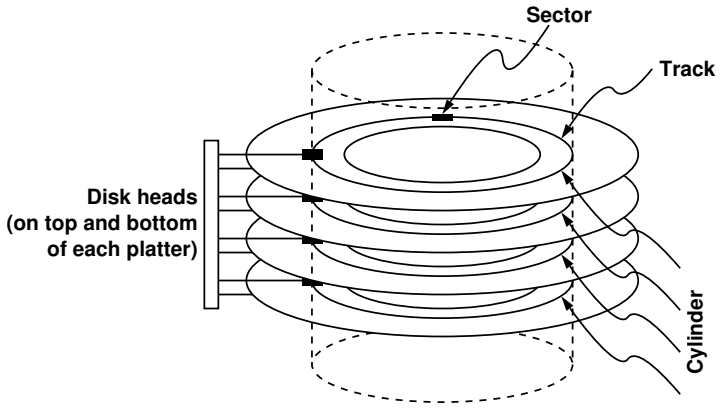
# 6.1 The Basics of File Systems

- UNIX's S5FS
- Disk Architecture
- *Problems with S5FS*
- Improving Performance






## Disk Architecture




- Smallest addressable unit is a *sector*
  - disk address = (*head/surface#*, *cylinder/track#*, *sector#*)






## Rhinopias Disk Drive

| | |
|---|---|
| Rotation speed | 10,000 RPM |
| Number of surfaces | 8 |
| Sector size | 512 bytes |
| Sectors/track | 500-1000; 750 average |
| Tracks/surface | 100,000 |
| Storage capacity | 307.2 billion bytes |
| Average seek time | 4 milliseconds |
| One-track seek time | .2 milliseconds |
| Maximum seek time | 10 milliseconds |






## S5FS on Rhinopias (A Marketing Disaster ...)

- Rhinopias's maximum transfer speed?
  - 63.9 MB/sec
- S5FS's average speed on Rhinopias?
  - average seek time:
    - < 4 milliseconds (say 2)
  - average rotational latency:
    - ~3 milliseconds
  - per-sector transfer time:
    - negligible
  - time/sector: 5 milliseconds
  - effective transfer speed: 102.4 KB/sec (.16% of maximum)
- In general, we have:
  - *access time = seek time + rotational latency + data transfer time*
    - some people would use the term "response time" to mean "access time"






# 6.1 The Basics of File Systems

- UNIX's S5FS
- Disk Architecture
- Problems with S5FS
- *Improving Performance*






## What to Do About It?

- Hardware
  - employ pre-fetch buffer
    - filled by hardware with what's underneath head
    - helps reads a bit; doesn't help writes
- Software
  - better on-disk data structures
    - increase block size
    - minimize seek time
    - reduce rotational latency

# FFS

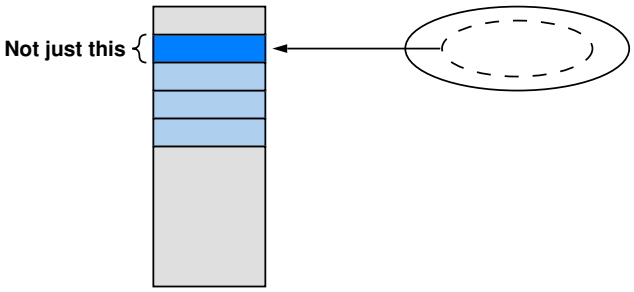
- Better on-disk organization
- Longer component names in directories
- Retains disk map of S5FS







# Larger Block Size

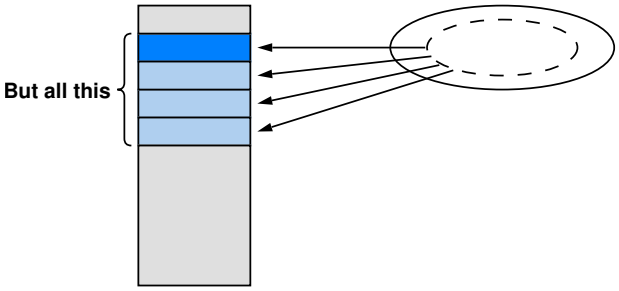









# Larger Block Size

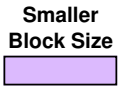









# The Down Side ...

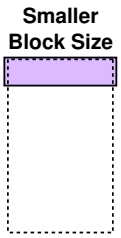









# The Down Side ...

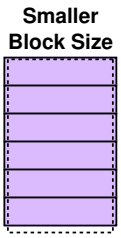




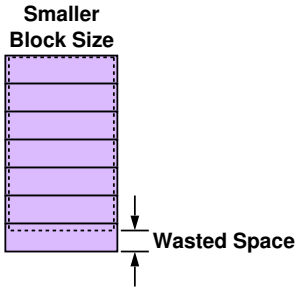





# The Down Side ...



## The Down Side ...

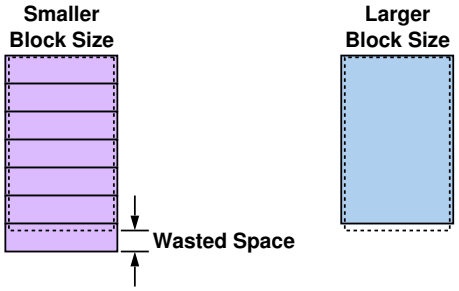



- internal fragmentation






## The Down Side ...

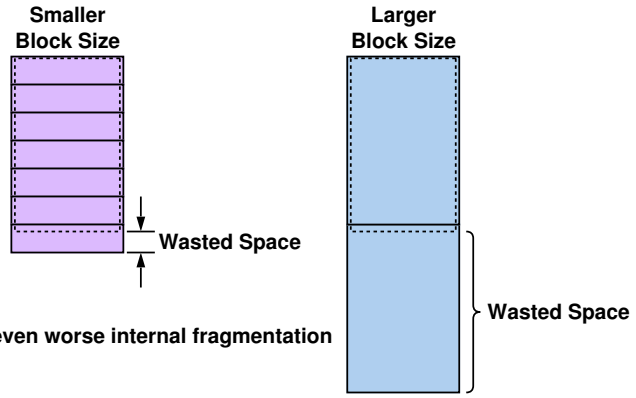








## The Down Side ...

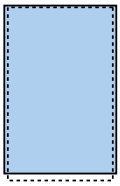



- even worse internal fragmentation





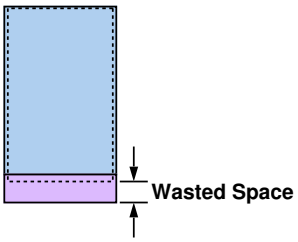

## Two Block Sizes ...








## Two Block Sizes ...




- e.g., 16KB blocks and 1KB fragments
- best of both worlds

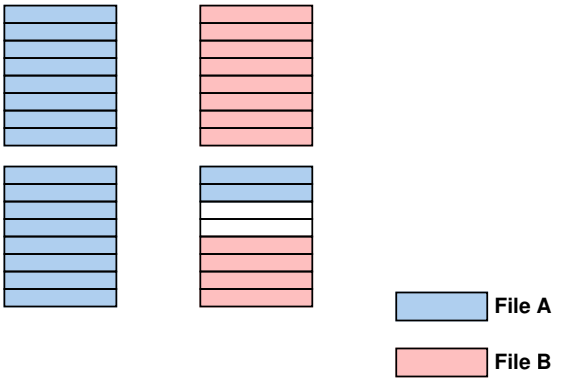





## Rules

- File-system blocks may be split into fragments that can be independently assigned to files
  - fragments assigned to a file must be contiguous and in order
- The number of fragments per block (1, 2, 4, or 8) is fixed for each file system
- Allocation in fragments may only be done on what would be the last block of a file, and only for small files

## Use of Fragments (1)

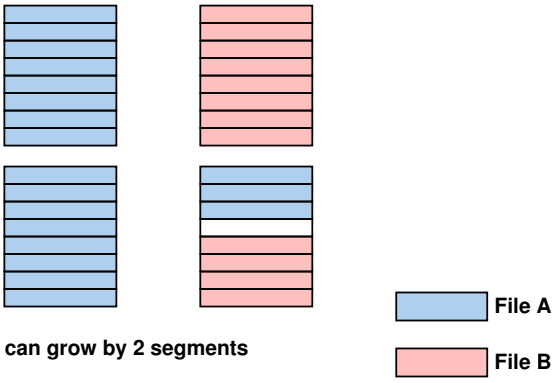








## Use of Fragments (2)

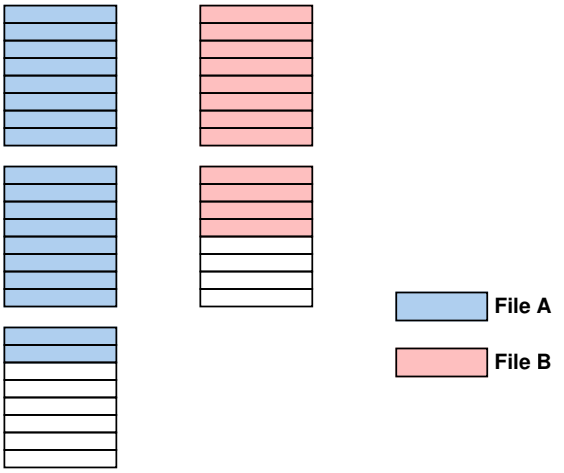



- A can grow by 2 segments






## Use of Fragments (3)









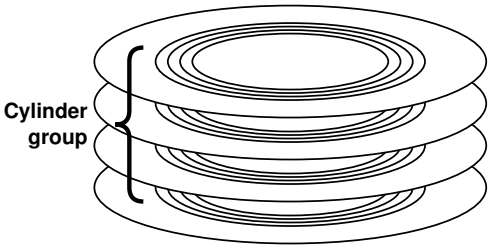
## Minimizing Seek Time

- Keep related things close to one another
- Separate unrelated things






## Cylinder Groups

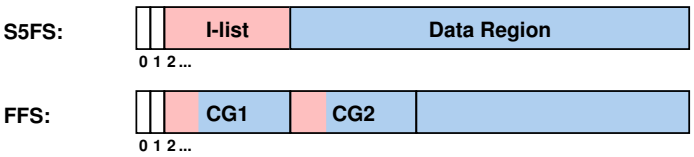



- recall that *seeking* to the *next cyliner/track* is much *faster*



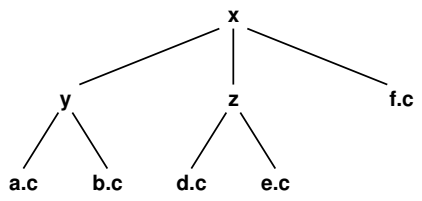



## Minimizing Seek Time




- The practice (heuristics):
  - attempt to put new inodes in the same cylinder group as their directories
  - put inodes for new directories in cylinder groups with "lots" of free space
  - put the beginning of a file (first 10KB, i.e., direct blocks) in the inode's cylinder group
  - put additional portions of the file (each 2MB) in cylinder groups with "lots" of free space

## Locality Of File Access

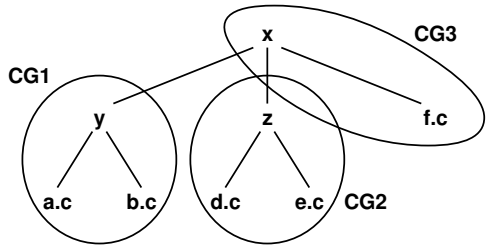



- if access "d.c", likely to access "e.c"






## Locality Of File Access




- if access "d.c", likely to access "e.c"





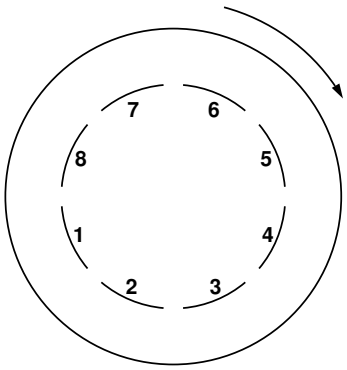

## How Are We Doing? (Part 1)

- Configure Rhinopias with 20 cylinders per group
  - 2-MB file fits entirely within one cylinder group
  - average seek time within cylinder group is ~.3 milliseconds
  - average rotational delay still 3 milliseconds
  - .12 milliseconds required for disk head to pass over 8KB block
  - 3.42 milliseconds for each block
  - 2.4 million bytes/second average effective transfer speed
  - *factor of 20 improvement*
  - 3.7% of maximum possible






## Minimizing Latency









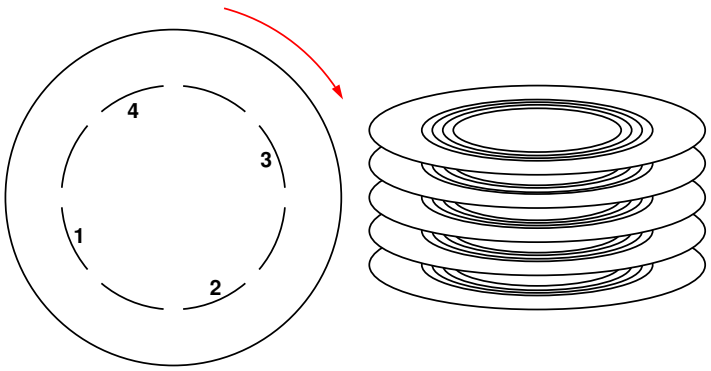
## Numbers

- Rhinopias spins at 10,000 RPM
  - 6 milliseconds/revolution
- 100 microseconds required to service disk-completion interrupt and start next operation
  - typical of early 1980s
- Each block takes 120 microseconds to traverse disk head
- Reading successive blocks is expensive!






## Minimizing Latency




- Block interleaving

## How're We Doing Now? (Part 2)

- Time to read successive blocks (two-way interleaving):
  - after request for second block is issued, must wait 20 microseconds for the beginning of the block to rotate under disk head
  - *factor of 15 improvement*
    - together with other improvements, overall, a factor of 300 improvement






## How're We Doing Now? (Altogether)

- Same setup as before
  - 2-MB file within one cylinder group
  - actually fits in one cylinder
  - block interleaving employed: every other block is skipped
  - .3-millisecond seek to that cylinder
  - 3-millisecond rotational delay for first block
  - 50 blocks/track, but 25 read in each revolution
  - 10.24 revolutions required to read all of file
  - 32.4 MB/second (50% of maximum possible)






## Further Improvements?

- S5FS: 0.16% of capacity
- FFS without block interleaving
  - factor of 20 improvement
  - reached 3.8% of capacity
- FFS with block interleaving
  - another factor of 15 improvement
  - reached 50% of capacity
- What next?






## Larger Transfer Units

- Allocate in whole tracks or cylinders
  - too much wasted space
- Allocate in blocks, but group them together
  - transfer many at once






## Block Clustering

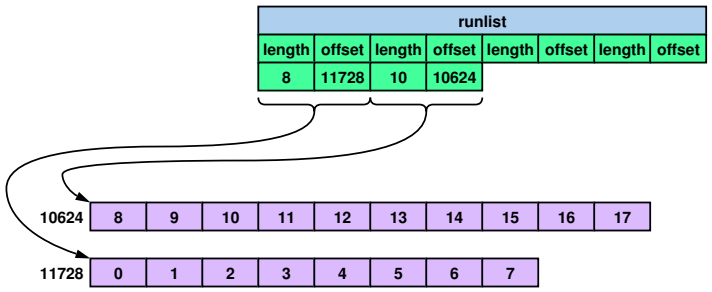
- Allocate space in blocks, eight at a time
- Linux's Ext2 (an FFS clone):
  - allocate eight blocks at a time
  - extra space is available to other files if there is a shortage of space
- FFS on Solaris (~1990)
  - delay disk-space allocation until:
    - 8 blocks are ready to be written
    - or the file is closed






## Extents

- Windows



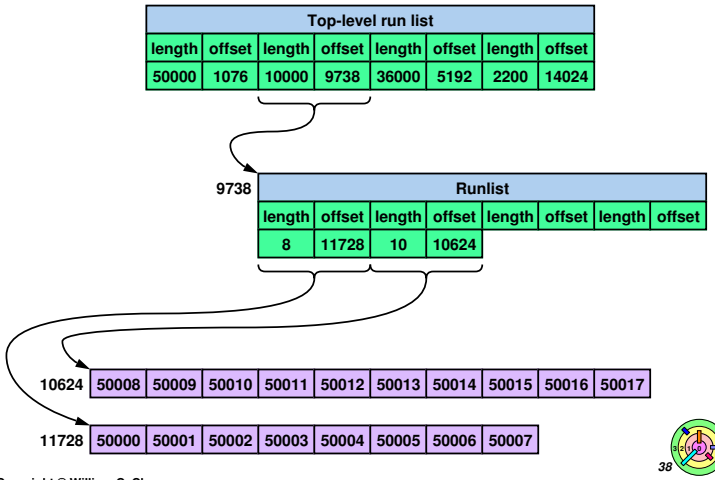
# Problems with Extents

- Could result in highly fragmented disk space
  - lots of small areas of free space
    - external fragmentation
  - solution: use a *defragmenter* to *coalesce* free space
- Random access
  - *linear search* through a long list of extents
  - solution: *multiple levels*
    - usually two levels






# Extents in NTFS

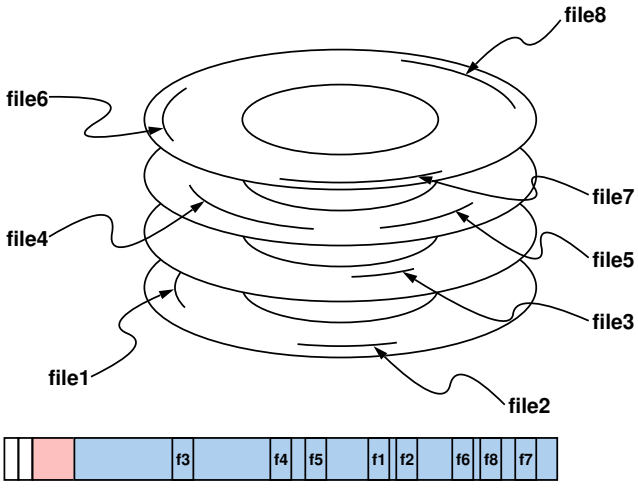








# Are We There Yet?

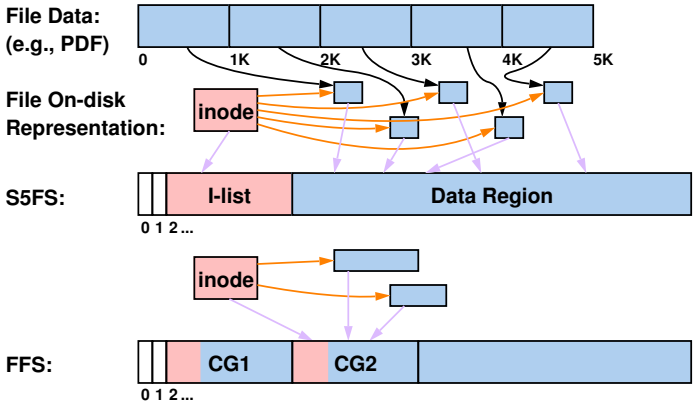








# Recall What A File Look Like in S5FS & FFS

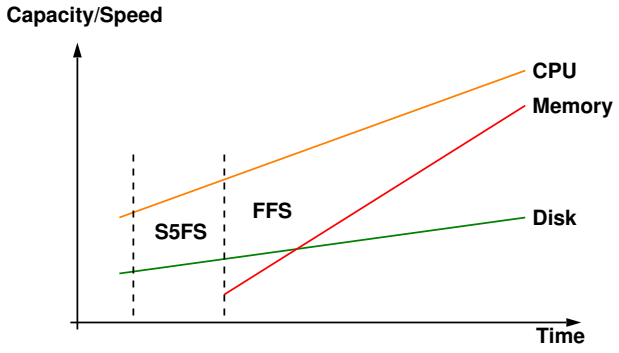








# CPU, Memory, Disk Speeds Over Time

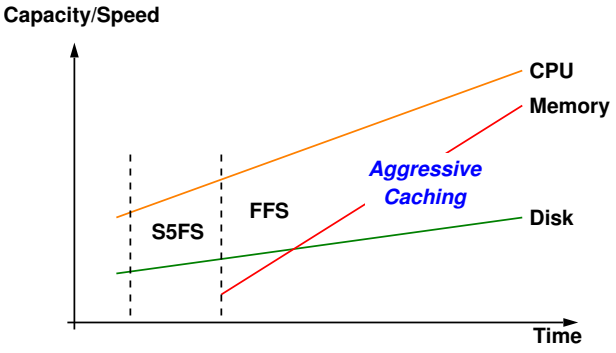








# CPU, Memory, Disk Speeds Over Time



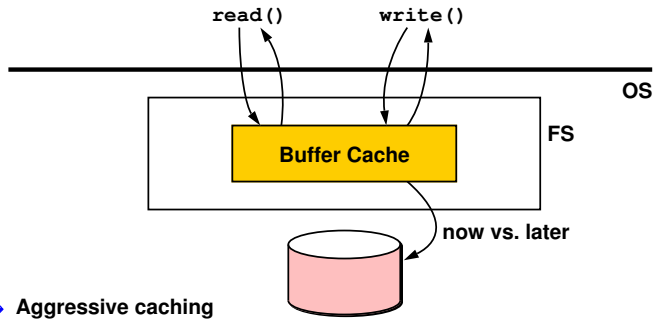
## A Different Approach

- We have lots of primary memory
  - enough to cache all commonly used files
- Read time from disk doesn't matter
- Time for writes does matter






## The Buffer Cache

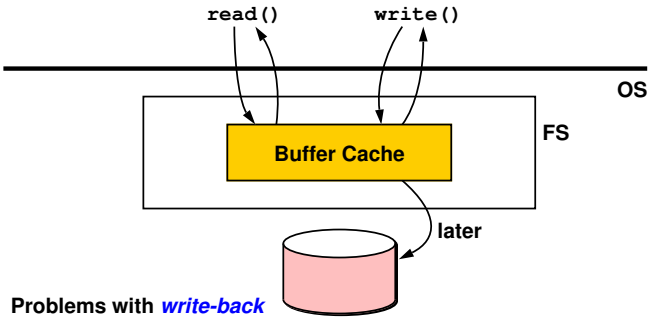



- Aggressive caching
  - most read and write will have a *cache hit*
  - for writes, need to update the disk
    - *write through* vs. *write back*






## The Buffer Cache



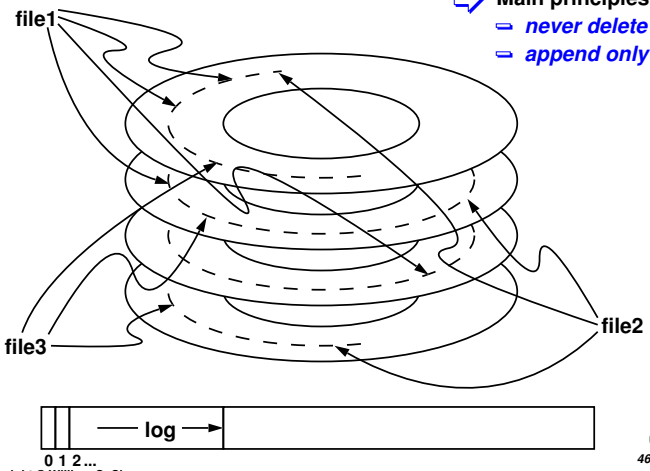

- Problems with *write-back*
  - writes to the disk can wait, may be for quite a while
    - longer the wait, higher the *risk*
- Need a file system optimized for *writing!*
  - how?
    - you organize the disk as a very long *log*






## Log-Structured File Systems

- Main principles
  - *never delete*
  - *append only*



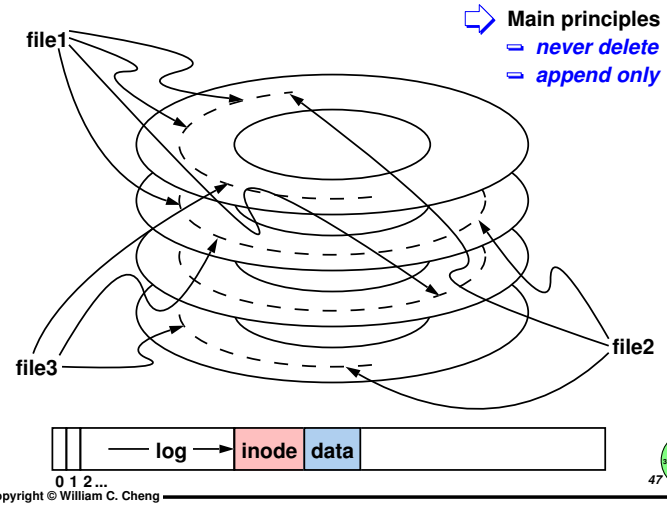






## Log-Structured File Systems

- Main principles
  - *never delete*
  - *append only*







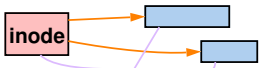


## Log-Structured File Systems

- How does "never delete" and "always append" help with performance?
  - minimize seek latency
  - minimize rotational latency
    - write a cylinder at a time
- *Sprite FS* (a log-structured file system)
  - through batching, a single, long write can write out everything

# LFS Data Placement Example

File On-disk Representation:




LFS:



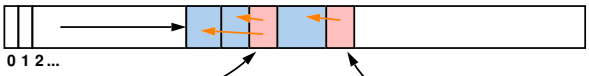





# LFS Data Placement Example

- What happens if you want to modify the file?
  - how does "append-only" really work?
- Ex: you create file A and then file B

LFS:




*Inode Map:* A B

- you modify file A, e.g., append to the last block of file A
- the new file will be referred as A'



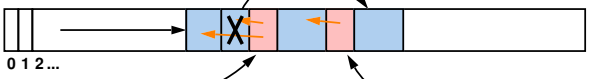




# LFS Data Placement Example

- What happens if you want to modify the file?
  - how does "append-only" really work?
- Ex: you create file A and then file B

LFS:




*Inode Map:* A B

- you modify file A, e.g., append to the last block of file A
- the new file will be referred as A'



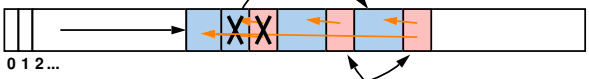




# LFS Data Placement Example

- What happens if you want to modify the file?
  - how does "append-only" really work?
- Ex: you create file A and then file B

LFS:




*Inode Map:* A B

- you modify file A, e.g., append to the last block of file A
- the updated file is still file A
  - but the inode has changed



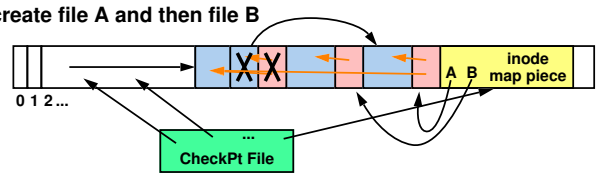




# LFS Data Placement Example

- What happens if you want to modify the file?
  - how does "append-only" really work?
- Ex: you create file A and then file B

LFS:




*Inode Map:*

- you modify file A, e.g., append to the last block of file A
- the updated file is still file A
  - but the inode has changed
- a *piece* of the *inode map* is appended to the log
  - fixed regions (previous version and current version) on the disk keeps track of *all* the *inode map pieces*
    - known as *checkpoint file*

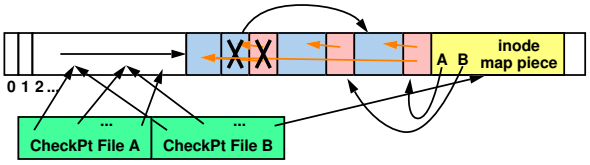






# More On Inode Map

- *Inode Map* cached in primary memory
  - indexed by inode number
  - points to inode on disk
  - written out to disk in pieces as updated
  - *checkpoint file* contains locations of pieces
    - written to disk occasionally
    - two copies: current and previous
    - outside of the "log" part of the LFS




- Commonly/Recently used inodes and other disk blocks cached in primary memory

## LFS Summary

- Advantages
  - good performance for writes
  - can recover from crashes easily through the use of checkpoint files
- Disadvantages
  - can waste a lot of disk space






# Extra Slides





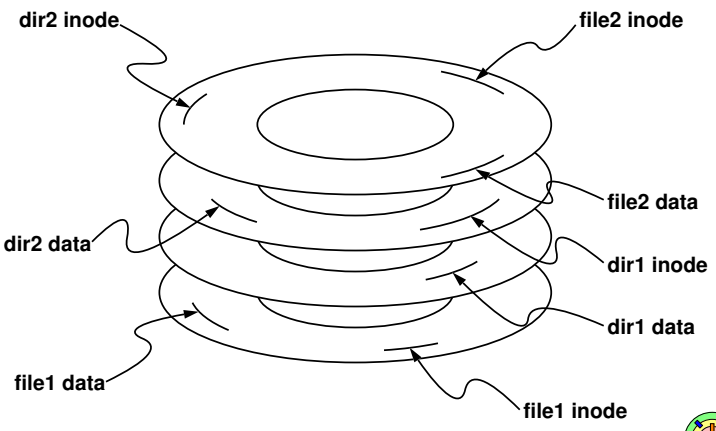

## Example

- We create two single-block files
  - dir1/file1
  - dir2/file2
- FFS
  - allocate and initialize inode for file1 and write it to disk
  - update dir1 to refer to it (and update dir1 inode)
  - write data to file1
    - allocate disk block
    - fill it with data and write to disk
    - update inode
  - six writes, plus six more for the other file
    - seek and rotational delays






## FFS Picture









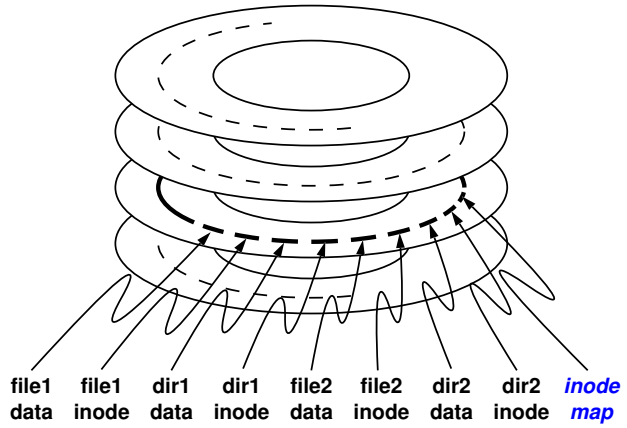
## Example (Continued)

- Sprite (a log-structured file system)
  - one single, long write does everything

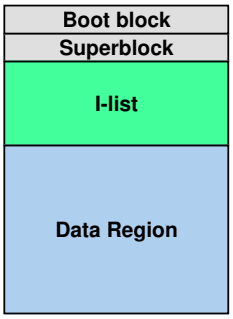





## Sprite Picture



## S5FS Layouts

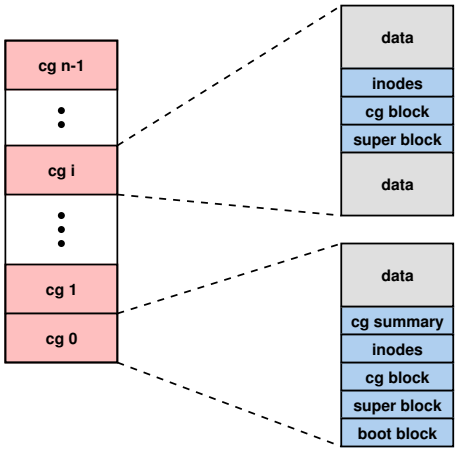








## FFS Layout









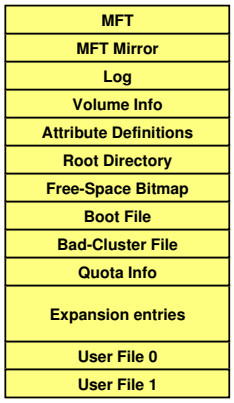
# 6.1 The Basics of File Systems

- UNIX's S5FS
- Disk Architecture
- Problems with S5FS
- Improving Performance
- *Dynamic Inodes*






## NTFS Master File Table

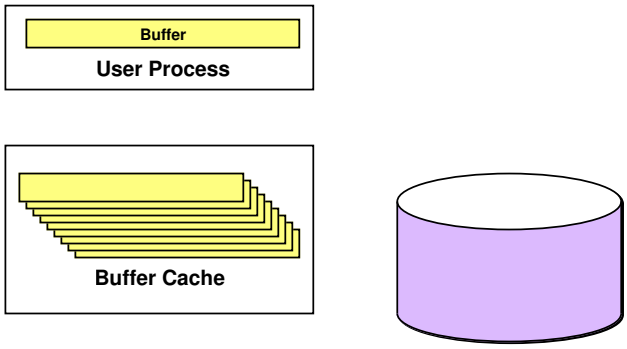








## The Buffer Cache

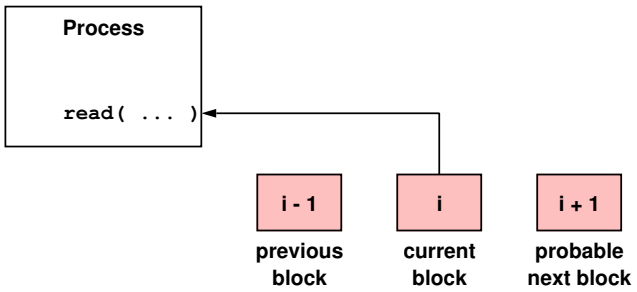








## Multi-Buffered I/O



# Maintaining the Cache