File Systems

Kartik Gopalan

Chapter 4

From Tanenbaum's Modern Operating System

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.

- A File System defines
  - Format of the data objects.
  - The format/meaning of meta-data associated with each data object. E.g. File name, permissions, and size of a file.
  - Location of the individual data blocks of for each data object.
  - A framework to manage free space on the raw storage.

Metadata — Examples

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File Types

- Regular files
  - Contains actual data

- Directories
  - Files that help locate other files

- Character Special Files
  - Typically for byte-oriented I/O operations

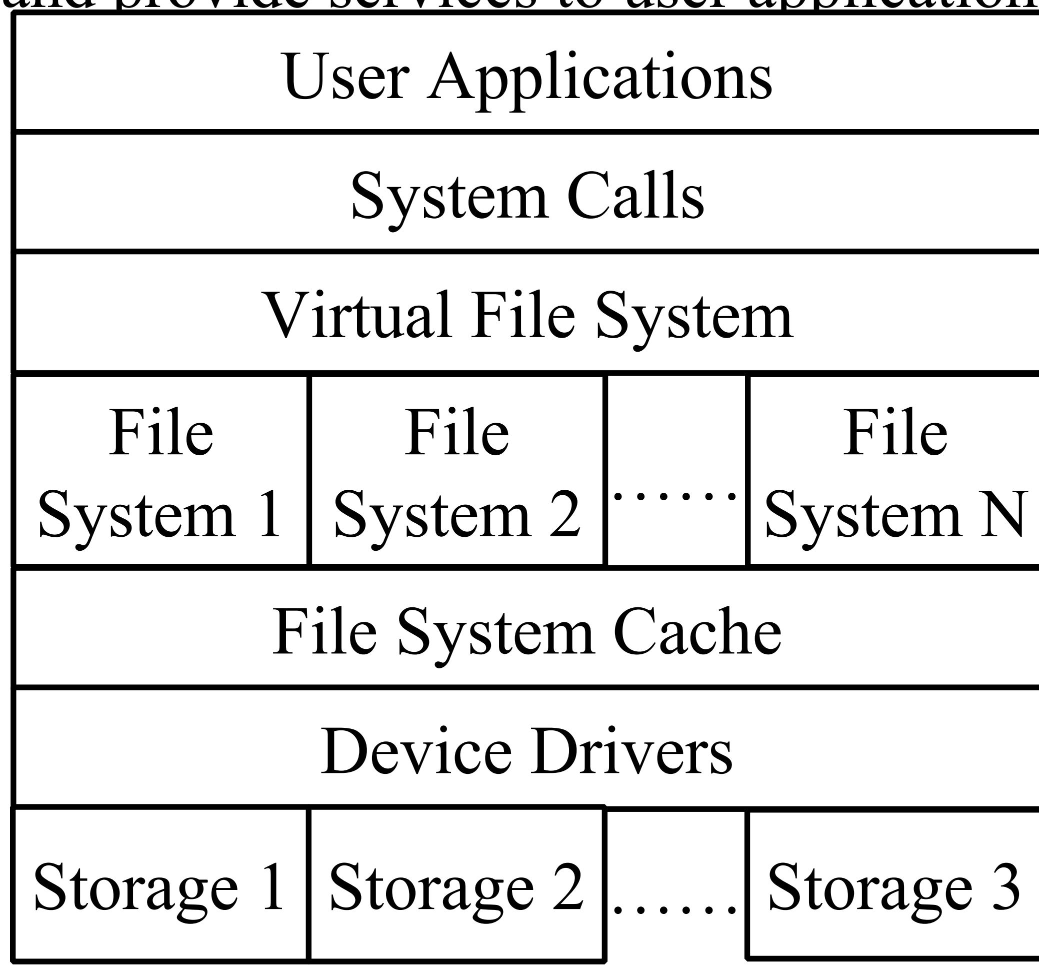
- Block special files
  - For block-oriented I/O operations

Basic File System Operations

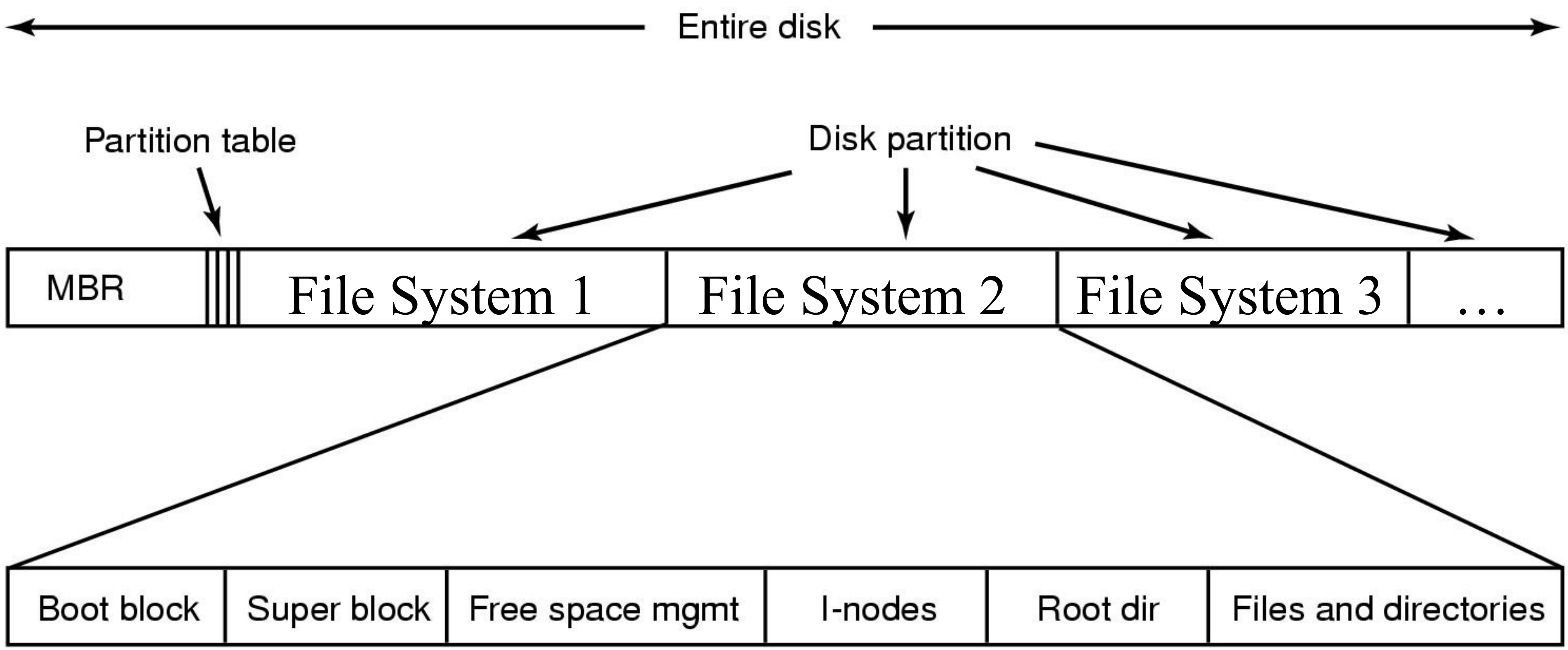
- Create a file
- Open an existing file
- Write to a file
- Read from a file
- Seek to somewhere in a file
- Close an open file
- Delete a file

Virtual File System (VFS)

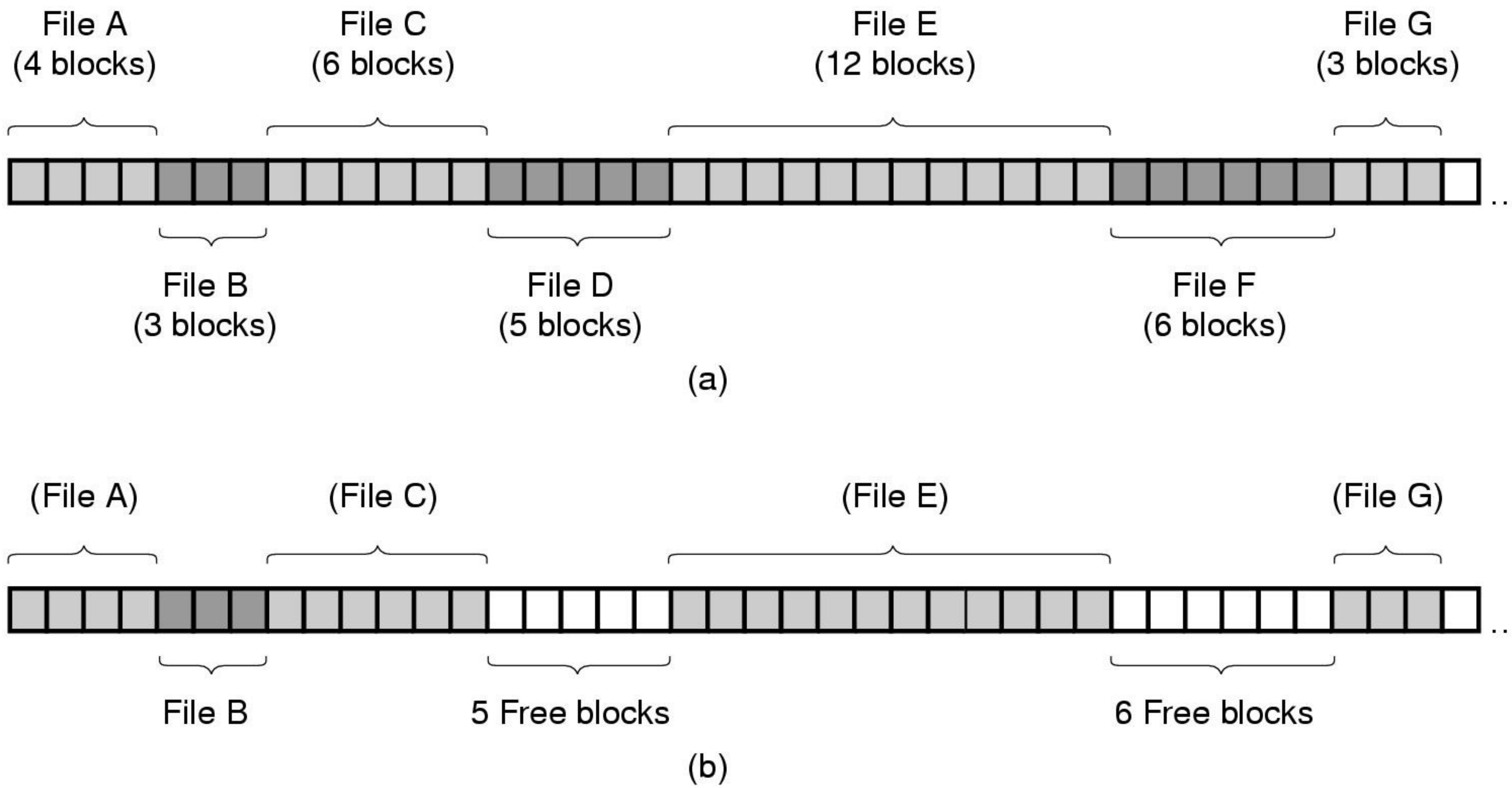
- VFS provides
  1. A common system call interface to user applications to access different file systems implemented in the OS.
  2. A common interface to file systems to “plug into” the operating system and provide services to user applications.



Partitions and File-system Layout



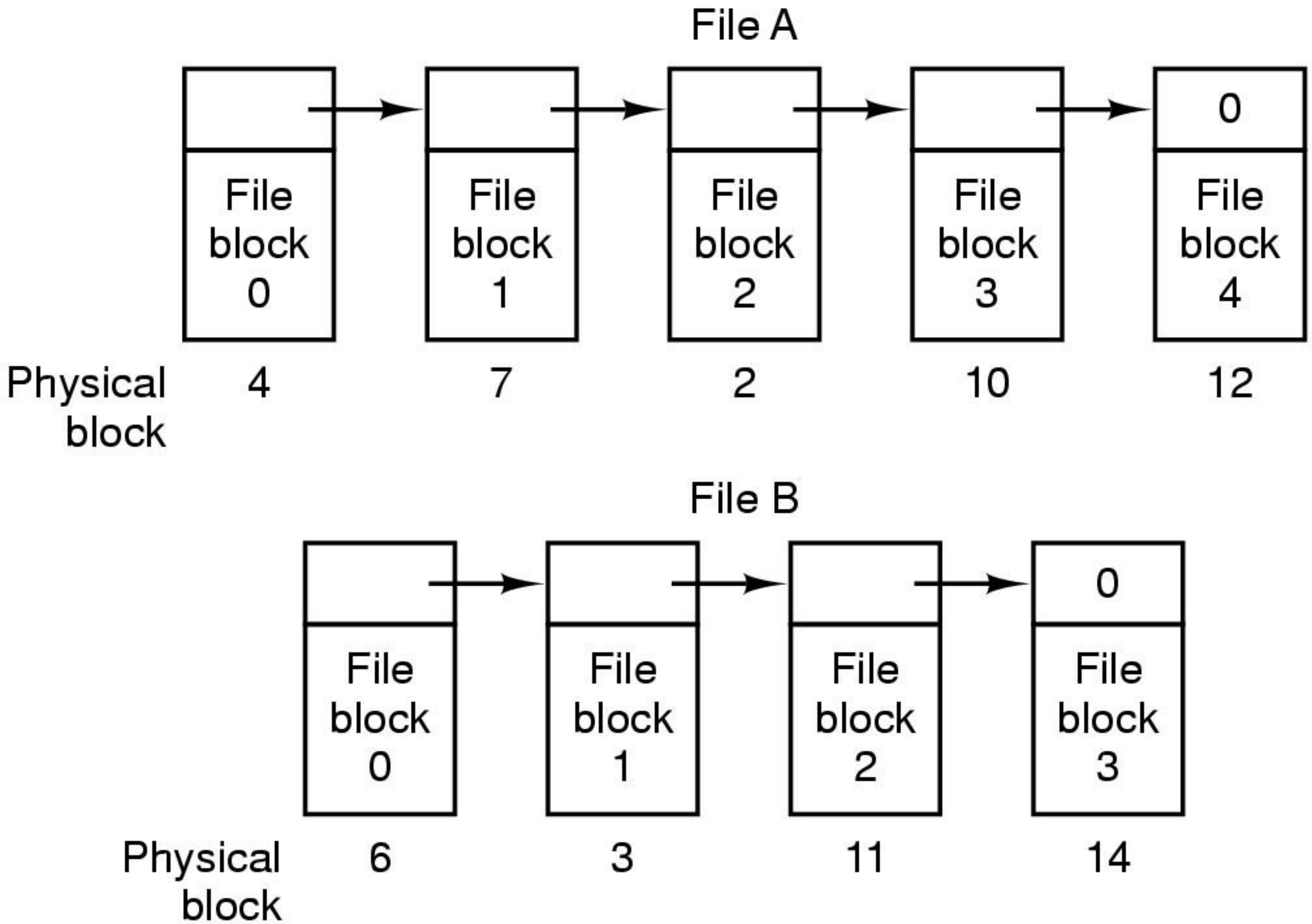
Organizing Files on Disk: Contiguous Allocation



(a) Contiguous allocation of disk space for 7 files

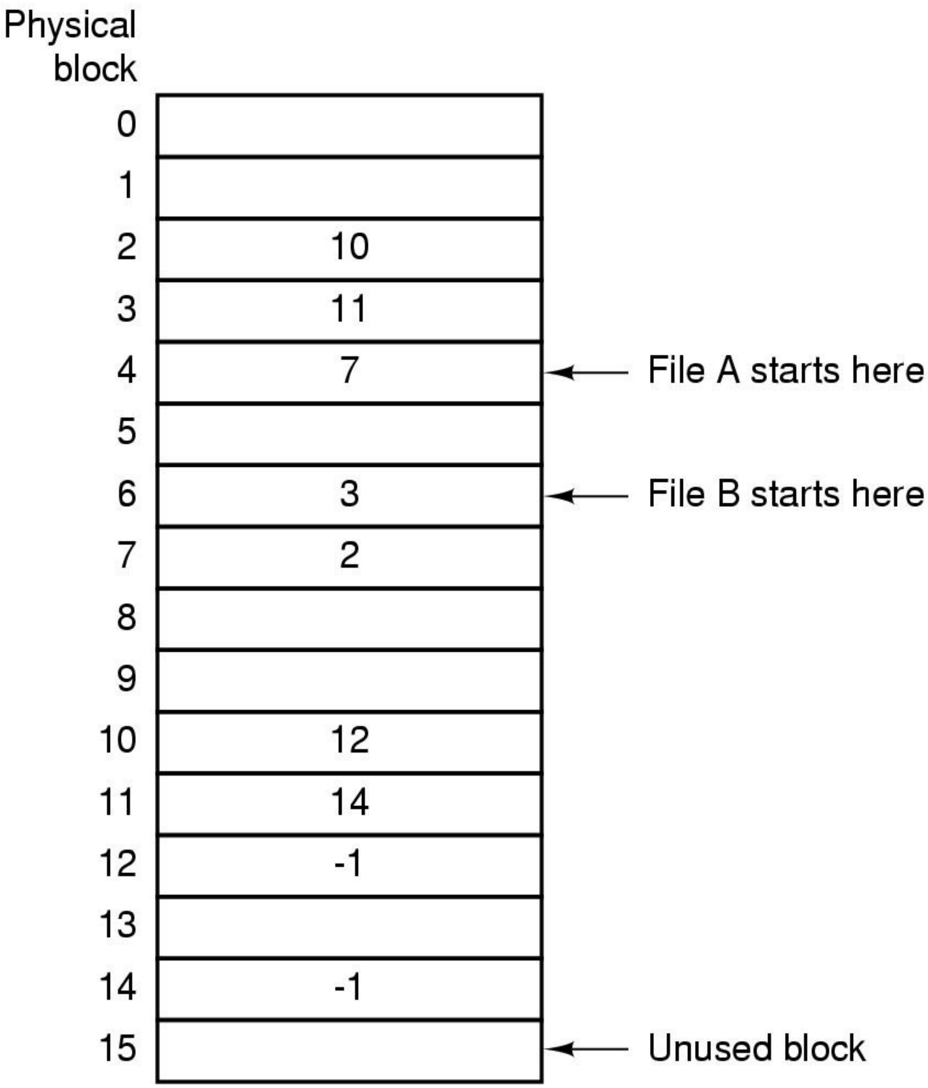
(b) State of the disk after files D and E have been removed

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

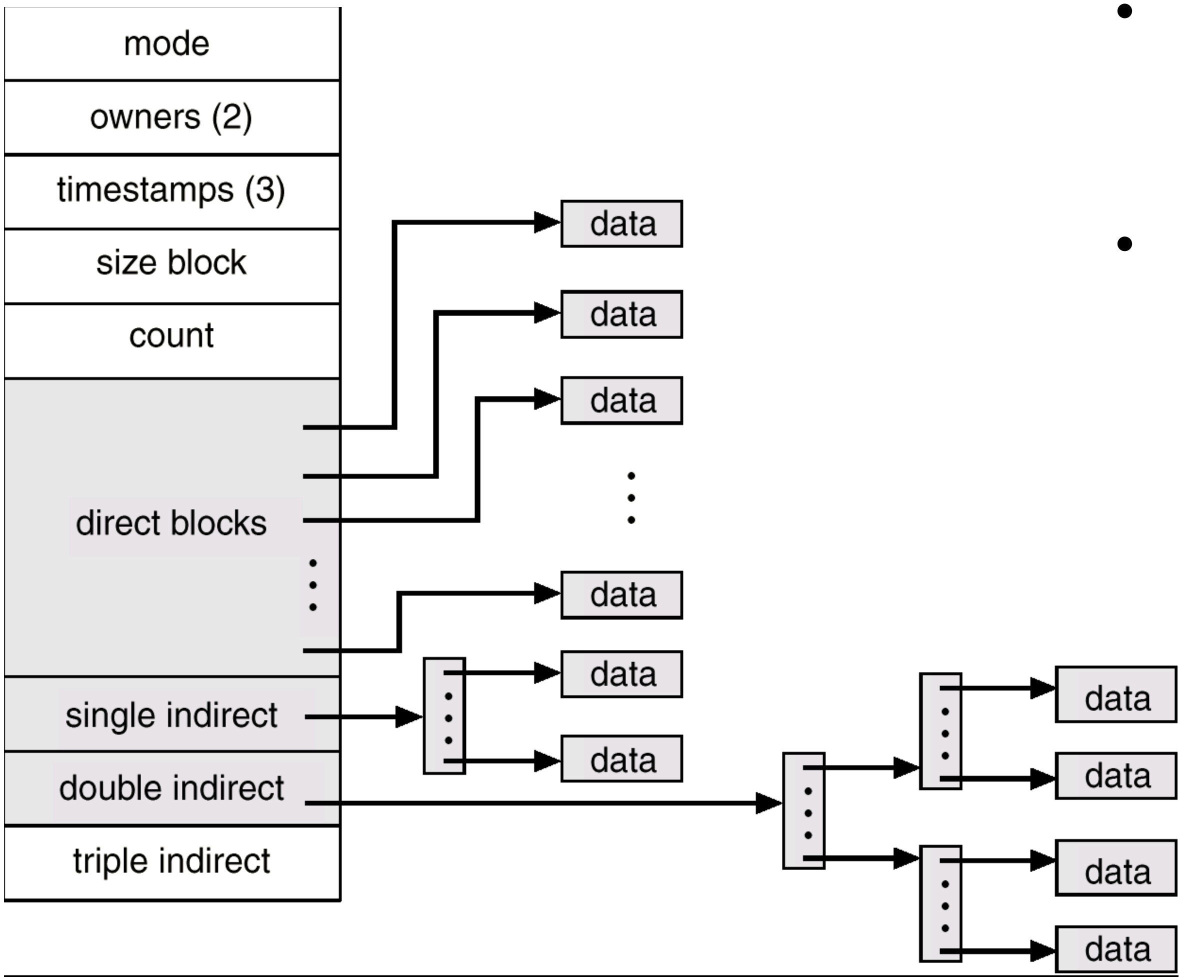
i-nodes (index nodes)

- 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

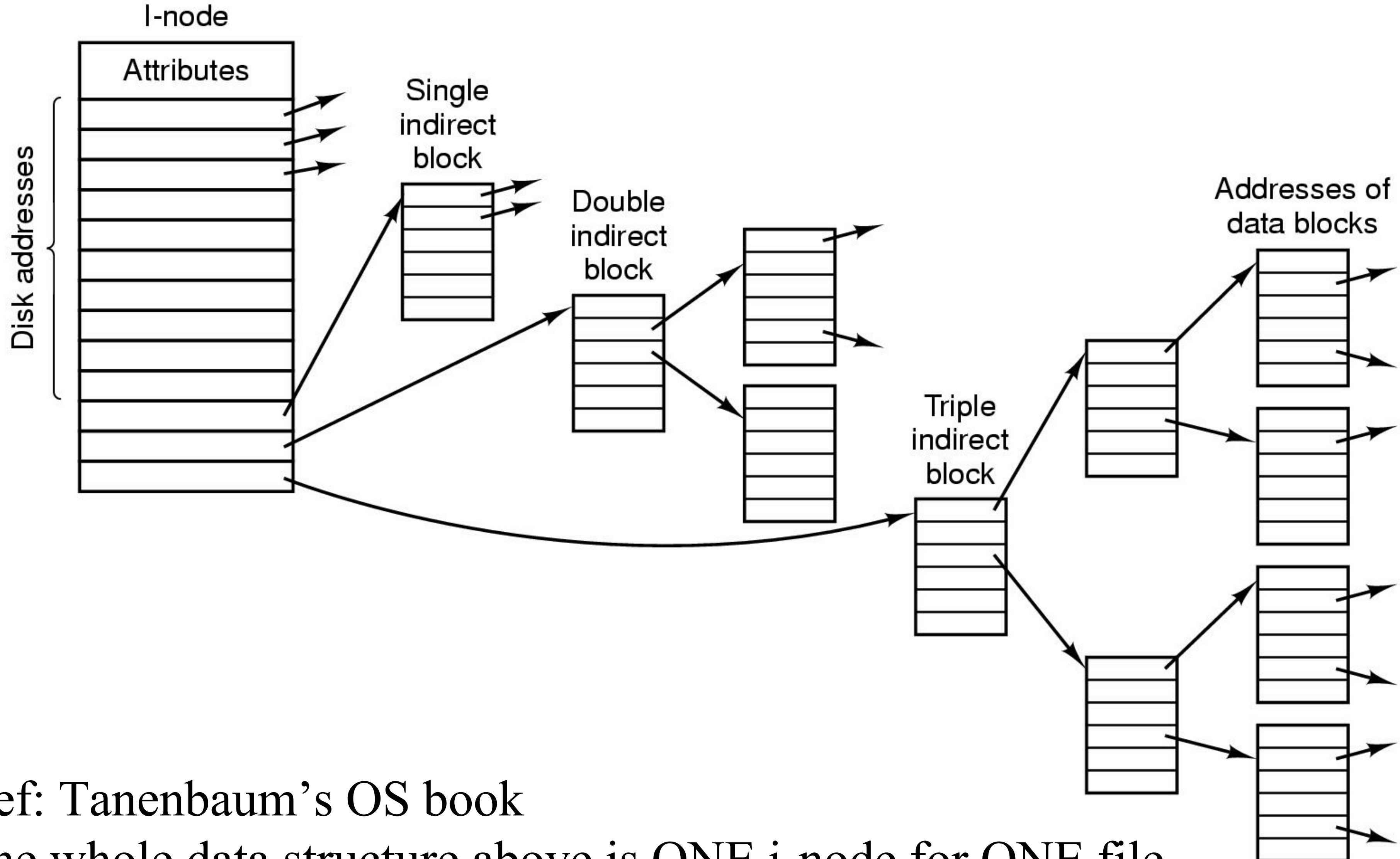
  - Page table maps
    - virtual page number in a virtual address space ——> physical page frame number in DRAM
  - i-node maps
    - logical block number in a file ——> physical block location on disk

Unix i-node (index node)



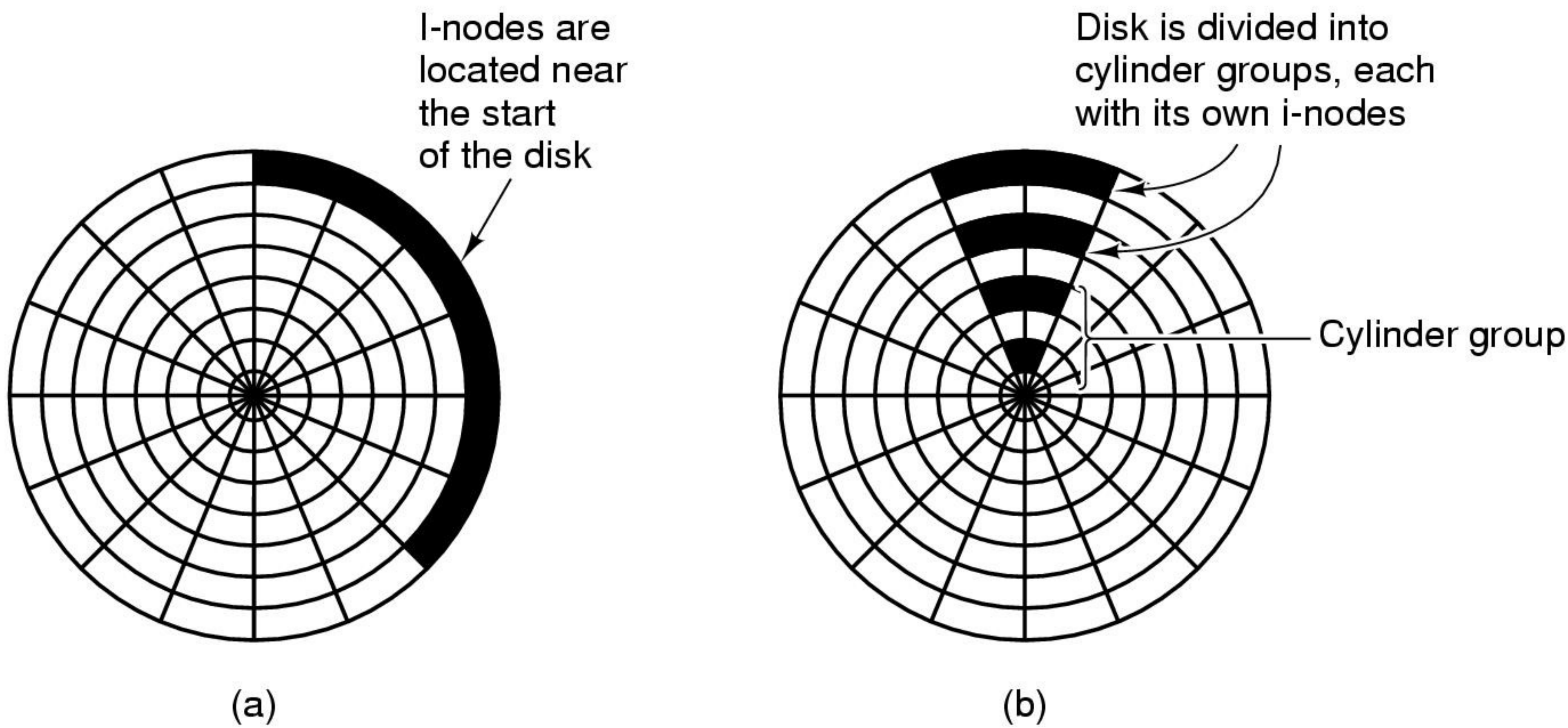
- Small files can be accessed quickly
- If each block is 4KB
  - First 48KB of file reachable from 12 direct blocks
  - Next 4MB available from single-indirect blocks
  - Next 4GB available from double-indirect blocks
  - Next 4TB available through the triple-indirect blocks
- Any block can be found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

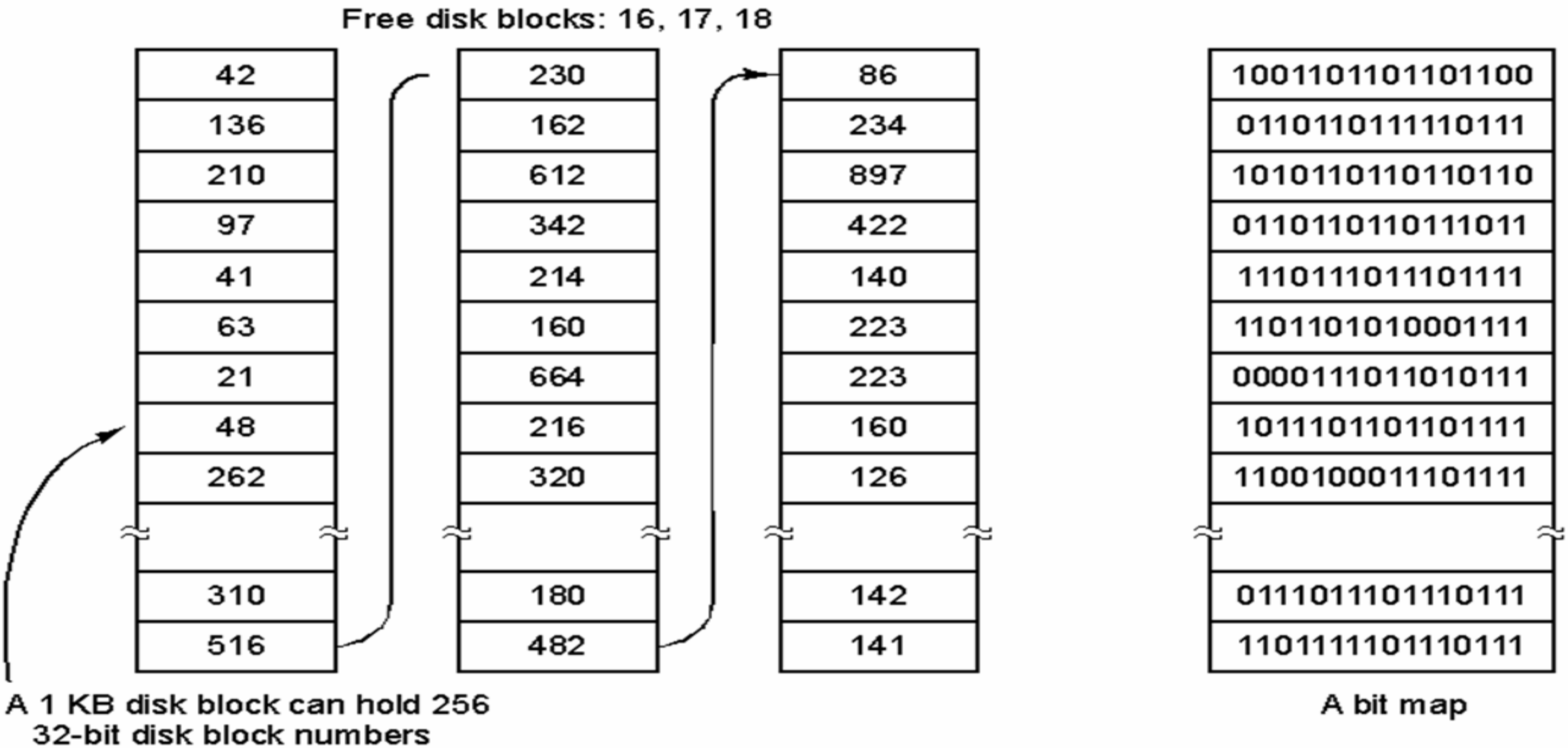
Performance impact of i-node placement



- I-nodes placed at the start of the disk
- Longer average seek times

- Each track on disk has its own blocks and i-nodes
- Shorter average seek times

Managing Free Space on Disk



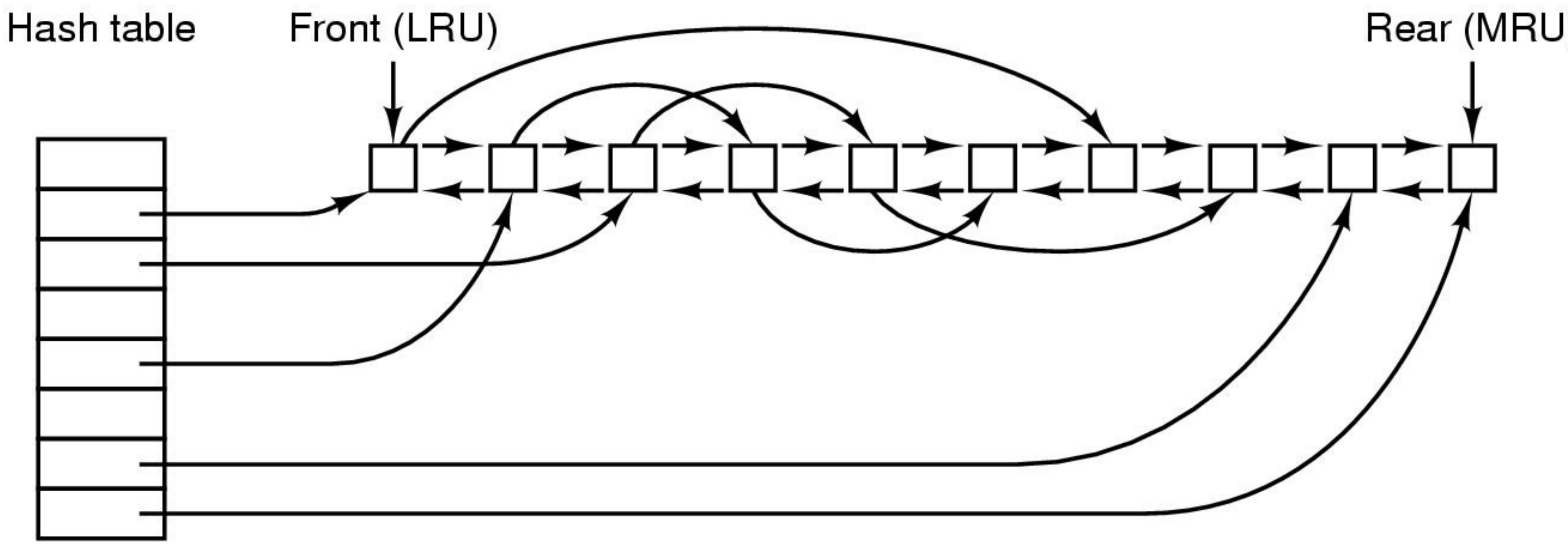
(a) Linked list

(b)

File System Cache

- Small area in main memory that stores frequently accessed data blocks in the file system.
- 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

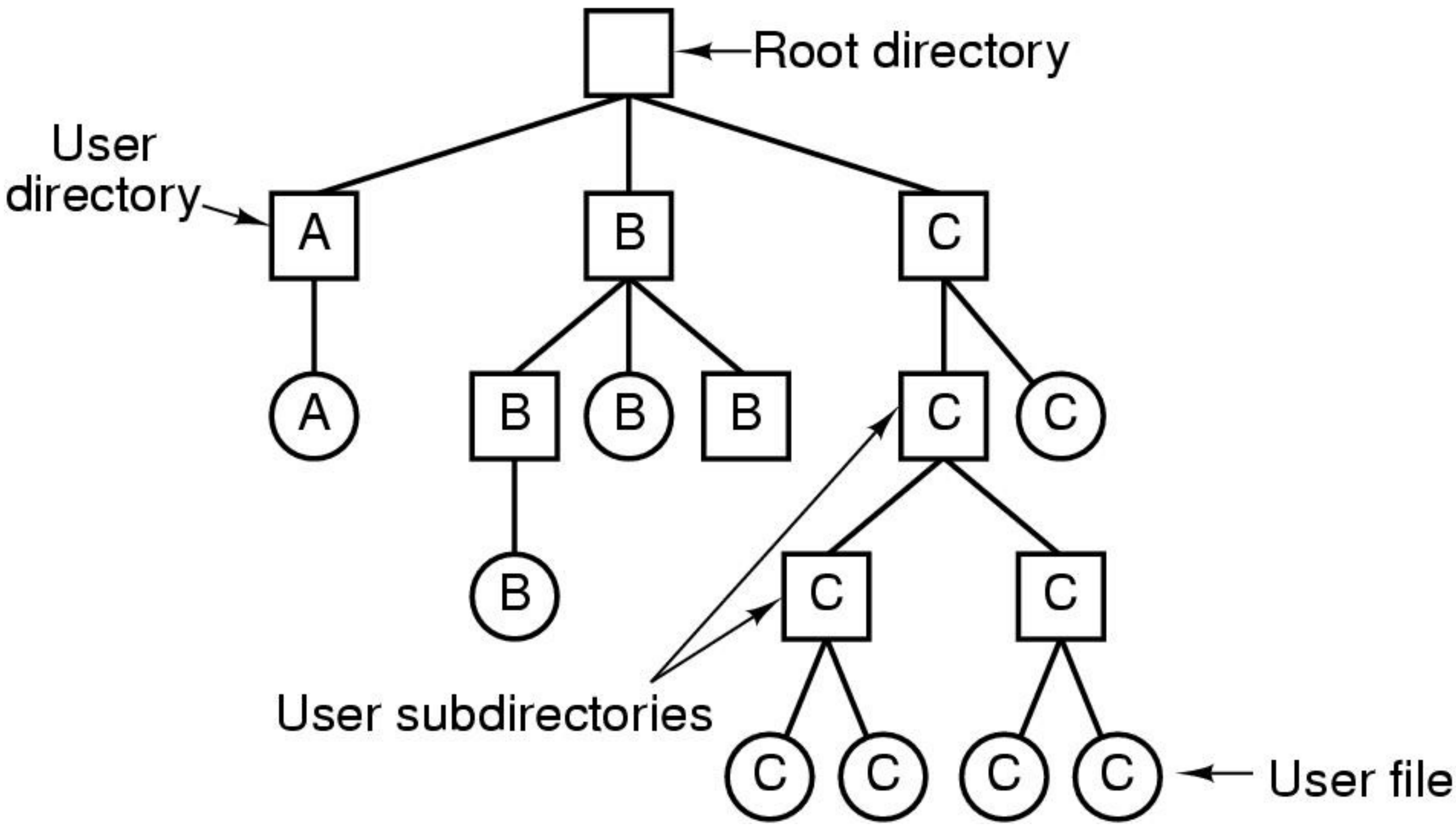
- Often they are managed in a unified manner
- Meaning: one common page-cache is used for managing pages for both virtual memory and file system.
- For example, Linux maintains one common set of data structures to keep track of active and inactive (LRU) pages.

Log-Structured File Systems

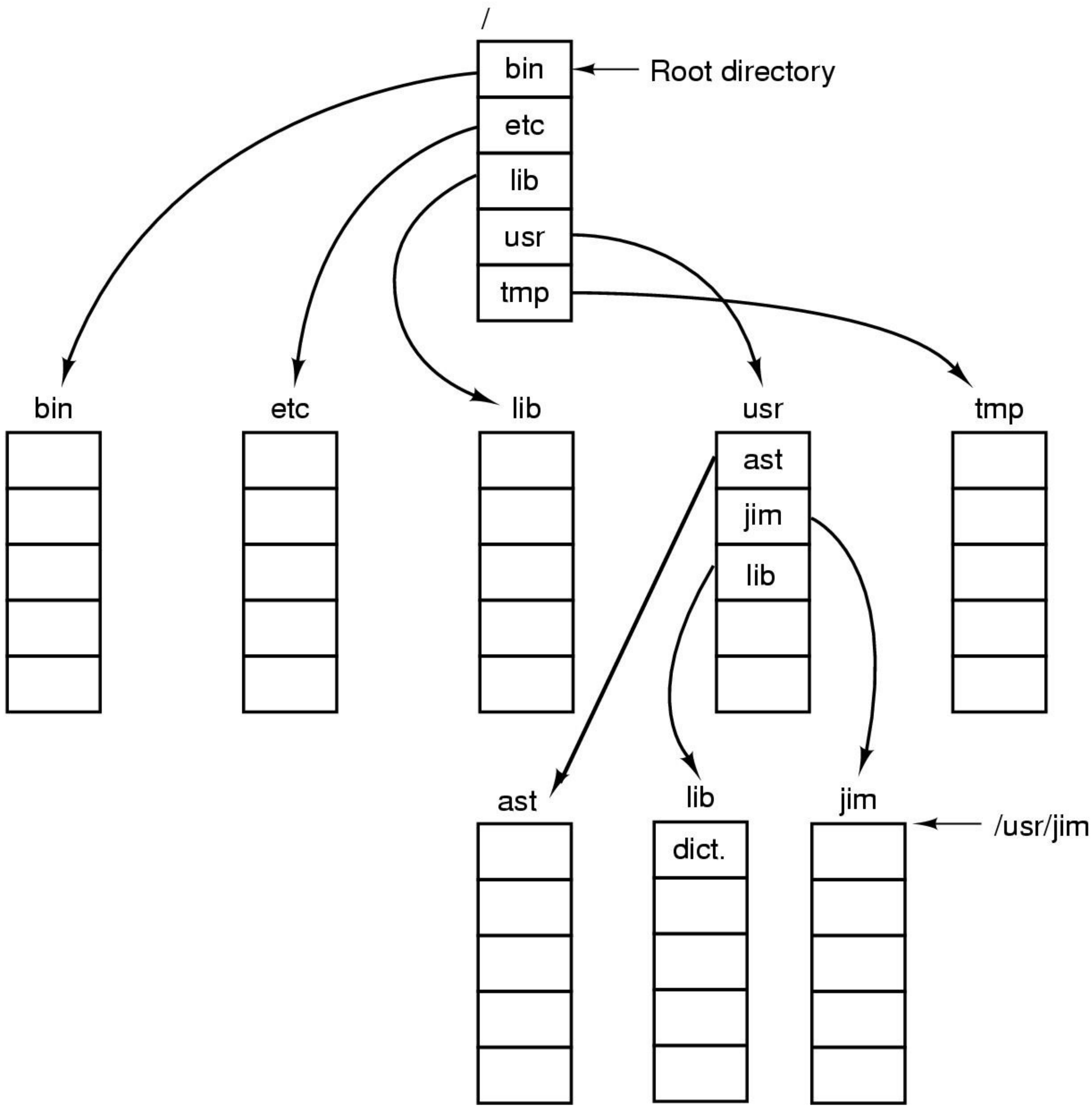
- With CPUs faster, memory larger
  - disk caches are also getting larger
  - increasing number of read requests come from file system cache
  - Thus, most disk accesses will be writes

- 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

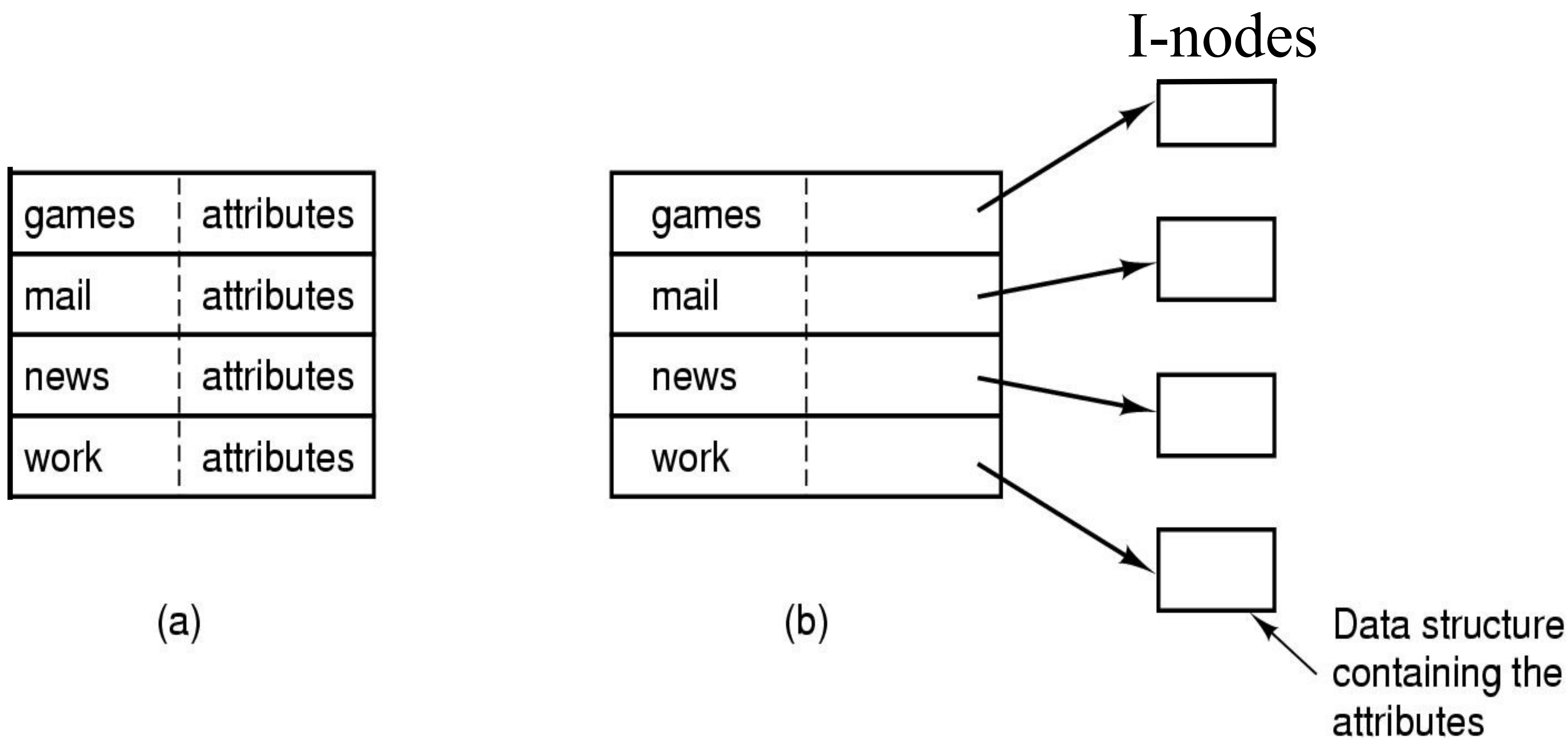
Hierarchical Directory Systems



Path Names


/
bin
etc
lib
usr
tmp
Root directory
bin
etc
lib
usr
tmp
ast
jim
lib
ast
lib
jim
dict.
/usr/jim

Implementing Directories



(a) A simple directory with fixed size entries. Disk addresses and attributes are stored in directory entry

(b) Directory in which each entry just refers to an i-node

Path lookup in a typical Unix File System

The steps in looking up */usr/ast/mbox*

Root directory

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

Looking up usr yields i-node 6

I-node 6 is for /usr

Mode size times
132

I-node 6 says that /usr is in block 132

Block 132 is /usr directory

6	•
1	••
19	dick
30	erik
51	jim
26	ast
45	bal

/usr/ast is i-node 26

I-node 26 is for /usr/ast

Mode size times
406

I-node 26 says that /usr/ast is in block 406

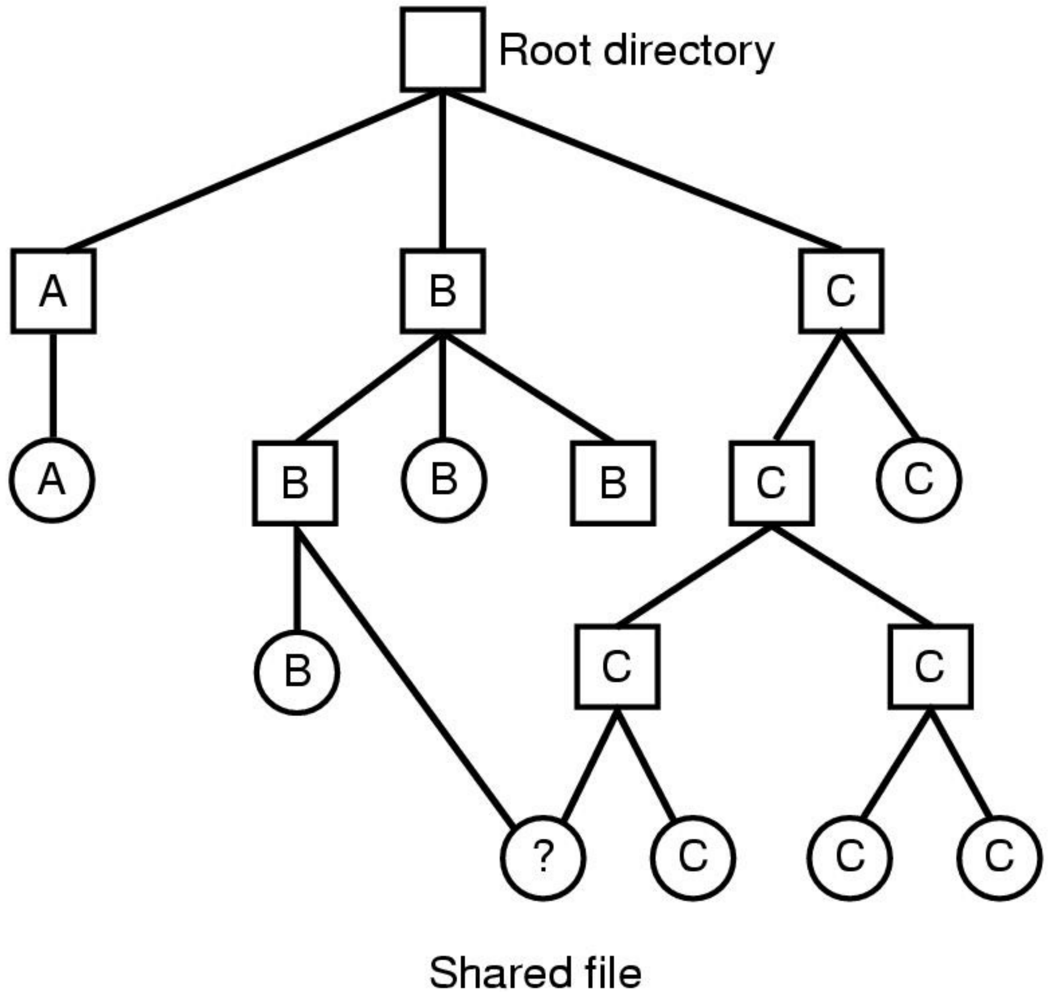
Block 406 is /usr/ast directory

26	•
6	••
64	grants
92	books
60	mbox
81	minix
17	src

/usr/ast/mbox is i-node 60

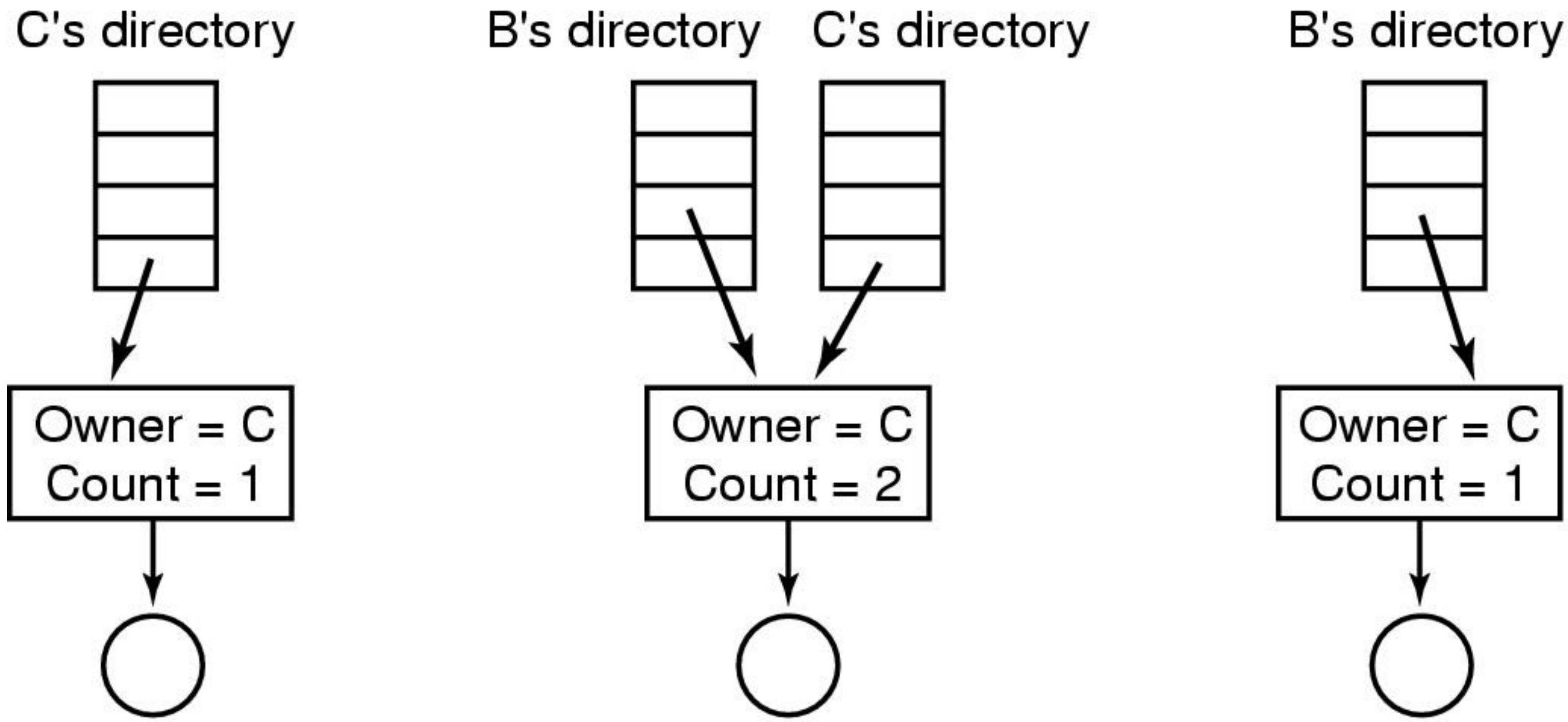
Next, lookup i-node 60 to locate the data blocks of the file /usr/ast/mbox

Shared Files — Hard Links



A file shared between two directories

Shared Files – Hard Links



(a) Situation prior to linking

(b) After the hard link is created

(c) After the original owner removes the file

File still exists, but with count decremented

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?