Sistemas Operativos: File Systems

Fast File System (FFS) - 80's

May 28, 2020

File System Implementation: Original Design (Ken Thompson)

- This is essentially the design we presented:

  - Free lists were **embedded** in inodes and data blocks
  - Block size was 512 bytes

Advantage simplicity

- Most previous file systems were record based

Issue Performance:

- On a new file system it was 17.5% of disk bandwidth (the upper bound)
- On FS a few weeks old it was 3% of disk bandwidth.

File Fragmentation

File fragmentation over time, file blocks become scattered over the disk. Example:

A1	A2	B1	B2	C1	C2	D1	D2

After deletion of D:

A1	A2	B1	B2	C1	C2		

After deletion of B:

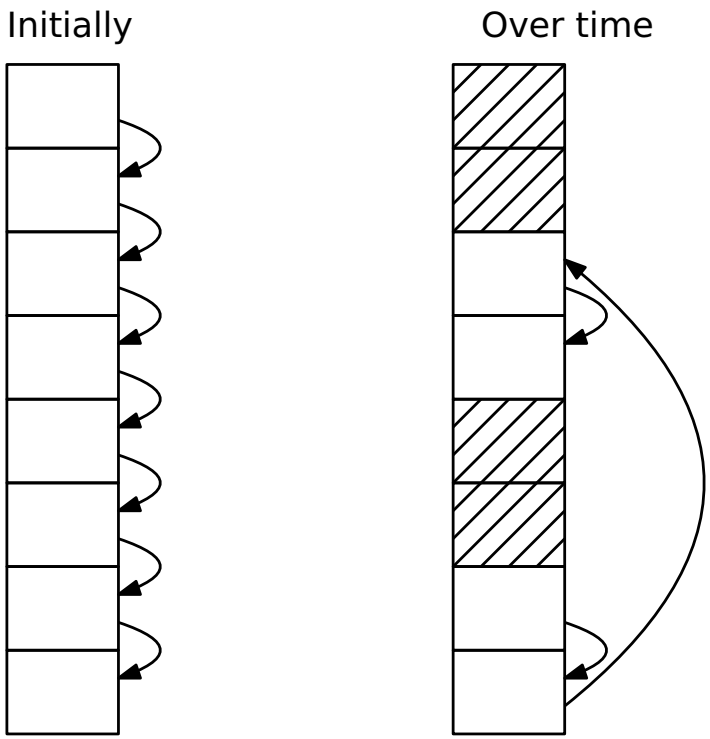
A1	A2			C1	C2		

and the creation of a 4-block file E:

A1	A2	E3	E4	C1	C2	E1	E2

File Fragmentation: Root Cause

Use of a list to keep track of free data blocks



Hacky workarounds Ocasionally

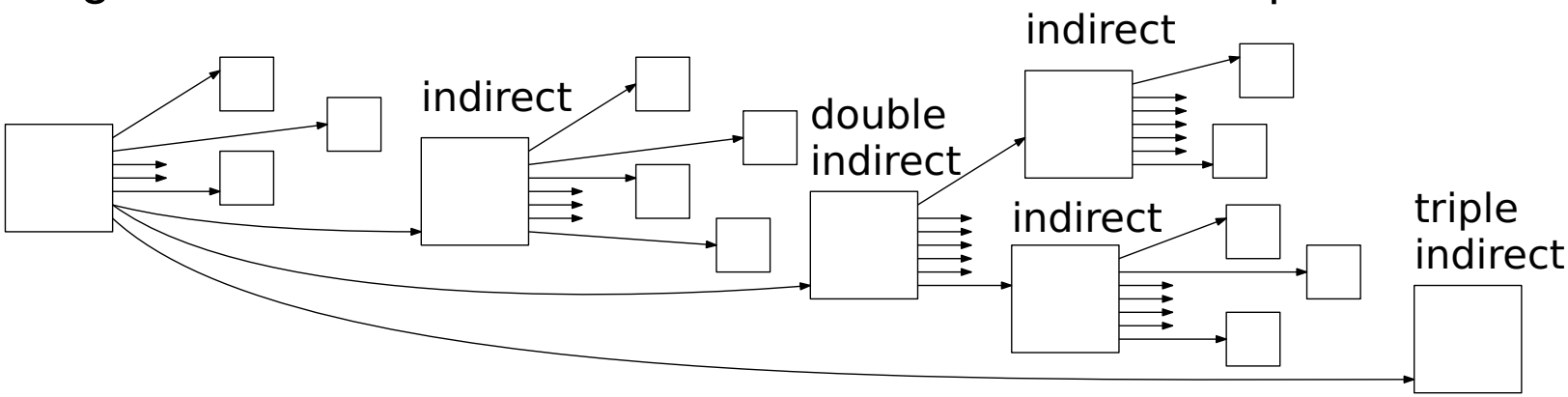
- Move data blocks around to compact files (**defragmentation**)
- Sort the free list

Fix use a bit map instead

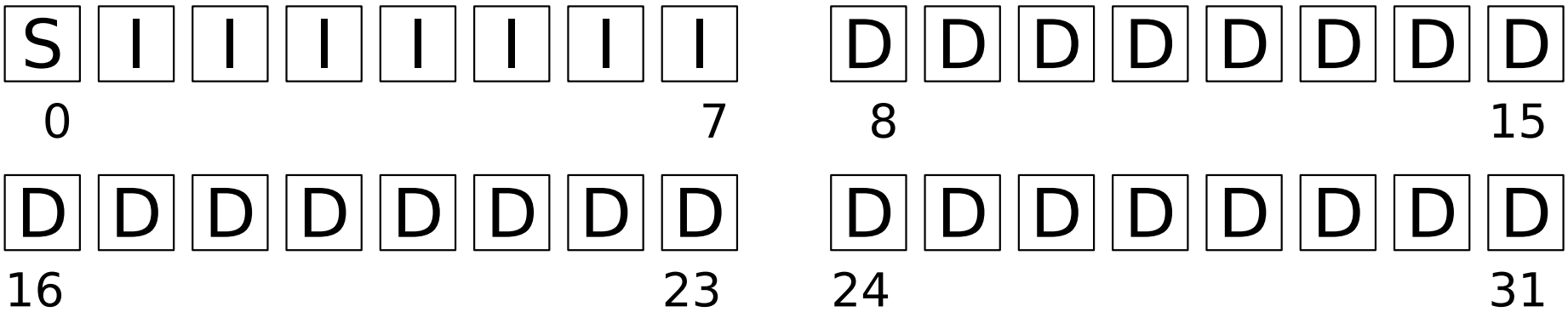
- Makes it easier to find consecutive free data blocks

Too Small Block Size

- The original block size was too small: 512 byte
- Just doubling its value more than doubled the speed:
  1. Each disk access allows to transfer double the data
     - Most blocks were scattered over the disk
  2. Larger blocks reduce the number of indirect blocks required



File System Layout



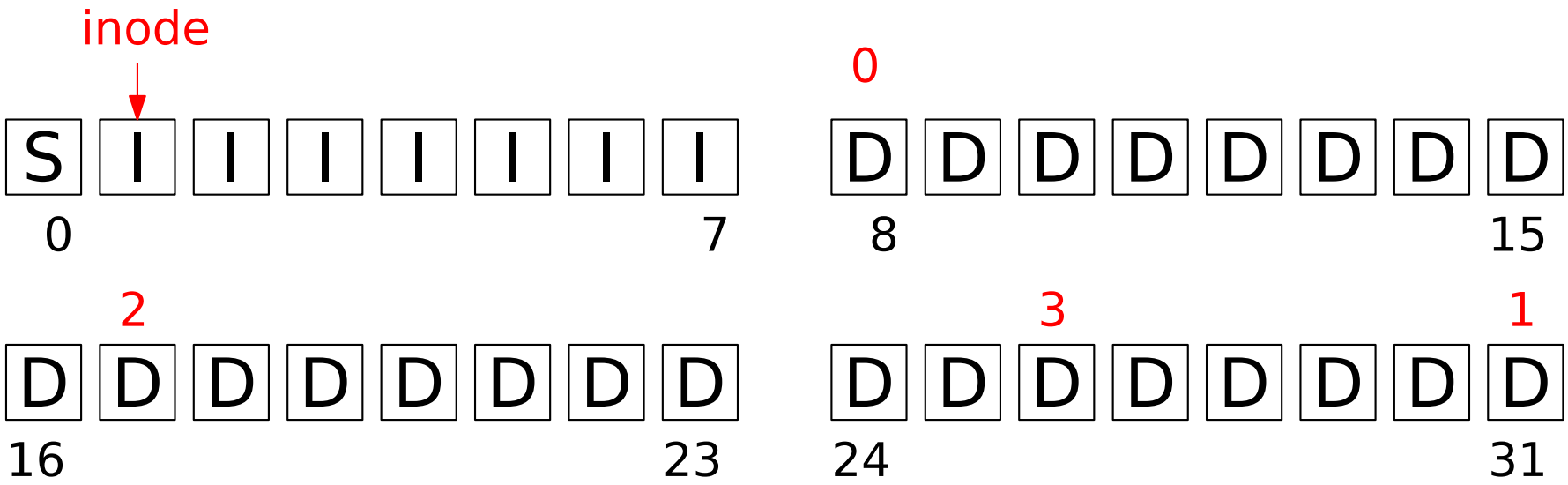
Issue blocks are laid out poorly

1. long distance between inodes and data
2. related inodes are not close to one another

Policy: Which inode and data blocks? (1/3)

Assumption empty filesystem

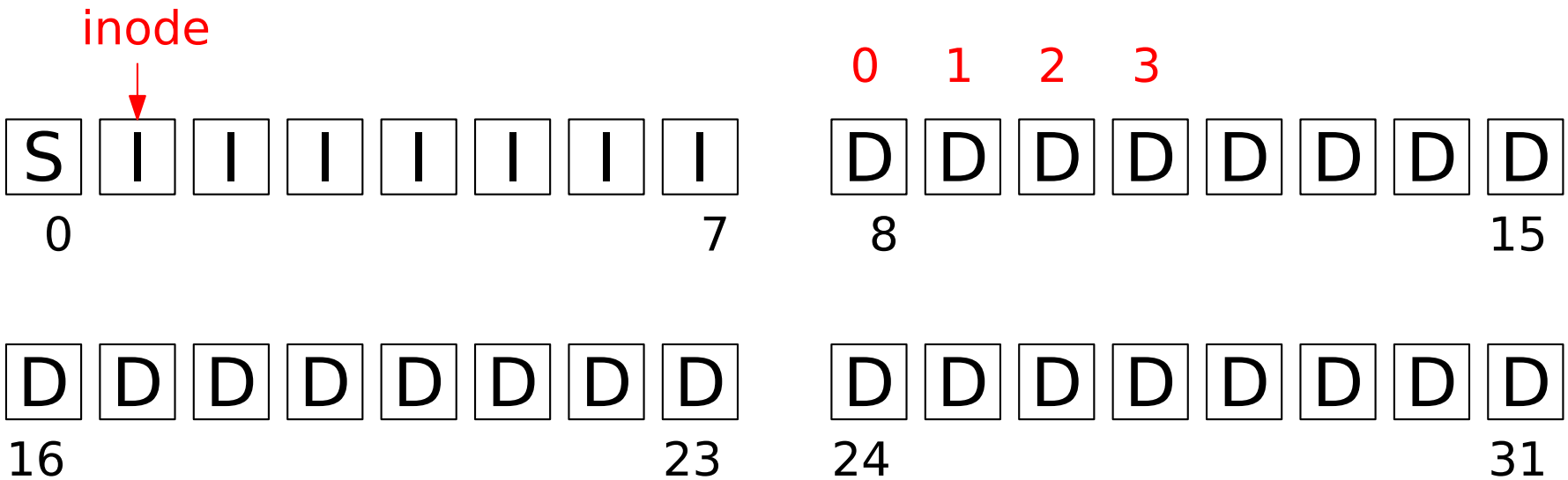
Bad file layout:



Policy: Which inode and data blocks? (2/3)

Assumption empty filesystem

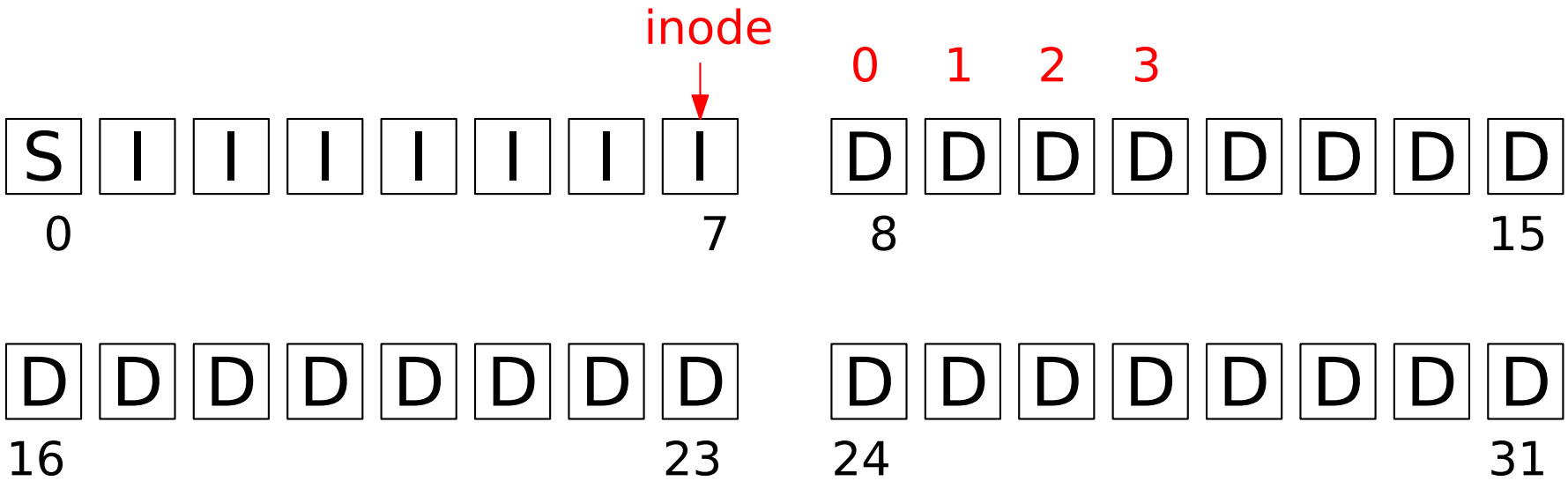
Better file layout:



Policy: Which inode and data blocks? (3/3)

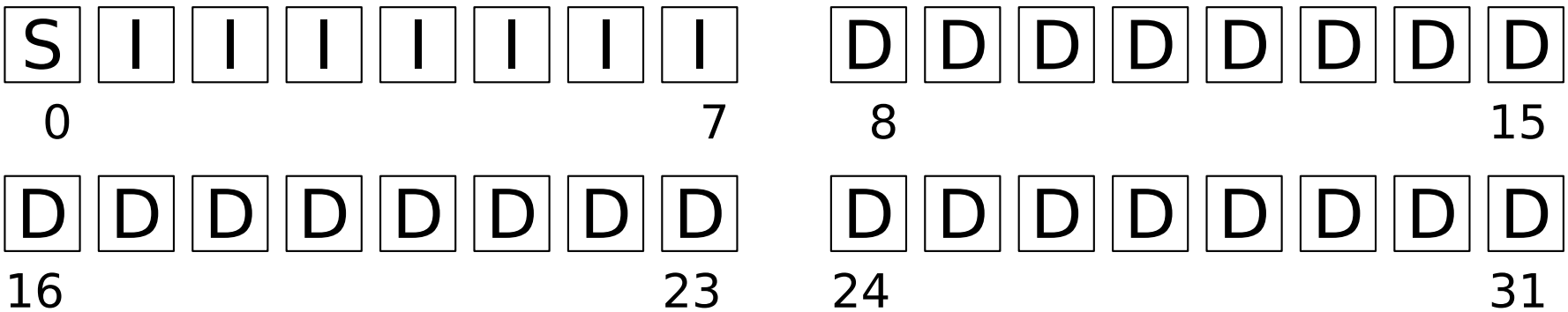
Assumption empty filesystem

Best file layout:



But cannot use it for all files :(

Policy: Is Inode Layout Important?



What does the FS do for

- ls
- ls -l

Conclusion Inodes in same diretory should be near one another.

Original File System

Free list becomes scrambled over time

- Simple allocation policy (first available) leads to random allocations

Small blocks 512 bytes

Blocks laid out poorly

- inodes and respective data blocks may be far away
- related inodes may be far away

Result **2%** of potential performance (or worse) over time

Problem Original FS treats disk like RAM

Solution: Fast File System

Use bitmaps instead of free-lists

- Easier to find contiguous free blocks

Use a disk-aware layout

Where to place meta-data and data on disk?
How to use big blocks without wasting space?

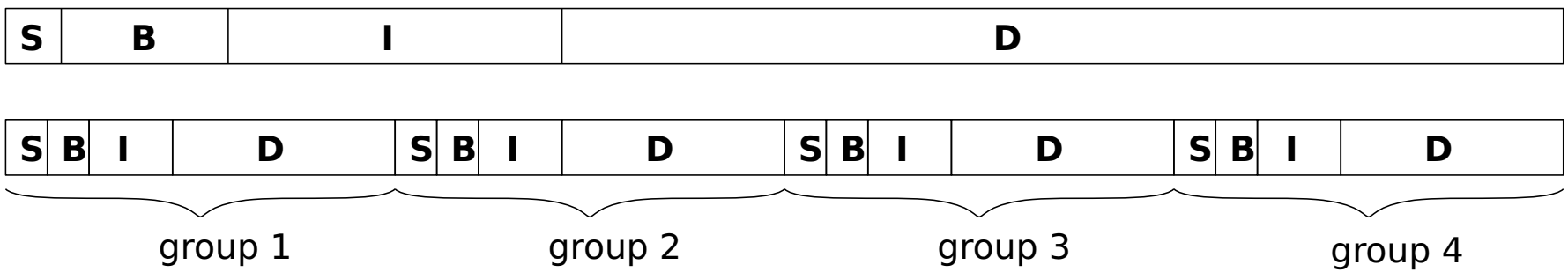
Where to Place Meta-data and Data on Disk?

How to avoid seeks when:

- Accessing data after accessing metadata?
  - Keep the inode of a file close to its data blocks
- Accessing a data block after accessing another data block?
  - Keep data blocks of a file close to each other.

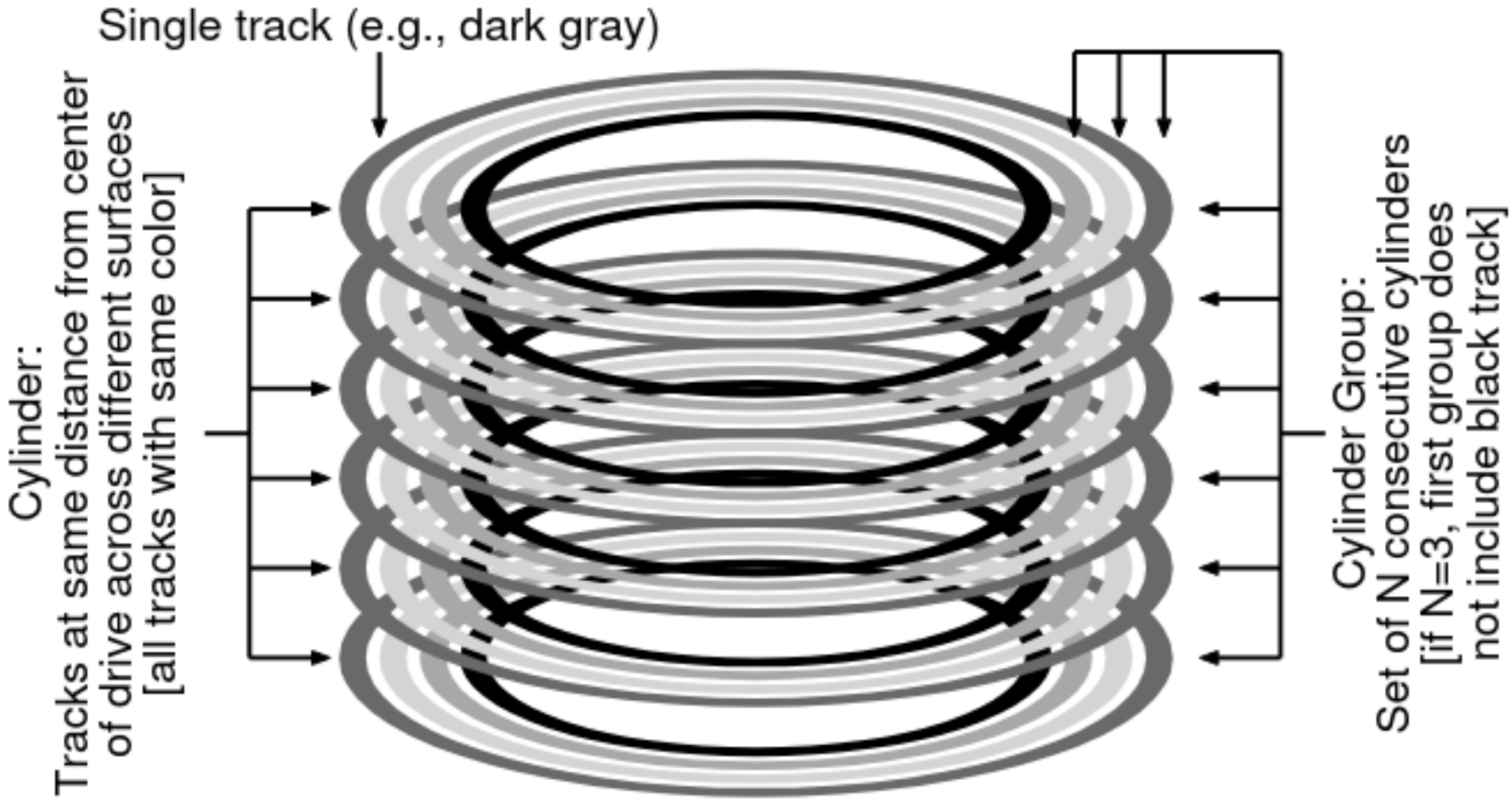
Split the disk in regions each with:

- its data blocks
- its inodes
- its bitmaps
- its superblock copy (for fault tolerance)



Try to place a file's inode and data blocks in the same region

Regions are Groups



FFS uses cylinder groups

ext2, ext3 and ext4 use block groups instead

- Modern disks hide their geometry

Policy: File and Directory Placement

Principle Keep related stuff together

Issue What is related?

Directories Directory inodes may be in a group different from their parents:

- Choose a group with a low number of directories and a high number of free inodes
- Place the directory data in the same group as the inode

Files

1. Places the inodes in the same group as its parent directory
2. Tries to place the data blocks in the same group as its inode

Policy: File and Directory Placement: Example

Assumptions

Groups with:

- 10 inodes
- 10 data blocks

Directories `/`, `/a` and `/b`

- Each with just one data block

Files `/a/c`, `/a/d`, `/a/e` and `/b/f`

- Each with two data blocks

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

Policy: Large File Exception

Issue If files are too large they may use up all the data blocks in its inode's group

Solution Allocate first indirect block, and the data blocks it points to, in a different group

- Change group after every 1 MiByte

Policy Summary

File Inodes allocate in **same** group with parent data and inode

Directory inodes allocate in **new** group

- Pick with fewer used inodes

First data block of file or directory

- Near its inode

Other data blocks allocate near previous block

Large file data blocks

- After 48 KB, use new group. (Move every subsequent 1 MB.)
- Pick group with fewer used data blocks

Block Size (1/2)

Observation in a previous change to the old file system speed more than doubled by doubling the block size (512 bytes)

- A significant part of this improvement was because of fewer accesses via indirect blocks

Design Decision Use at most double indirect blocks.

- The block size is a file system parameter – typical value is 4 KiByte. The block size affects:
  - Performance
  - Maximum file size
    - What's the maximum file size, assuming 4 byte block addresses?
    - What's the maximum size of a file system?

Block Size (1/2)

Why not larger? Most files were very small

- This is still true, even though nowadays many files are multimedia
- Large filesystem blocks lead to internal fragmentation
  - On average half a block per file is wasted
  - Measured waste:

Organization	% Waste
Data only	0.0
Data only, file starts on 512 byte boundary	4.2
Data + inodes, 512-byte blocks	6.9
Data + inodes, 1024-byte blocks	11.8
Data + inodes, 2048-byte blocks	22.4
Data + inodes, 4096-byte blocks	45.6

Solution Use **fragments**, i.e. sub-blocks

- The fragment size is a file system parameter, just like the block size
- Disk space waste is similar to that of a file system whose block size is equal to the fragment size

Fragments

Assumptions

Block size 4096 bytes

Fragment size 1024

Data Block Bitmap Must have a bit per fragment

- Blocks must be aligned
  - Not all free consecutive fragments are treated like a block

Addresses specify the fragment

- Whether an address is the address of a block or of a fragment is implicit
  - Determined by the rules for fragment allocation
    - E.g., fragments are used only for data that can fit in 3 or less blocks
- This and other rules
  - E.g. all fragments of a file must be in the same block

require copying of fragments to a new block, if there is no room to grow

- To reduce waste of space, a block may hold fragments of different files

Fast File System Conclusion

First disk-aware file system

- Bitmaps
- Locality groups
- Large blocks with fragments
- Smart file/directory placement

Lesson each hardware is unique

- Treat disk like disk
- Treat flash like flash