

# Lecture 17: Unix Fast Filesystem

601.418/618 Operating Systems

David Hovemeyer

April 14, 2025

# Agenda

- ▶ Unix FS performance problems, FFS solutions
  - ▶ Small block size
  - ▶ Free space management
  - ▶ Locality

Acknowledgments: These slides are shamelessly adapted from [Prof. Ryan Huang's Fall 2022 slides](#), which in turn are based on [Prof. David Mazières's OS lecture notes](#).

# File Systems Examples

## BSD Fast File System (FFS)

- ▶ What were the problems with the original Unix FS?
- ▶ How did FFS solve these problems?

## Log-Structured File system (LFS) – next lecture

- ▶ What was the motivation of LFS?
- ▶ How did LFS work?

# Original Unix FS

From Bell Labs by Ken Thompson

Simple and elegant:

Unix disk layout



## Components

- ▶ Data blocks
- ▶ Inodes (directories represented as files)
- ▶ Free list
- ▶ Superblock. (specifies number of blks in FS, counts of max # of files, pointer to head of free list)

Problem: slow

- ▶ Only gets 2% of disk maximum (20Kb/sec) even for sequential disk transfers!

# Why So Slow?

Problem 1: blocks too small (512 bytes)

- ▶ File index too large
- ▶ Require more indirect blocks
- ▶ Transfer rate low (get one block at time)

Problem 2: unorganized freelist

- ▶ Consecutive file blocks not close together
  - ▶ Pay seek cost for even sequential access
- ▶ Aging: becomes fragmented over time

Problem 3: poor locality

- ▶ inodes far from data blocks
- ▶ inodes for directory not close together
  - ▶ poor enumeration performance: e.g., "ls", "grep foo \*.c"

# FFS: Fast File System

Designed by a Berkeley research group for the BSD UNIX

- ▶ A classic file systems paper to read: [McKusick et. al.](#)<sup>1</sup>

Approach:

- ▶ **measure** a state of the art systems
- ▶ identify and understand the fundamental problems
  - ▶ **The original FS treats disks like random-access memory!**
- ▶ get an idea and **build** a better systems

Idea: design FS structures and allocation polices to be “disk aware”

Next: how FFS fixes the performance problems (to a degree)

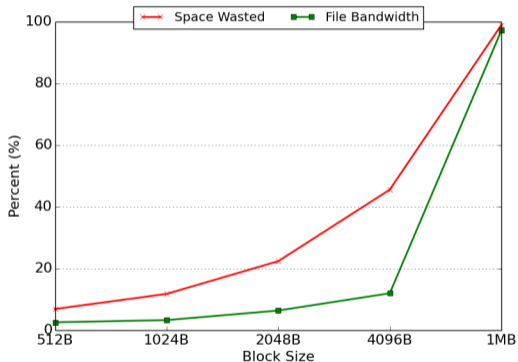
---

<sup>1</sup>Marshall Kirk McKusick, William N. Joy, Samuel J. Leffler, Robert S. Fabry, *A Fast File System for UNIX*

## Problem 1: Blocks Too Small

Bigger block increases bandwidth, but how to deal with wastage (“**internal fragmentation**”)?

Measurement:

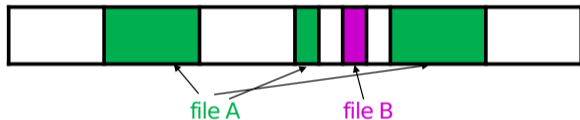


- ▶ Use idea from malloc: split unused portion

## Solution: Fragments

BSD FFS:

- ▶ Has large block size (4096B or 8192B)
- ▶ Allow large blocks to be chopped into small ones called “fragments”
- ▶ Ensure fragments only used for little files or ends of files



- ▶ Fragment size specified at the time that the file system is created
- ▶ Limit number of fragments per block to 2, 4, or 8

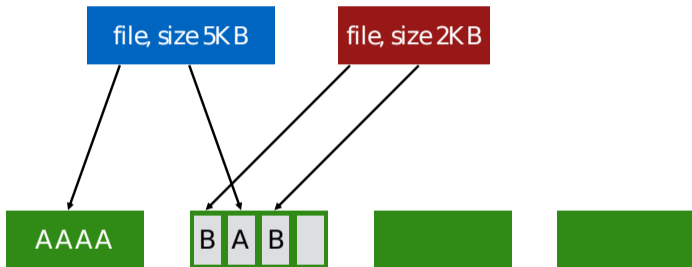
Pros

- ▶ High transfer speed for larger files
- ▶ Low wasted space for small files or ends of files



## Fragment Example

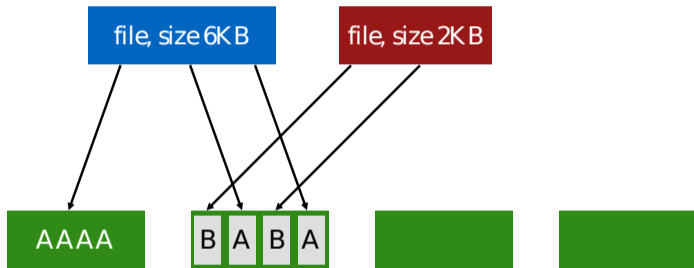
Block size: 4096 B  
Fragment size: 1024 B



# Fragment Example

```
write(fd1, "A"); // append A to first file
```

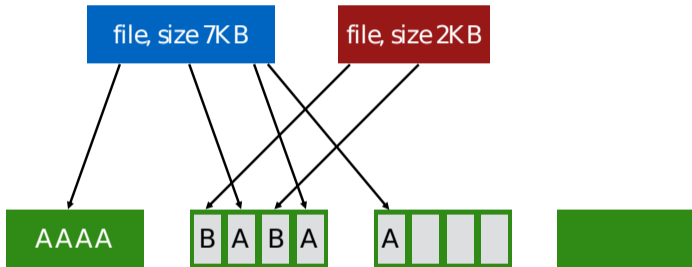
Block size: 4096 B  
Fragment size: 1024 B



# Fragment Example

```
write(fd1, "A"); // append A to first file  
write(fd1, "A");
```

Block size: 4096 B  
Fragment size: 1024 B



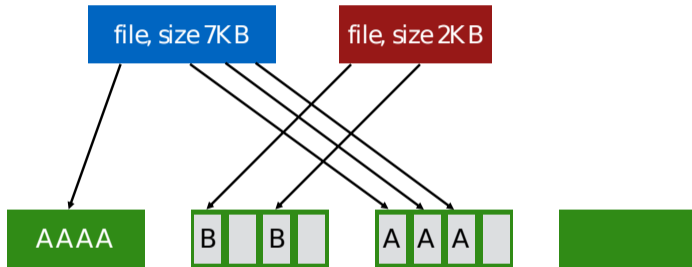
Not allowed to use fragments across multiple blocks!

What to do instead?

# Fragment Example

```
write(fd1, "A"); // append A to first file  
write(fd1, "A");
```

Block size: 4096 B  
Fragment size: 1024 B

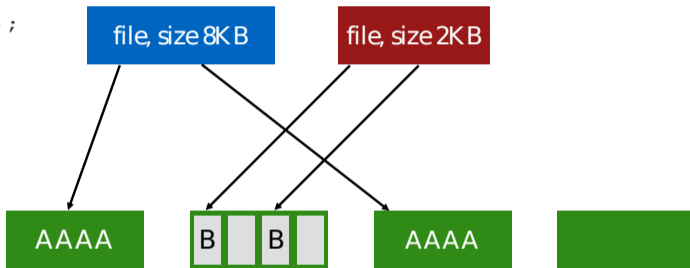


copy old fragments to new block  
new data use remaining fragments

# Fragment Example

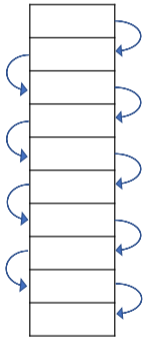
```
write(fd1, "A"); // append A to first file  
write(fd1, "A");  
write(fd1, "A");
```

Block size: 4096 B  
Fragment size: 1024 B

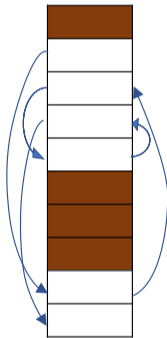


## Problem 2: Unorganized Freelist

Leads to random allocation of sequential file blocks overtime



Initial performance good



Get worse over time

### Measurement:

- New FS: **17.5%** of disk bandwidth
- Few weeks old: **3%** of disk bandwidth

# Fixing the Unorganized Freelist

Periodical compact/defragment disk

- ▶ Cons: locks up disk bandwidth during operation

Keep adjacent free blocks together on freelist

- ▶ Cons: costly to maintain

FFS: bitmap of free fragments

- ▶ Each bit indicates whether fragment is free
  - ▶ E.g., 1010101111111000001111111000101100
- ▶ Easier to find contiguous blocks  
(all aligned fragments 0 → block is free)
- ▶ Small, so usually keep entire thing in memory
- ▶ Time to find free blocks increases if fewer free blocks

Bits in map	XXXX	XXOO	OOXX	OOOO
Fragment numbers	0-3	4-7	8-11	12-15
Block numbers	0	1	2	3

# Using a Bitmap

Usually keep entire bitmap in memory:

- ▶ 4G disk / 4K byte blocks. How big is map?

Allocate block close to block  $x$ ?

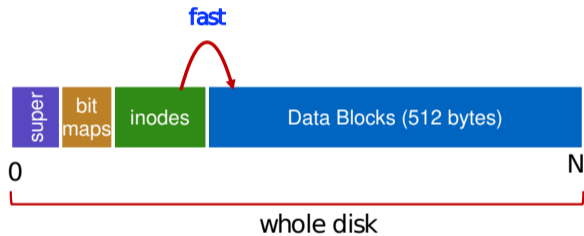
- ▶ Check for blocks near  $\text{bmap}[x/32]$
- ▶ If disk almost empty, will likely find one near
- ▶ As disk becomes full, search becomes more expensive and less effective

Trade space for time (search time, file access time)



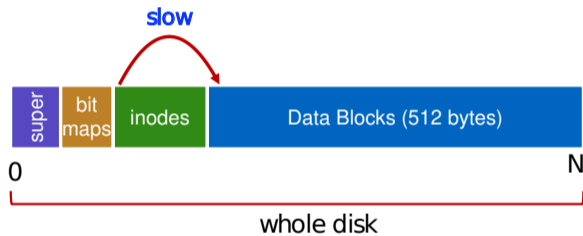


## Problem 3: Poor Locality



How to keep inode close to data block?

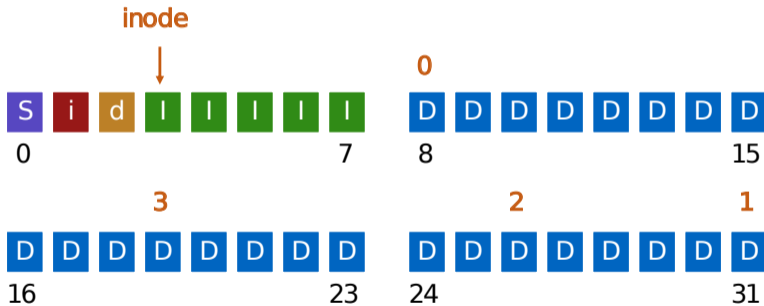
## Problem 3: Poor Locality



How to keep inode close to data block?

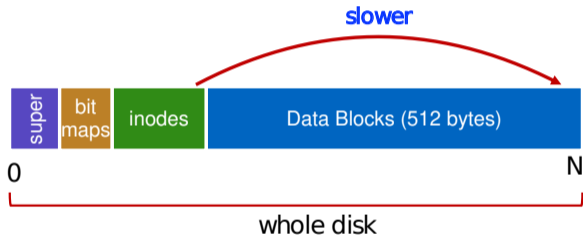
## Problem 3: Poor Locality

Example bad layout:



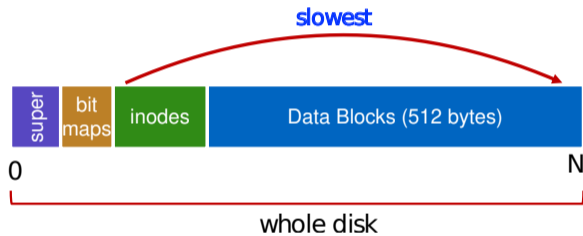
How to keep inode close to data block?

## Problem 3: Poor Locality



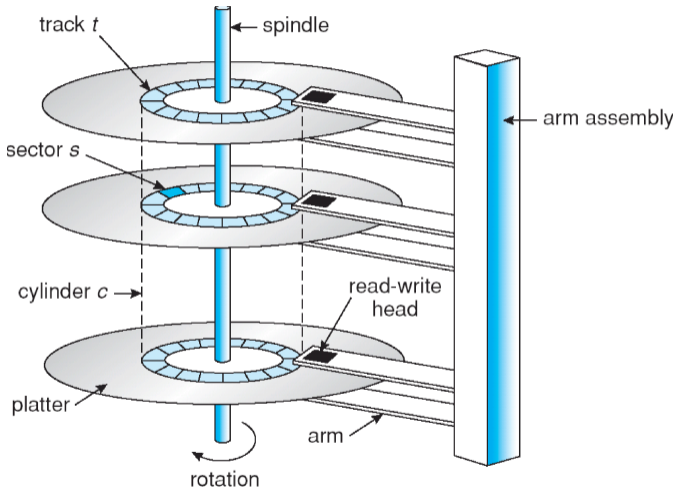
How to keep inode close to data block?

## Problem 3: Poor Locality



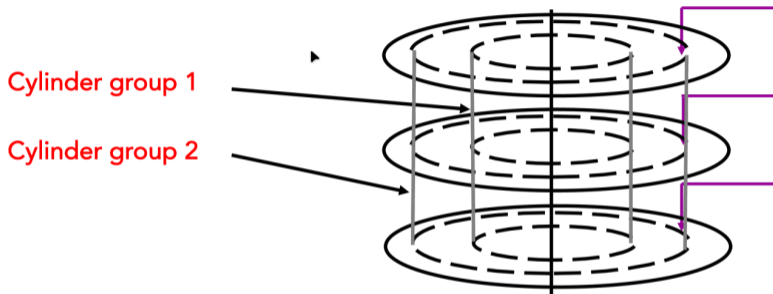
How to keep inode close to data block?

# Cylinders, Tracks, & Sectors



## FFS Solution: Cylinder Group

Group sets of consecutive cylinders into “cylinder groups”



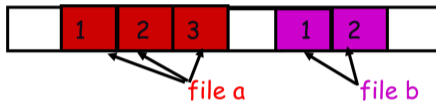
Key: can access any block in a cylinder without performing a seek. Next fastest place is **adjacent cylinder**.

- ▶ Tries to put everything related in same cylinder group
- ▶ Tries to put everything not related in different group

## Clustering in FFS

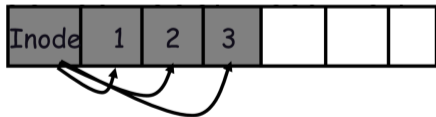
Tries to put sequential blocks in adjacent sectors

- ▶ (Access one block, probably access next)



Tries to keep inode in same cylinder as file data:

- ▶ (If you look at inode, most likely will look at data too)

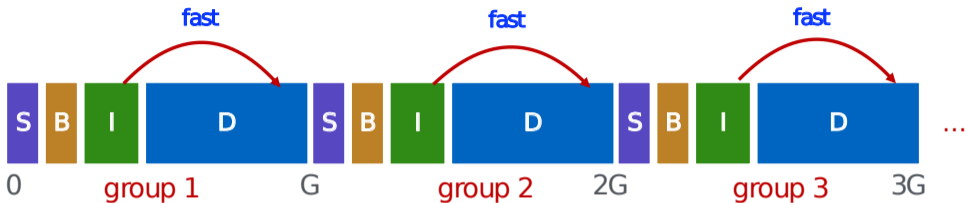


Tries to keep all inodes in a dir in same cylinder group

- ▶ Access one name, frequently access many, e.g., "ls -l"



## What Does Disk Layout Look Like Now?



How to keep inode close to data block?

- ▶ Answer: Use groups across disks
- ▶ Strategy: allocate inodes and data blocks in same group
- ▶ Each cylinder group basically a mini-Unix file system

Is it useful to have multiple super blocks?

- ▶ Yes, if some (but not all) fail

# FFS Results

## Performance improvements:

- ▶ Able to get 20-40% of disk bandwidth for large files
- ▶ 10-20x original Unix file system!
- ▶ Stable over FS lifetime
- ▶ Better small file performance (why?)

## Other enhancements

- ▶ Long file names
- ▶ Parameterization
- ▶ Free space reserve (10%) that only admin can allocate blocks from

## Next Time

Log Structured Filesystem (LFS)