocks of the disk/partition

Logical Block Access

- Disk drives are addressed as large 1-dimensional array of **logical blocks**.
- The 1-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially.
  - Block 0 is the first sector of the first track/head on the outermost cylinder.
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

Logical Block Access (cont..)

LBA	C	H	S
0	0	0	0
1	0	0	1
2	0	0	2
3	0	0	3
4	0	0	4
5	0	0	5
6	0	0	6
7	0	0	7
8	0	0	8
9	0	0	9
10	0	1	0
11	0	1	1
12	0	1	2
13	0	1	3
14	0	1	4
15	0	1	5
16	0	1	6
17	0	1	7
18	0	1	8
19	0	1	9
Cylinder 0			

LBA	C	H	S
20	1	0	0
21	1	0	1
22	1	0	2
23	1	0	3
24	1	0	4
25	1	0	5
26	1	0	6
27	1	0	7
28	1	0	8
29	1	0	9
30	1	1	0
31	1	1	1
32	1	1	2
33	1	1	3
34	1	1	4
35	1	1	5
36	1	1	6
37	1	1	7
38	1	1	8
39	1	1	9
Cylinder 1			

Disk Anatomy – Abstractions (Recap)

- For simplicity, we'll consider the disk as a long, flat array of disk blocks/clusters
  - All blocks have the same size (although some modern disks can be low-level formatted to have different block sizes)
  - Note: Each block made up of multiple (fixed number) fragments
  - Note: Each fragment made up of multiple (fixed number) sectors
  - Note: Fragments have addresses, from 0 to size of disk (in fragments); blocks are addressed with the address of their 1st fragment

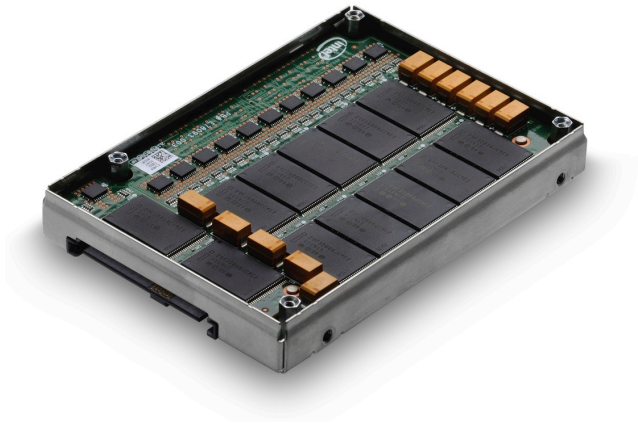
- Hmmm… This sounds a lot like RAM…
  - Think: sector → memory bit; fragment → memory byte; block → memory word
  - Unlike RAM, **not** all sectors have the same natural performance characteristics
    - Think: sectors at the outer part of the disk vs. those at the centre…
    - How can we provide a (semi-)constant data rate?

Disk Anatomy -- Scheduling

- A typical disk may have many I/O requests waiting to be served
  - The average disk I/O takes several ms
  - Several processes may have executed in that time
  - Many of them may have issued I/O requests

- Where there's a queue, there's a ~~delay~~scheduler…
  - FCFS (First-Come-First-Served)
  - SSTF (Shortest-Seek-Time-First): Choose request with lest seek distance from current head position
  - SCAN/LOOK (a.k.a. the *elevator* algorithm): Choose next request in the direction of the motion of the head (SCAN: till end of disk; LOOK: till furthest request), then reverse direction of motion and continue
  - C-SCAN/C-LOOK (circular SCAN/LOOK): Like SCAN/LOOK but reset to centre of disk instead of reversing direction
  - Which one is the best?

Solid-State Disks (SSD)



SSDs (cont..)

- Non-volatile memory used like a hard drive
  - Most of them use NAND-based flash memory
- Less fragile than HDDs
- More expensive per GB
- May have shorter life span
- But much faster and less energy consumption
- No moving parts, so no seek time or rotational latency

SSDs (cont..)

- NAND Flash stores data in a large array of cells.
- Each cell stores data
  - Single-Level Cell (SLC), 1 bit
  - Multi-Level Cell (MLC), 2 bits
  - Three-level Cell (TLC), 3 bits
  - Four-level Cell (QLC), 4 bits

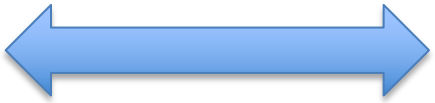
- LBA -> blocks translated into sets of cells.

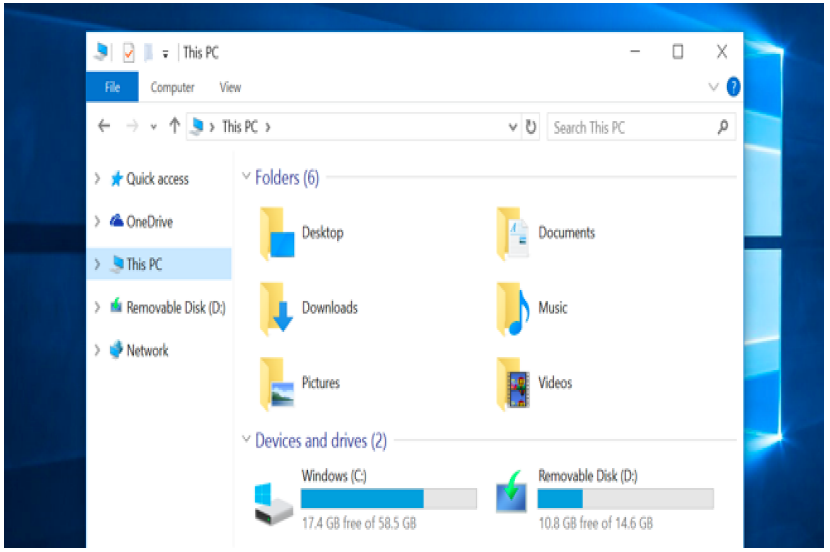
- NAND cells are not designed to last forever!
  - NAND storage devices have a limited number of write cycles
  - Approx. 100,000 cycles

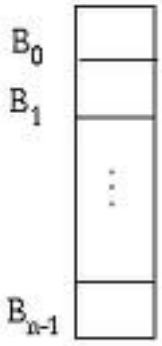
Magnetic Tape

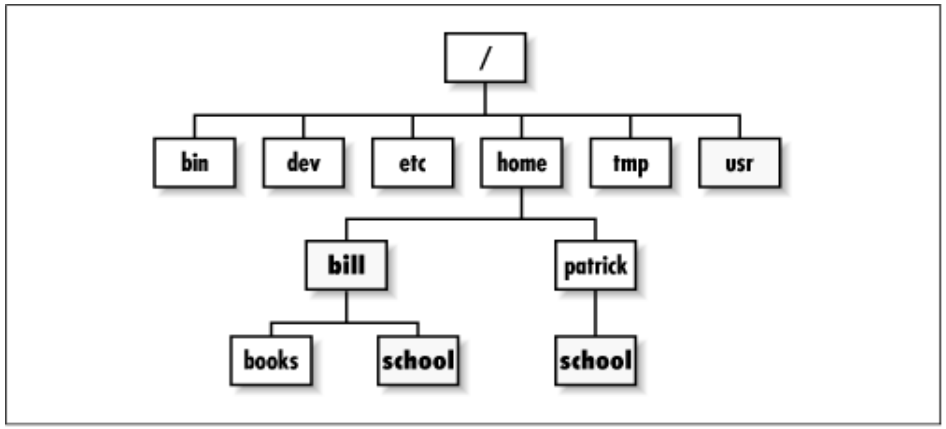
- Was early secondary-storage medium
  - Evolved from open spools to cartridges
- Relatively permanent and holds large quantities of data
- Slow access time
  - Random access ~1000 times slower than disk
- Mainly used for backup, storage of infrequently-used data, transfer medium between systems

From Blocks to Files?







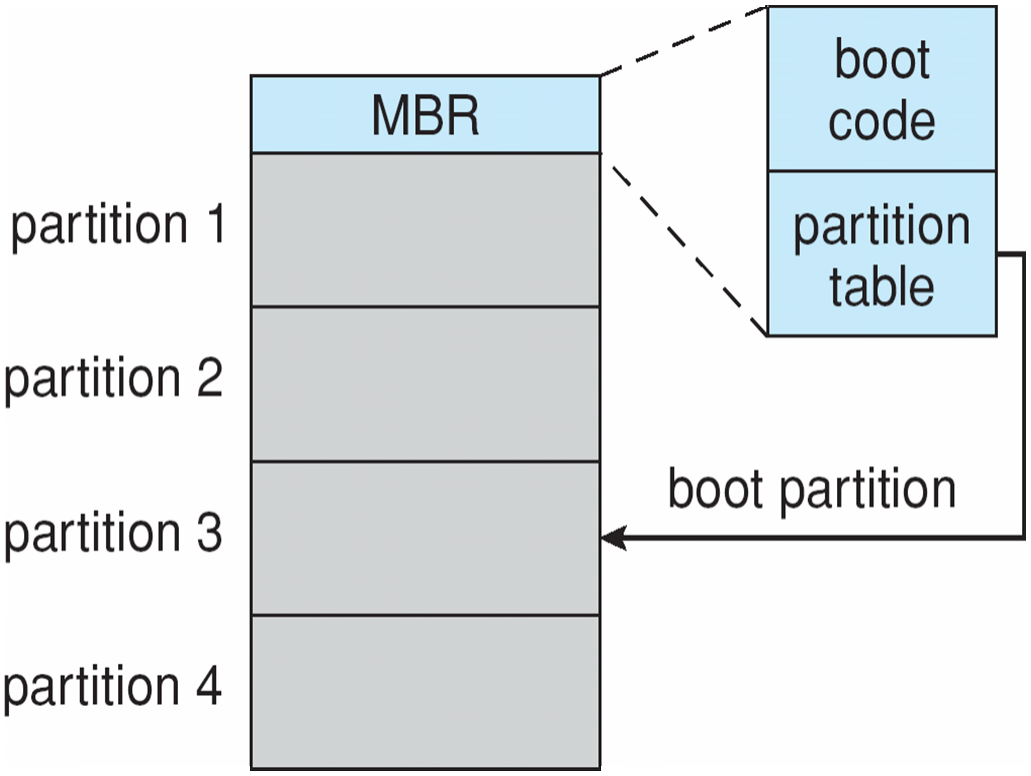


From Blocks to Files? (cont..)

- In order for a disk to hold files, the operating system also needs to record its own data structures on the disk….
  - **Partition** the disk into one or more groups of contiguous blocks
  - **Logical formatting** or "making a file system"
  - To increase efficiency most file systems group blocks into **clusters**

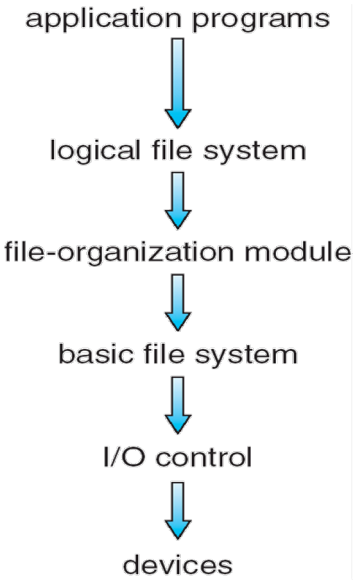
OS Booting process

- Master Boot Record (MBR) initializes system
  - At sector/block 0
  - Holds partition table
  - Bootstraps OS
  - 32/64-bit disk signature
  - Cluster size



File Systems

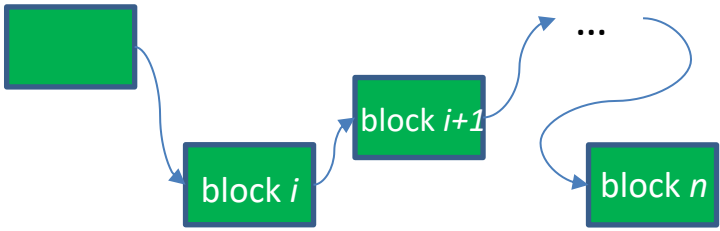
- Platters/tracks/sectors… Too complicated for us to *keep track of*!
- What do we want?
  - Ability to organise data in files
  - Impression of contiguous logical address space for each file
  - High-level API that builds on this abstraction
    - Ability to add/delete/update data in a file
  - Ability to organise/access files through an intuitive interface
  - Access control/security
- Filesystem layers
  - Logical file system:
    - Provides naming and logical organisation (files, directories, access rights)
    - Presents user-facing API and keeps track of per-process open files
    - Passes all I/O operations to the layer below
  - Virtual file system (optional):
    - Presents to the logical layer a unified view of possibly multiple physical layer implementations
  - Physical file system:
    - Handles disk blocks -- reading/writing blocks, buffering and *memory* management
    - Interfaces with the device driver/hardware



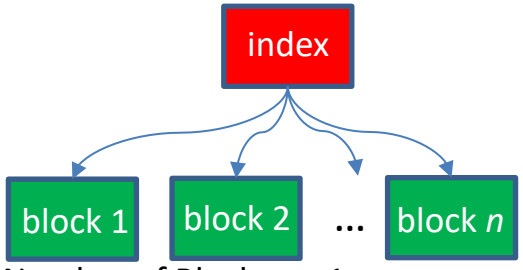
Physical Filesystem

- *File* is a set of *blocks* of data
  - Note: blocks are *data agnostic*
- How do we allocate blocks to files?
  - **Contiguous (neighboring) allocation**: A file contains spatially consecutive blocks (a.k.a. self-navigation file)
  - **Linked allocation**: Each block i has a pointer to the logically next block i+1 anywhere on the disk -- i.e., a linked list of blocks; navigation information is stored in each block of the file
  - **Indexed allocation**: there exists a specific block (index block -- can be anywhere on the disk), which maintains a list of pointers pointing to the physical address of each block; navigation information stored in the index block
- Which one is the best?
  - Depends on file access patterns
    - Contiguous: great for sequential and random I/O
    - Linked: fairly good for sequential I/O, bad for random I/O
    - Indexed: as good as linked for sequential I/O; as good as contiguous for random I/O

Number of Blocks: *n*



Number of Blocks: *n*



Number of Blocks: *n*+1

Widely used File Systems

- FAT32 (Windows, Linux, OSx) – FAT12 1977
- NTFS (Windows) - 1993
- EXT4 (Linux, Android from 2.3) – EXT2 1993, EXT4 2008
- HFS+ (OS X, iOS) - 1998

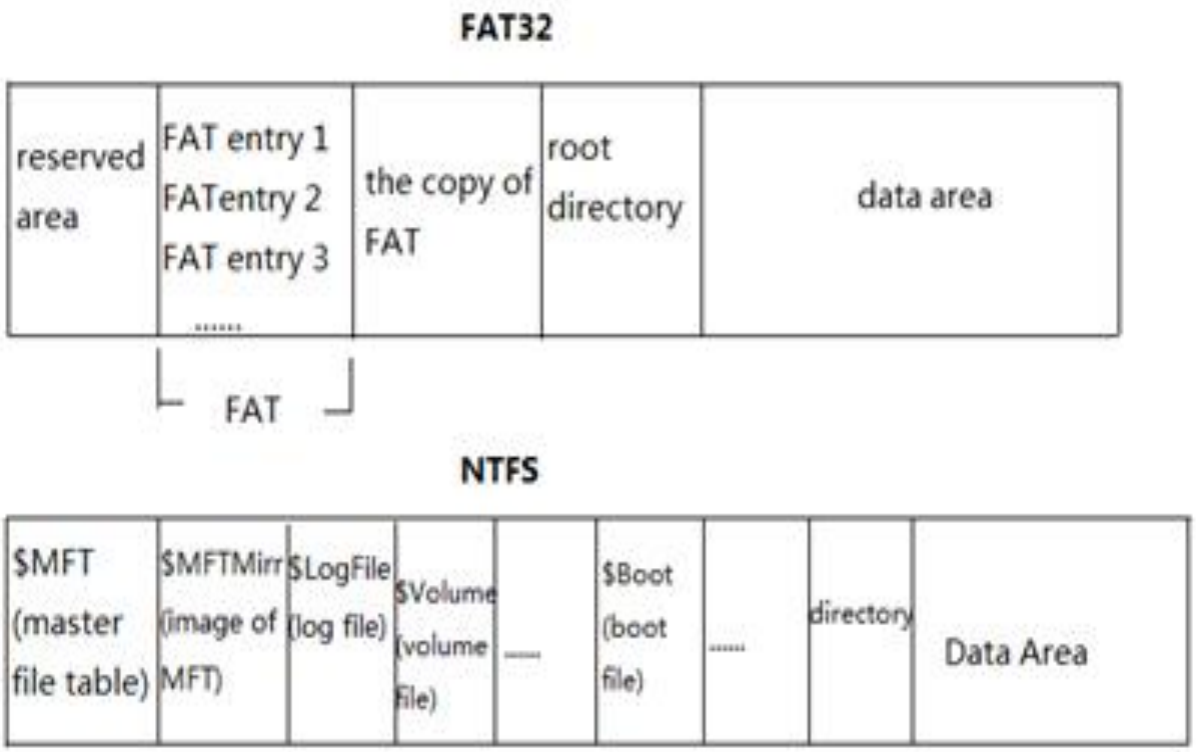
- **Focus on:**

  1) Partition structure

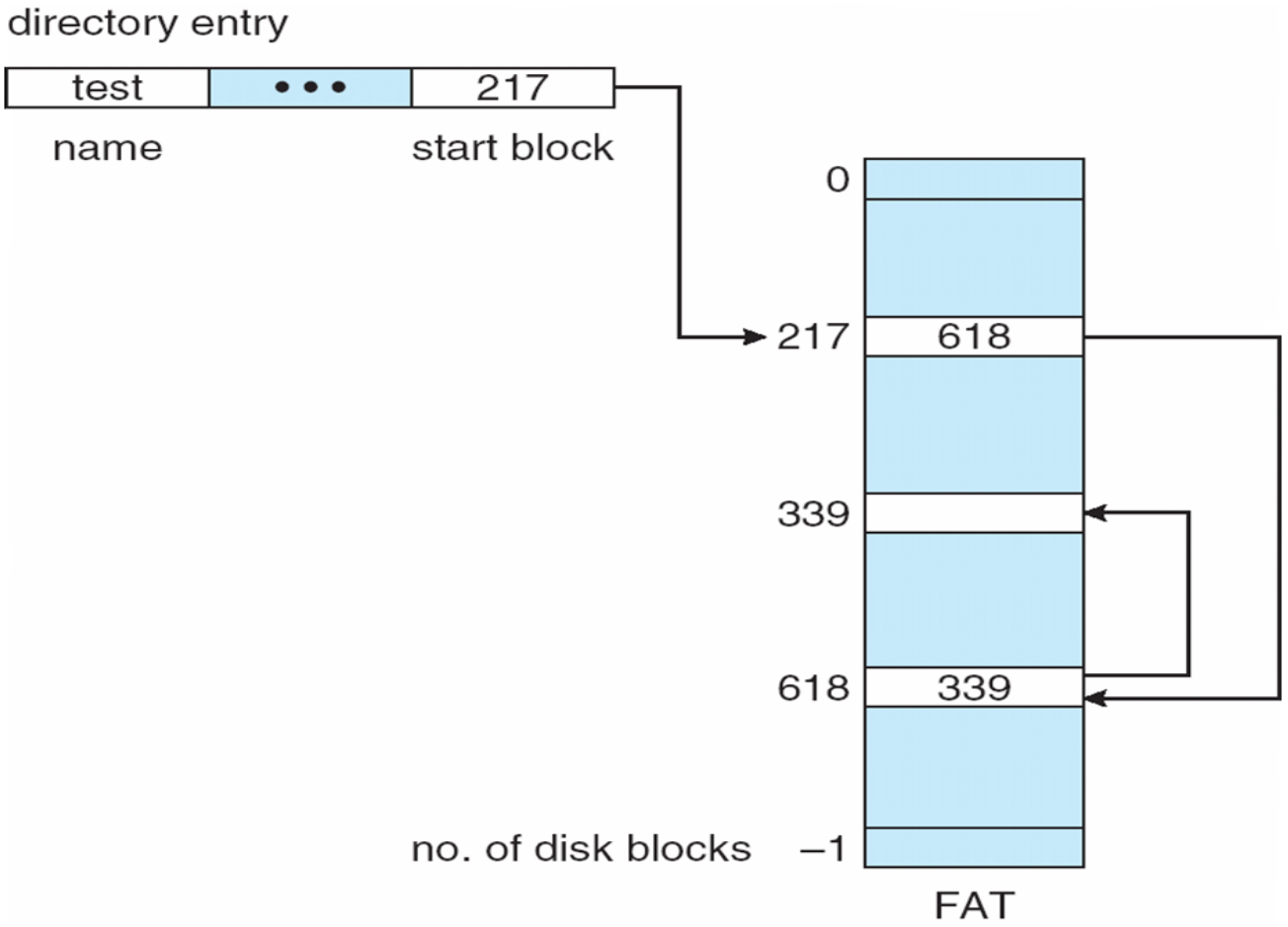
  2) How information about which clusters/blocks a file uses is kept.

  3) What do directory/folder entries contain?

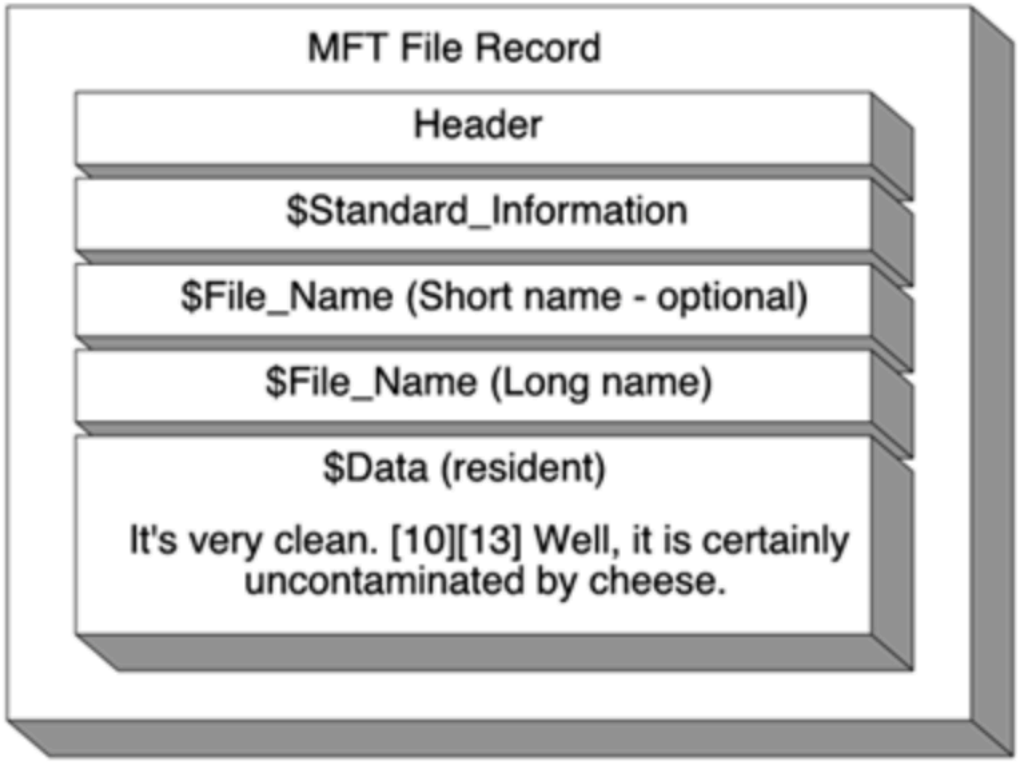
FAT32 & NTFS Partition



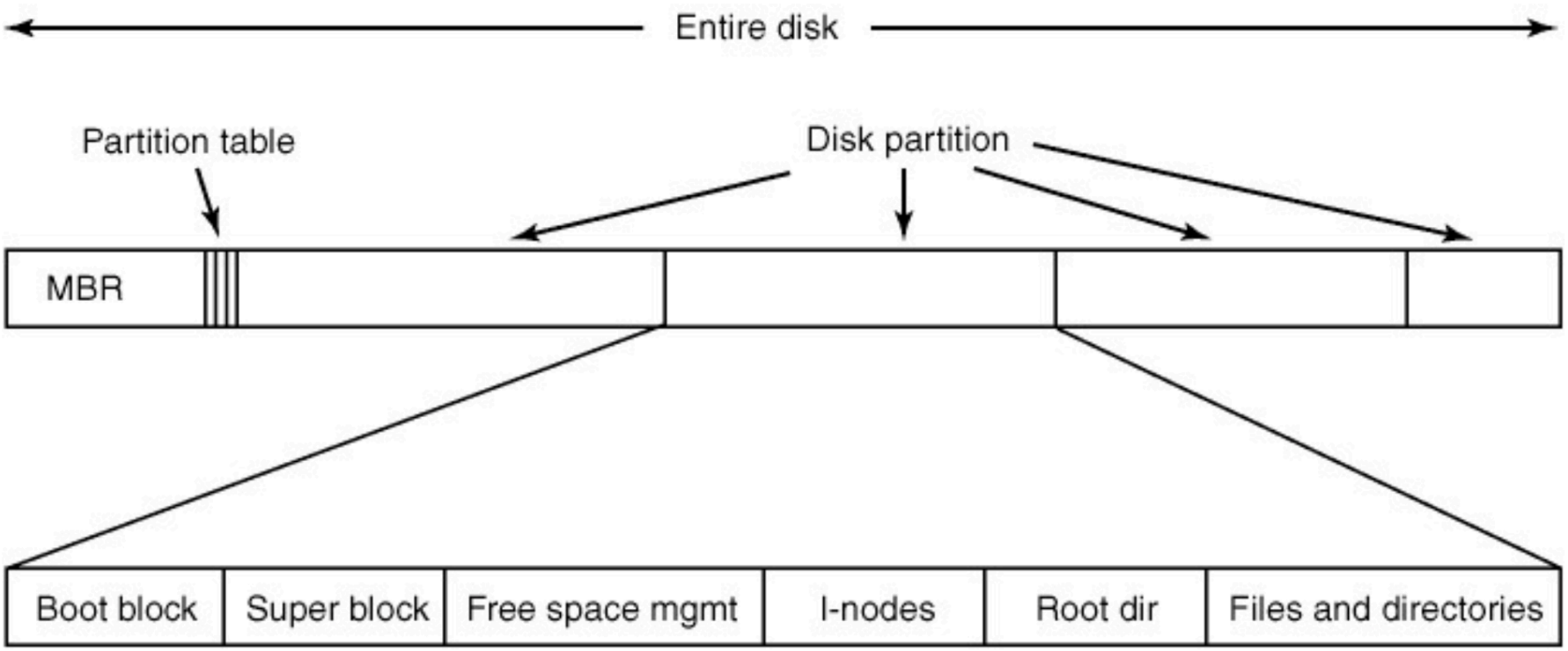
Linked Allocation: MS-DOS FAT32



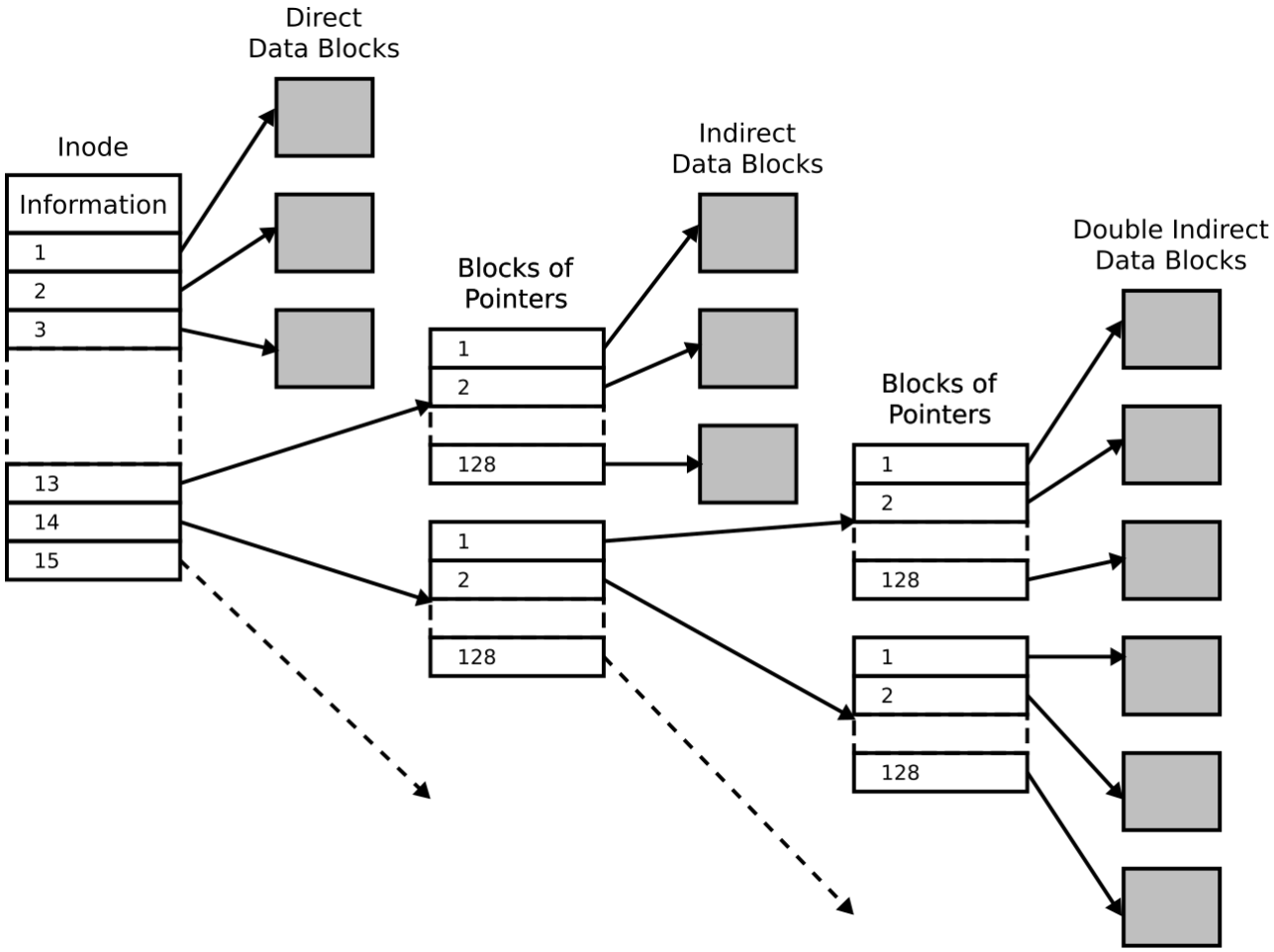
NTFS


MFT File Record
Header
$Standard_Information
$File_Name (Short name - optional)
$File_Name (Long name)
$Data (resident)
It's very clean. [10][13] Well, it is certainly uncontaminated by cheese.

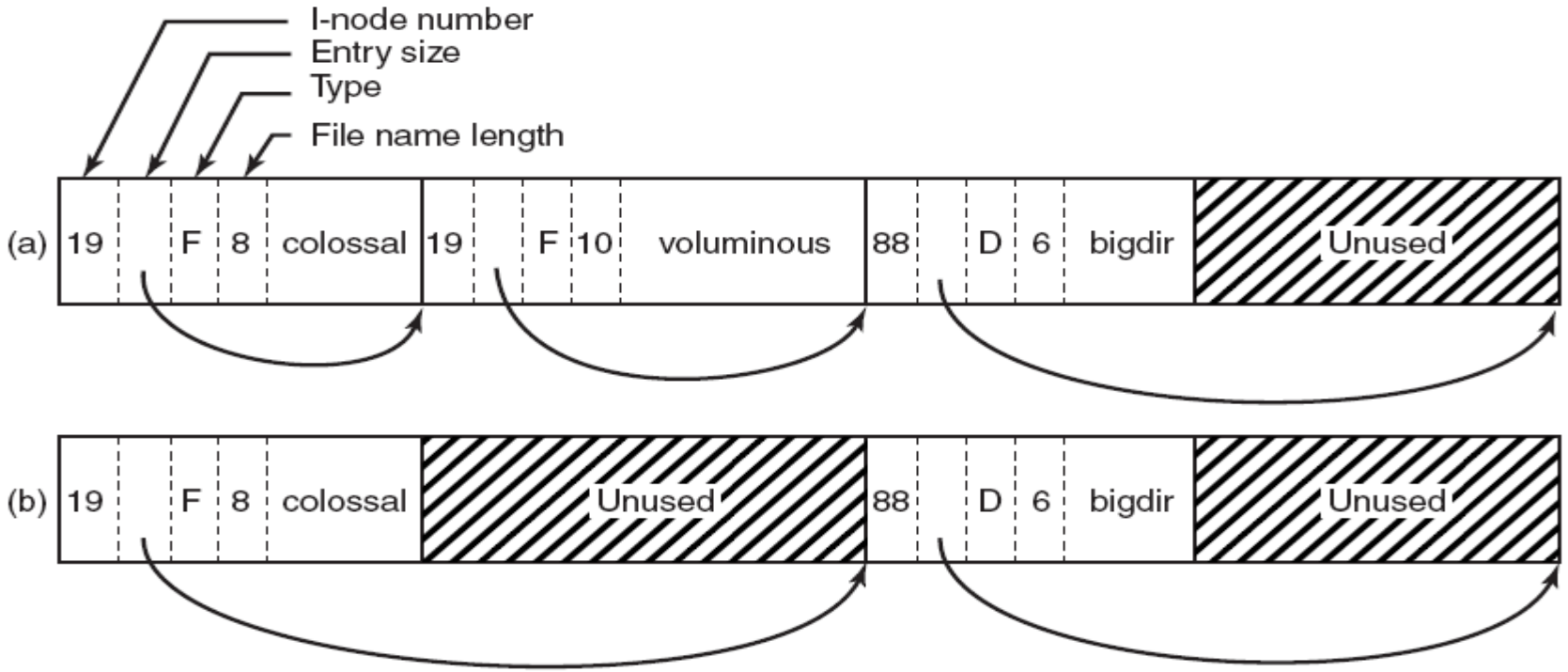
EXT


Entire disk
Partition table
Disk partition
MBR
Boot block
Super block
Free space mgmt
I-nodes
Root dir
Files and directories

Indexed Allocation: The *nix i-node



Example Ext2 Directory Structure



(a) A Linux directory with three files. (b) The same directory after the file voluminous has been removed.

Free-Space Management

- File system maintains list of unallocated blocks
  - Bit vector or bit map (1 bit per block)
  - List of ranges of empty blocks
  - List of pairs containing address of first empty block in a range, plus count of empty blocks after it
    - Note: “list” not referring to the list data structure -- these “lists” are often loaded into balanced tree data structures at runtime
    - Note: doesn’t have to be across the full disk -- divide and conquer to the rescue…

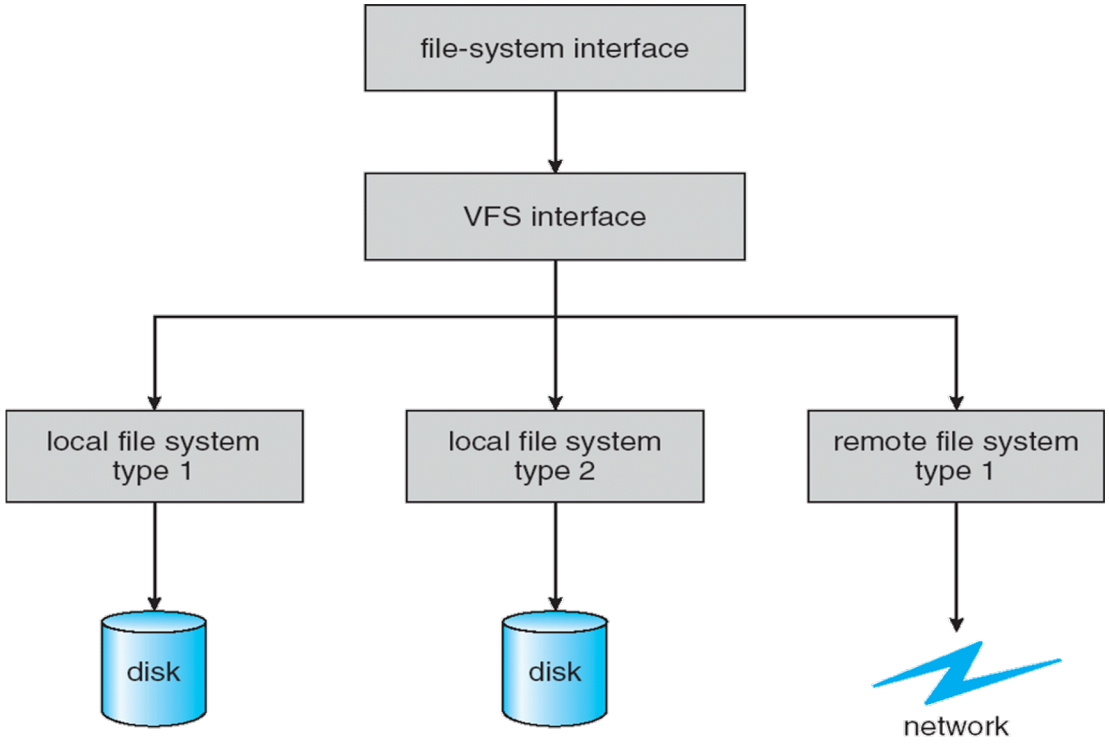
Disk I/O is the worst…

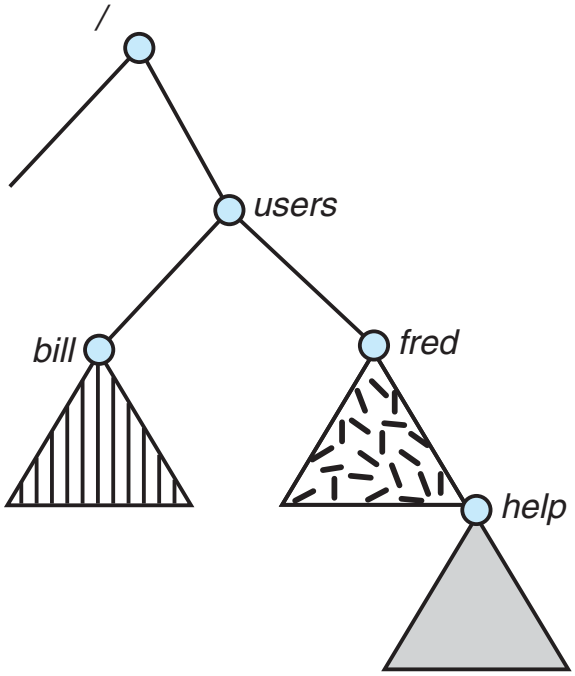
- Sequential I/O can be fast, but still several orders of magnitude slower than RAM
- Random I/O is slow (often by several orders of magnitude, compared to sequential I/O)
- How can we make this work in practice?
  - Reduce preliminary seeks
    - Keep data and metadata "close" together
  - Reduce data access seeks
    - E.g., journaling, log-structured merge trees (LSM), etc. ➔ Turn random I/O to sequential I/O plus indexing
    - Allow space for files to grow ➔ alleviate fragmentation
  - Caching (buffering)
    - Read-ahead (fetch and cache blocks before you need them)
    - Free-behind (remove a block from cache as soon as process moved on to next block)

Access Types

- Sequential vs Random
  - Sequential
    - Start from point A in a file, read/write all data until point B
  - Random
    - Read/write from/to random positions in the file -- i.e., seek to a position, read/write, seek to another position, read/write, etc.
  - Could be a mix of both
    - Seek to a position, then read several bytes sequentially, etc.
- Buffered vs Unbuffered
  - Unbuffered:
    - Read/write bytes as needed
  - Buffered:
    - Use part of memory (buffer) to store data coming from/going to the disk
    - Read from the disk more than needed and store in buffer -- subsequent reads don't need to go to disk, until the buffer is exhausted
    - For writes, store data in the buffer and write it out to disk when the buffer is full or after some time
    - Result: disk I/O is amortised across time and requests

Virtual Filesystem





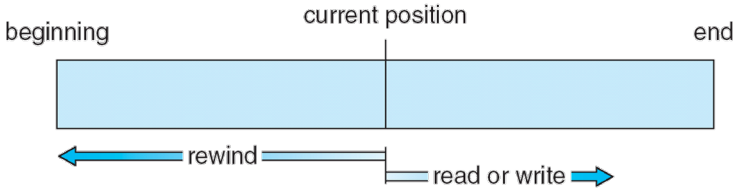
Logical Filesystem

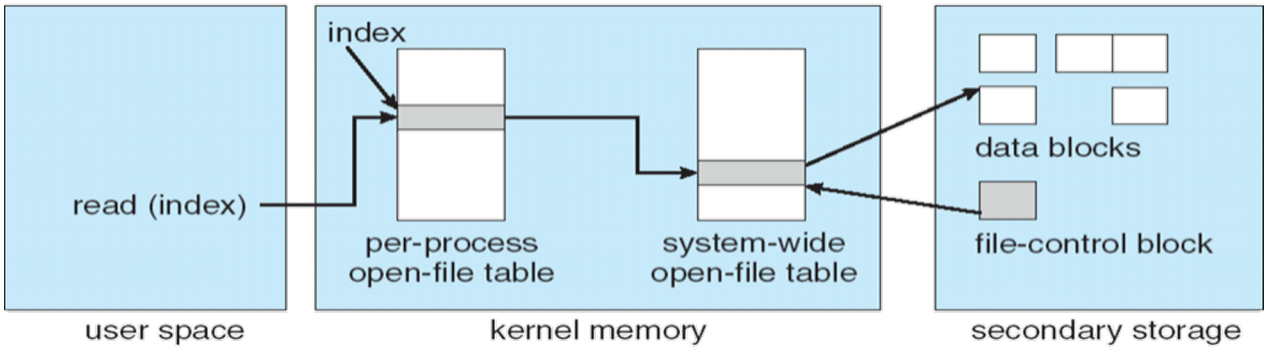
- File structure represented via *File Control Blocks*
- Each FCB (one per file) contains:
  - Access control
    - Owner/group/etc. IDs
    - Access rights descriptors
      - *nix: 9 bits -- owner R/W/X, group R/W/X, others R/W/X
      - E.g., `rw-r--r--` (=0644 (octal)); `rwx------` (=0700)
  - Dates
    - Creation, last access, last modification, etc.
  - Size of file (in bytes)
  - Location and organization of blocks of the file
    - i-node for *nix, location of first block for FAT, etc.
  - …

- Note: in *nix, *everything is a file*
  - Files, directories, devices, sockets, …

Logical Filesystem

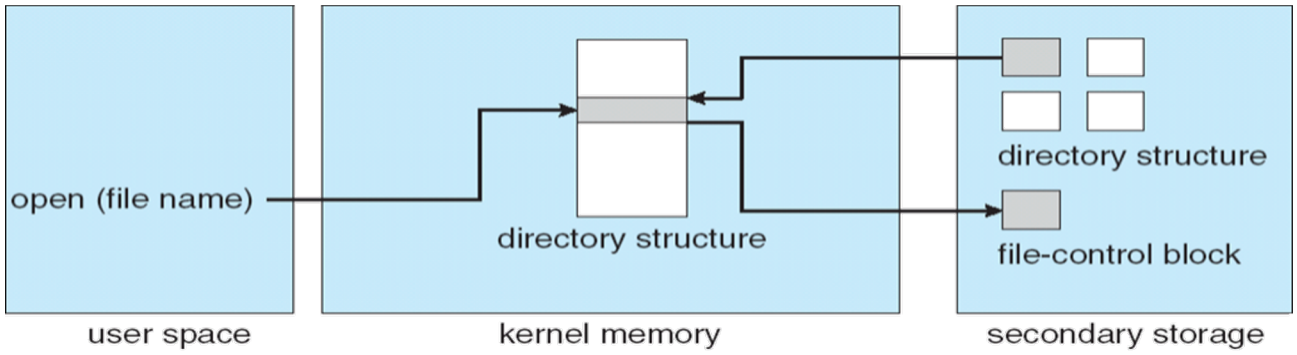
- Several pieces of data are needed to manage open files
  - Open-file table
    - Contains file control blocks (FCBs) of open files
    - Tracks open files
    - One global plus one per process
    - Per-process table (aka the *file descriptor table -- fdtable)* entries contain pointer to global fdtable ones
  - File pointer
    - Pointer to last location read/written in a file
    - Per file and per process that has the file open (i.e., part of the per-process fdtable)
  - File-open count
    - Counter of number of times a file is open
    - Per file (i.e., part of the global fdtable)
    - Allows removal of fdtable entry from global fdtable when last process closes it



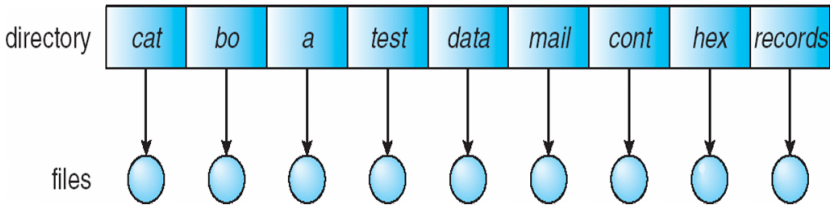


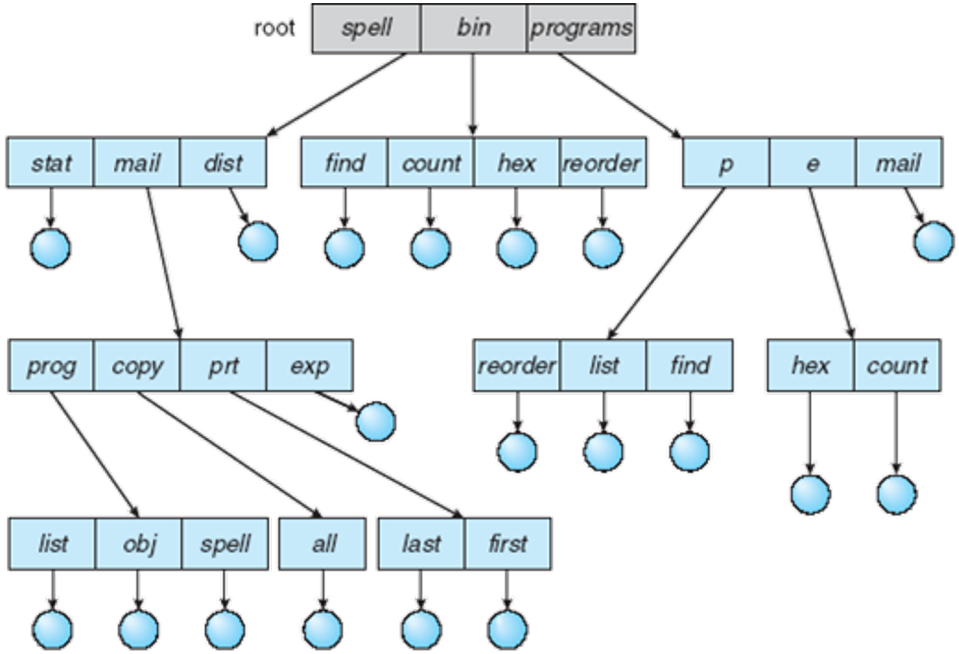
Logical Filesystem -- Directory Structure

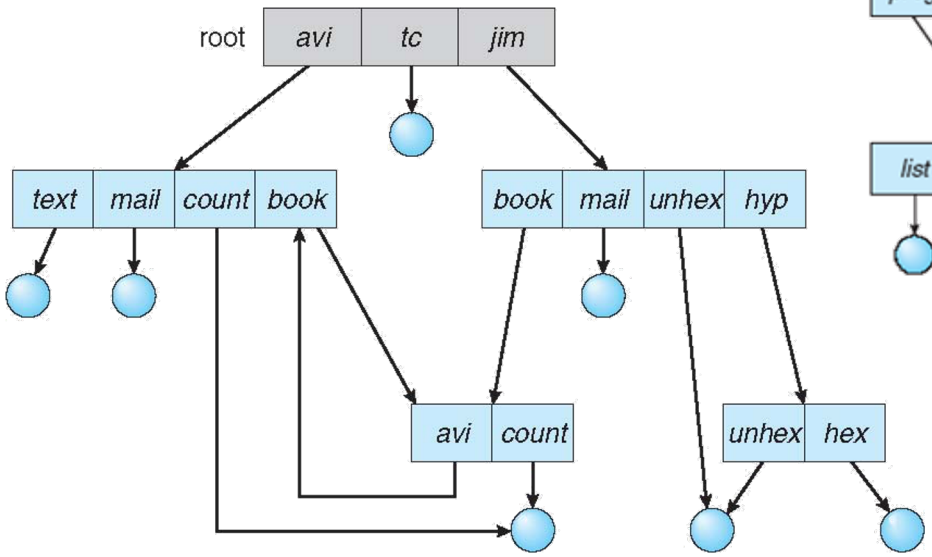
- What is a directory then?
- Just a file…
  - Contains pairs of type {*file name*, *FCB block location*}
  - Entries can point to files, other directories, etc.
  - Entries may be shared across directories ➔ *hard links*
    - E.g., *dir1/foo* and *dir2/bar* could point to the same file (FCB block location)
    - *Soft/symbolic link* = a special file containing a pointer (i.e., the path) to another file



Directory Hierarchy/Organisation







Recommended Reading

- Silberschatz, Galvn and Gagne, *Operating Systems Essentials, Chapter 11 , Sections 11.1, 11.2, 11.4, Chapter 10, Sections 10.1, 10.2, 10.4*