Modern Operating Systems

Chapter 4 – File System Management

Zhang Yang

Spring 2022

Content of the Lecture

- 4.4 File System Management and Optimization
  - Disk Space Management
  - File System Backups
  - File System Consistency
  - File System Performance

Disk Space Management

- Block Size
- Keeping Track of Free Blocks
- Disk Quotas

Block Size (1)

- Must choose a disk block size...
  - = Page Size?
  - = Sector Size?
  - = Track size?
- Large block sizes
  - Internal fragmentation
  - Last block has (on average) 1/2 wasted space
  - Lots of very small files; waste is greater.
- Small block sizes
  - More seeks; file access will be slower.

Block Size (2)

- Tanenbaum's Result (2006)
  - Studied the file-size distribution.

Length	VU 1984	VU 2005	Web
1	1.79	1.38	6.67
2	1.88	1.53	7.67
4	2.01	1.65	8.33
8	2.31	1.80	11.30
16	3.32	2.15	11.46
32	5.13	3.15	12.33
64	8.71	4.98	26.10
128	14.73	8.03	28.49
256	23.09	13.29	32.10
512	34.44	20.62	39.94
1 KB	48.05	30.91	47.82
2 KB	60.87	46.09	59.44
4 KB	75.31	59.13	70.64
8 KB	84.97	69.96	79.69
16 KB	92.53	78.92	86.79
32 KB	97.21	85.87	91.65
64 KB	99.18	90.84	94.80
128 KB	99.84	93.73	96.93
256 KB	99.96	96.12	98.48
512 KB	100.00	97.73	98.99
1 MB	100.00	98.87	99.62
2 MB	100.00	99.44	99.80
4 MB	100.00	99.71	99.87
8 MB	100.00	99.86	99.94
16 MB	100.00	99.94	99.97
32 MB	100.00	99.97	99.99
64 MB	100.00	99.99	99.99
128 MB	100.00	99.99	100.00

Figure 4-20. Percentage of files smaller than a given size (in bytes).

Block Size (3)

- Tanenbaum’s Result (2006) (ctd.)
  - Conclusions
    - With a block size of 1KB, only about 30-50% of all files fit in a single block.
    - With a block size of 4KB, about 60-70% of all files fit in a single block.
    - Other data show that with a 4-KB block, 93% of the disk blocks are used by the 10% largest files.
      - Wasting space at the end of each small file hardly matters.

Block Size (4)

- Performance and space utilization are inherently in conflict.

Higher block size: higher data rate, lower space utilization

Lower block size: lower data rate, higher space utilization

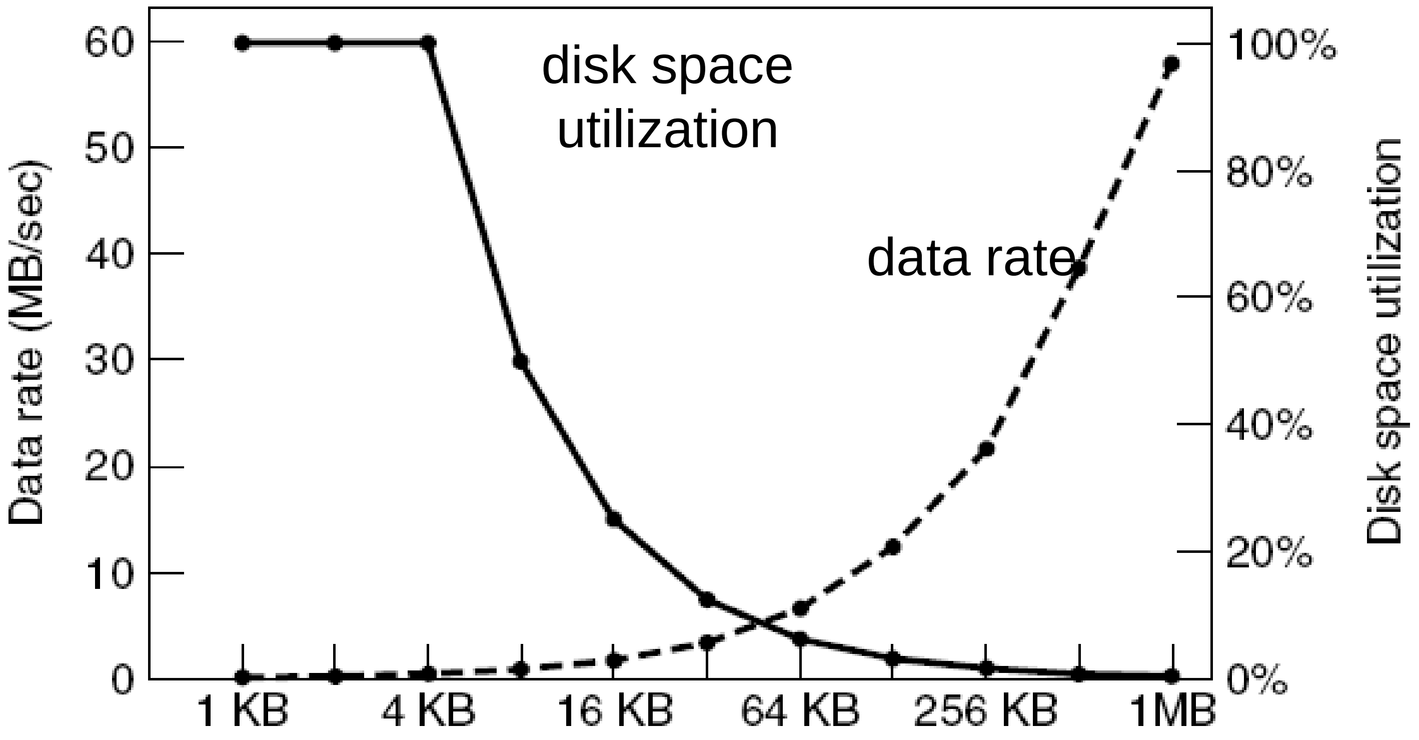


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

Block Size (5)

- Typical Block Sizes
  - Historically, file systems have chosen sizes in the 1-KB to 4-KB range, but with disks now exceeding 1 TB, it might be better too increase the block size to 64KB and accept the wasted disk space.
  - Unix
    - Block size in UNIX is basically the size of chunk memory in which, block devices transfer data inside the system.
    - Usually in modern system it is 4KB, 8KB etc.
  - MS-DOS
    - Variable, N*512 ("cluster size" represents the OS "Block Size")

Block Size (6)

- Typical Block Sizes (ctd.)
  - MS-DOS (ctd.)
    - Determined by other issues
    - Limited number of bits for block addresses.
    - To accommodate larger disk sizes-->use bigger blocks
      - FAT-12: 512B, 1KB, 2KB, 4KB
      - FAT-16: 2KB, 4KB, 8KB, 16KB, 32KB
      - FAT-32: 4KB, 8KB, 16KB, 32KB
  - Windows
    - Default 4KB
    - Go to "My Computer"   --> manage --> Disk Defragmenter --> select any disk --> click on "analyze" button , then it will present us a report and in that report "Cluster Size“.

Keeping Track of Free Blocks (1)

- Bit Map or Bit Vector
  - A disk with n blocks requires a bit map with n bits.
  - Free blocks are represented by 1's.
  - Allocated blocks represented by 0's.
  - Example: Calculate the number of blocks needed to hold the disc map.
    - 16GB disk has 2^{24} 1-KB block and requires 2^{24} bits ➔ 2048 blocks

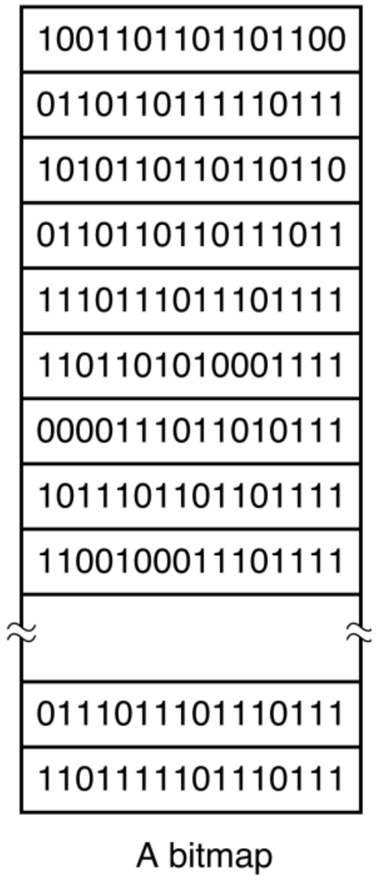


Figure 4-22 (b) A bit map

Keeping Track of Free Blocks (2)

- Bit Map (ctd.)
  - Advantages
    - Simple to understand.
    - Easy to find first free block.
    - Supports contiguous allocation, easy to find n consecutive free blocks.
  - Disadvantage
    - Bit maps are inefficient unless the entire bit map is kept in main memory (and is written to disk occasionally for recovery needs).
    - Keeping it in main memory is possible for smaller disks but not necessarily for larger ones.

Keeping Track of Free Blocks (3)

- Linked List
  - Each block holds as many free disk block numbers as will fit.
  - Example: Calculate the maximum number of blocks we need to hold a complete free list.
    - 16GB disk has 2^{24} 1-KB block and requires $2^{24}/255 = 65793$ blocks. (Assuming 32-bit block number)

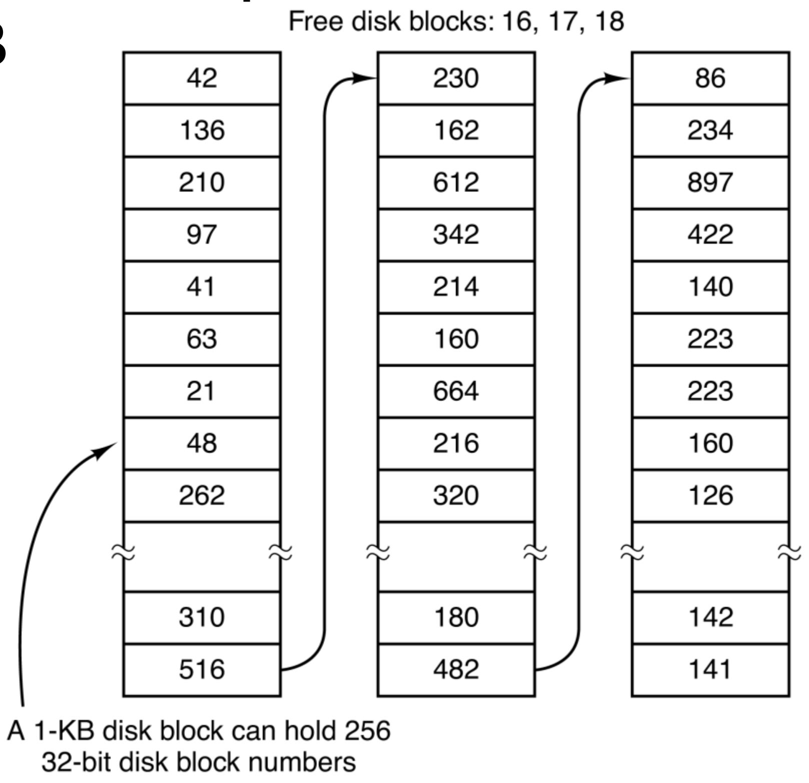


Figure 4-22

(a) Storing the free list on a linked list

Keeping Track of Free Blocks (4)

- Linked List (ctd.)
  - Disadvantage
    - Traversing the list and/or finding a contiguous block of a given size are not easy.
- Comparison
  - The bitmap requires less space.
  - Only if the disk is nearly full (i.e., has few free blocks) will the linked list scheme require fewer blocks than the bitmap.

Keeping Track of Free Blocks (5)

- Counting
  - Rather than keeping a list of n free disk addresses, we can keep the address of the first free block and the number n of free contiguous blocks that follow the first block.
  - Each entry in the free-space list then consists of a disk address and a count.
  - Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

Problem of Linked List Method (1)

- Only one block of pointers need be kept in main memory. Under certain circumstance, this method leads to unnecessary disk I/O.
  - When the block of pointers is almost empty, a series of short-lived temporary files can cause a lot of disk I/O.
  - Example: Figure 4-23 (a) (b)
- Solution
  - Keep most of the pointer blocks on disk full (to minimize disk usage), but keep the one in memory about half full.
  - Example: Figure 4-23 (c)

Problem of Linked List Method (2)

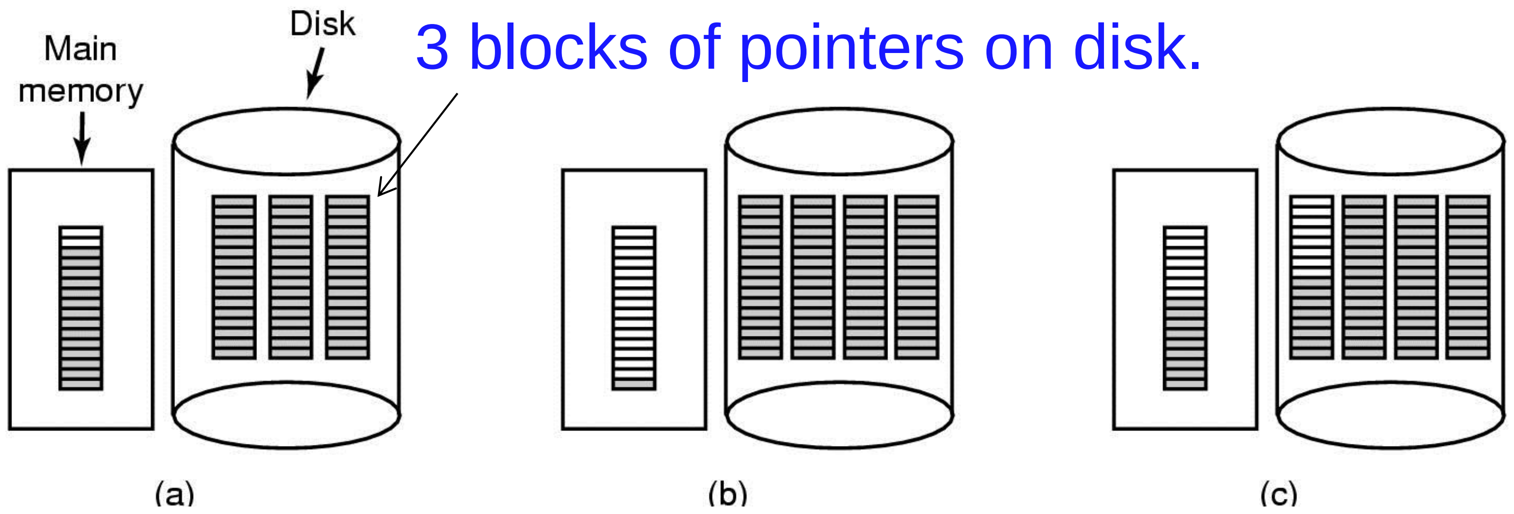


Figure 4-23

(a) Almost-full block of pointers to free disk blocks in RAM.

(b) Result of freeing a 3-block file

(c) Alternative strategy for handling 3 free blocks

shaded entries are pointers to free disk blocks

Disk Quotas (1)

- Prevents people from getting too much disk space.
- Use a second table in the open file table structure. For each user...
    - OS will maintain a record.
    - Example
        - Amount of disk space used (in blocks)
            - Current
            - Maximum allowable
        - Number of files
            - Current
            - Maximum allowable
    - Soft Limits
        - When exceeded, print a warning.
    - Hard Limits
        - May not be exceeded, error.

Disk Quotas (2)

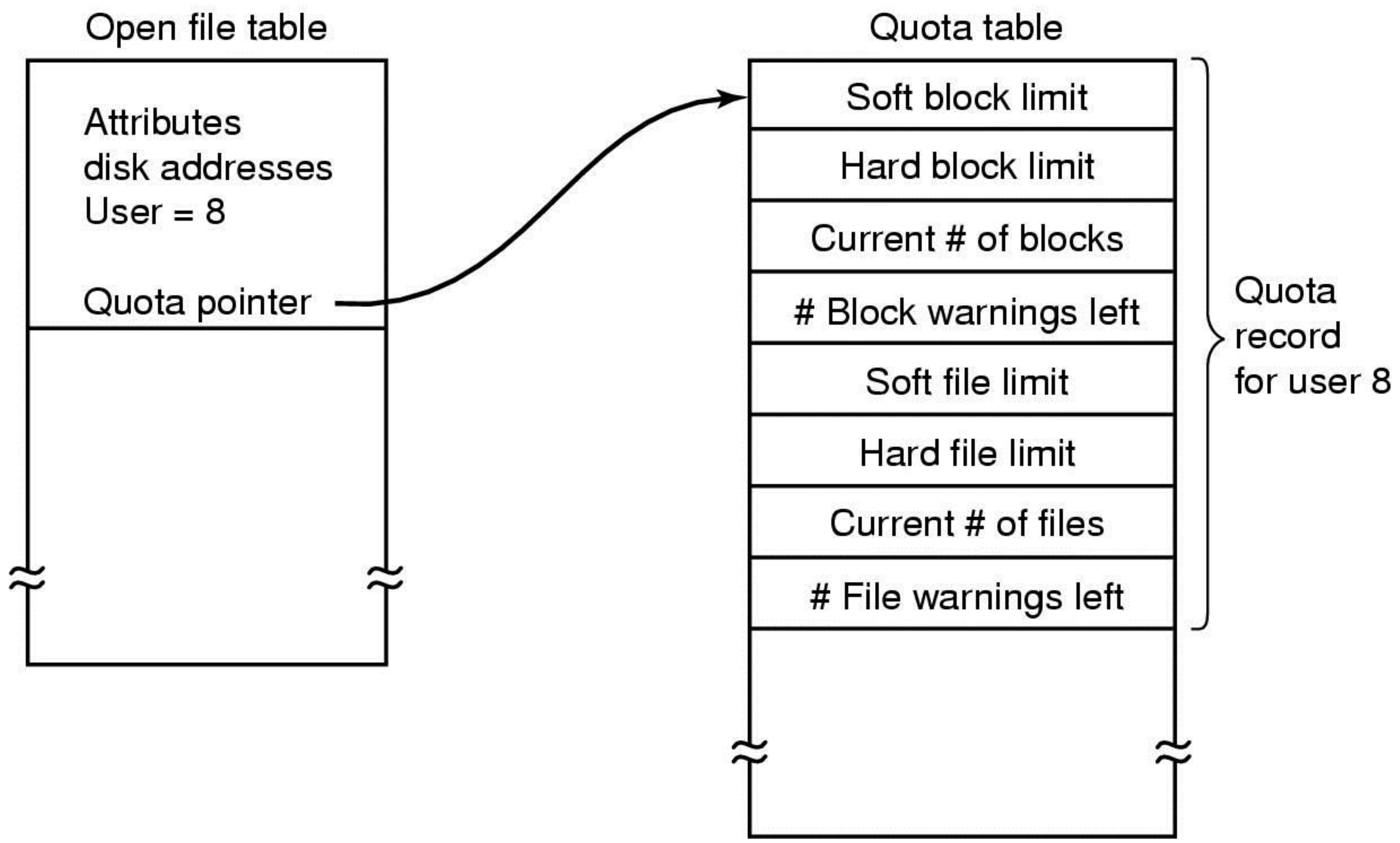
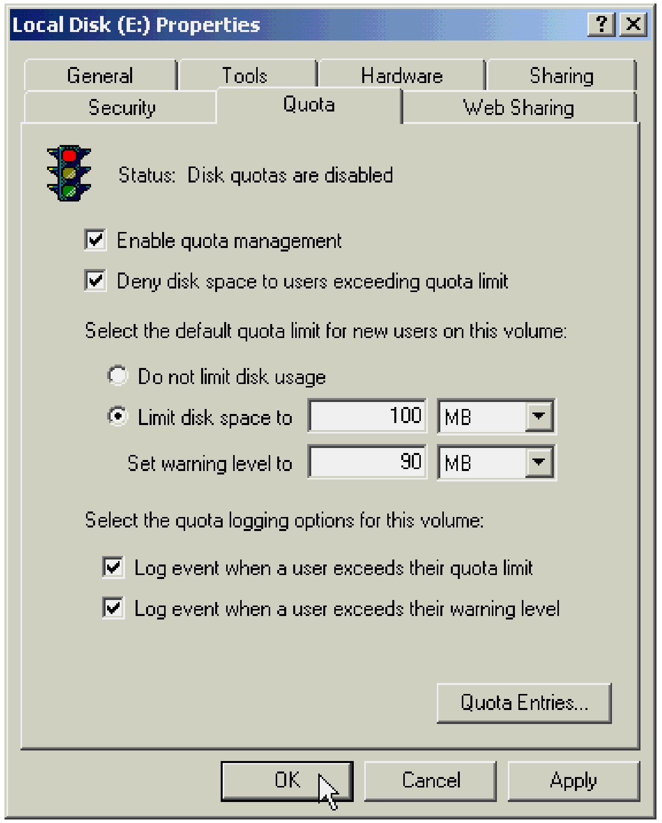


Figure 4-24 Quotas for keeping track of each user’s disk use

Disk Quotas (3)

- Windows Disk Quota Management
  - *Enable quota management:* Sets up quota management and starts tracking disk usage.
  - *Deny disk space to users exceeding quota limits:* Users can't write new information after reaching their quotas.
  - *Do not limit disk usage:* Tracks disk usage without imposing quotas.
  - *Limit disk space to:* Sets the default amount of disk space for all users.



Status: Disk quotas are disabled

[✓] Enable quota management

[✓] Deny disk space to users exceeding quota limit

Select the default quota limit for new users on this volume:

○ Do not limit disk usage

● Limit disk space to 100 MB

Set warning level to 90 MB

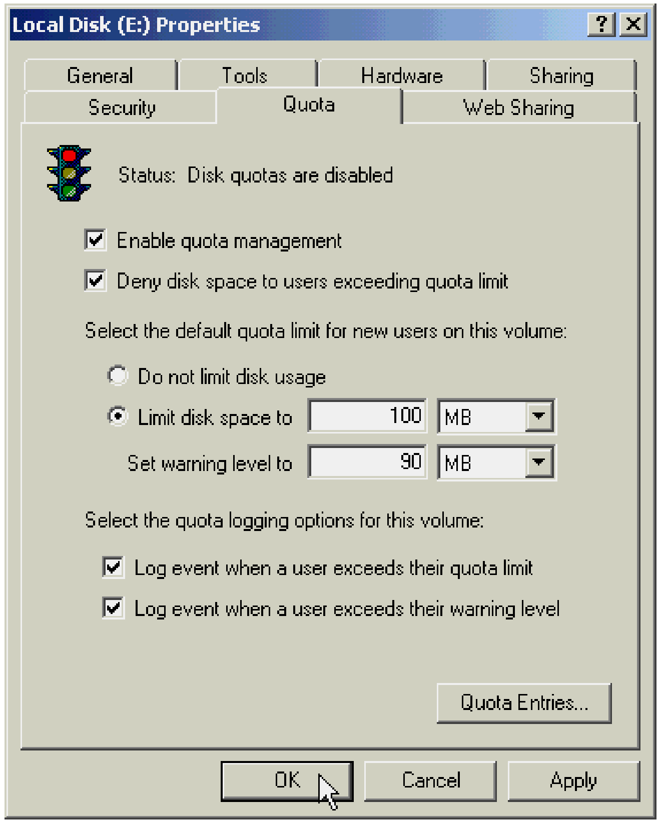
Select the quota logging options for this volume:

[✓] Log event when a user exceeds their quota limit

[✓] Log event when a user exceeds their warning level

Disk Quotas (4)

- Windows Disk Quota Management (ctd.)
  - *Set warning level to:* Sets the default disk space that users can occupy that will trigger a warning message.
  - *Log event when a user exceeds their quota limit:* An event is entered in the System log when a user reaches his or her quota.
  - *Log event when the user exceeds the warning level:* An event is entered in the System log when a user receives a warning that he or she is approaching the quota.



Status: Disk quotas are disabled

[✓] Enable quota management

[✓] Deny disk space to users exceeding quota limit

Select the default quota limit for new users on this volume:

○ Do not limit disk usage

● Limit disk space to 100 MB

Set warning level to 90 MB

Select the quota logging options for this volume:

[✓] Log event when a user exceeds their quota limit

[✓] Log event when a user exceeds their warning level

File System Reliability

- File System Back Up
  - Data in a file system should be permanent.
  - Implemented through backup and replication.
- File System Consistency
  - Data in a file system should be good.
  - Implemented through logging , transactions, and shadowing.

File System Backups (1)

- Backup to Tapes
  - Recover from disaster
    - Get the computer running again after a disk crash, fire, flood, or other natural catastrophe.
  - Recover from stupidity
    - Users often accidently remove files that they later need again.
- Backup Issues
  - Backup all or part of the system? Back up only specific directories and everything in them rather than the entire file system.
  - Don’t backup file if not changed. Incremental Dumps.

File System Backups (2)

- Backup Issues (ctd.)
  - Compression of backup or not? A single bad spot on the backup tape can foil the decompression algorithm. The decision to compression should be carefully considered.
  - Difficulty of backup while file system active.
  - Physical security of backup media.
- Two Strategies for Dumping a Disk to Tape
  - Physical Dump
  - Logical Dump

File System Backups (3)

- Physical Dump
  - Approach
    - Start at block 0 on the disk.
    - Copy each block, in order.
  - Concerns
    - Blocks on the free list?
      - Should avoid copying them.
    - Bad sectors on disk?
      - Controller remaps bad sectors.
      - Backup utility need not do anything special!
        - OS handles bad sectors.
        - Backup utility must avoid copying them!

File System Backups (3)

- Physical Dump (ctd.)
  - Advantages
    - Simple
    - Great speed (run at the speed of disk)
  - Disadvantages
    - Inability to skip selected directories.
    - Inability to make incremental dump.
    - Inability to restore individual files upon request.

File System Backups (4)

- Logical Dump
  - Most common form.
  - Start at one or more specified directories and recursively dumps all files and directories found there that have changed since some given base date.
- Logical Dump: Incremental dumping of files and directories
  - Will copy only files that have been modified since last incremental backup.
  - Must also copy the directories containing any modified files.

File System Backups (5)

- Logical Dump Algorithm
  - Phase 1: begin at the starting directory and examines all the entries in it. Marked each modified file and each directory (whether or not modified).
  - Phase 2: recursively walks the directory tree, unmarking any directories that have no modified files or directories in them or under them.
  - Phase 3: dump all the marked directories
  - Phase 4: dump all the marked files
  - Example: Figure 4-25, 4-26

File System Backups (6)

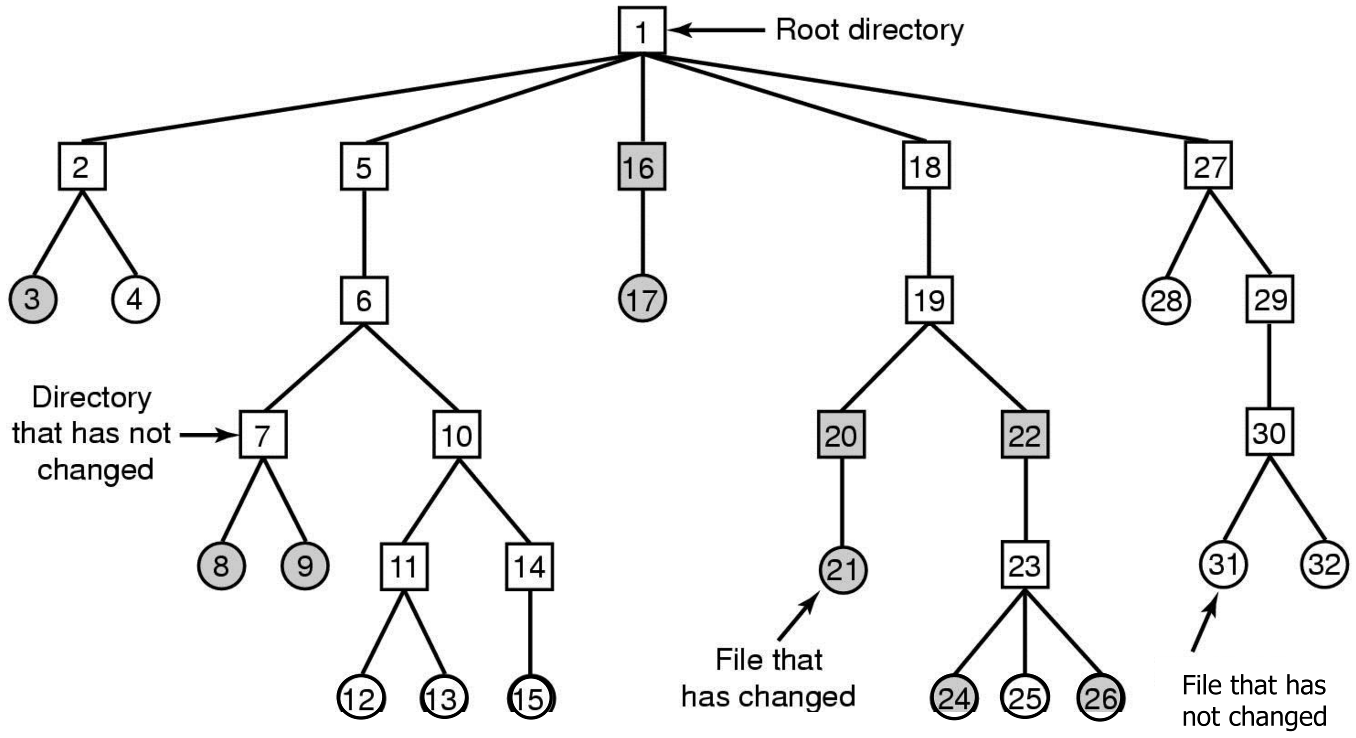


Figure 4-25 A file system to be dumped

- squares are directories, circles are files
- shaded items, modified since last dump
- each directory & file labeled by i-node number

File System Backups (7)

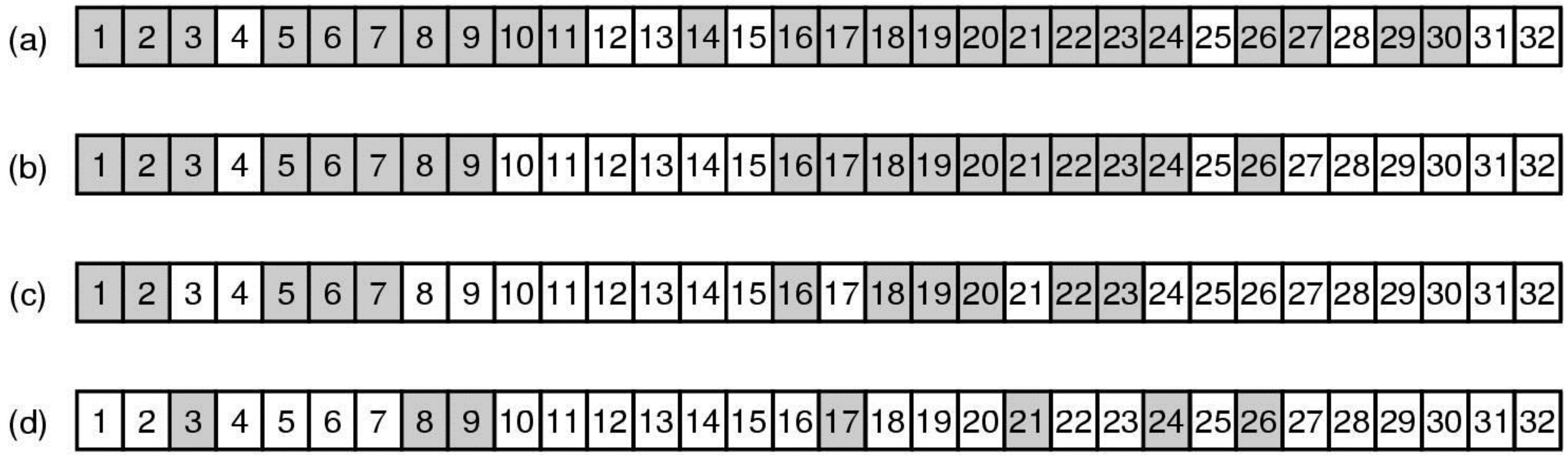


Figure 4-26 Bit maps used by the logical dumping algorithm

File System Consistency (1)

- Multi-step updates cause problems for consistency.
  - If crash happens in the middle.
  - E.g. transfer $100 from my account to Janet’s
    - (1) deduct $100 from my account
    - (2) add $100 to Janet’s account
  - What happens if you crash between step 1 and 2?

File System Consistency (2)

- Most systems have a utility program that checks file system consistency.
  - Windows: scandisk
  - UNIX: fsck
    - Checking blocks.
    - Checking directories.
- fsck: checking blocks
  - Basic fact: Each disk block must be in a file (or directory), or on the free list.
  - Build 2 tables, each contains a counter for each block.
  - Reads all the i-nodes and mark used blocks.
  - Examines the free list and mark free blocks.

File System Consistency (3)

- Block Inconsistent States
  - Some block is not in a file or on free list (“missing block”)
    - Example: Figure 4-27 (b)
  - Add it to the free list.

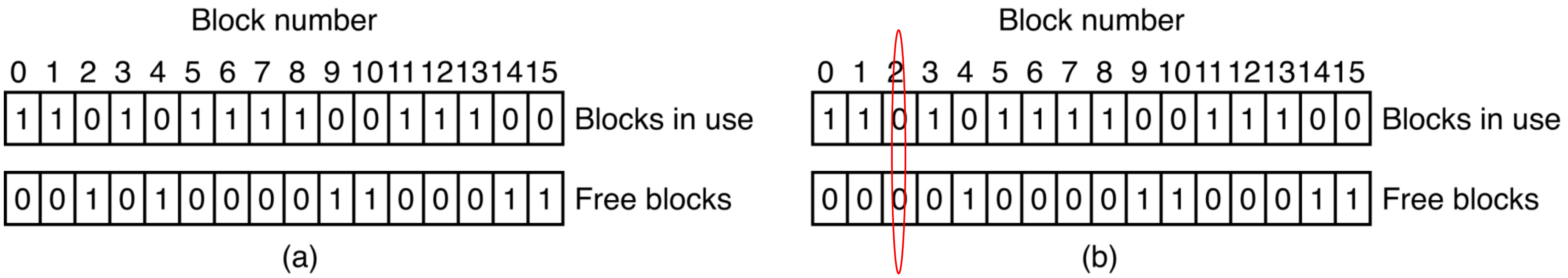


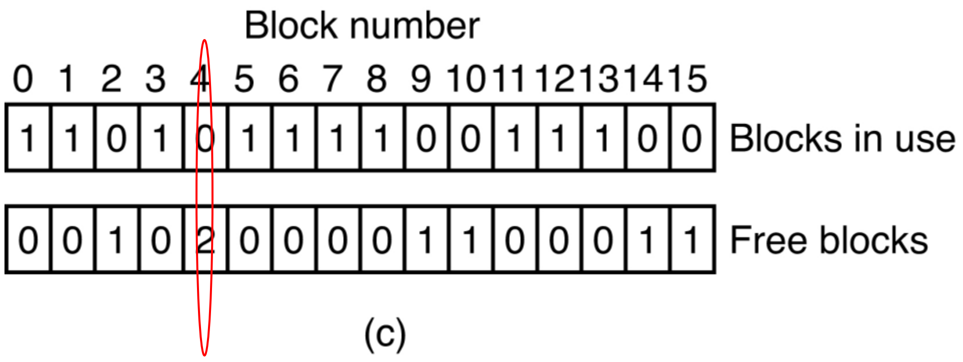
Figure 4-27 File System States

(a) consistent

(b) missing block

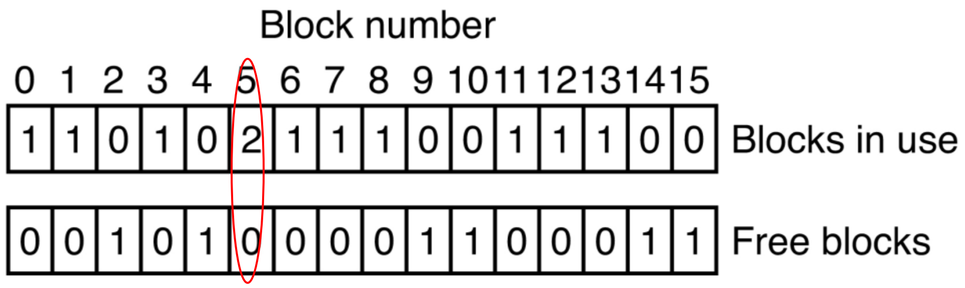
File System Consistency (4)

- Block Inconsistent States (ctd.)
  - Some block is on the free list more than once
    - Can't happen when using a bitmap for free blocks.
    - Example: Figure 4-27 (c)
  - Fix the free list so the block appears only once.



File System Consistency (5)

- Block Inconsistent States (ctd.)
  - Some block is in more than one file.
    - Example: Figure 4-27 (d)
  - Allocate another block.
  - Copy the block.
  - Put each block in each file.
  - Notify the user that one file may contain data from another file.



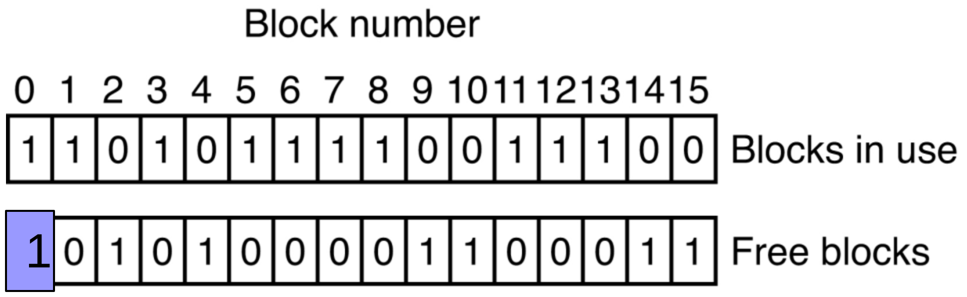
Block number

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Blocks in use	1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0
Free blocks	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(d)

File System Consistency (6)

- Block Inconsistent States (ctd.)
  - Some block is on free list and is in some file
    - Example



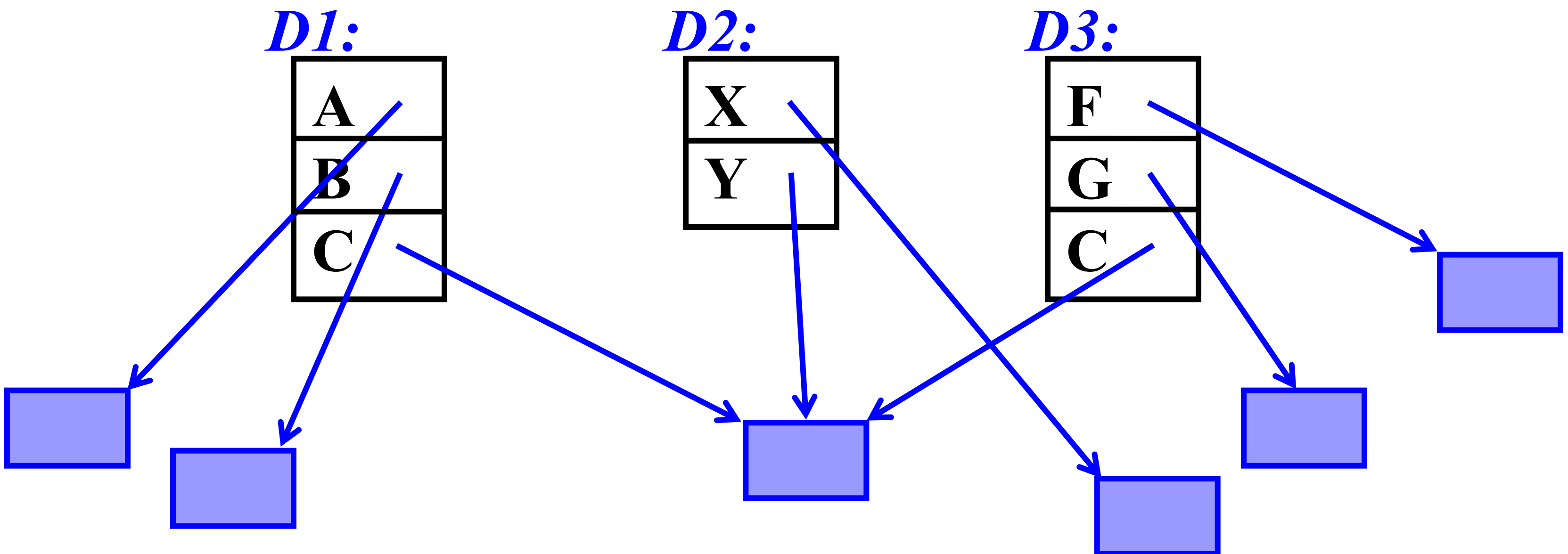
    - Remove it from the free list.

File System Consistency (7)

- Exercise
  - Problem: Consider Fig. 4-27. Is it possible that for some particular block number the counters in both lists have the value 2?
  - Solution:
    - It should not happen, but due to a bug somewhere it could happen. It means that some block occurs in two files and also twice in the free list.
    - The first step in repairing the error is to remove both copies from the free list. Next a free block has to be acquired and the contents of the sick block copied there. Finally, the occurrence of the block in one of the files should be changed to refer to the newly acquired copy of the block. At this point the system is once again consistent

File System Consistency (8)

- fsck: checking directories
  - Recursively descends the tree from the root directory, for file counting the number of links each file. One counter for each file.
  - Compare these numbers with the link counts stored in the i-nodes.
  - The reference count in each i-node must be equal to the number of hard links to the file.



File System Consistency (9)

- Directory Inconsistent State
  - Two Inconsistent States Solution
    - Force the link count in the i-node to the actual number of directory entries.
  - Reference count is too large
    - The “rm“ command will delete a hard link.
    - When the count becomes zero, the blocks are freed.
    - Permanently allocated; blocks can never be reused.
  - Reference count is too small
    - When links are removed, the count will go to zero too soon!
    - The blocks will be added to the free list, even though the file is still in some directory!

File System Performance (1)

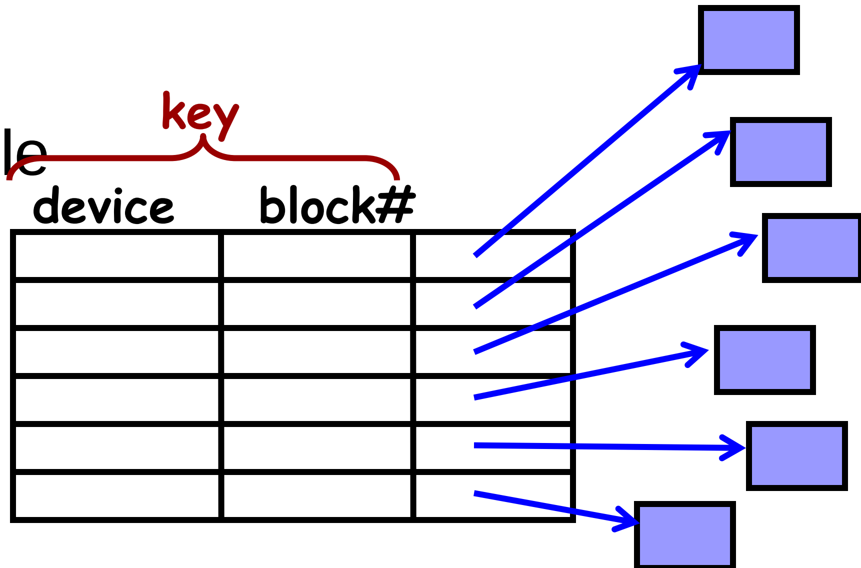
- Caching
  - Avoid disk I/O overhead
- Block Read Ahead
  - Improve buffer cache hit rate
- Reducing Disk Arm Motion
  - Avoid seek overhead

Caching (1)

- Many files are used repeatedly.
  - Option: read it each time from disk.
  - Better: keep a copy in memory.
- File System Cache
  - Maintain a block cache or buffer cache in memory.
  - A cache is a collection of blocks that logically belong on the disk but are being kept in memory for performance reasons.
  - Set of recently used file blocks.
  - Keep blocks just referenced.
  - When a process tries to read a block check the cache first.

Caching (2)

- File System Cache (ctd.)
  - If it is, the read request can be satisfied without a disk access.
  - If the block is not in the cache, it is first read into the cache, and then copied to wherever it is needed.
- Cache Organization
  - Many blocks (e.g., 1000s)
  - Indexed on block number
  - For efficiency, use a hash table



Caching (3)

- The buffer cache data structures
  - Collision chain
  - Bidirectional list: running through all the blocks in the order of recently used usage, with the least recently used block on the front of this list and the most recently used block at the end of the list.

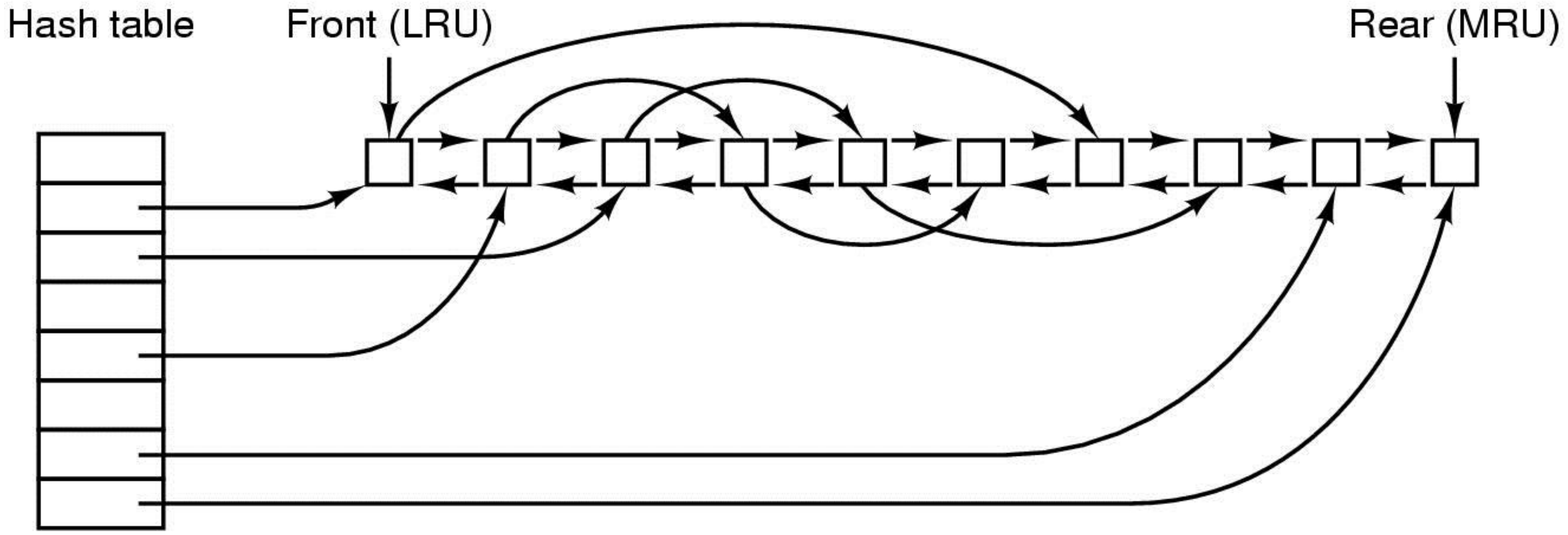


Figure 4-28 The buffer cache data structures

Caching (4)

- What if the cache full and the user request a new block not in cache?
  - Throw out old, unused blocks.
  - Same kinds of algorithms as for virtual memory. Similar to paging system, FIFO, Second Chance, LRU, etc. can be used.
  - More effort per reference is OK: file references are a lot less frequent than memory references.
  - LRU is a reasonable *block replacement policy.*
    - Can be implemented precisely: Bidirectional list in figure 4-28

Caching (5)

- To implement precise LRU, consider the following situations:
  - If a critical block (such as File Control Block, or i-node) is read into the cache and modified, but not re-written to the disk, a crash will leave the file system in an inconsistent state.
  - If the i-node block is put at the end of the LRU chain, it may be quite a while before it reaches the front and is rewritten to the disk.
  - Some blocks, such as i-node blocks, are rarely referenced two times within a short interval.

Caching (6)

- Consider a modified LRU scheme, taking two factors into account:
  - 1. Is the block likely to be needed again soon?
  - 2. Is the block essential to the consistency of the file system?
- For factors 1 and 2
  - Blocks can be divided to categories
    - i-node blocks, indirect blocks, directory blocks, full data blocks, partially full data blocks, etc.
  - Blocks that will probably not be needed again soon go on the front, rather than the rear of the LRU list.
  - Blocks that might be needed again soon go on the end of the list.

Caching (7)

- For factor 2
  - If the block is essential to the file system consistency, and it has been modified, it should be written to disk immediately, regardless of which end of the LRU list it is put on.
  - Okay to delay writes to data blocks.
  - Note: It is undesirable for keeping data blocks in the cache too long before writing them out.
    - 30 seconds Later
      - The Unix "sync" system call
    - Immediately
      - MS-DOS "Write-through cache"

Block Read Ahead (1)

- This method means
  - Try to get blocks into the cache before they are needed.
  - Increase hit rate by prefetching anticipated blocks.
- Approach
  - User requests ‘k’ block.
  - FS gets ‘k’ block.
  - FS checks of ‘k+1’ block is in cache, if not, FS will get ‘k+1’ block from disk anticipating that it will be needed in the future.

Block Read Ahead (2)

- Advantage
  - If user needs ‘k+1’ block, the access is fast – great method for video playback.
- Disadvantage
  - If user does not need ‘k+1’ block, extra unnecessary work has been done.
- This technique helps only if access is sequential.

Block Read Ahead (3)

- What if the file is *not* being read sequentially?
  - FS keeps track of access patterns to open files
    - Sequential access mode
    - Random access mode
  - FS may use a bit associated with each file to keep track of access pattern (1 for sequential access, 0 for random access)

Reducing Disk Arm Motion (1)

- Seek time dominates disk access time.
  - Reducing disk arm motion improved performance.
  - Reducing disk arm motion by putting blocks that are likely to be accessed in sequence close to each other, preferably in the same cylinder.
- When an output file is writing
  - If the system uses a bitmap for the free blocks, it can allocate a new block for a file close to the previous block (guessing that the file will be accessed sequentially).

Reducing Disk Arm Motion (2)

- When an output file is writing (ctd.)
  - If the system uses a link list for the free blocks, the system can perform allocations in super-blocks, consisting of several contiguous blocks.
    - The block cache and I/O requests are still in blocks not super-blocks.
    - If the file is accessed sequentially, consecutive blocks of a super-block will be accessed in sequence and these are contiguous on the disk.

Reducing Disk Arm Motion (3)

- In systems using i-nodes or anything equivalent to i-nodes, another performance bottleneck is that reading even a short file requires two disk accesses.
  - Normally all i-nodes go together, and all data go together.
  - Idea is to put i-node closer to actual file data.
- Improvement
  - Solution 1: Put the i-nodes in the middle of the disk, thus reducing the average seek between the i-node and the first block by a factor of two.
  - Solution 2: Break disk into regions--"Cylinder Groups". Put blocks that are "close together" in the same cylinder group.
    - Example: figure 4-29 (b)
    - Try to allocate i-node and blocks in the file within the same cylinder group.

Reducing Disk Arm Motion (4)

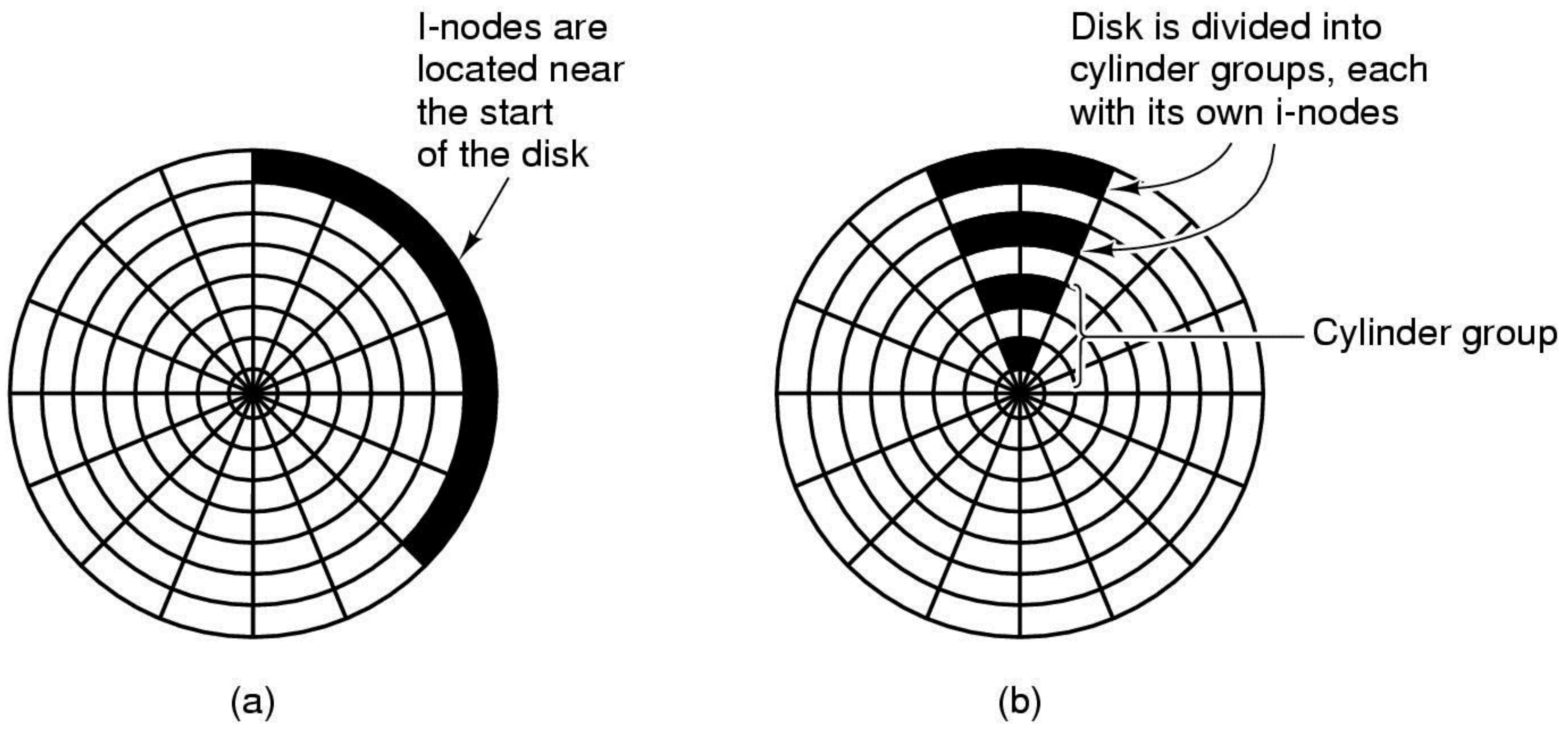

I-nodes are located near the start of the disk
Disk is divided into cylinder groups, each with its own i-nodes
Cylinder group
(a)
(b)

Figure 4-29

(a) i-nodes placed at the start of the disk

(b) Disk divided into cylinder groups, each with its own blocks and i-nodes

Summary

- File System Management and Optimization
  - Disk Space Management
    - Block Size
    - Keeping Track of Free Blocks
  - File System Backups
    - Logical Dump Algorithm
  - File System Consistency
    - Unix fsck: block consistency, directory consistency
  - File System Performance
    - Caching
    - Block Read Ahead
    - Reducing Disk-Arm Motion

Homework

- P334 23, 25, 31, 34