Universidade Estadual de Campinas
Instituto de Computação
MC504 Sistemas Operacionais

T31 Locality and The Fast File System

Referência principal
Ch.41 of *Operating Systems: Three Easy Pieces* by Remzi and Andrea Arpaci-Dusseau (pages.cs.wisc.edu/~remzi/OSTEP/)

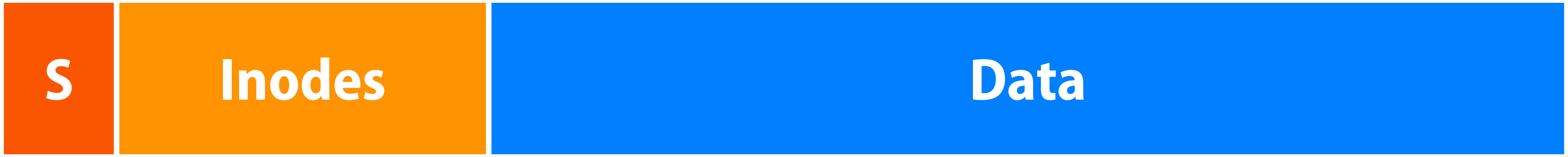
Discutido em classe em 21 de novembro de 2018

Arthur João Catto, PhD

2º semestre de 2018

Unix operating system

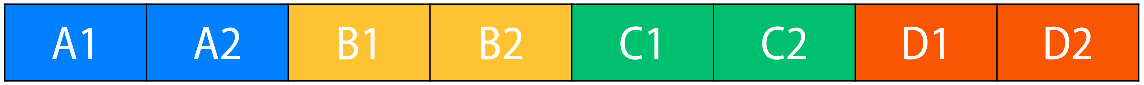
- Original Unix data structures on disk



- The good thing about the old file system
  - Simple and supported the basic abstractions, i.e. files and directories
  - Easy to use
  - A step-forward from earlier approaches
- The Problem
  - Terrible performance
    - Started off bad and got worse over time

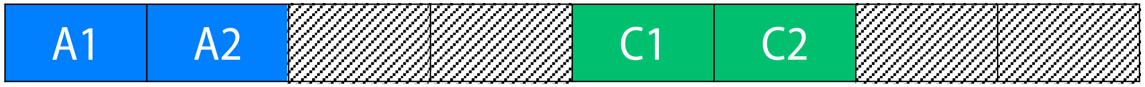
Issues of the old Unix file system

- Treated the disk as a random-access memory
    - Data was spread all over the disk without regard to positioning time
- File system easily fragmented due to uncareful free space management
    - The policy was simply to take the next free block
        - For example, consider 4 files A, B, C and D, each with 2 blocks.



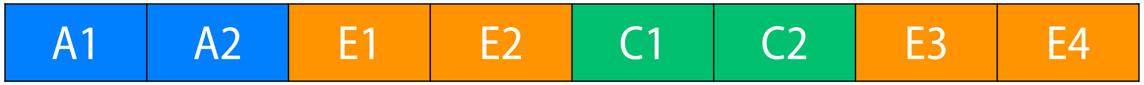
A1	A2	B1	B2	C1	C2	D1	D2

        - If B and D are deleted, the layout becomes



A1	A2	▨	▨	C1	C2	▨	▨

        - Now, if we create a 4-block file E we get



A1	A2	E1	E2	C1	C2	E3	E4

        - And E is spread across the disk.

Issues of the old Unix file system

- The original block size was too small (512 bytes)
  - Good because it reduced internal fragmentation
  - Bad because disk transfer was inefficient.

How to organize on-disk data to improve performance?

What types of allocation policies are required?

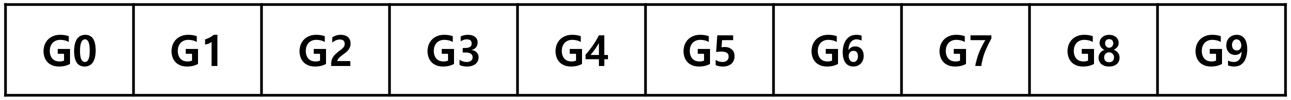
How do we make the file system “disk aware”?

Fast File System

- FFS was designed by a group at Berkeley in the early 80's.
- FFS structures and allocation policies were designed to be "disk aware" and improve performance.
    - It kept same API (`open()`, `read()`, `write()`, etc)
    - The internal implementation was heavily changed.

Organizing Structure: The Cylinder Group

- FFS divides the disk into a bunch of groups. **(Cylinder Group)**
  - Modern file system call cylinder group as block group.



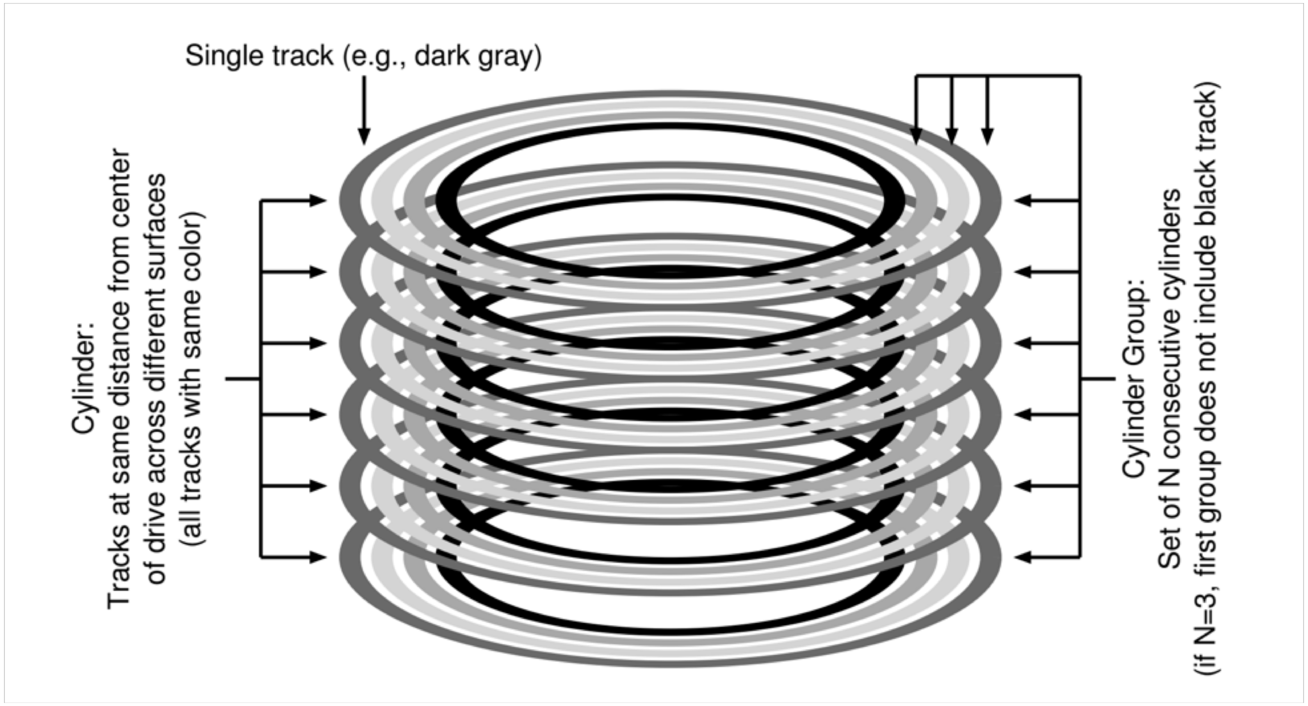
G0	G1	G2	G3	G4	G5	G6	G7	G8	G9

- These groups are uses to improve seek performance.
  - By placing two files within the same group.
  - Accessing one after the other **will not be long seeks** across the disk.
  - FFS needs to allocate files and directories within each of these groups.

Organizing Structure

Cylinder Groups

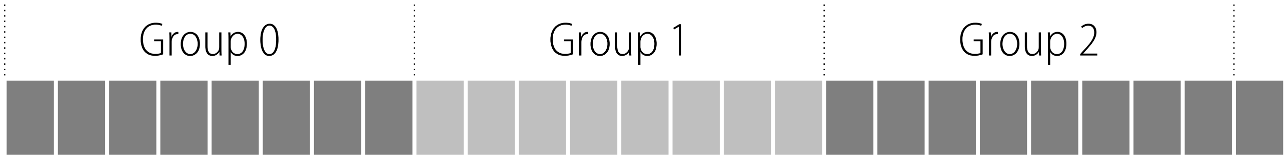
- FFS divides the disk into groups of N consecutive cylinders.


Single track (e.g., dark gray)
Cylinder:
Tracks at same distance from center
of drive across different surfaces
(all tracks with same color)
Cylinder Group:
Set of N consecutive cylinders
(if N=3, first group does not include black track)

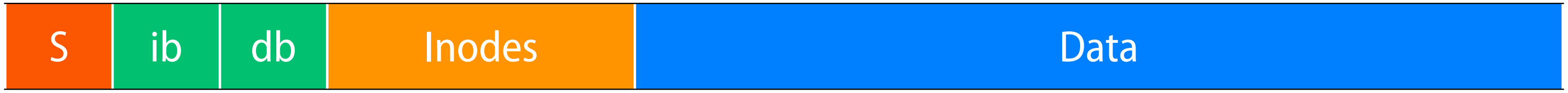
  - Modern drives do not show such details about their geometry.
    - Instead, they export a logical address space of blocks and current file systems organize the drive into groups of consecutive blocks.

Organizing Structure

Block Groups



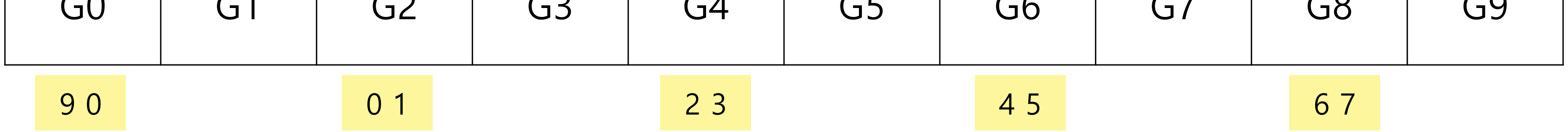
- By placing two files within the same group, FFS ensures that accessing one after the other will not involve long seeks across the disk.
- To be able to place and manage files and directories into a group, FFS includes in it all the structures of a file system.



- Data structures for each block group
  - A copy of the **super block(S)** for reliability reasons
  - **inode bitmap(ib)** and **data bitmap(db)** to track free inodes and data blocks
  - **inodes** and **data blocks** are like those in the very-simple file system (VSFS)

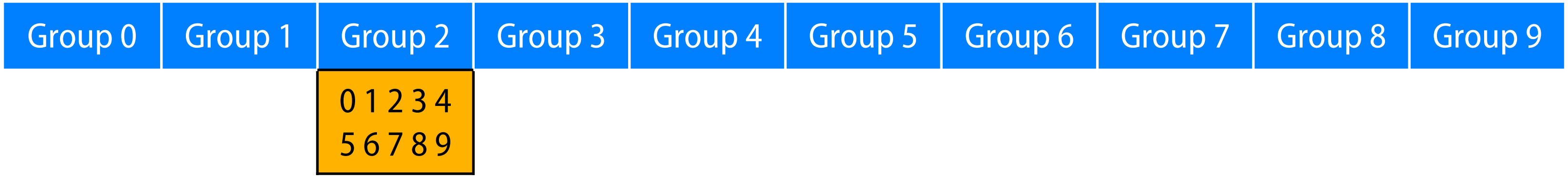
How To Allocate Files and Directories?

- Policy is “**keep related stuff togethe**r”
  - But… what does “related” mean?
- The placement of directories…
  - Find a block group with a low number of allocated directories and a high number of free inodes.
  - Put the directory data and inode in that group.
- The placement of files…
  - Allocate data blocks of a file in the same block group of its inode
  - Place all files in the same block group as their directory

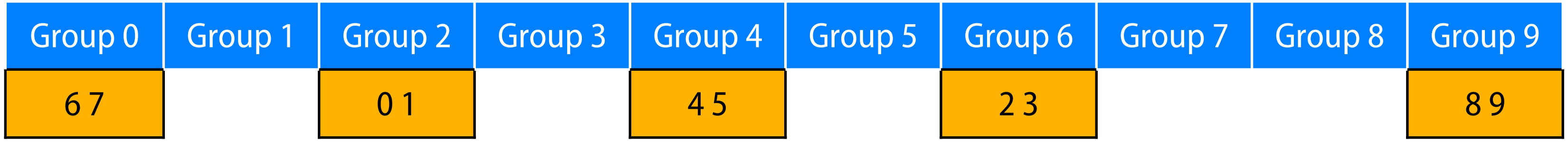


The Large-File Exception

- If the general policy of file placement is applied, a large file might
  - Entirely fill the block group it is first placed within
  - Prevent subsequent "related" files from being placed within this group and, thus, hurt file-access locality

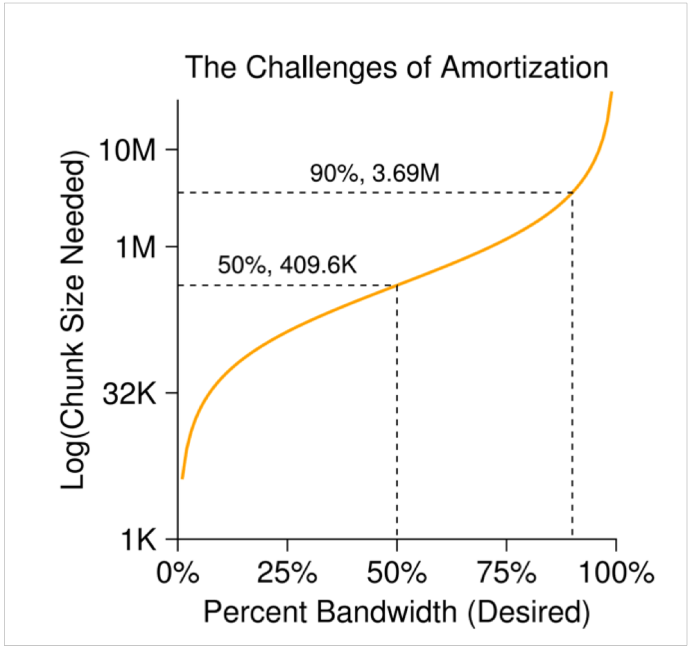


- For large files, chunks are spread across the disk
  - Hurts performance, but this can be addressed by choosing chunk size
  - Amortization: reducing overhead by doing more work



Amortization: How Big Do Chunks Have To Be?

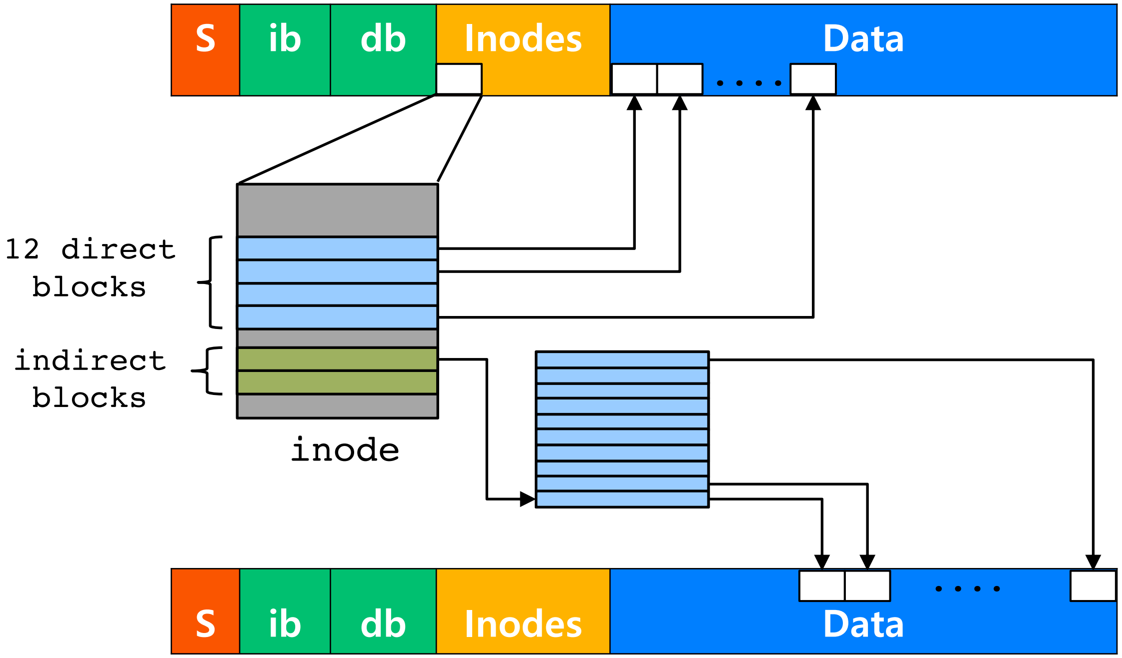
- Estimating chunk size to achieve 50% of peak disk performance
  - half of time seeking and
  - half of time transferring
  - Disk bandwidth: 40 MB/s
  - Positioning time: 10ms



- How much data can this disk transfer in 10ms?
  - $40\frac{MB}{s}\times 0{,}01s = 0{,}4\ MB = 0{,}4\times 1024\ KB = 409{,}6\ KB$

The Large-File Exception in FSS

- For large files, FFS took a simpler approach based on the inode structure
  - The first 12 direct blocks were placed in the same group as the inode
  - Each subsequent indirect block, and all the blocks it pointed to, were placed in a different block group.
    - With 4KB-blocks and 32-bit addresses, every 1024 blocks (4MB) of the file were in separate groups



A few other things about FFS

- Internal fragmentation due to large (at that time!) block size
  - Solution: 512-byte sub-blocks
    - E.g. to create a file with 1 KB, use two sub-blocks, not an entire 4-KB block
- Parameterization
- Track buffers
- Long file names
  - Enabling more expressive names in the file system
- Symbolic links