The UNIX Time-Sharing System

Dennis M. Ritchie and Ken Thompson
Bell Laboratories

Communications of the ACM
July 1974, Volume 17, Number 7

UNIX overview

- Unix is a general-purpose, multi-user, interactive operating system

- Originally developed for DEC PDP-7, -9, and -11 computers

- PDP-11/45
  - 16-bit word (8-bit byte) computer
  - 144KB main memory
  - UNIX occupies 42KB
  - 1MB fixed head disk
  - Four 2.5MB removable disk cartridges
  - One 40MB disk pack
  - Also Picturephone, voice response, voice synthesizer, photo typesetter, digital switching network etc.

- Written in C language

- Now widely supported across almost all hardware platforms in various variants
  - System V, FreeBSD, Linux, Solaris etc.

Major Innovations

- Hierarchical file system
- Compatible file, device, and inter-process I/O
- Background (asynchronous) and foreground processes
- Interactive Shell

Original use around 1969-70s

- “preparation and formatting of patent applications and other textual material, the collection and processing of trouble data from various switching machines within the Bell System, and recording and checking telephone service orders”

Now

- Widely used in
  - Servers
  - Desktops
  - Laptops, Netbooks
  - PDAs, phones
  - Mainframes
  - Embedded systems
  - Real-time systems
  - Switches, routers
  - Almost any computing platform you can think of

- "*Perhaps paradoxically, the success of UNIX is largely due to the fact that it was not designed to meet any predefined objectives.*"

Inexpensive!

- "UNIX can run on hardware costing as little as $40,000"

- "Less than two man years were spent on the main system software."

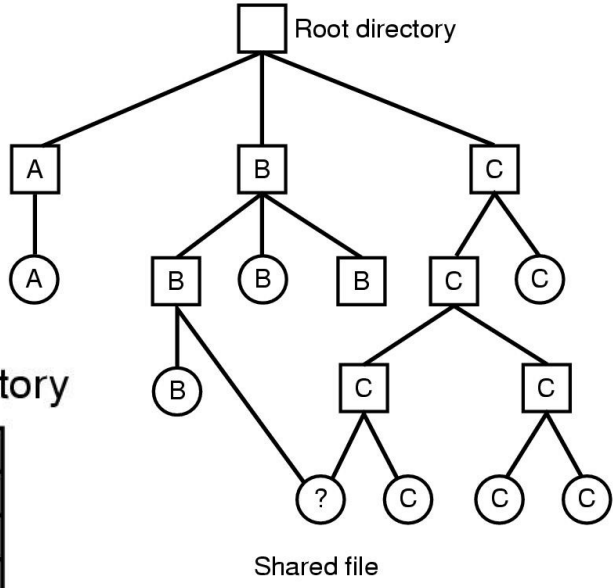
- "users of UNIX will find that the most important characteristics of the system are its simplicity, elegance, and ease of use."

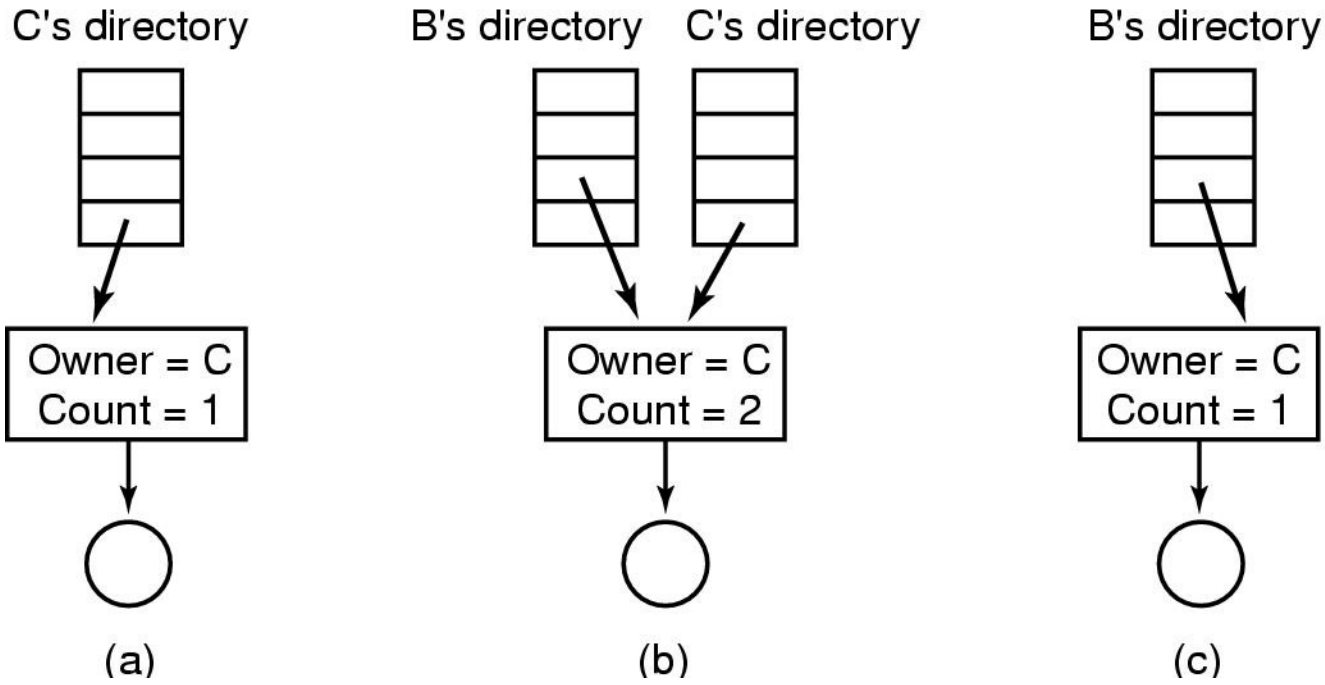
File System

- "Most important role of UNIX is to provide a file-system".
  - Why?

- Three types of files
  - Ordinary files
    - No particular structure imposed by OS
  - Directories
    - Mapping between filenames and files
  - Special files
    - I/O devices

Shared Files – Hard Links





prior to linking

After the hard link is created

After the original owner removes the file

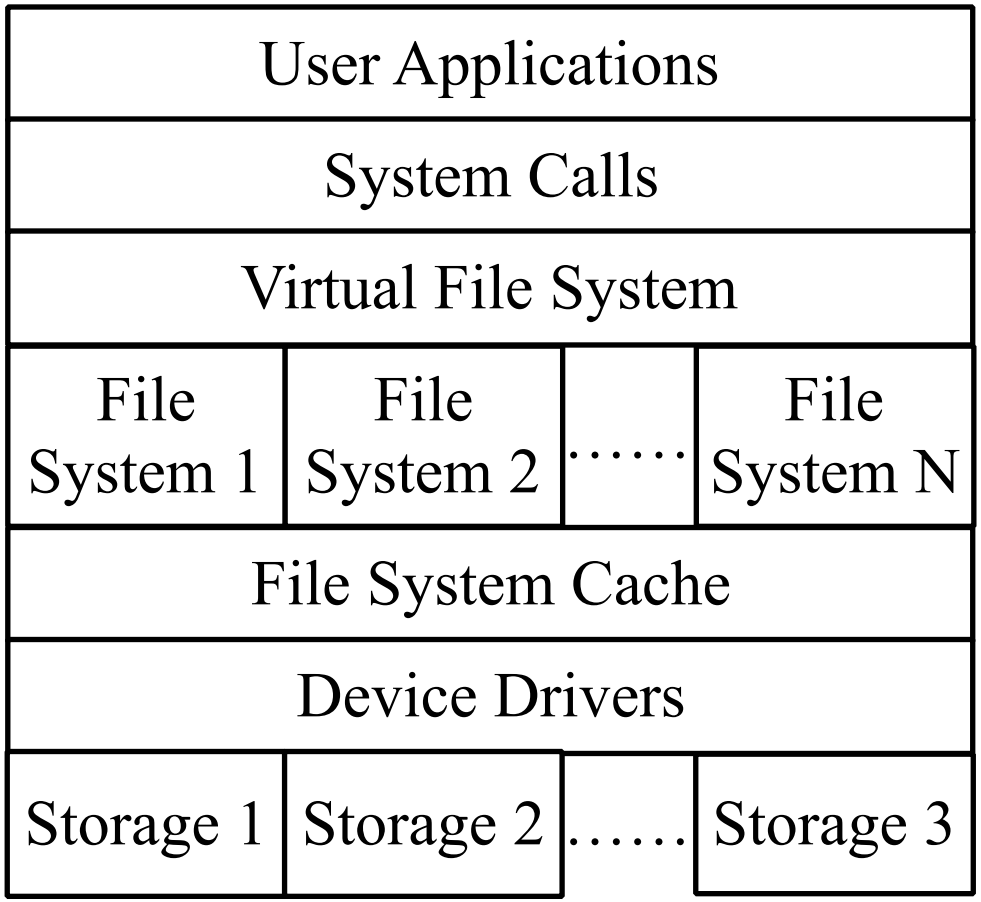
File still exists, but with count decremented

What is a File System?

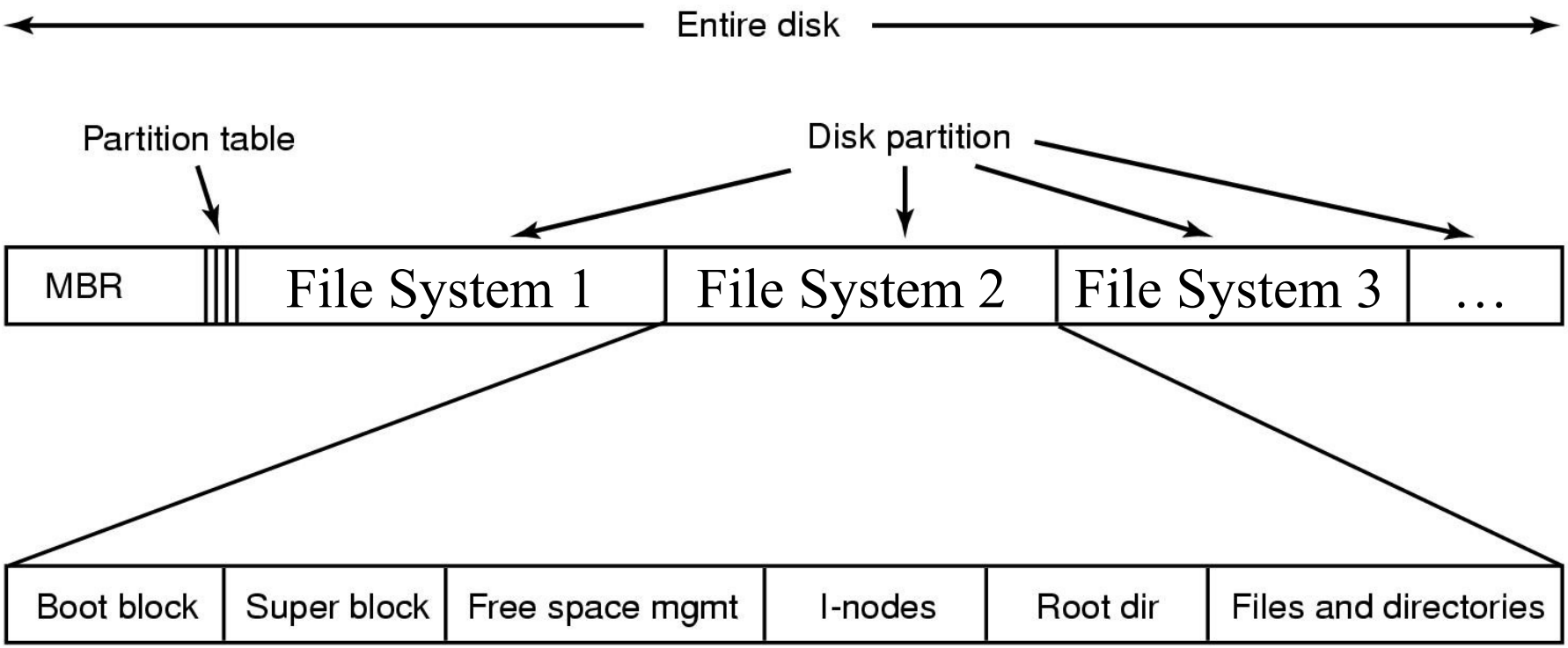
- File system is the OS component that organizes data on the raw storage device.
- **Data**, by itself, is just a meaningless sequence of bits and bytes.
- **Metadata** is the information that describes the data.
  - Also called attributes.
  - Without meta-data, data itself will be incomprehensible.
- Responsibilities of a file system
  - Defines the format of the data objects.
  - Defines the format & meaning of meta-data associated with each data object. E.g. File name, permissions, and size of a file.
  - Manages the location of the individual data blocks of each data object.
  - Manages free space on the storage device.

Virtual File System (VFS)

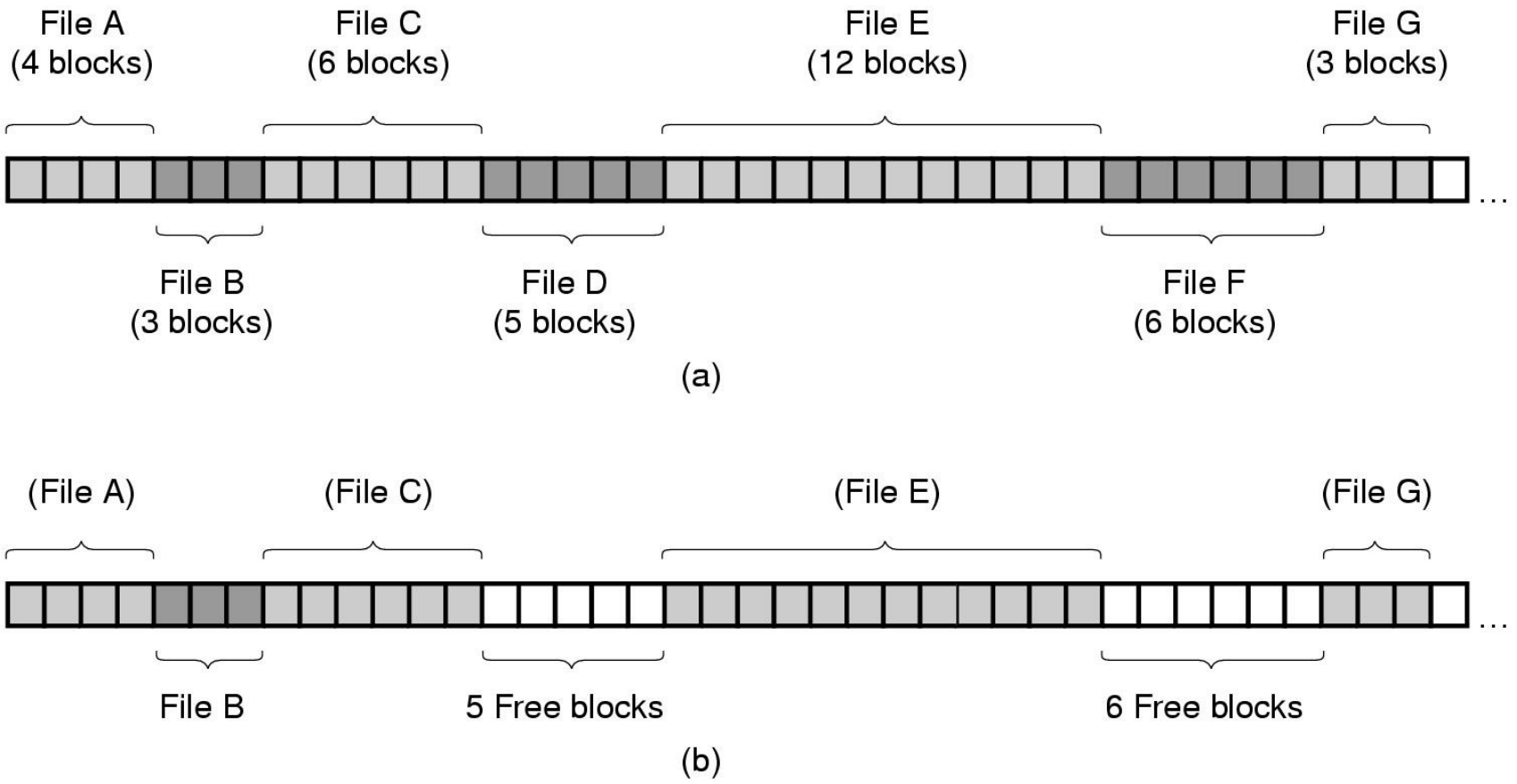
- VFS provides
  1. A common system call interface to user applications to access different file systems implemented in the OS.
  2. A common interface to file systems to "plug into" the operating system and provide services to user applications.



Partitions and File-system Layout



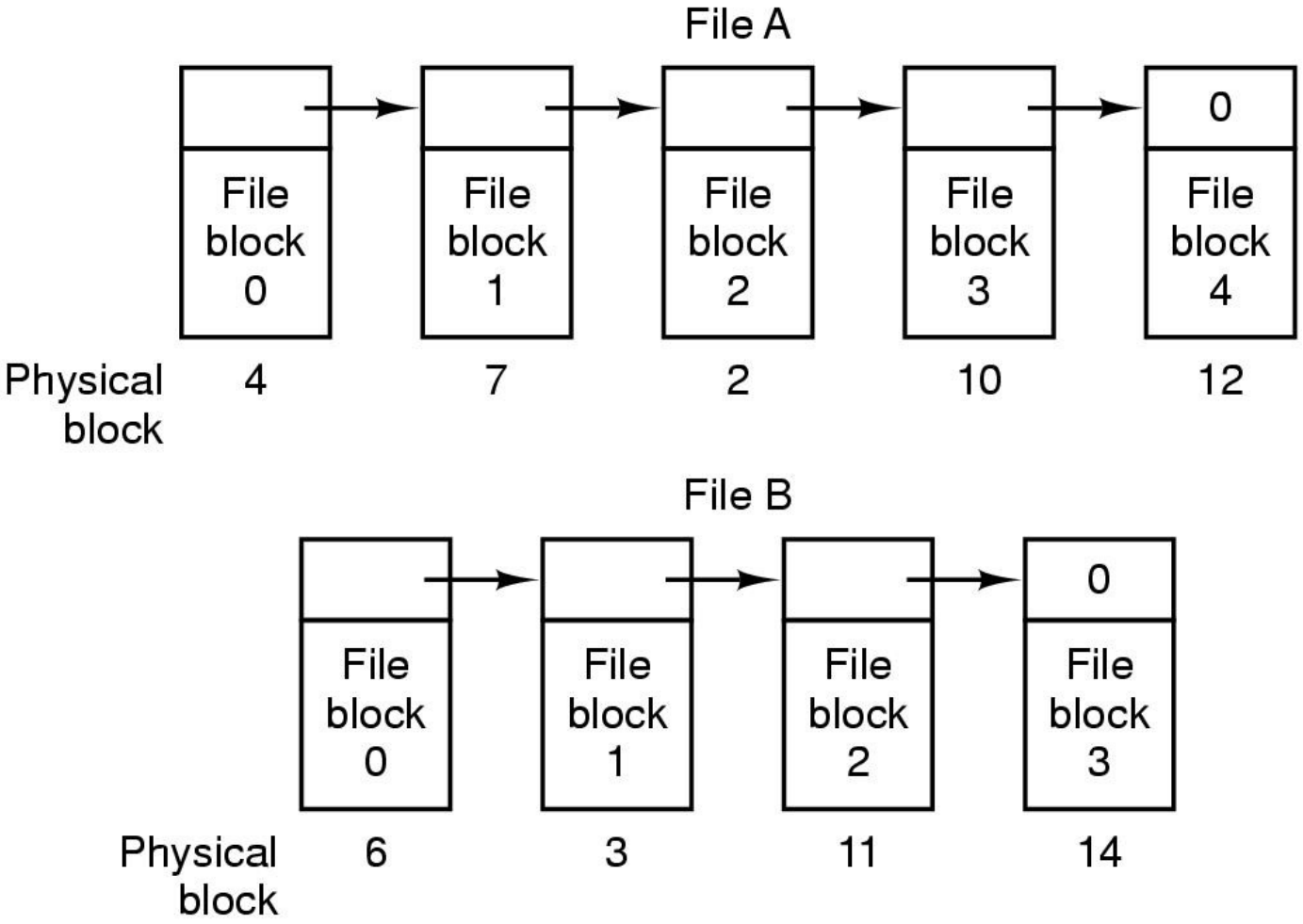
Organizing Files on Disk: Contiguous Allocation



(a) Contiguous allocation of disk space for 7 files

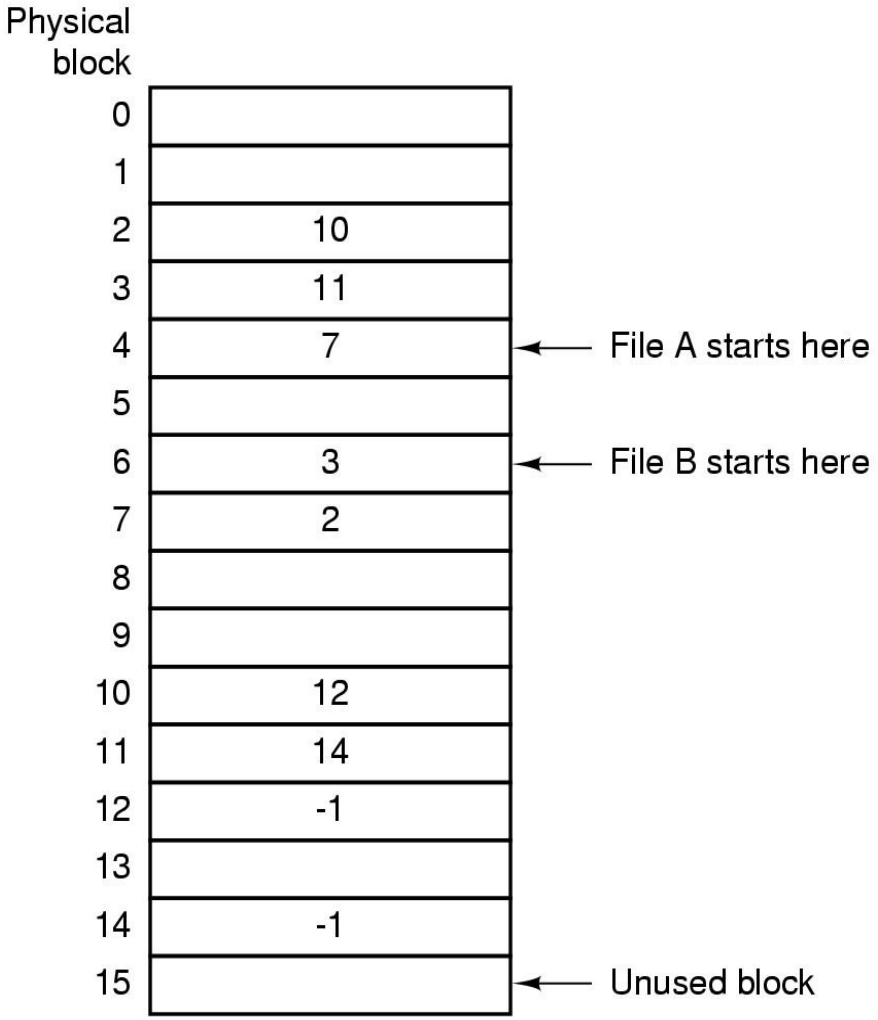
(b) State of the disk after files D and E have been removed

Organizing Files on Disk: Singly Linked List of Blocks



- Advantage: Logically contiguous blocks can be discontiguous on disk
- Disadvantage: Random seeks are expensive. Requires traversal from the start.

Organizing Files on Disk: File Allocation Table



- Linked list allocation using a file allocation table in RAM
- Doesn't need a separate "next" pointer within each block
- But random seeks are still expensive

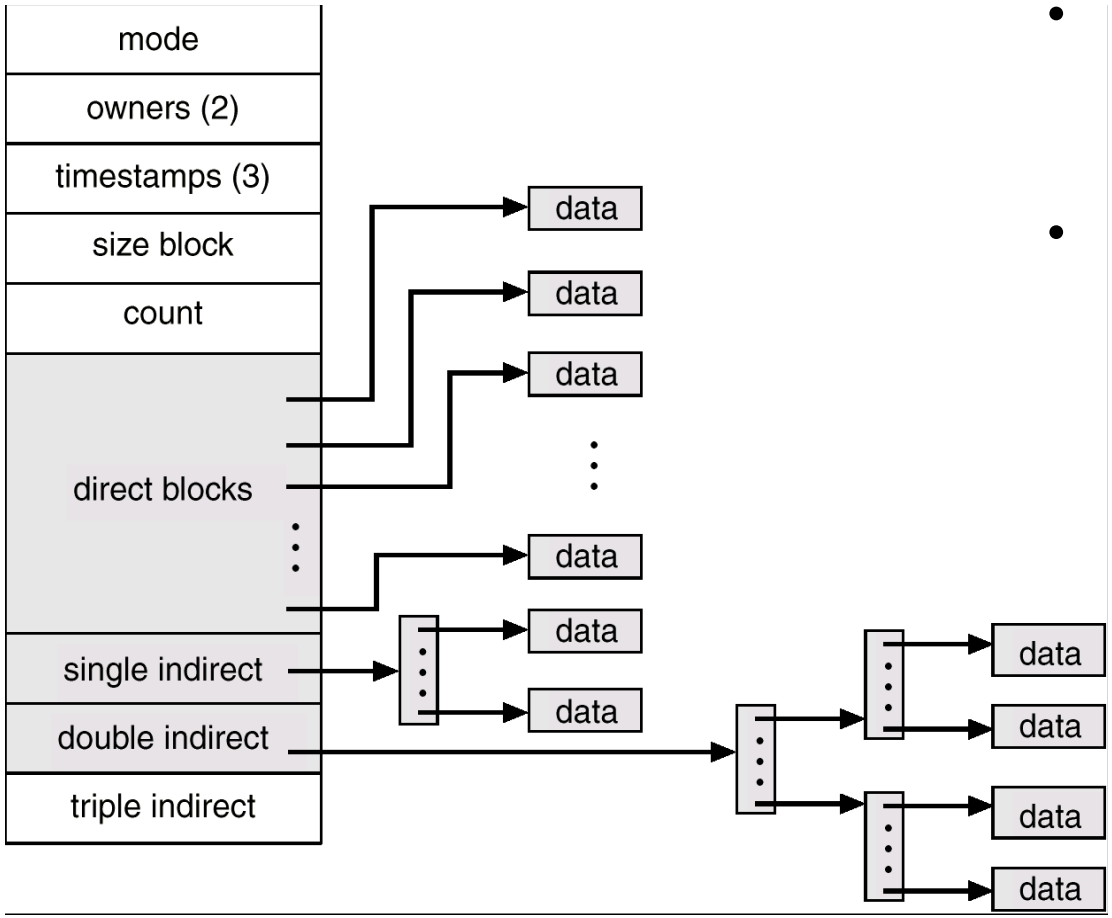
i-nodes (index nodes)

- Each file is described by an i-node
- i-node stores the metadata for the file
- Metadata = file info + location of data blocks
- i-nodes are stored on the disk

- An i-node is to a file what a page-table is to a virtual address space

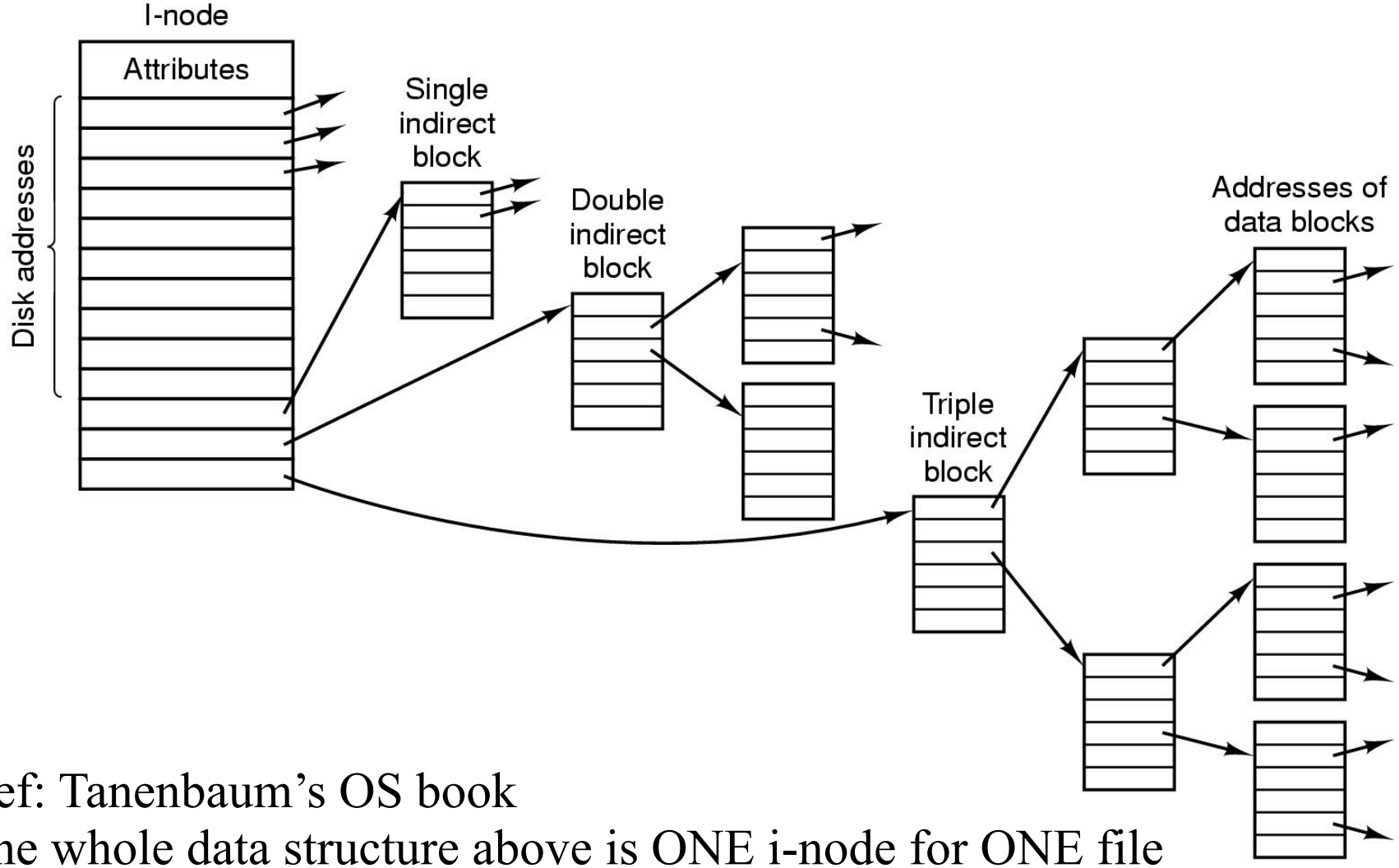
  - Page table maps
    - virtual page number in a virtual address space ——> physical page frame number in DRAM
  - i-node maps
    - logical block number in a file ——> physical block location on disk

Unix i-node (index node)



- Small files can be accessed quickly
- If each block is 4KB
  - First 48KB of file reachable from 12 direct blocks
  - Next 4MB available from single-indirect blocks
  - Next 4GB available from double-indirect blocks
  - Next 4TB available through the triple-indirect blocks
- Any block can be found with at most 3 disk accesses

Another view of a UNIX i-node



- Ref: Tanenbaum's OS book
- The whole data structure above is ONE i-node for ONE file
- i-node grows dynamically as the file grows
- Just like page-tables, i-node tracks a giant array broken up into many pieces

Special files

- “most unusual feature”
- Located in /dev
- Just like ordinary files.
- But read/write requests result in activation of the associated device.
- Advantages of treating I/O devices as files:
  - File and device I/O are similar
  - File and device names have same syntax and meaning.
    - Programs that operate on files can also operate on devices.
  - Same protection mechanisms as regular files

Removable file system

- Different parts of filesystem can be on different devices

- *mount*
  - Allows a storage device to be referenced as a subtree of existing rooted filesystem.

- Hard links across mounted file systems disallowed. Why?

Protection

- Each user has a userid
- Files marked as owned with userid of the creator
- Seven protection bits
  - Six bits for read, write, and execute permissions for user, group, and others
  - 7^{th} bit $\rightarrow$ set-user-ID bit
    - Temporarily change the userid of current user to the owner when file is executed as a program.
    - Allows the safe use of privileged programs that require access to special system files (e.g. system logs).
    - Actual userid of the invoker is available to the program for credential checks.

I/O calls

- filep = *open* (name, flag)
- *Create* system call creates and opens a new file. Truncates to zero if file exists.
  - *filep* is a file descriptor
    - Notion of a file descriptor hasn't changed over the years

- No locking provided by OS for multi-user access
  - Lets users figure out synchronization.
  - Locks are "neither necessary nor sufficient". Why?
  - Internal OS locks for consistency of data structures

- n = *read*(filep, buffer, count)
  - Read returns 0 when end of file (EOF) reached
- n = *write*(filep, buffer, count)
  - Write beyond end of file grows the file automatically

- location = *seek*(filep, base, offset)
  - For random I/O

Processes and images

- processid = fork(label)
- filep = pipe( )
- execute(file, args, argo, ..., arg,+)
- processid = wait( )
- exit (status)

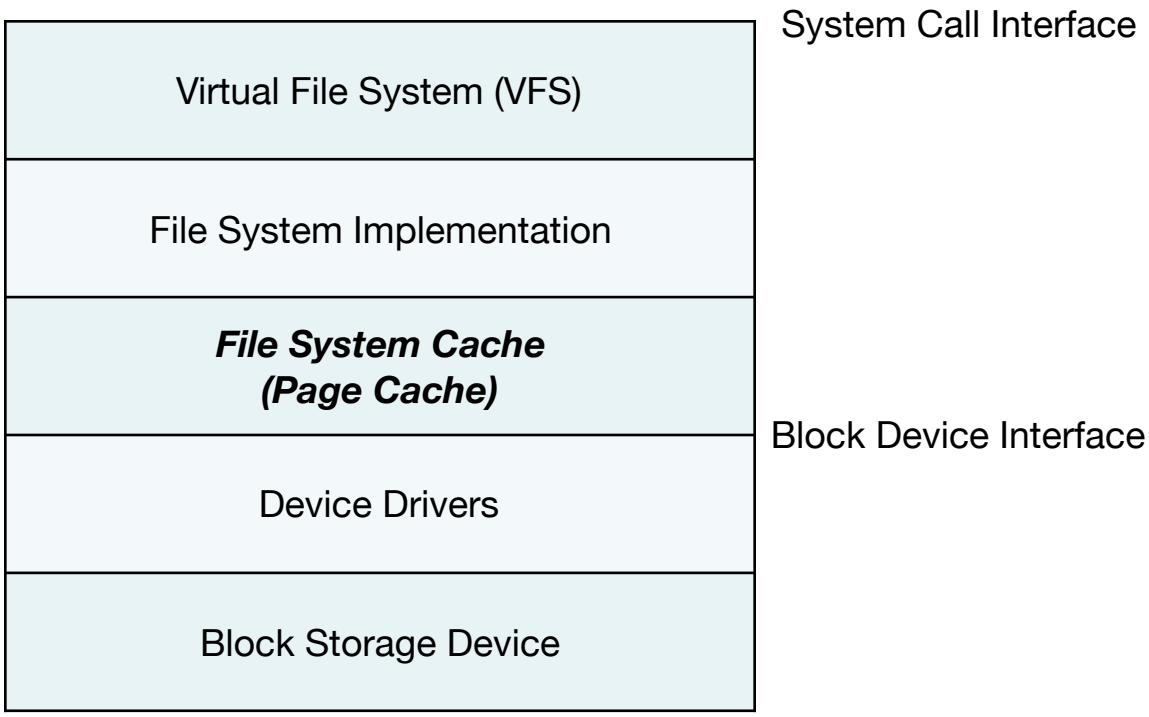
Shell

- Triggered upon login by the init process
  - Can be replaced by other commands
- command arg1 arg2 • • - argn
- ls > there
- ed < script
- Filters: ls | grep foo | wc -l
- Multitasking
  - ls; ed
  - as source > output &
  - as source > output & ls > files &
  - (date; ls) > x &

read/write buffering

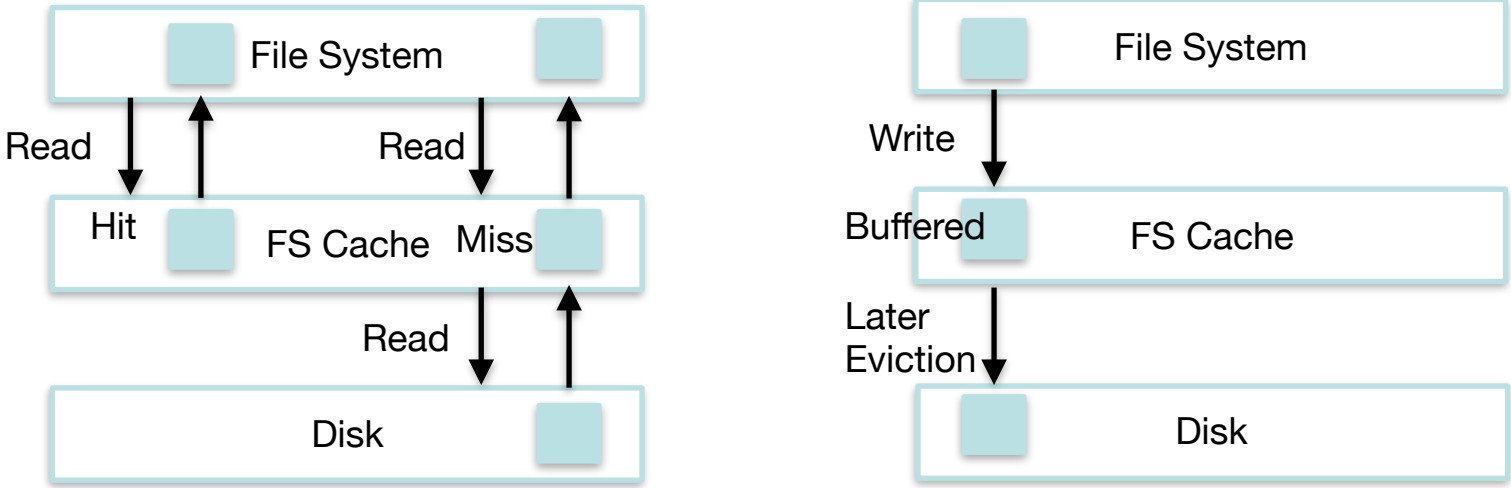
- System buffers to hide block I/O activity
- Users can read/write at any byte granularity
- OS converts these to block-level I/O

File System Cache



- FS Cache is a part of main memory used to store frequently accessed data blocks from the disk

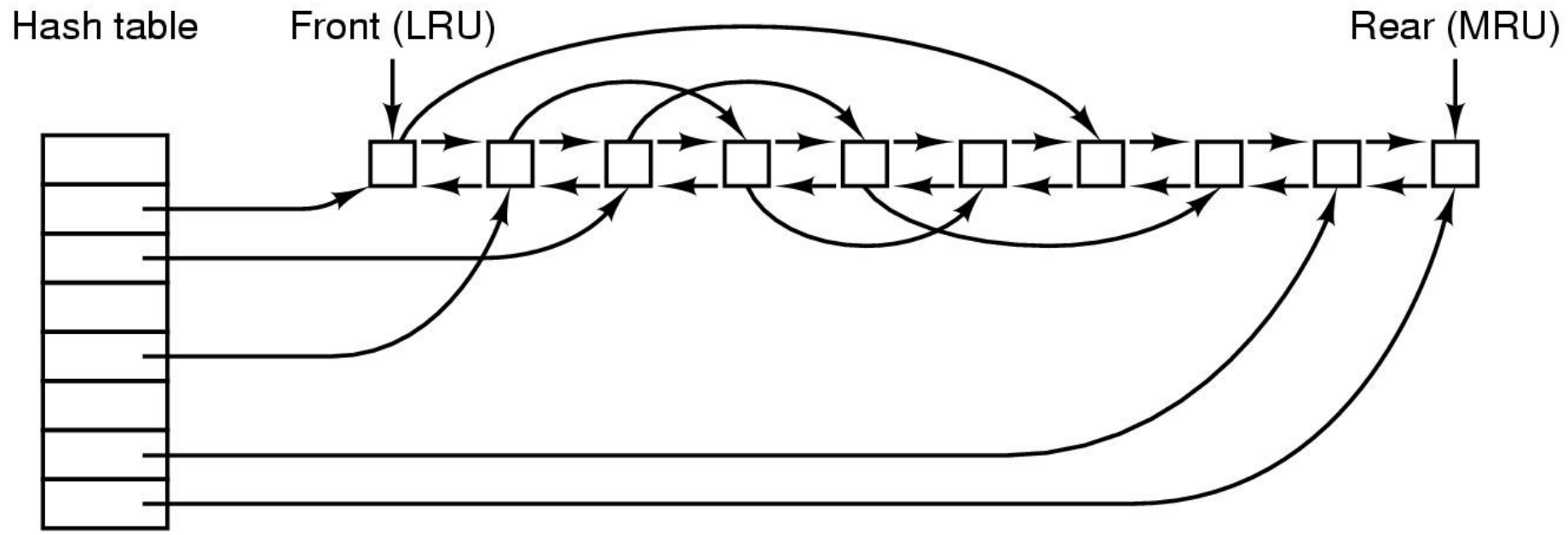
File System Cache- How it works



- How it works:
  - Before accessing the disk, look in the FS cache.
  - If data block is in FS cache, no need to go to disk.
  - Periodically, purge the cache of infrequently used data blocks.

- Claim: If the cache works well, then most I/O accesses to the physical disk will be writes. Why?

Data Structure for File-System Cache



- Cache Lookup:
  - Pages in cache are looked up via a hash table for fast access.
- Cache Eviction:
  - Another doubly-linked list maintained to identify least-recently used (LRU) victim pages that are periodically purged.
  - Is the victim page dirty? Then write to disk. Else discard.

Virtual memory page cache and FS cache

- Often they are managed in a unified manner
- Meaning: one common page-cache is used for managing pages for both virtual memory and file system.
- For example, Linux maintains one common set of data structures to keep track of active and inactive (LRU) pages.

Log-Structured File Systems

- With CPUs faster, memory larger
  - disk caches are also getting larger
  - increasing number of read requests come from file system cache
  - Thus, most disk accesses will be writes

- LFS treats the entire disk as a log
  - all writes are initially buffered in memory
  - periodically commit the writes to the end of the disk log
  - When file is opened, locate i-node, then find blocks

Quiz on Inodes

- All blocks in a disk are of size 4KB (4096 bytes). A disk block can store either data or metadata (but not both).
- Each block address, i.e. a block's location on the disk, is 8-bytes in size
- Assume that all file attributes other than data block addresses, take up negligible space in the top-level block of inode block.
- Last three entries of the top-level block of inode contain single, double, and triple indirect block addresses.
- Rest of the space in the top-level inode (between end of attributes and single-indirect block address) is used to store direct block addresses.

- Question 1: What is the largest size of a file that can be accessed through
  - direct block entries?
  - direct + single indirect block entries?
  - direct + single + double indirect block entries?
  - direct + single + double + triple indirect block entries?

- Question 2: What is the size of an inode (in bytes) for a 32GB file?