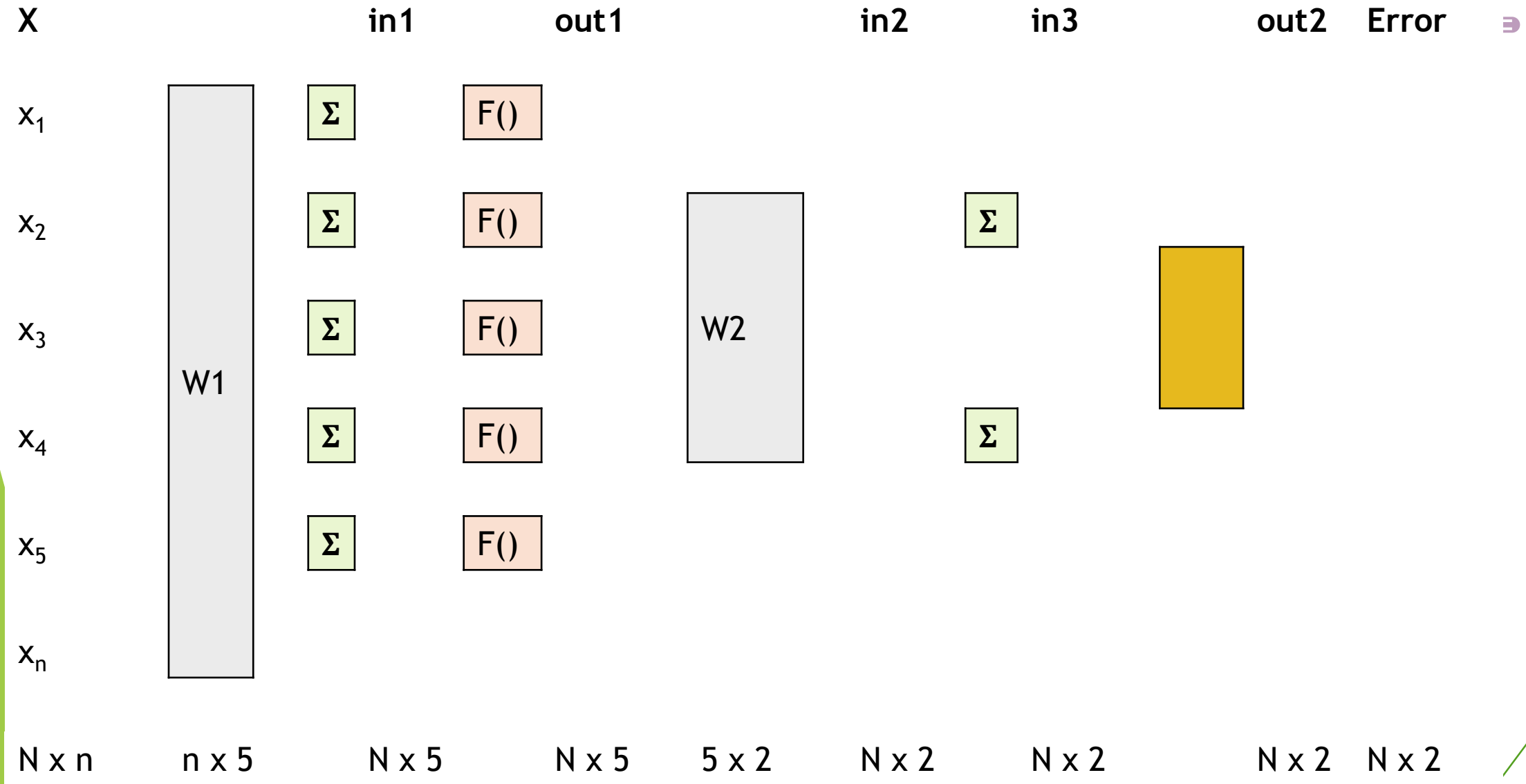
File Systems - II

Table of Contents


X
in1
out1
in2
in3
out2
Error
x1
x2
x3
x4
x5
xn
W1
Σ
F()
W2
N x n
n x 5
N x 5
N x 5
5 x 2
N x 2
N x 2
N x 2
N x 2

File Crash

- To recap, a file is stored as
  - An entry in the directory
  - One or more blocks to store attributes and disk addresses (i-node in case of linux, MFT record in NTFS)
  - Data blocks (where file contents are actually stored)
- Deleting a file shall require
  - Deleting respective entry from the directory. (the entry may be marked as deleted instead of replacing this by some randomized data or zeros)
  - Unassign i-node and mark it free to be assigned for next file.
  - Mark data blocks as free and add to the list of free blocks through necessary changes in bitmap.

File Crash (contd.)

- Consider following steps for removal of files in Linux:
  - Remove the entry corresponding to file from the directory.
  - Release the i-node to the pool of free i-nodes.
  - Return all the disk blocks to the list of free disk blocks.
- If there is no crash, these operations can be carried out in any order.

- Crash happens after first step:
  - Blocks corresponding to i-node as well as data blocks are not marked free.
  - The i-node and file blocks will not be accessible from any file, but will also not be available
  - Available disk storage is reduced but no effect on file system consistency.
- Crash happens after second step:
  - If the crash occurs after the second step, only the data blocks are lost.

File Crash (contd.)

- Consider the steps for removal of files in Linux are carried out in following order:
  - Release the i-node to the pool of free i-nodes.
  - Remove the entry corresponding to file from the directory.
  - Return all the disk blocks to the list of free disk blocks.
- If there is no crash, these operations can be carried out in any order.

- Crash happens after first step:
  - i-node is released first, and returned to pool of free i-nodes, the old directory entry will continue
  - If i-node is reassigned (assigned to new file), undeleted directory entry shall point to this new file (the wrong file).
  - Inconsistent file system. A user may have access to file of another user.
  - Data blocks are not returned to free list and disk storage is reduced.
- Crash happens after second step:
  - Data blocks are not returned to free list resulting in reduction in effective disk capacity.

File Crash (contd.)

- Consider the steps for removal of files in Linux are carried out in following order:
  - Return all the disk blocks to the list of free disk blocks.
  - Release the i-node to the pool of free i-nodes.
  - Remove the entry corresponding to file from the directory.
- If there is no crash, these operations can be carried out in any order.

- Crash after first step:
  - Valid directory entry pointing to i-node that points to data-blocks in free list. When these blocks are assigned to other file(s), directory entry is pointing to one or more file
  - two or more files randomly sharing the same blocks
  - File system is inconsistent.
- Crash after second step
  - Directory entry is still valid and reassignment of i-node shall mean that it is pointing to a new file.
  - File system is inconsistent.

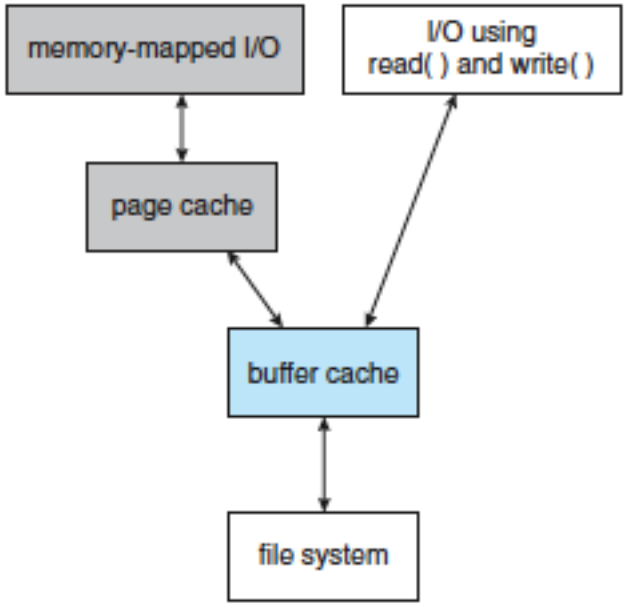
File Recovery: Journaling File System

- Write a log entry listing the three actions to be completed.
- The log entry is then written to disk.
- After the operations complete successfully, the log entry is erased.
- If the system now crashes, upon recovery the file system can check the log to see if any operations were pending.
- If so, all of them can be rerun (multiple times in the event of repeated crashes) until the file is correctly removed.
- To make journaling work, the logged operations must be idempotent, which means they can be repeated as often as necessary without harm.
- Operations such as ‘‘Update the bitmap to mark i-node k or block n as free’’ can be repeated
- Searching a directory and removing any entry called foobar is also idempotent.
- Adding the newly freed blocks from kth i-node to the end of the free list is not idempotent
- The more-expensive operation ‘‘Search the list of free blocks and add block n to it if it is not already present’’ is idempotent.
- For added reliability, a file system can introduce the database concept of an atomic transaction.

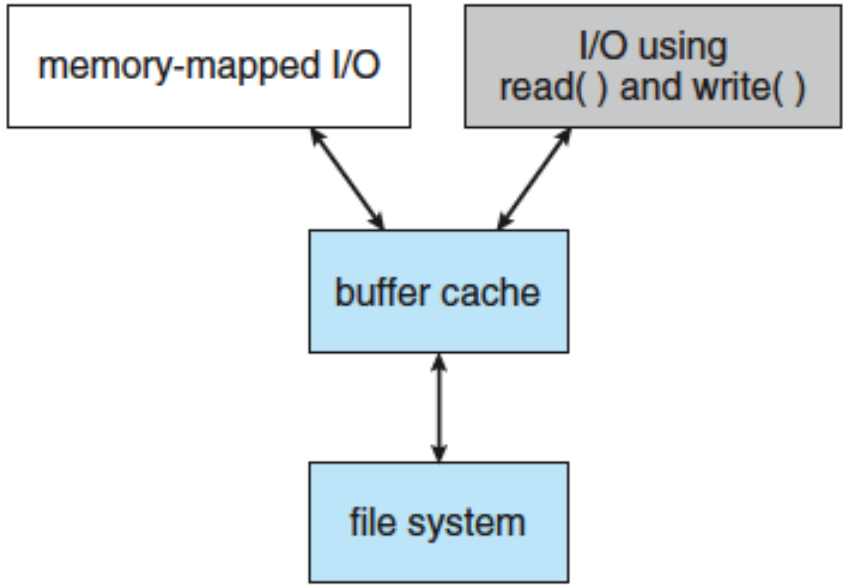
File System: Performance Issues

- Block Size:
  - Large block size means larger internal fragmentation. Small files waste large amounts of disk space;
  - Small block size: Most files will span multiple blocks; needing multiple seeks and rotation delays to read:
- Data blocks and i-blocks
  - If too far away, seek time shall increase.
- Solutions
  - Write blocks in a contiguous area (log) on disk and integrate it in file system periodically
  - Caching data at disk controller and memory

Caching



Separate memory area called buffer cache used to cache disk data.
Disk caching not integrated with Virtual memory management.
Requires double-caching.
Wastes memory as well as CPU and I/O cycles
Inconsistencies between two caches can result in corrupt files.



A unified buffer cache allows both memory mapping and the read() and write() system calls use the same page cache.
Avoids double caching improving efficiency and CPU+I/O utilization.
Allows the virtual memory system to manage file-system data.

Log Structured File Systems (LFS)

- Caching and read-ahead means that many disk blocks are cached into disk controller as well as main memory cache, Disk read requests are largely met from cache itself.
- Most disk accesses are writes – writing to i-node of directory, directory entry, i-node of file, data block(s) of file. Of these, first three are small writes and scattered over disk. Delaying these writes may lead to inconsistent file system in case of crash. Writing these on any modification leads to inefficiency.
- The basic idea is to structure the entire disk or part thereof as a great big log. The log may be in a separate section of the file system or even on a separate disk spindle.
- Periodically, all the pending writes are collected into a single segment and written at the end of the log in the disk.
  - This segment is collection of i-nodes, directory blocks, and data blocks, all mixed together.
  - Each segment has a summary at the beginning to signify what is in a segment.
  - Each segment also contains i-node map to locate -nodes are scattered within the segment
- Once the changes are written to this log, they are considered to be committed,
- Meanwhile, these log entries are replayed across the actual filesystem structures.; a pointer is updated to indicate which actions have completed.
- A committed transaction is removed from the log file, (maintained as a circular buffer).

Free Space Management

- Bit Vector or Bitmap
- Linked List
- Grouping
- Counting
- Space Maps

Bit Vector or Bitmap

- Free space list is implemented as a bit vector – one bit per block. Also called bit map.
- If bit = 1, block is free else it is allocated. First block is block zero.
- Bit vector for a disk where blocks 2, 4, 5, 7, 9, 10, 11, 12, 13, 17, 18, 23, 25, 26, and 27 are free and the rest are allocated. 00101101011111000110000101111000000 ...
- Advantages
  - Simplicity
  - Less space requirement
  - Efficiency in finding the first free block or n consecutive free blocks on the disk. (first non-zero word needs to be scanned)
- Limitations:
  - For efficiency, entire bit vector needs to be in main memory.
  - For large disks, this may not be practical.
  - Inefficient updation of bitmaps especially when allocating/freeing large number of data blocks (scattered over entire disk) 1 GB of data on a 1-TB disk could.
- Bit map size
  - Disk size = 1 GB, Block size =512 bytes, Number of blocks = 2^{21}, Bit map size = 2^{21} bits = 2^{18} bytes = 256 KB
  - Disk size = 1 TB, Block size =4 KB, Number of blocks = 2^{30}, Bit map size = 2^{30} bits = 2^{27} bytes = 128 MB
  - Disk size = 16 TB, Block size =4 KB, Number of blocks = 2^{32}, Bit map size = 2^{32} bits = 2^{29} bytes = 512 MB
- Reducing Bit map size: Clustering blocks: 1 bit per cluster

Linked List

- Create a linked list of all the free disk blocks,
- Header of this list (i.e. first free block) is stored at a predefined location in hard disk and cached in memory for faster access.
- This first block contains a pointer to the next free disk block, and so on.
- When a block is freed, it is returned to the free list.
- Simple but not efficient
  - Need to read each block for traversal, which requires substantial I/O time. (Most often, traversing the free list is not a frequent action)
  - Does not keep any information on contiguous blocks
- Another limitation
  - blocks in the list may not be ordered (may result in spaghetti like structure); keeping ordered list may require time for insertion
  - blocks allocated to a file may not be in near vicinity increasing seek time.

Grouping

- Store addresses of n free blocks in the first free block.
  - First n−1 of these blocks are actually free.
  - Last address points to a block having addresses of another n free blocks, and so on.
  - We shall call this block as address block.
- Advantage:
  - The addresses of a large number of free blocks can now be found quickly, unlike linked list case.
- Issues:
  - If some entries in an address block are utilized for allocating blocks, how and when are these replaced.
  - Whenever one free block is allocated, should OS search for address of a free block and put it in this address block? Or just mark this address as non-free block and replace all entries for allocated blocks by addresses of free blocks only when the number of entries for free block go below a certain threshold?
  - When to load next address block?
  - Maintaining information about contiguous blocks?

Counting

- Previous approaches do not specifically keep information on contiguous blocks.
- When a file is deleted, all its blocks are freed simultaneously, and some of these may be contiguous.
- This scheme keeps 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 is an address and a count.
- Improves time-efficiency but space required per entry is more.
- Algorithms for managing entries is complex as whenever a block is freed, status of its adjacent block(s) need to be checked.
- For efficient lookup, insertion, and deletion, entries can be stored in a balanced tree, rather than a linked list,

Space Maps

- Suited for file system with large number of files and directories. Objective is to reduce I/O overhead.
- Physical disk space divided into metaslabs (chunks of manageable sizes)
- Free space management
  - Counting algorithm is used for free block management.
  - Log-structured file system technique used to record changes to spacemap in memory.
  - Time-sequenced log of all block activity (allocating and freeing) in counting format is maintained.
  - Log is written to in-memory space map
- Spacemap
  - Each metaslab has one associated in-memory space map.
  - Loaded into memory as a balanced tree structure (for very efficient operation), indexed by offset,
  - The space map keeps time-ordreed information on all block activity (allocating and freeing) in counting format.
  - In-memory spacemap is an accurate representation of the allocated and free space in the metaslab
- Spacemap is condensed as much as possible by combining contiguous free blocks into a single entry.
- Free-space list on disk is updated as part of the transaction-oriented operations
- In essence, the log plus the balanced tree is the free list.

File Consistency Check

- Consistency checks: block consistency check, file consistency check
- Block consistency check program builds two tables. First table keeps track of how many times each block is present in a file; second table records how often each block is present in the free list (or the bitmap of free blocks).
- The program then reads all the i-nodes to identify all blocks allocated to a file. As each block number is read, its counter in the first table is incremented.
- The program then examines the free list or bitmap to find all the blocks that are not in use. Increments respective counter in second table.
- If the file system is consistent, each block will have a 1 either in the first table or in the second table
- A block not appearing in either table is marked as a missing block. While missing blocks do no real harm, they waste space and thus reduce the capacity of the disk. File system checker just adds them to the free list.
- Issues: Same data block is present in two or more files. If either of these files is removed, block shall be in both allocated as well as free list. If both files are removed, the block will be put onto the free list twice. This is dealt with by allocating a free block, copy the contents of shared block, and insert the copy into one of the files. File-system is consistent though files are not. User is informed of this so that he can check which file is corrupted.
- File-system checker also checks the directory system. Creates a table of counters per file. Traverses directory structure from root directory and for every i-node, increments counter for respective file.
- Compares count of a file with link count in its i-node. If these match, file system is consistent.
- Errors
  - Link count (i-node) > count: even if all the linked files are removed, the count will still be nonzero and the inode will not be removed.
  - Link count (i-node) < count: deletion of as many linked files as link count shall result in release of i-node and data blocks of the file but some directory entry shall still point to this file. Reassignment of i-node shall make this directory entry point to another file.
- In either case, the solution is just to force the link count in the inode to the actual number of directory entries.

File Backup

- Backup allow recovery from accidental deletes and/or disastor
- Backups are expensive in terms of time as well as space.
- Issues:
  - It is usually desirable to back up only specific directories and everything in them rather than the entire file system. As long as source files are maintained, object and executable files need not be stored. Similarly downloaded software need not be backuped unless a specific version no more available online is needed. Temporary files need not be backuped.
  - Second, it is wasteful to back up files that have not changed since the previous backup. To save time and space, incremental backups should be carried out. Backup only those files that have changed since they were last dumped.
    - **Makes recovery more complicated, because first the most recent full dump has to be restored, followed by all the incremental dumps in reverse order.**
    - **Software for Incremental backup needs to traverse file system to identify what has changed recently. Keeping hash of directory and files can make this time efficient.**
  - To save space, compressing data being backup is desirable. Any error in writing compressed data may fail recovery.
  - Backup needs to be offline ie when file system is not in use. Otherwise, if files and directories are being added, deleted, and modified during the dumping process, the resulting dump may be inconsistent. Keeping system offline for backup duration (often in hours) is a challenge.
  - Backups need to be secured. Any backup disk left unattended may lead to data theft.

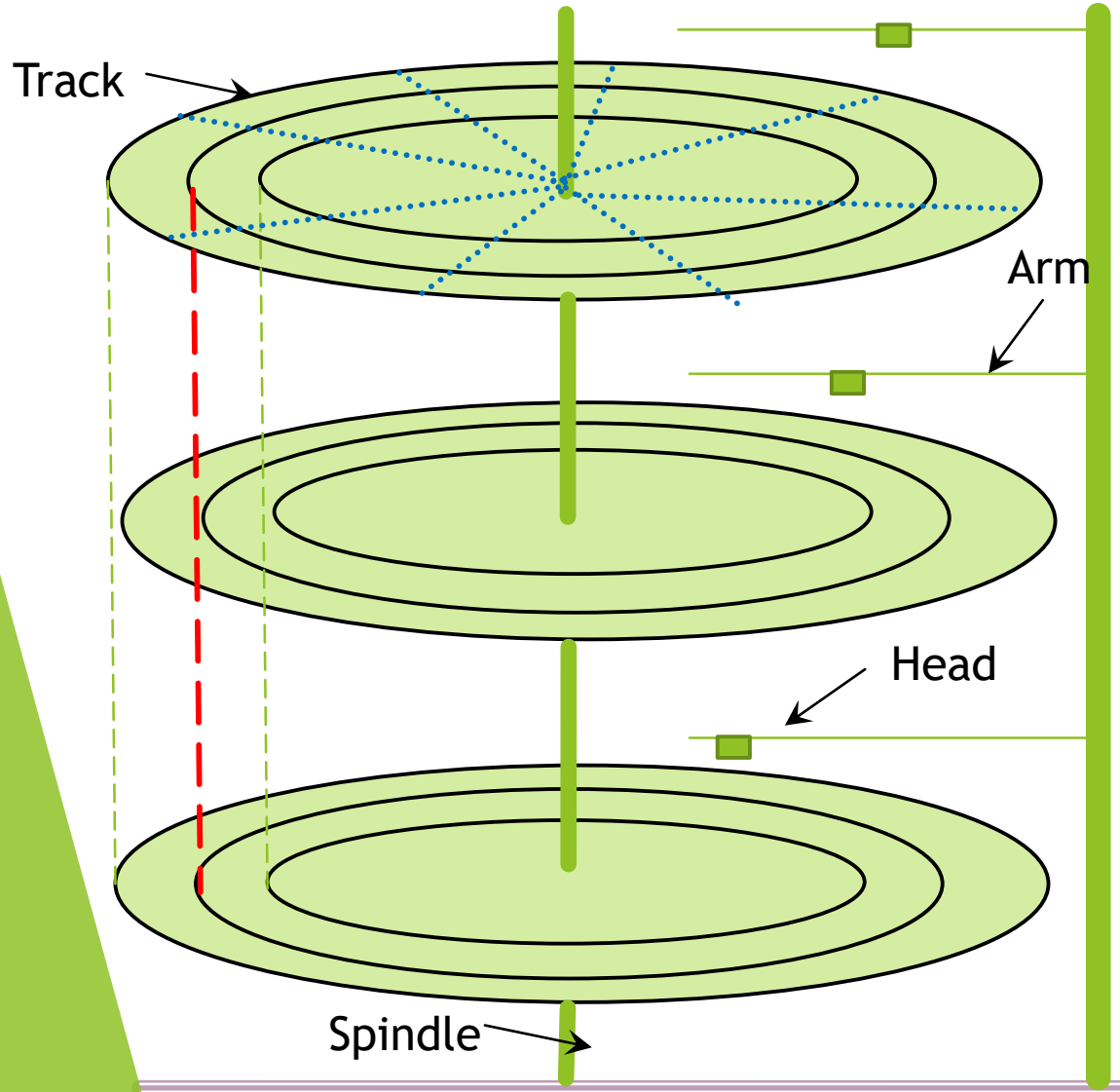
File Backup – Physical Dump

- Two strategies are used: **(1) Physical dump and (2) Logical dump**
- Physical dump: A physical dump starts at block 0 of the disk, writes all the disk blocks onto the output disk in order, and stops when it has copied the last one.
- Advantages:
  - Simple program that need not be aware of specifics of file structure and how OS stores files, directories and file systems.
  - Backup can be done at high speed as blocks need to be copied only
- Should avoid backup of
  - unused disk blocks. skipping unused blocks requires writing the number of each block in front of the block as kth block on disk may not correspond to kth block on backup (due to skipping unused blocks)
  - bad blocks
  - Internal files such as paging and hibernation files (windows)
- To avoid backup of unused and bad blocks, dumping program need to be aware of how OS stores can obtain access to the free-block data structure, it can avoid dumping
- Limitations: (As physical dump is not aware of specifics of file system as well as file/directory structure)
  - inability to skip selected directories,
  - make incremental dumps,
  - restore individual files upon request.

File Backup – Logical Dump

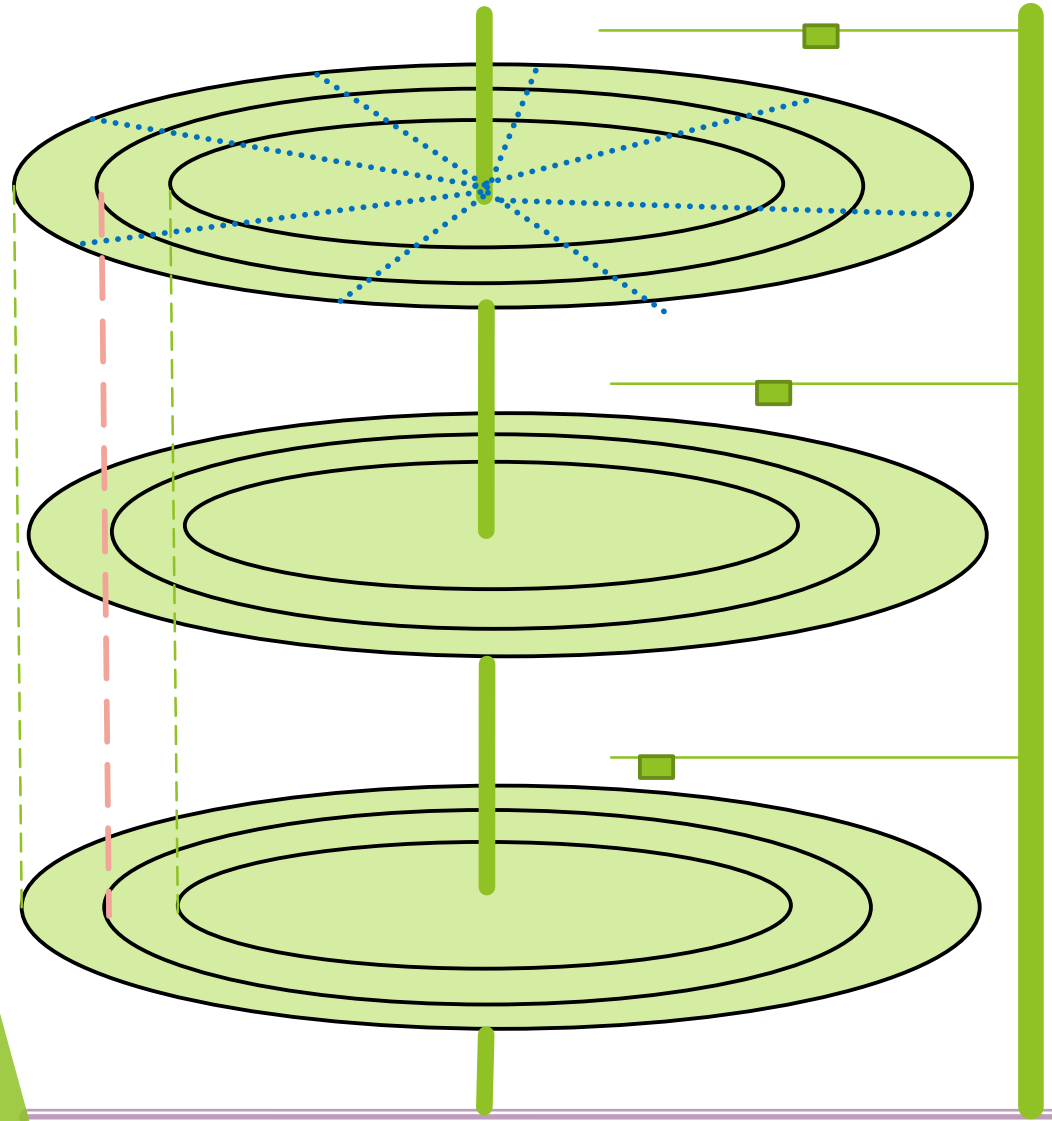
- The dump algorithm maintains a bitmap indexed by i-node number with several bits per i-node. Bits will be set and cleared in this map as the algorithm proceeds.
- The algorithm operates in four phases.
  - Traverses directory structure (through any tree traversal) from the starting directory. For each modified file, its i-node is marked in the bitmap. Each directory is also marked and then recursively inspected. All modified files are identified and marked in the bitmap.
  - Recursively walks the tree again, unmarking any directories that have no modified files or directories in them or under them.
  - Scans the i-nodes in numerical order and dumps all the marked directories. Each directory is prefixed by the directory's attributes (owner, times, etc.) so that they can be restored.
  - All marked files (prefixed by their attributes) are also dumped, again This completes the dump.
- Restoring a file system from the dump disk
  - An empty file system is created on the disk.
  - The most recent full dump is restored – first directories to help create a skeleton of the file system and then the files themselves are restored.
  - This process is then repeated with the first incremental dump made after the full dump, then the next one, and so on.
- Issues
  - Free block list (not being a file) is not dumped and is reconstructed as the complement of the set of blocks contained in all the files combined.
  - Links: A file linked to two or more directories is restored only one time
  - Holes: UNIX files may contain holes (open a file, write a few bytes, then seek to a distant file offset and write a few more bytes) The blocks in between are not part of the file and should not be dumped and must not be restored.
  - Special files, named pipes, and anything that is not a real file should never be dumped

Structure of Hard Disk



- Consists of a stack of disks (coated with magnetic surface), connected to a spindle.
- Each disk has concentric tracks (where data is actually recorded)
- Head moves along an arm to read data from a specific track.
- One head/disk surface
- Corresponding tracks on all surface constitute a cylinder (Red dotted line)
- Each track divided radially into equal number of sectors (blue dotted line)
- Data density (bits per unit length) increases as we move inward from outermost track
- Tracks near center not used as reliable read/write not warranted (owing to physical size of head)

Structure of Hard Disk (contd.)



- Disk rotating at constant angular speed; time to read from/write to a sector is same
- Read/Write
  - Move head along arm to position on track
  - When requisite sector comes under head, start transferring data
  - Identification of a sector is through a format string written at the beginning of a sector
  - Marking sectors through distinguishing sequence of bits is formatting
- Read/write time
  - Seek Time : time to move head to a track
  - Rotational Delay: time to bring a sector under track
  - Data transfer time: time to transfer data to/from sector

Disk Access Time

- Disk access time = seek time + latency time (avg. rotational delay) + data transfer time
- Avg. rotational delay = time to rotate half a track
- Disk rotating at 6000 rpm (revolutions per minute); Number of sectors/track = 8;
- In 60 second, disk is rotated 6000 times
- 6000 tracks transferred in 60 seconds
- Time to transfer one track = 1/100 seconds = 10ms
- Avg. Rotational Delay = 10/2 ms = 5ms
- Data transfer time = 1.25ms per sector
- If sectors being read are non-contiguous, seek time adds to delay for each access.

Disk Addressing

- Track 0 is outermost track
- Consider a large file with contiguous allocation. Two possible ways of storing file on hard disk
  - Surface wise: Requires multiple seek times (moving from track to track) while reading data. Disk address <surface number, track number, sector number>
  - Cylinder wise: Preferred as it requires less number of seeks. <cylinder number, surface number, sector number>
- Most modern drives provide access to logical address space (without any information on how disk address is related to disk geometry). OS may not be able to implement cylinder-wise addressing.

Disk Volume

- Volume is divided into partitions
  - First sector of volume is Master Boot Record (MBR)
  - MBR also contains partition table – information on disk partitions – type of partition, disk addresses for start and end of partition, whether partition is active or not
- Partition
  - The first sector of each partition is a boot record.
  - This is followed by information on file system. Format of this information is specific to file system.

Modern Disks

- By the time a track is read and head moves to next track, there is a delay (seek time). During this time, disk has rotated past the first sector on next track.
- Solutions:
- Cylinder Skew:
  - Disk speed= 10,000 rpm, 300 sectors per track, seek time = 800 μ sec
  - Time to move over 300 sectors = time for one reovolution = 60/10,000 = 6000 μ sec
  - Time to move over one sector = 20 μ sec
  - Sectors moved over in 800 μ sec = 40
  - Cylinder Skew = 40 sectors
- Head skew needs to be taken into consideration: as switching between heads also takes a finite time

Cylinder Skew

- Home Assignments: Compute cylinder skew
  - Disk speed= 6,000 rpm, 200 sectors per track, seek time = 1000 μ sec
  - Disk speed= 7,500 rpm, 240 sectors per track, seek time = 800 μ sec
  - Disk speed= 8,000 rpm, 320 sectors per track, seek time = 600 μ sec

Disk Scheduling

- FCFS (first come first served)
- SSTF (shortest seek time first)
- SCAN
- LOOK
- C-SCAN
- C-LOOK

FCFS

- Disk requests are served in the order these are received.
- This algorithm is intrinsically fair.
- No optimization in respect of seek time or rotational delay.
- May lead to excessive head movements and higher seek time.
- Example:
  - Head is at track 50.
  - Requests received are from tracks 23, 34, 5, 78, 29, 63, 43, 15
  - Total head movement = |23-50| + |34-23| + |5-34| + |78-5| + |29-78| + |63 – 29| + |43 – 63| + |15 – 43|
  - 27+11+29+73+49+34+20+28
  - 271

SSTF

- A greedy algorithm aims at reducing seek time overhead.
- Next disk request served is that of the nearest track.
- Not optimal.
- May lead to starvation.
- Example:
  - Head is at track 50.
  - Requests received are from tracks 23, 34, 5, 78, 29, 63, 43, 15
  - As track 43 is nearest to 50,it is selected at first.
  - Total head movement: |43-50|+|34-43|+|29-34|+|23-29|+|5-23|+|63-5|+|78-63|
  - = 7+9+5+6+18+58+15
  - = 118

SCAN

- Disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
- At the other end, the direction of head movement is reversed, and servicing continues.
- The head continuously scans back and forth across the disk.
- SCAN algorithm is called the elevator algorithm.
- Example:
  - Head is at track 50. Tracks are numbered 1-99.
  - Requests received are from tracks 23, 34, 5, 78, 29, 63, 43, 15
  - As track 43 is nearest to 50,it is selected at first. And disk arm moves towards track 1.
  - Total head movement: |43-50|+|34-43|+|29-34|+|23-29|+|5-23|+|0-5|+|63-0|+|78-63|
  - = 7+9+5+6+18+5+63+15
  - = 128

LOOK

- Similar to SCAN but the arm moves to the last request along a direction.
- Disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the last request along that direction.
- And then, the direction of head movement is reversed, and servicing continues.
- The head continuously scans back and forth across the disk.
- Example:
  - Head is at track 50. Tracks are numbered 1-99.
  - Requests received are from tracks 23, 34, 5, 78, 29, 63, 43, 15
  - As track 43 is nearest to 50,it is selected at first. And disk arm moves towards track 5 (the last request).
  - Total head movement: |43-50|+|34-43|+|29-34|+|23-29|+|5-23|+|63-5|+|78-63|
  - = 7+9+5+6+18+58+15
  - = 118

Other variants

- For a uniform distribution of requests for cylinders, SCAN algorithm and the head reaching one end; there is no point in servicing disk requests in reverse direction as this area was just recently served. The heaviest density of requests is at the other end of the disk. These requests have waited the longest. So should served next.
- C-SCAN
  - Cylinder numbers are considered circular. Once the disk arm reaches the last track along a direction; it is moved to the opposite end without servicing any request on return trip and scan starts.
- C-LOOK
  - Similar to look but as in C-SCAN, once the last request in a direction has been served; the head reverses direction but no request is serviced in return trip.
- N-Step SCAN
  - Same as SCAN but disk requests are served in a batch of N requests.

Solid State Devices

- Solid-state disks (SSD)
  - are more reliable as no moving parts
  - and faster because no seek and rotational delay latency
  - consume less power.
  - more expensive per megabyte and have less capacity
  - shorter life spans than hard disks
- Random accesses are just as fast as sequential ones and many of the problems of traditional disks go away.
- Issues
  - each block can be written only a limited number of times, so writes to a given block needs to be avoided and blocks belonging to frequently updated files may need to be moved
  - Secure deletion of file can not be warranted (Home Assignment: Explain why this should be the case)

Home Assignments

- Write notes on
  - Swap space management.
  - RAID
  - Keyboard and Mouse Interface

Thank you.