FFS: Fast File System

Williams College
CS333

This Video

- FFS: Fast file system
  - General FFS structure
  - Overview of FFS goals
  - Explanation of the FFS allocation heuristics and their implications on performance

Recall: Key **Ref. FS** Data structures

- Inode
  - Persistent information about a single file
- Superblock
  - Persistent information about entire file system
- Allocation structures
  - Inode bitmap, data bitmap

Today: Key **FFS** Data structures

- Inode
  - Persistent information about a single file
- Superblock
  - Persistent information about entire file system
- Allocation structures
  - Inode bitmap, data bitmap

FFS set the stage for FS design

The FFS Designers:

- Thought hard about HDD performance
- Abstracted common file system structures & methods
- Identified performance bottlenecks
- Implemented "Common sense" heuristics
  - Use *device awareness* to improve performance
    - Downsides to device awareness?

Problem 1: Dependent Reads

- To read file data, must first read the inode

Core issue: data and metadata separation can cause long seeks.

Problem 2: Small Block Size

- How does **`grep`** work?
  - $ **`grep -r “pattern” .`**
- We want our file system’s I/O to be purely sequential
  - Or at least, we want to maximize the amount of time spent transferring data relative to time wasted doing I/O setup (seeking+rotating)
- Because FS blocks are small, we can’t afford to seek for each LBA

Problem 3: Free Space Fragmentation

- In order to allocate an object, we need to find a region of free space to hold it.

- Can't we just use first fit LBA allocation?

    - Allocating blocks one at a time leads to significant fragmentation!
        - Blocks within a single file may be scattered
        - Related files may be far away from each other

Ideas

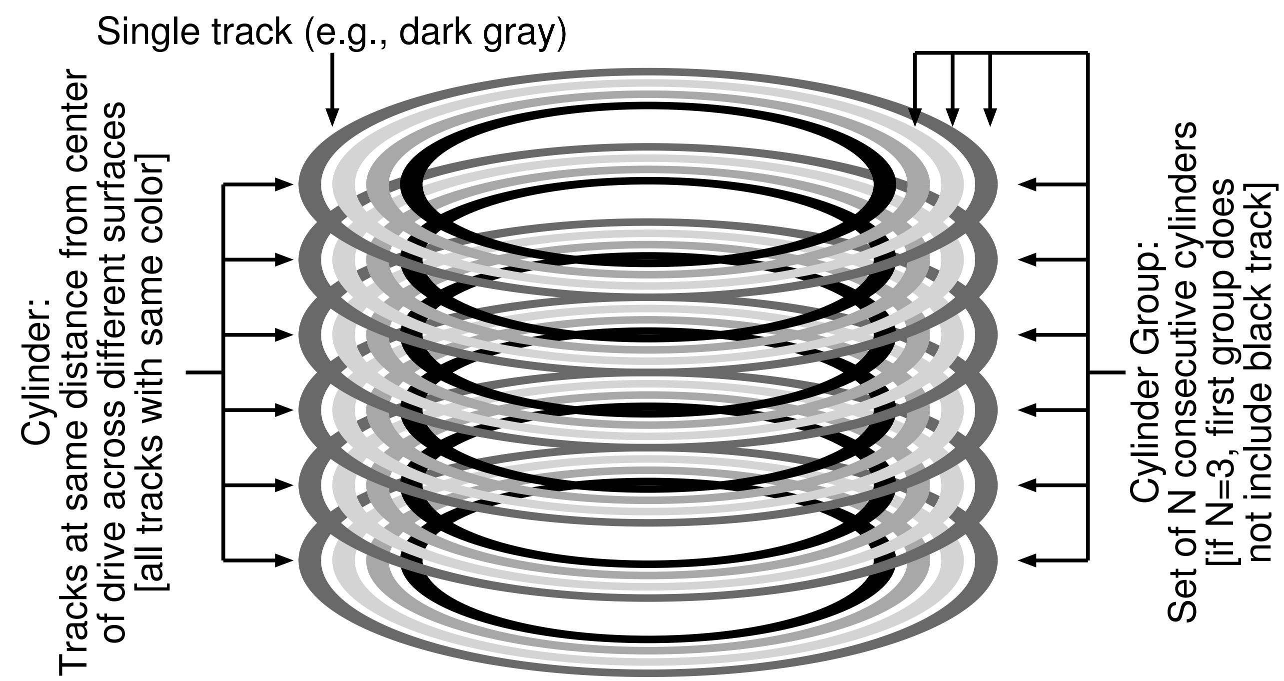
Key idea: Keep related things together

FFS Techniques:

- Cylinder groups (block groups)
- Set of coordinated allocation heuristics for:
  - Directory allocation
  - File Allocation
  - File block allocation

Cylinder Groups

- Disks don't just have one platter: they have an array of platters and disk arms that move as a unit.
- A cylinder group is a vertical slice through the platters, where each slice consists of contiguous tracks
  - A "narrow seek window"
- Since neighboring LBAs should correspond to neighboring tracks in a cylinder group, we can abstract this idea into a block group: a contiguous region of the LBA space

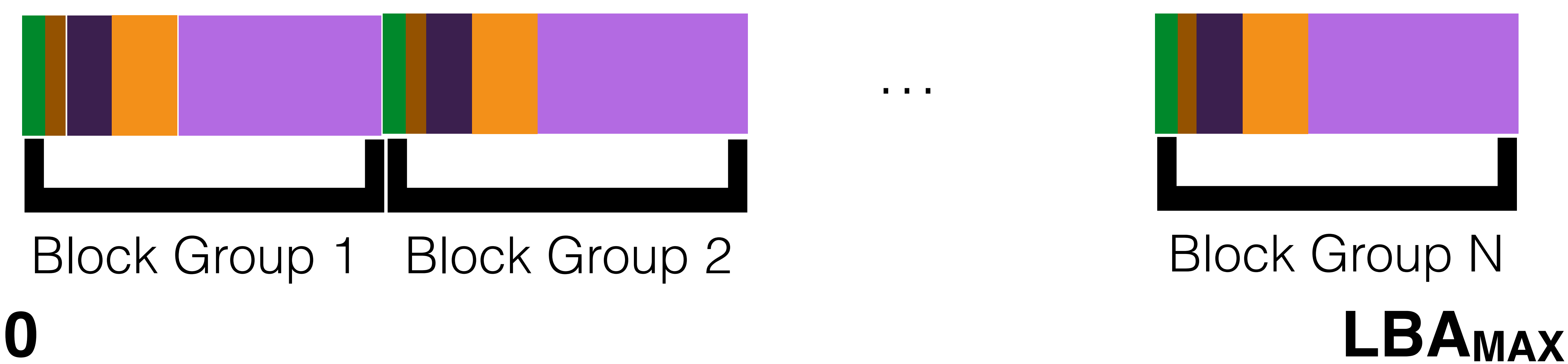


[OSTEP ch 41]

Block Groups to Improve Locality

- Recall the on-disk format of our reference file system design:

- FFS replicates this format in each block group:



Block Groups to Improve Locality

- Superblock is replicated in each block group
  - SB is very important: this is simply done for redundancy
- Inode table is "distributed" across block groups
  - Goal is to keep inodes close to the data they describe
  - Only one copy of each inode is stored
  - Fixed number of inodes stored per block group
- When possible, allocate a file's first blocks in the same block group that its inode is stored
  - Minimizes the distance to seek when reading the inode and reading the beginning of the file (very common op)

Coordinated Allocation Heuristics

Goal: store related things near each other

- What things are “related”?
  - A single file’s blocks
    - Why? We often read files in their entirety
  - Files in the same directory (siblings)
    - Why? Directory hierarchy expresses relationships

Goal: Store Related Things Near Each Other

When allocating files in the same directory (siblings):

- Strategy 1: Allocate a *directory's* inode in the B.G. with the most free inodes
  - Hopefully, this leaves room for its future children
- Strategy 2: Allocate a *file's* inode in the same B.G. as its parent directory's inode

Goal: Store Related Things Near Each Other

When Allocating a single file's blocks

- Strategy: Allocate "chunks" of data together
  - Put all direct-pointed-to blocks in inode's B.G.
  - Allocate successive "chunks" together
    - All blocks a single indirect block points to

➡ *For each dependent read, we get a chunk of data from the seek*

- Why not put all of a file's blocks in the same B.G.?
  - Large files "crowd out" other files in B.G.
  - Don't necessarily know how large a file will ultimately grow
  - This is called the "large file exception"—for large files, a single seek still finds a significant amount of data

Allocation Examples (simplified)

- Three directories:
  - `/`
  - `/a`
  - `/b`
- Four files:
  - `/a/c`
  - `/a/d`
  - `/a/e`
  - `/b/f`
- Suppose they were allocated and written in the order above. Let's apply the rules

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

[OSTEP ch 41]

Problems with FFS?

- All rules are heuristics… there are no guarantees!
- Once you make a decision, you're stuck with it
- Aging: file system performance degradation over time
  - What operations/workloads might cause problems over time?
  - What is defragmentation?
    - Relocating blocks on disk to restore locality

Recap

- FFS is an influential File System Design—it is the canonical "update-in-place" file system
    - Once we place a block, updates to that block are made "in-place" by overwriting the contents
- Created commonsense rules to improve performance *based on observations about application behaviors*
- Despite limitations, FFS ideas still used in practice today
    - ext4 builds on key components of FFS design