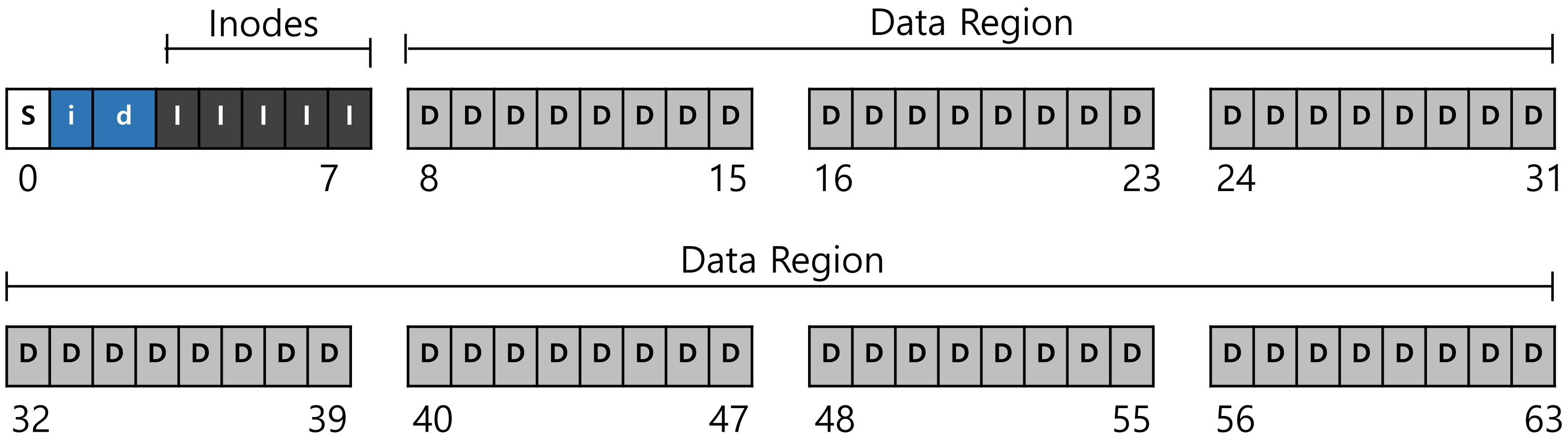
Persistence: Fast File System

Sridhar Alagar

Very Simple File System

- On-disk structures:
  - Super block, bitmap, inode table, directories, indirect block pointers



create /foo/bar: What needs to be read and written?

data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data
		read				
					read	
			read			
						read
	read					
	write					
						write
				read		
				write		
			write			

open /foo/bar

data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data
		read					
					read		
			read				
						read	
				read			

read /foo/bar - assume opened

data	inode	root	foo	bar	root	foo	bar
bitmap	bitmap	inode	inode	inode	data	data	data
				read			
							read
				write			

write to /foo/bar (assume file exists and has been opened)

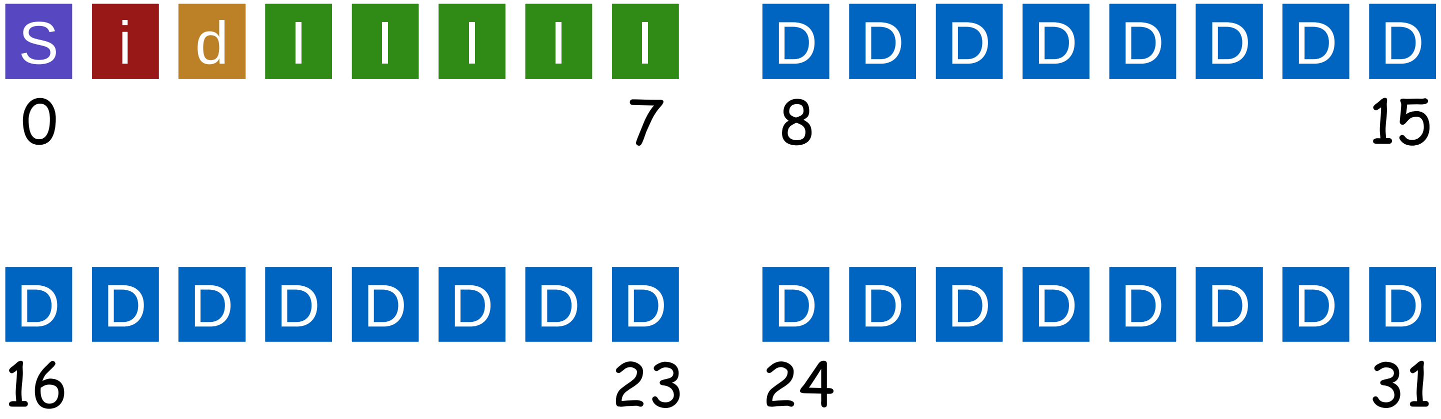
data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data
				read			
read							
write							
							write
				write			

close /foo/bar

data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data

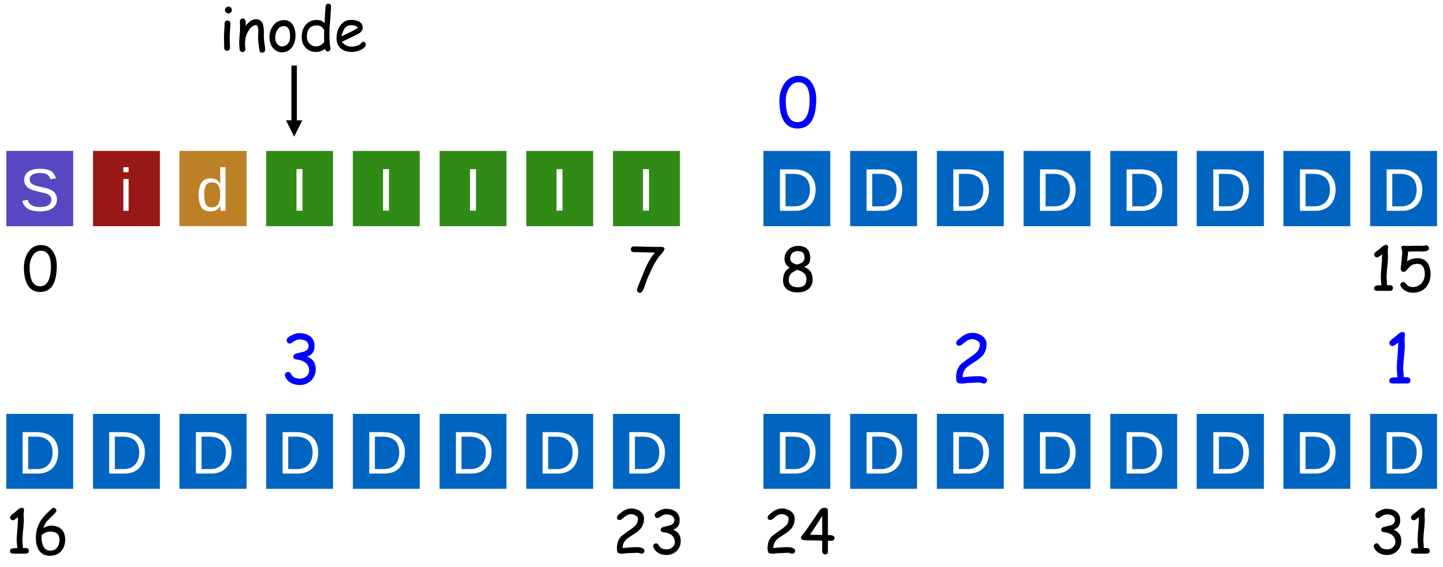
nothing to do on disk!

Performance

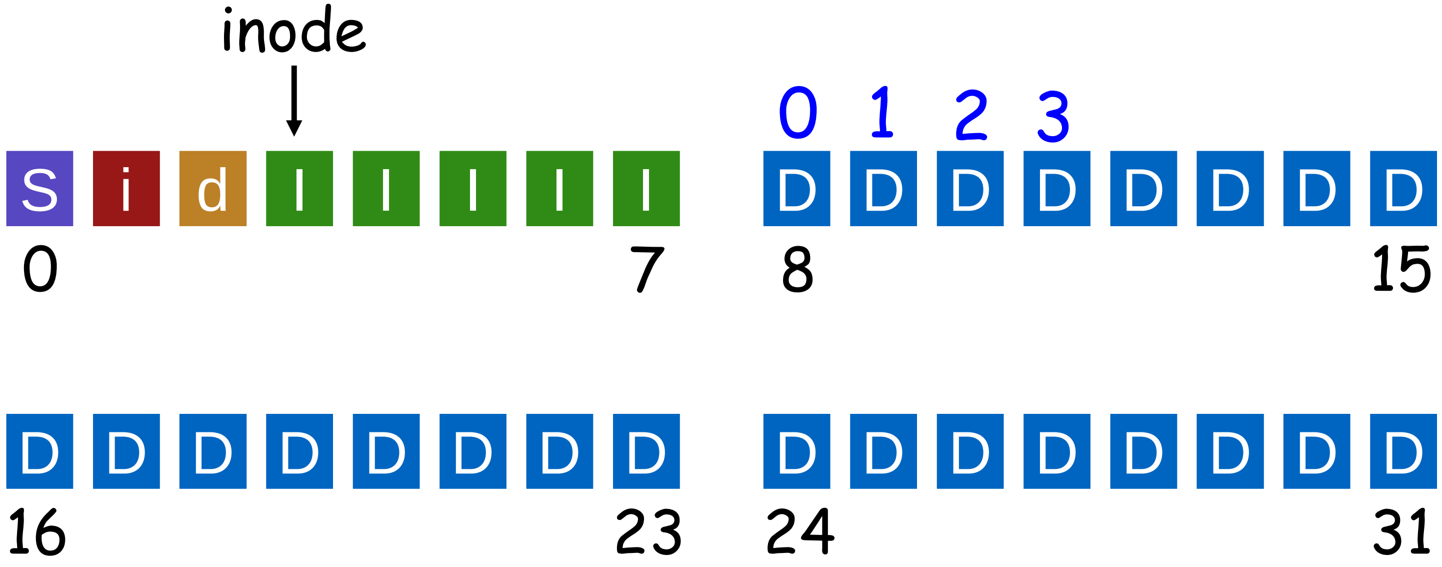


Which inodes, data blocks should be chosen for good performance?

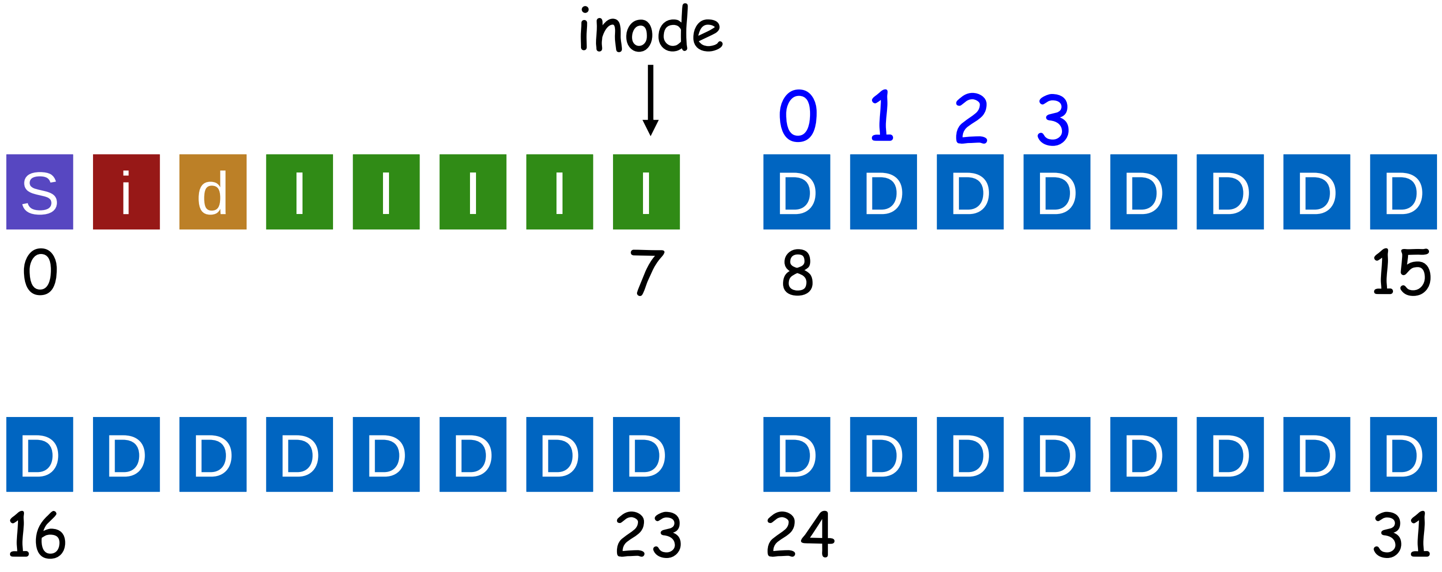
Bad Allocation



Better Allocation



Best Allocation



Can't do this for all files ☹

What is the right block size?

- Large block size (4k)
  - Better transfer rate
  - Internal fragmentation

- Small blocks (512 bytes)
  - Reduces internal fragmentation
  - Bad for transfer rate

Old FS Issues

- Block size issues (small vs big)
- Blocks laid out poorly
  - long distance between inodes/data
  - related inodes not close to one another

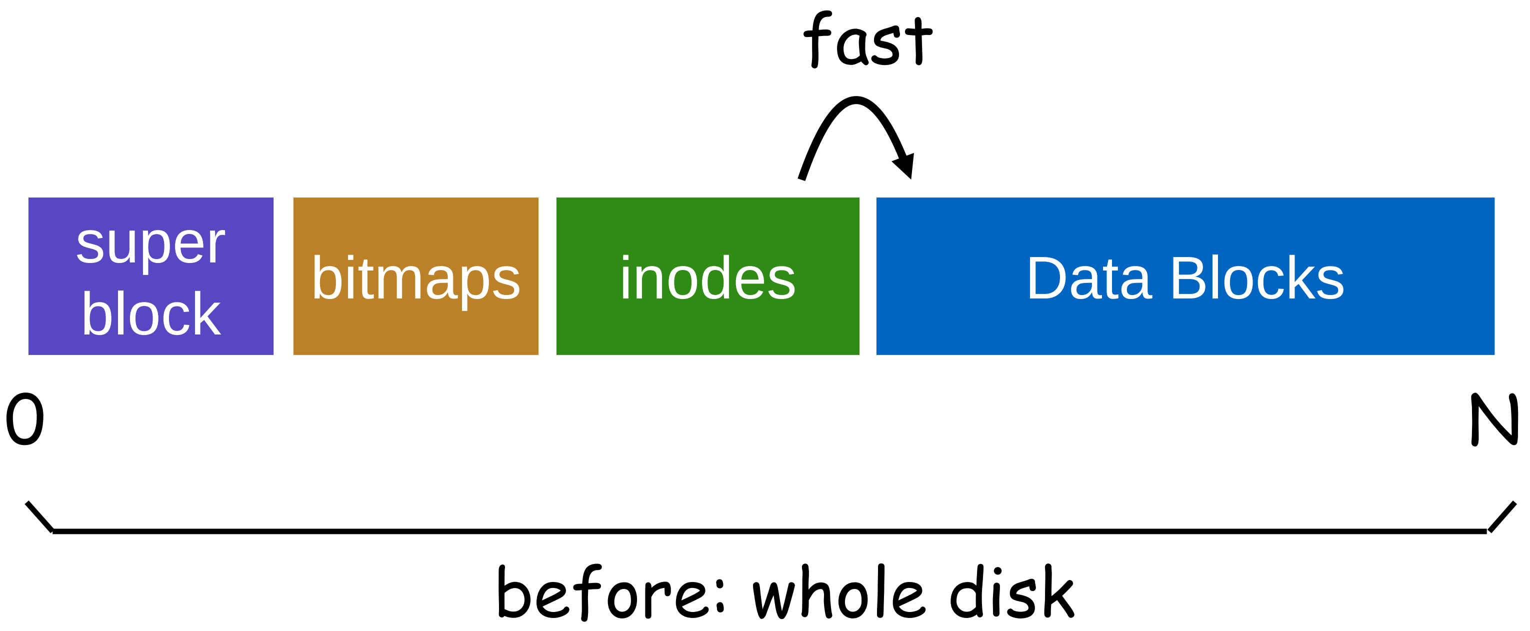
- Result: 2% of potential performance! (and worse over time)

Problem: old FS treats disk like RAM!

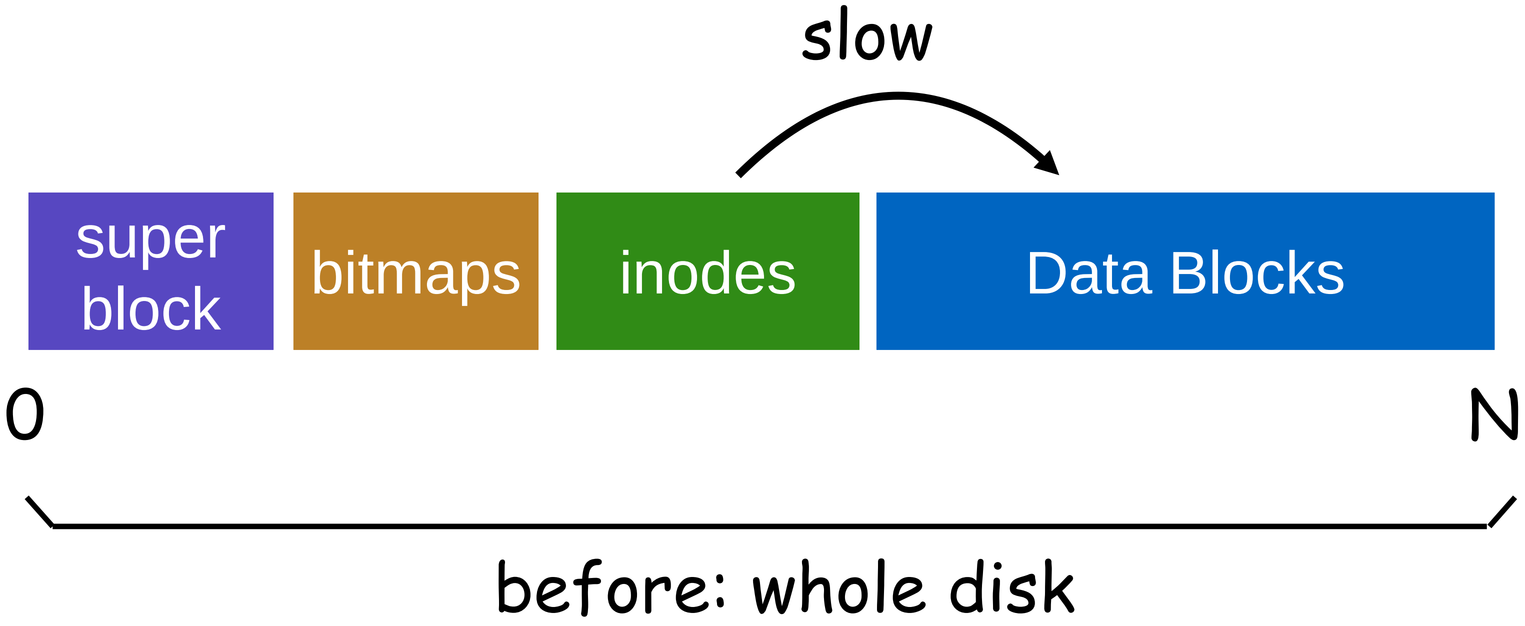
FFS: Disk Aware

- By a team at UC Berkeley
- Design questions:
  - Where to place meta-data and data on disk?
  - How to use big blocks without wasting space?

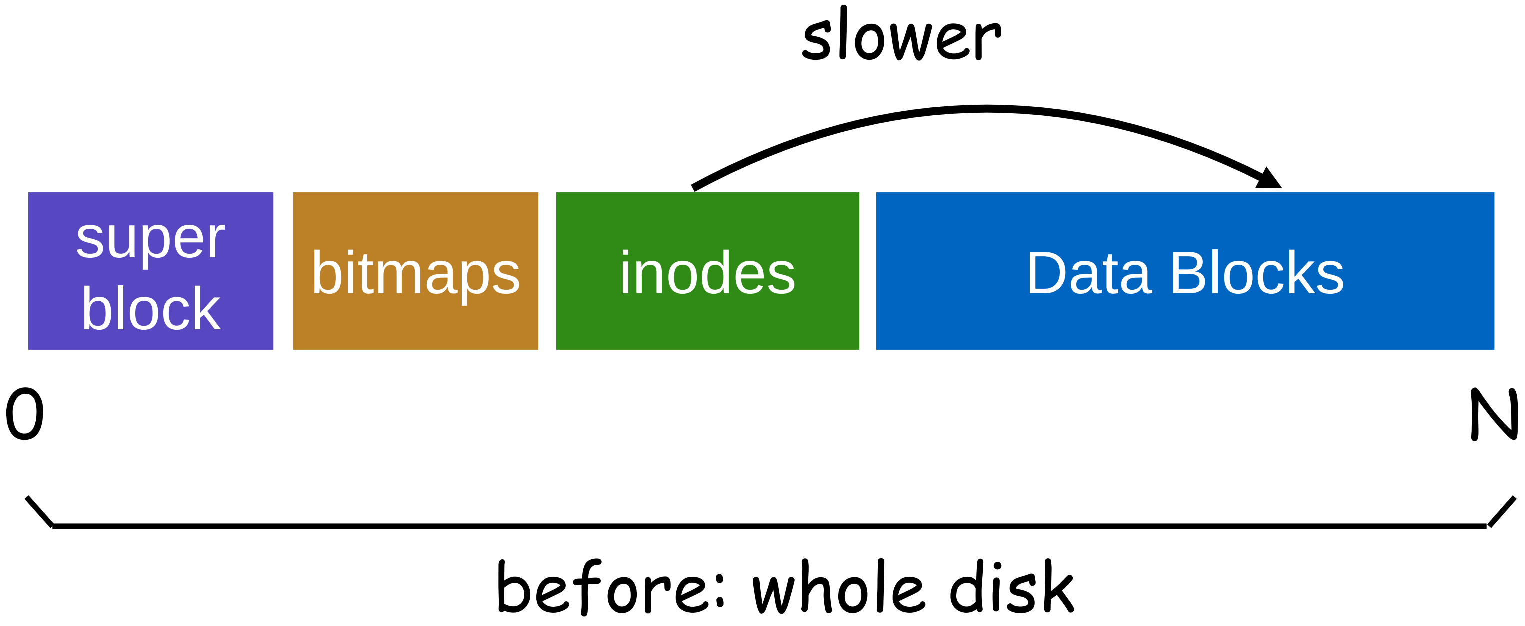
How to keep inode close to data?



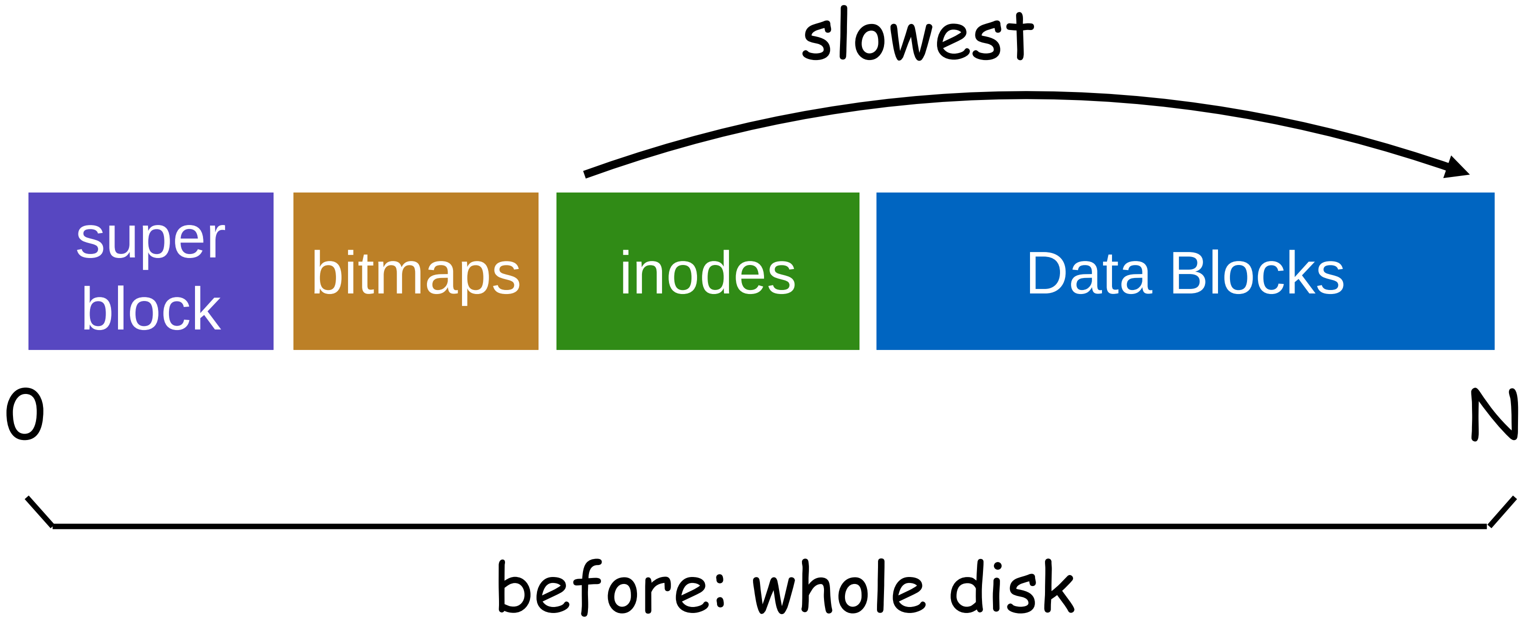
How to keep inode close to data?



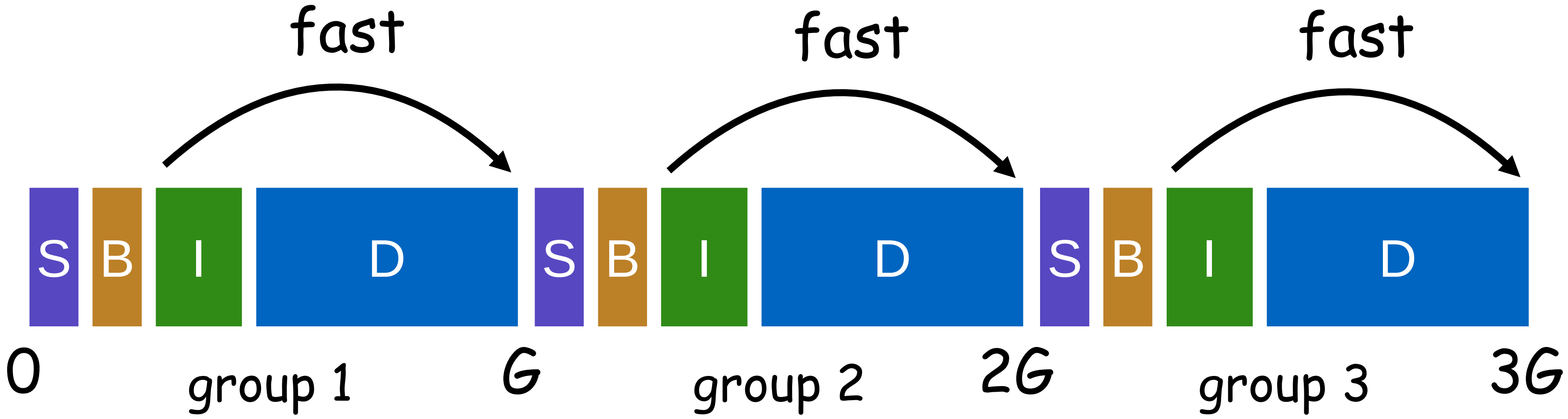
How to keep inode close to data?



How to keep inode close to data?



Organize Disk into Groups



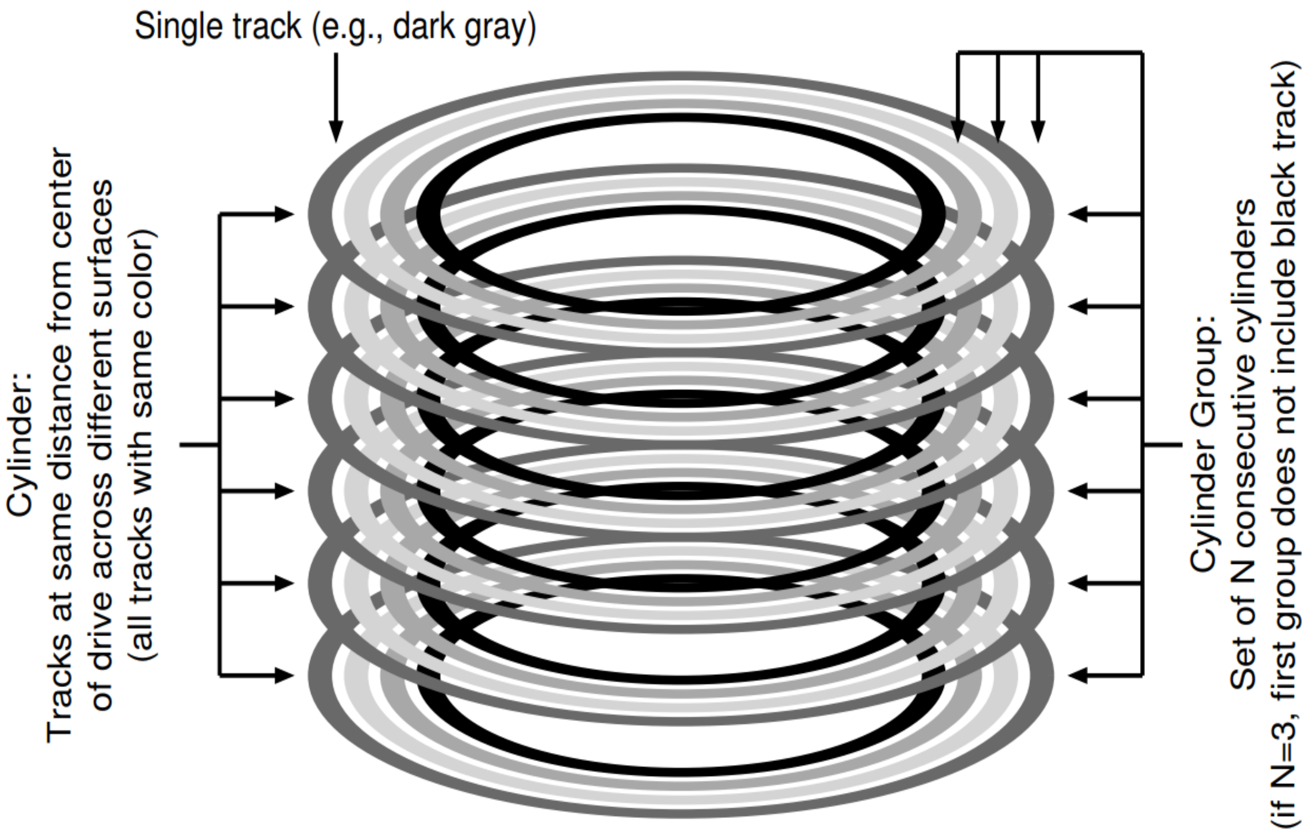
strategy: allocate inodes and related data blocks in same group

Super block replication provides fault tolerance

How to Group?

- FFS organized into cylinder groups

- Modern FS (ext2-4) organize into block groups


Single track (e.g., dark gray)
Cylinder:
Tracks at same distance from center
of drive across different surfaces
(all tracks with same color)
Cylinder Group:
Set of N consecutive cylinders
(if N=3, first group does not include black track)

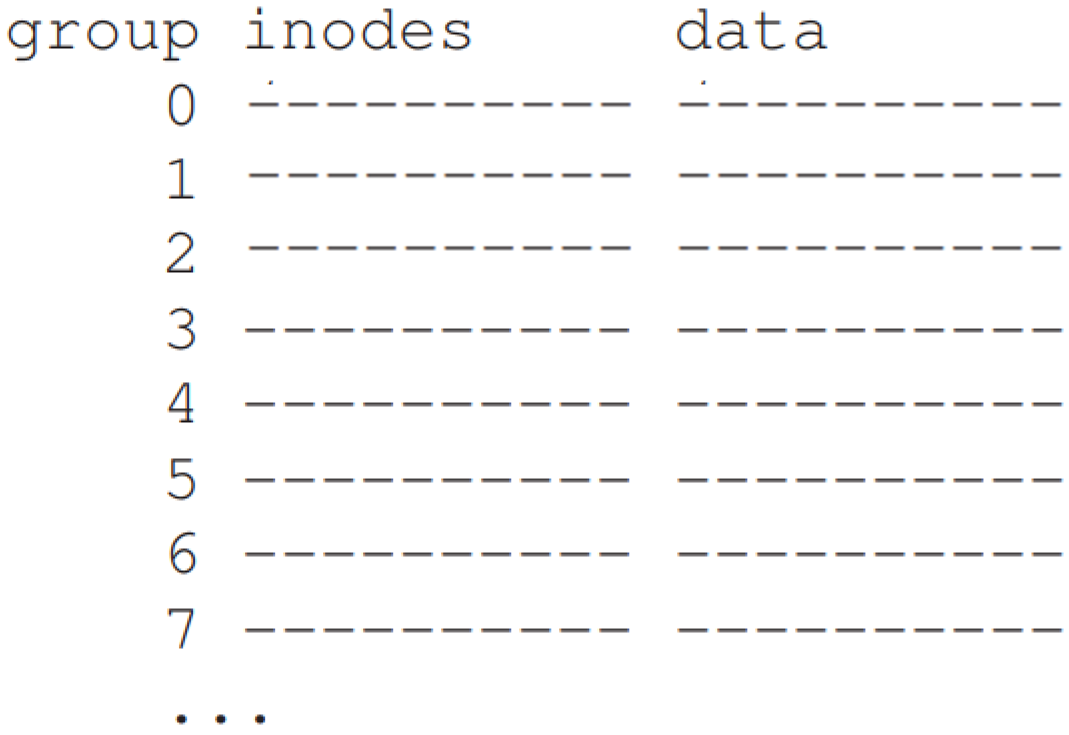
How to allocate files and directories?

- Policy

- Keep related entities together (principal of locality)

- Keep unrelated entities far apart

How to allocate files and directories?



```
group inodes      data
    0 ----------- -----------
    1 ----------- -----------
    2 ----------- -----------
    3 ----------- -----------
    4 ----------- -----------
    5 ----------- -----------
    6 ----------- -----------
    7 ----------- -----------
    ...
```

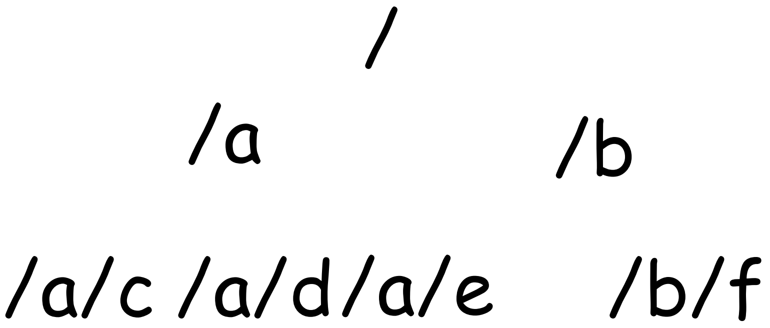
/

/a /b

/a/c /a/d /a/e /b/f

How to allocate files and directories?

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



Allocation Heuristic

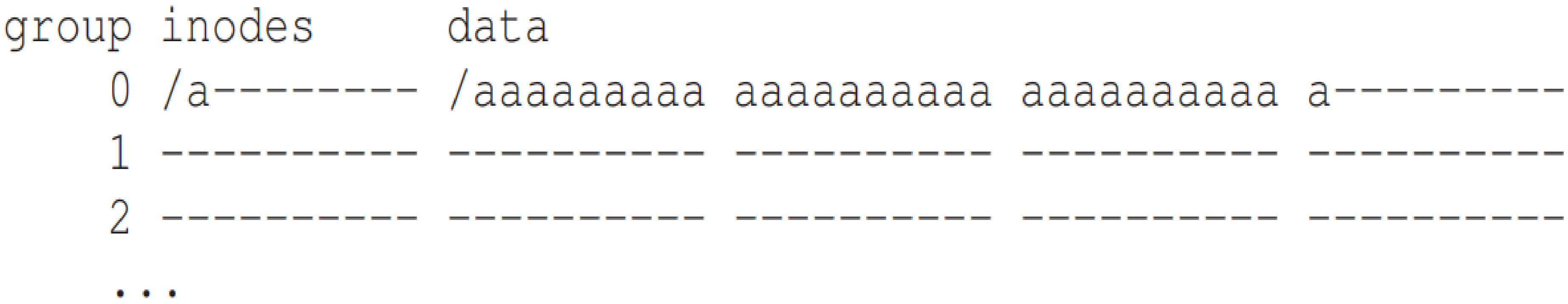
File inodes: allocate in <u>same</u> group with dir

Dir inodes: allocate in <u>new</u> group with fewer used inodes than average group

First data block: allocate near inode

Other data blocks: allocate near previous block

What happens if a file is large?



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

Not enough blocks for other files in the same directory

What happens if a file is large?

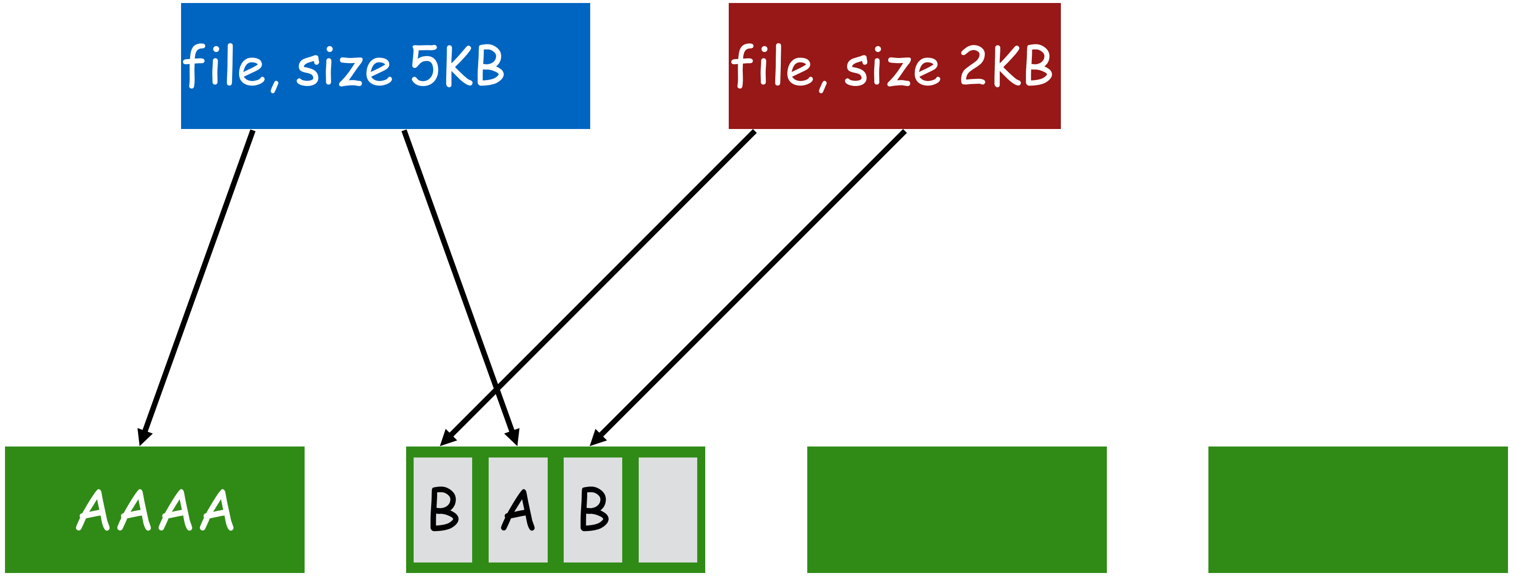
```
group inodes     data
    0 /a-------- /aaaaa---- ---------- ---------- ----------
    1 ---------- aaaaa----- ---------- ---------- ----------
    2 ---------- aaaaa----- ---------- ---------- ----------
    3 ---------- aaaaa----- ---------- ---------- ----------
    4 ---------- aaaaa----- ----------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

Divide large files into reasonable sized chunks

transfer time more than seeking time (amortized)

Block Size: Hybrid

- Keep 4k blocks
- Divide some 4k blocks into 8 sub-blocks of 512 bytes
- Allocate sub-blocks for small files
- What happens when a small file grows?

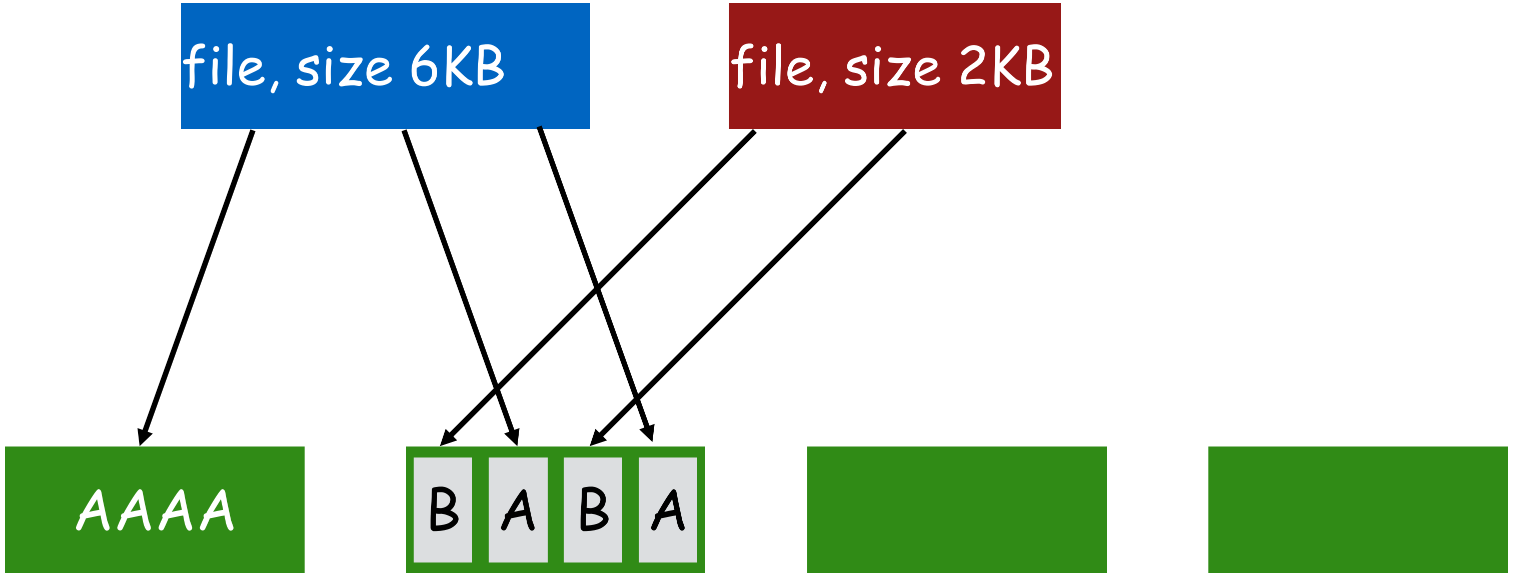


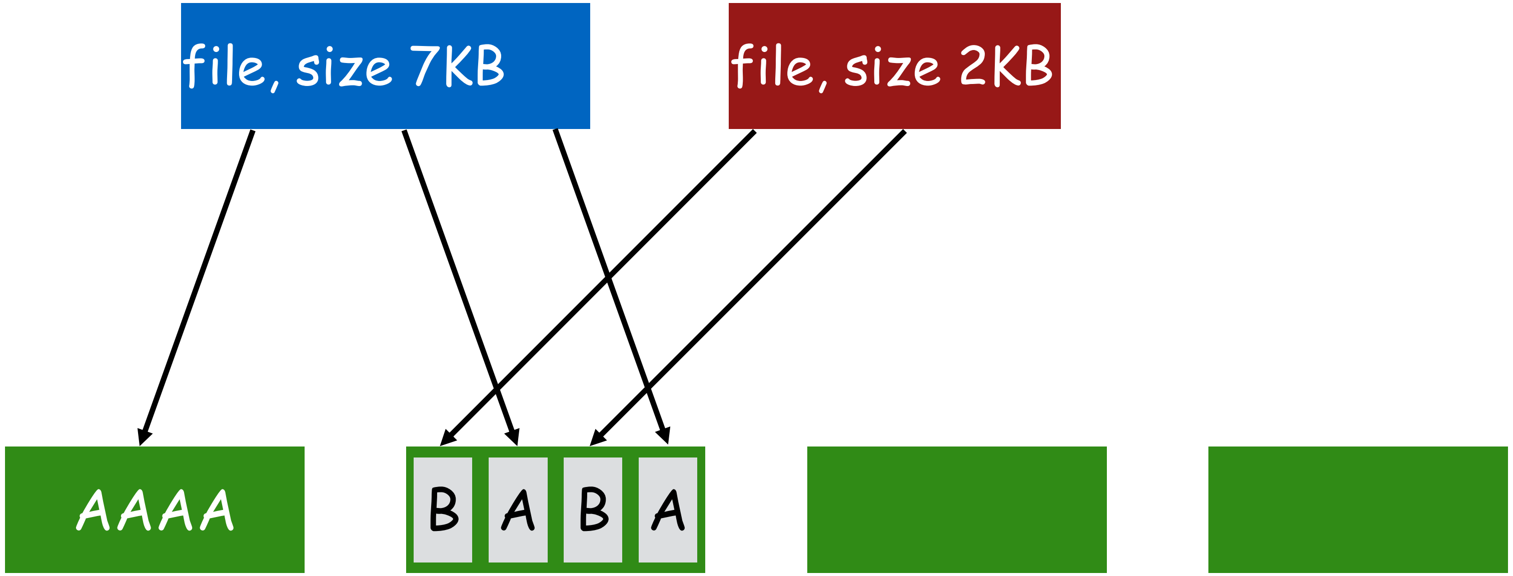
file, size 5KB

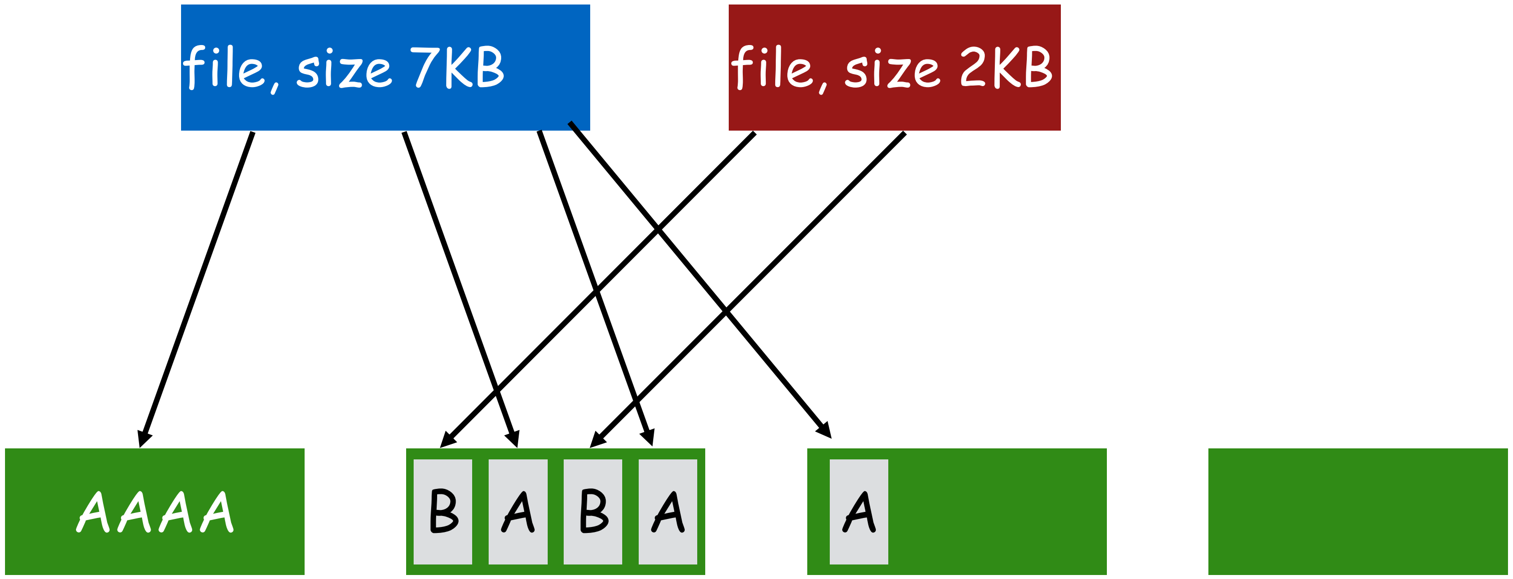
file, size 2KB

AAAA

B A B

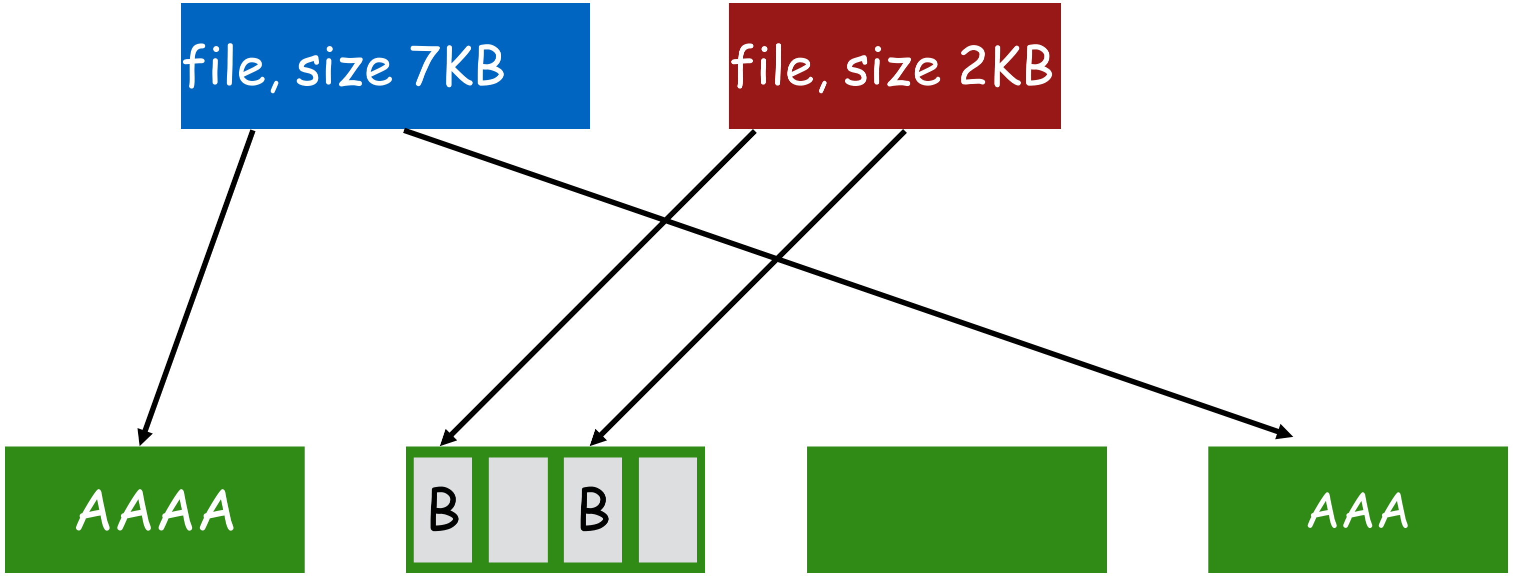

file, size 6KB
file, size 2KB
AAAA
B
A
B
A


file, size 7KB
file, size 2KB
AAAA
B
A
B
A



Not allowed to use fragments across multiple blocks!

What to do instead?


file, size 7KB
file, size 2KB
AAAA
B
B
AAA

Conclusion

- FFS is the first disk-aware file system
  - Bitmaps
  - Locality groups
  - Replicated superblocks
  - Large blocks
  - Fragmention
  - Smart allocation policy
- FFS inspired modern files systems, including ext2 and ext3
- FFS also introduced several new features:
  - long file names
  - atomic rename
  - symbolic links

Disclaimer

- Some of the materials in this lecture slides are from the lecture slides by Prof. Arpaci, Prof. Youjip, and other educators. Thanks to all of them.