# File System Implementation

# Unix operating system


Data structures

- The Good Thing
  - Simple and supports the basic abstractions
  - Easy to use file system
- The Problem
  - Terrible performance

# Problem of Unix File System

- Unix file system treated the disk as a random-access memory
  - Example of random-access blocks with four files
    - Data blocks for each file can accessed by going back and forth the disk, because they are are contiguous

| A1 | A2 | B1 | B2 | C1 | C2 | D1 | D2 |
|---|---|---|---|---|---|---|---|

    - File b and d is deleted

| A1 | A2 | | | C1 | C2 | | |
|---|---|---|---|---|---|---|---|

    - File E is created with free blocks (spread across the block)

| A1 | A2 | E1 | E2 | C1 | C2 | E3 | E4 |
|---|---|---|---|---|---|---|---|

- Other Problem is that the original block size was too small (512 bytes)

# FFS: Disk Awareness is Solution

- FFS is Fast File system designed by a group at UC Berkeley
- The design of FFS is that file system structures and allocation polices to be "disk aware" and improve performance
  - Keeps the same APIs with UNIX file system (open(), read(), write(), etc)
  - Changed the internal implementation

# Organizing Structure: 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 Group


- Data structure for each cylinder group
  - A copy of the super block(S) for reliability reason
  - inode bitmap(ib) and data bitmap(db) to track free inode and data block
  - inodes and data block are same to the previous very-simple file system(VSFS)

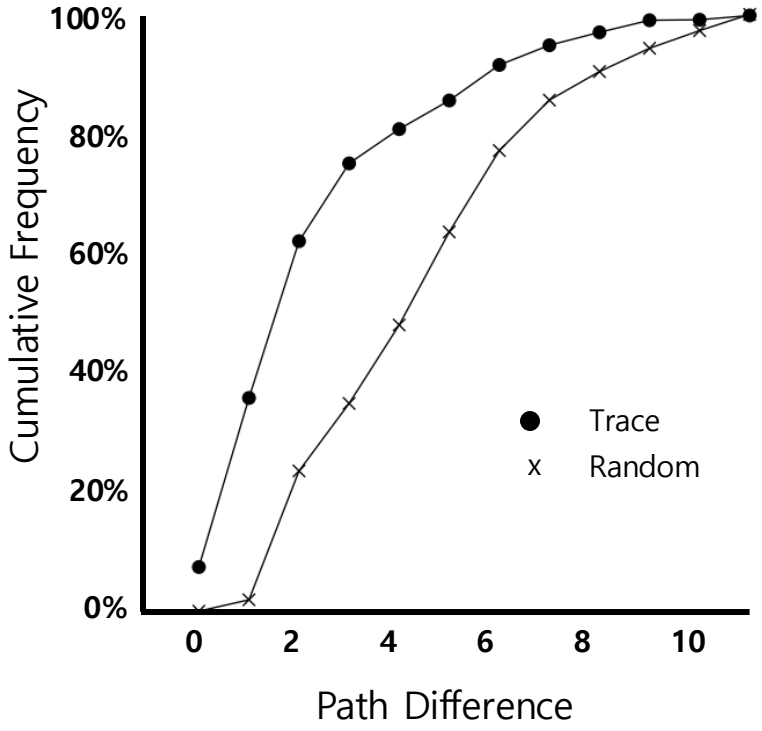
# How To Allocate Files and Directories?

- Policy is "keep related stuff together"
- The placement of directories
  - Find the cylinder 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 group as its inode
  - It places all files in the same group as their directory

# FFS Locality for SEER Traces

- How "**far away**" file accesses were from one another in the directory tree.

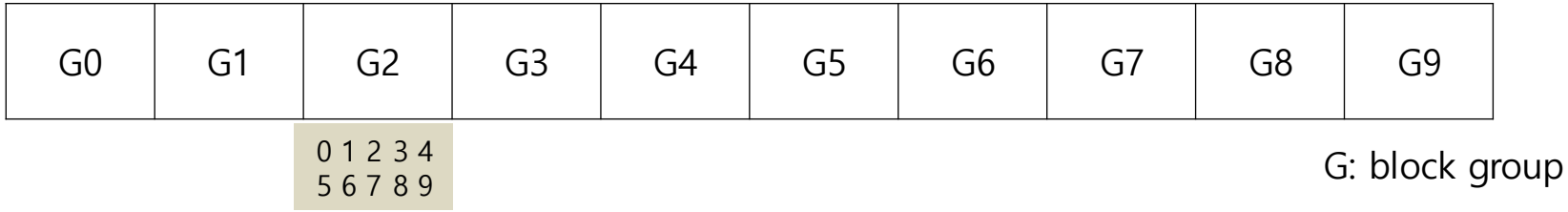
```
proc/src/foo.c
proc/src/bar.c
the distance of two file access is 1

proc/src/foo.c
proc/obj/foo.o
the distance of two file access is 2
```



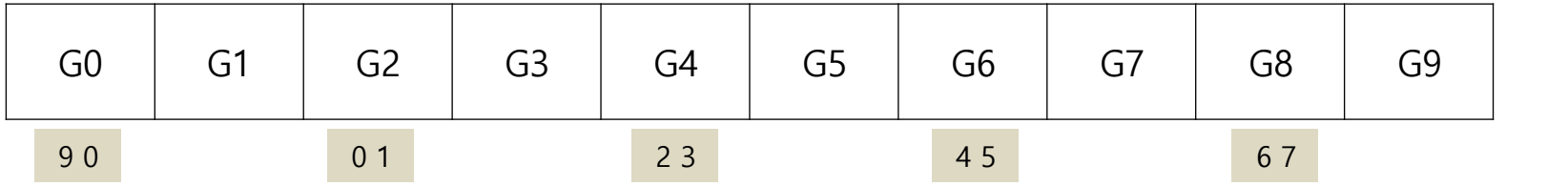

- 7% of file accesses to the same file
- Nearly 40% of file accesses in the same directory
- 25% of file accesses were two distances

# Large-File Exception

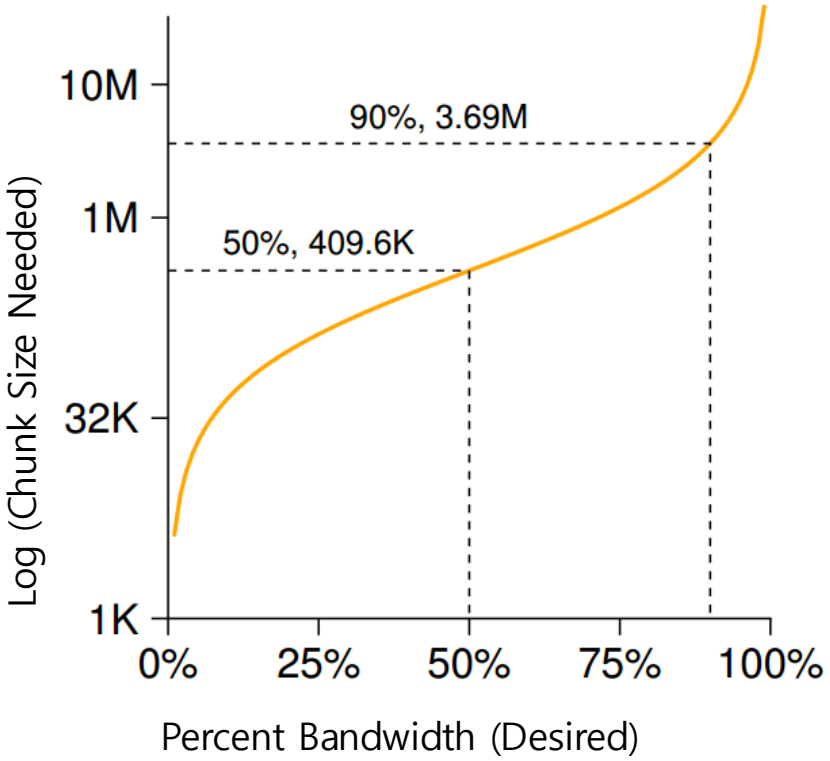
- General policy of file placement
  - Entierly fill the block group it is first place within
  - Hurt file-access locality from "related" file being placed




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



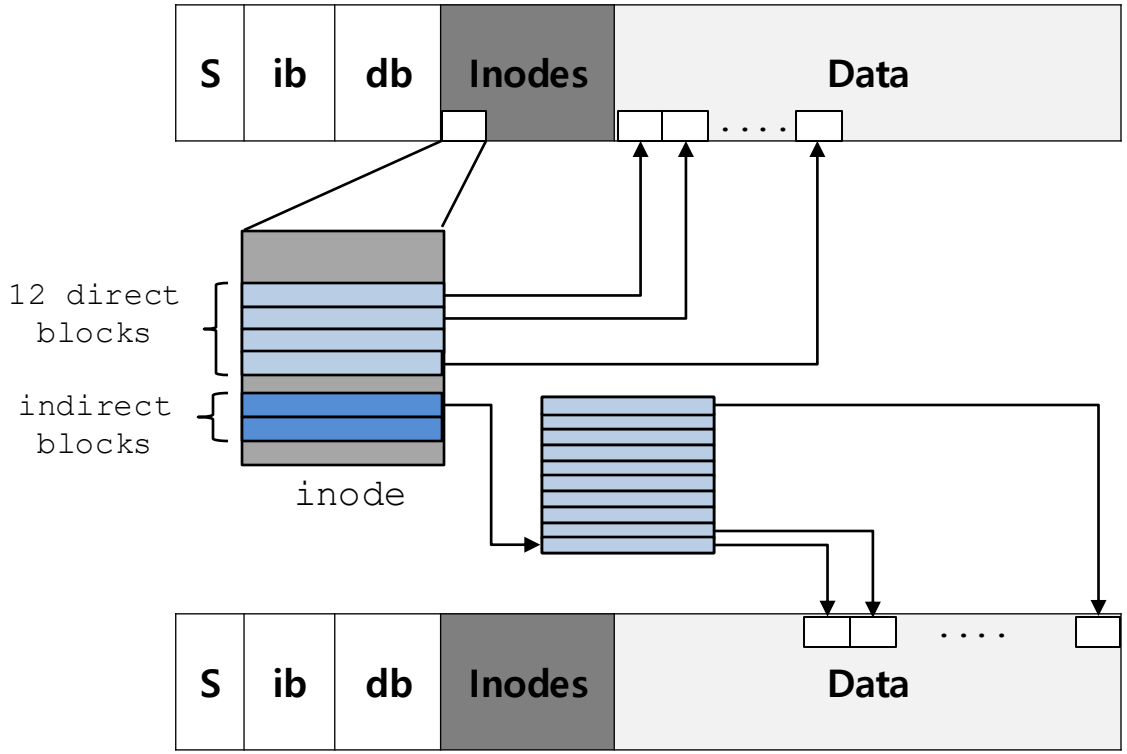
# Amortization: How Big Do Chunks Have To Be?




- Computation of the size of chunk
  - Desire 50% of peak disk performance
    - half of time seeking and half of time transferring
  - Disk bandwidth: 40 MB/s
  - Positioning time: 10ms
  - $\frac{40\ \cancel{MB}}{\cancel{sec}} \cdot \frac{1024\ KB}{1\ \cancel{MB}} \cdot \frac{1\ \cancel{sec}}{1000\ \cancel{ms}} \cdot 10\ \cancel{ms} = 409.6\ KB$
    - Transfer only 409.6 KB every time seeking
  - 90% of peak performance on 3.69MB chunk size

# Large-File Exception in FSS

- A simple approach based on the structure of inode
  - Each subsequent **indirect blocks**, and all the **blocks it pointed to**, placed in **a different block group**
  - Every **1024 blocks (4MB)** of the **file in a separate group**



# A Few Other Things about FFS

- Internal fragmentation
- Sub-blocks
  - Ex) Create a file with 1 KB : use two sub-blocks, not an entire 4-KB blocks
- Parameterization
- Track buffer
- Long file names
  - Enabling more expressive names in the file system
- Symbolic link