Locality and The Fast File System

First File System

- "old UNIX file system" by Ken Thompson

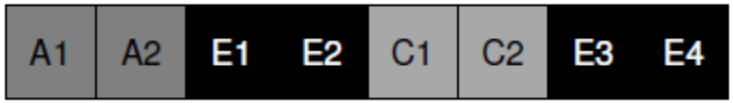
- simple
- supported files and the directory hierarchy

Kirk McKusick

- The problem: performance was terrible.
  - Performance started off bad and got worse over time, to the point where the file system was delivering only 2% of overall disk bandwidth! [Kirk McKusick '84]
- Treated the disk like it was a random-access memory; data was spread and thus had real and expensive positioning costs.
  - For example, the data blocks of a file were often very far away from its inode

First File System

- The file system would end up getting quite fragmented
  - free space was not carefully managed.
  - a logically contiguous file would be accessed by going back and forth across the disk, thus reducing performance dramatically.

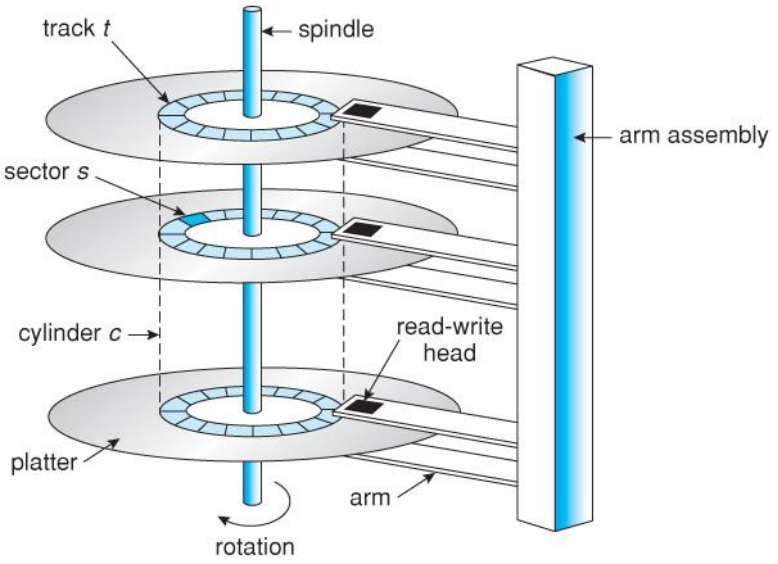


  - defragmentation tools can help
    - reorganize on-disk data to place files contiguously and make free space for one or a few contiguous regions, moving data around and then rewriting inodes and such to reflect the changes.

- One other problem: the original block size was too small (512 bytes).
  - Smaller blocks minimized internal fragmentation (waste within the block),
  - but bad for transfer as each block might require a positioning overhead to reach it.

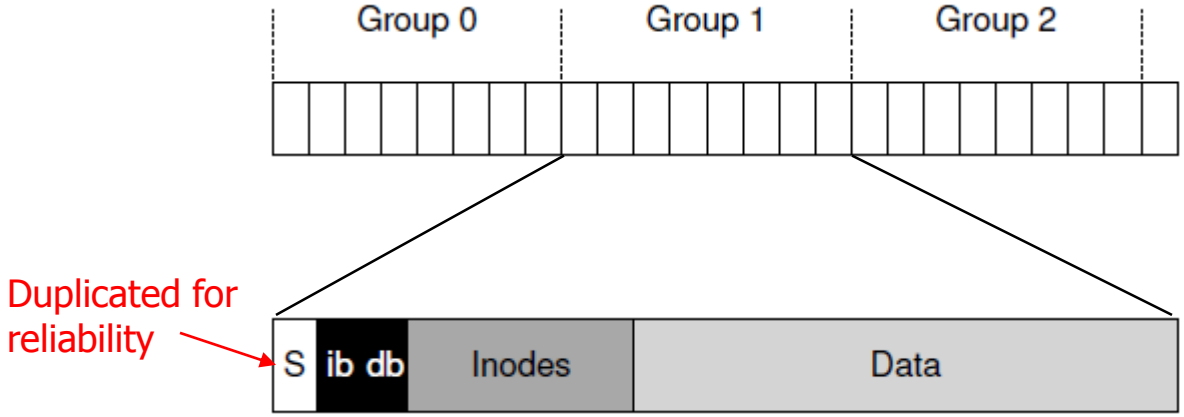
FFS: Disk Awareness Is The Solution

- Fast File System (FFS)
  - Design the file system structures and allocation policies to be "disk aware" and thus improve performance
  - Same APIs, but advanced internal implementation
- FFS divides the disk into a number of cylinder groups.
  - A single cylinder is a set of tracks on different surfaces of a hard drive that are the same distance from the center of the drive
  - aggregates each N consecutive cylinders into group



FFS: Disk Awareness Is The Solution

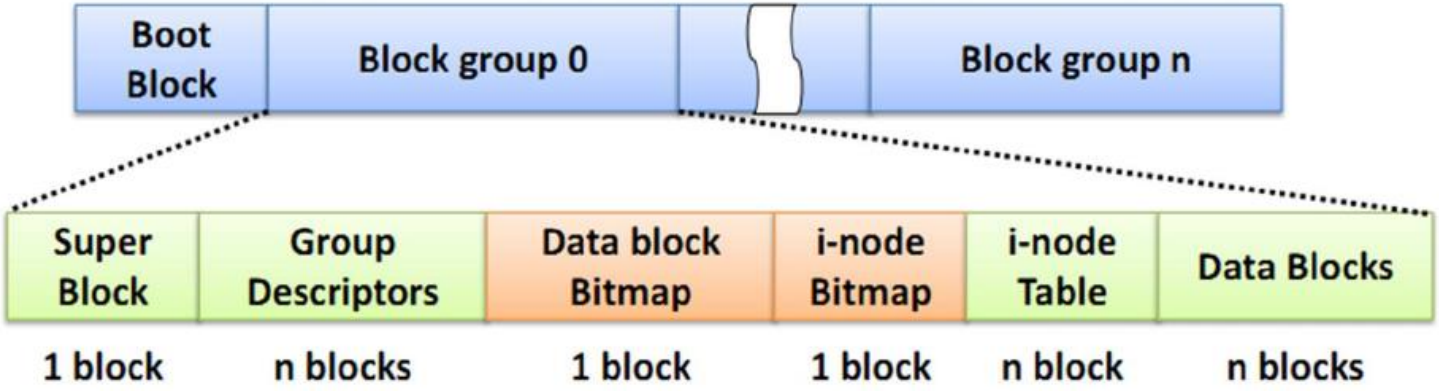
- Modern drives do not export enough information for the file system to truly understand whether a particular cylinder is in use
- Modern file systems (such as Linux ext2, ext3, and ext4) instead organize the drive into block groups
  - A block group is just a consecutive portion of the disk's address space.
- By placing two files within the same group, FFS can ensure that accessing one after the other will not result in long seeks across the disk.



per-group inode bitmap (ib) and data bitmap (db)

Linux File System

- **ext2/ext3/ext4**
  - Block group
    - Kernel tries to keep the data blocks belonging to a file in the same block group to reduce file fragmentation
    - All the block groups have the same size and are stored sequentially



Policies: How To Allocate Files and Directories

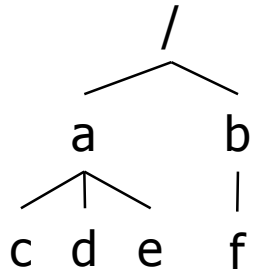
- Keep related stuff together (keep unrelated stuff far apart)!!!
- FFS has to decide what is "related" and place it within the same block group
- Simple placement heuristics
  - Placement of a directory
    - find the cylinder group with
    - a low number of allocated directories (to balance directories across groups)
    - a high number of free inodes (to subsequently be able to allocate a bunch of files)
    - other heuristics could be used here (e.g., a high number of free data blocks)
  - File
    - allocate the data blocks of a file in the same group as its inode
    - places all files that are in the same directory in the cylinder group of the directory they are in.
    - if a user creates four files, /a/b, /a/c, /a/d, and b/f, FFS would try to place the first three in the same group and the fourth in some other group.

Policies: How To Allocate Files and Directories

FFS policies

(1) The data blocks of each file are near each file's inode

(2) The files in the same directory are near one another



directory hierarchy

Are these policies still helpful for SSD-based systems?

```
group inodes     data
    0 /--------- /---------
    1 acde------ accddee---
    2 bf-------- bff-------
    3 ---------- ----------
    4 ---------- ----------
    5 ---------- ----------
    6 ---------- ----------
    7 ---------- ----------
    ...
```

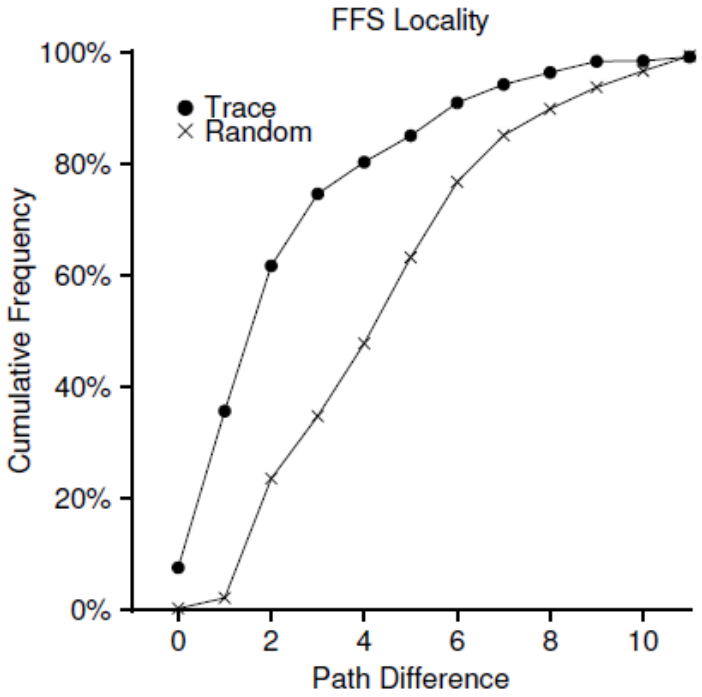
(1) & (2) (namespace-based locality)

```
group inodes     data
    0 /--------- /---------
    1 a--------- a---------
    2 b--------- b---------
    3 c--------- cc--------
    4 d--------- dd--------
    5 e--------- ee--------
    6 f--------- ff--------
    7 ---------- ----------
    ...
```

Only (1)

Measuring File Locality

- Let's see if indeed there is namespace locality
- Path difference
  - how "far away" file accesses were from one another in the directory tree
  - Same file accesses: 1
  - Different file in the same directory: 2
- SEER traces
  - about 7% of file accesses were to the file that was opened previously
  - nearly 40% of file accesses were to either the same file or to one in the same directory
  - 25% of file accesses were to files that had a distance of two.
    - a set of related directories and consistently jumps between them (e.g., /proj/src/foo.c and proc/obj/foo.o)
    - FFS does not capture this type of locality



The Large-File Exception

- Problem of FFS policies
  - If a large file would entirely fill the block group it is first placed within, it prevents subsequent "related" files from being placed within this block group, and thus may hurt file-access locality.

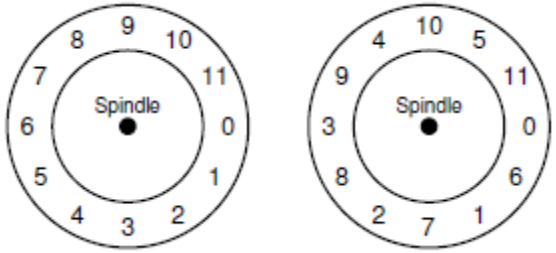
```
group inodes     data
    0 /a-------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa a---------
    1 ---------- ---------- ---------- ---------- ----------
    2 ---------- ---------- ---------- ---------- ----------
    ...
```

- Large-File Exception
  - The first twelve direct blocks are allocated into the first block group (48KB)
  - The next "large" chunk of the file in another block group.
    - Each subsequent indirect block, and all the blocks it pointed to. (1024 blocks = 4 MB)
  - If the chunk size is large enough, little seeking overhead

```
group inodes     data
    0 /a-------- /aaaaa----- ---------- ---------- ----------
    1 ---------- aaaaa------ ---------- ---------- ----------
    2 ---------- aaaaa------ ---------- ---------- ----------
    3 ---------- aaaaa------ ---------- ---------- ----------
    4 ---------- aaaaa------ ---------- ---------- ----------
    5 ---------- aaaaa------ ---------- ---------- ----------
    6 ---------- ---------- ---------- ---------- ----------
    ...
```

A Few Other Things About FFS

- Small files < 4KB
  - Internal fragmentation
  - sub-block allocation (512B)
  - if the file grows and requires a full 4KB of data, copy the sub-blocks into a 4KB block, and free the sub-blocks for future use.
  - To avoid a lot of extra I/O to perform the copy, the libc library would buffer writes and then issue them in 4KB chunks to the file system.
- Parameterized Placement
  - By skipping over every some other blocks, FFS has enough time to request the next block before it went past the disk head.
  - Modern disks read the entire track in and buffer it in an internal disk cache (track buffer)
- Made the system more usable
  - Long file name (only 8 characters in the traditional fixed-size approach)
  - Symbolic link
  - Atomic rename()
- Many recent file systems take cues from FFS



Standard vs. Parameterized Placement

Homework

- Homework in Chap 41 (FFS)