Persistence: Fast File System (FFS)

Questions answered in this lecture:

How to improve performance of complex system?

Why do file systems obtain worse performance over time?

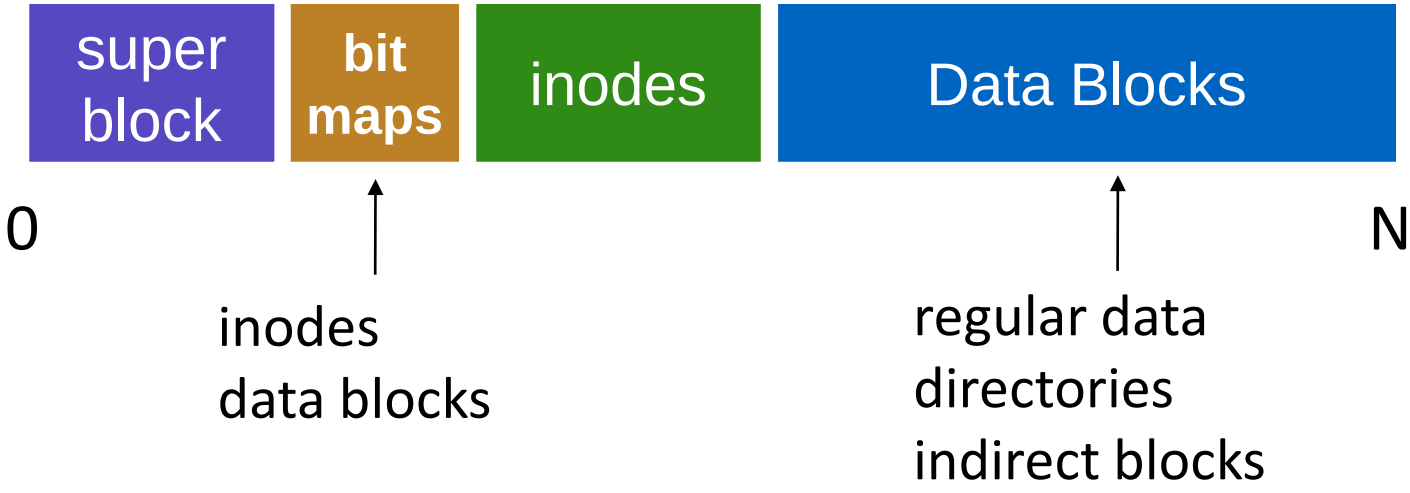
How to choose the right block size? How to avoid internal fragmentation?

How to place related blocks close to one another on disk?

File-System Case Studies

- **Local**
  - FFS: Fast File System
  - LFS: Log-Structured File System
- **Network**
  - NFS: Network File System
  - AFS: Andrew File System

Review: Basic Layout



What is stored as a data block?

REVIEW: create /foo/bar

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

create /foo/bar

[traverse]

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

Verify that bar does not already exist

create /foo/bar

[allocate inode]

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

create /foo/bar

[populate inode]

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

Why must read bar inode?

How to initialize inode?

create /foo/bar

[add bar to /foo]

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

Update inode (e.g., size) and data for directory

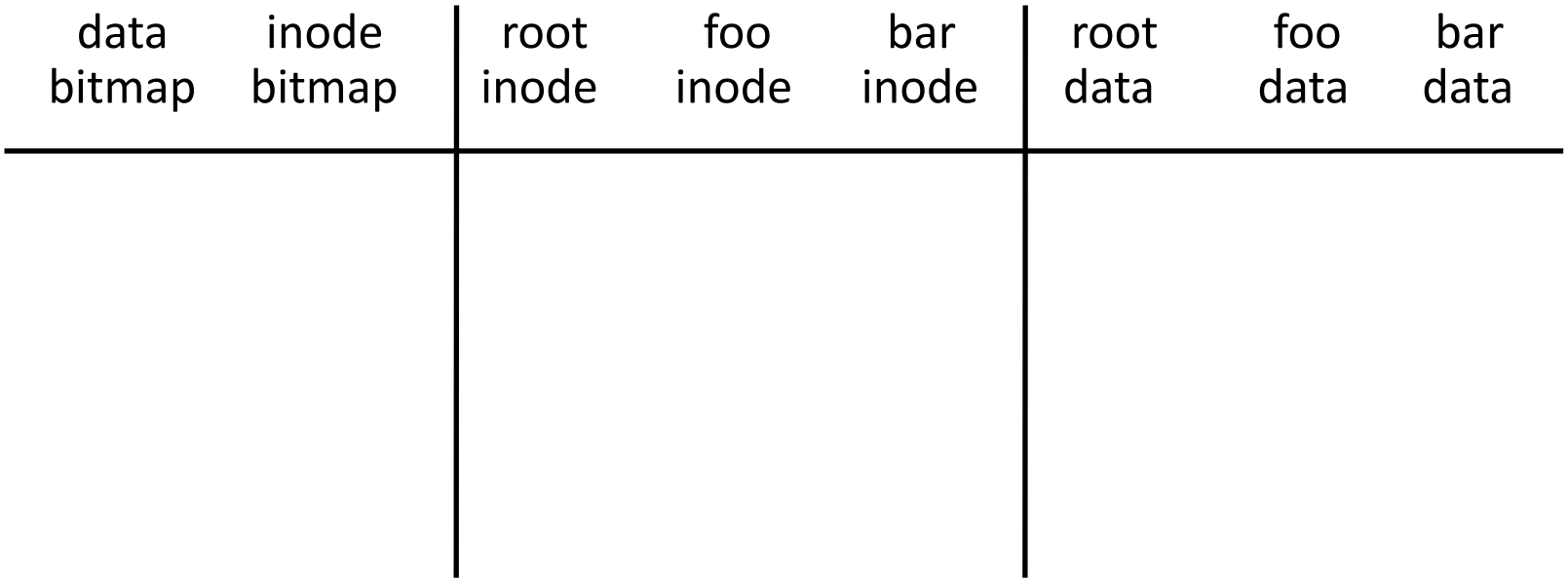
open /foo/bar

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

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			

append to /foo/bar



append to /foo/bar (opened already)

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

append to /foo/bar

[allocate block]

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

append to /foo/bar

[point to block]

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

append to /foo/bar

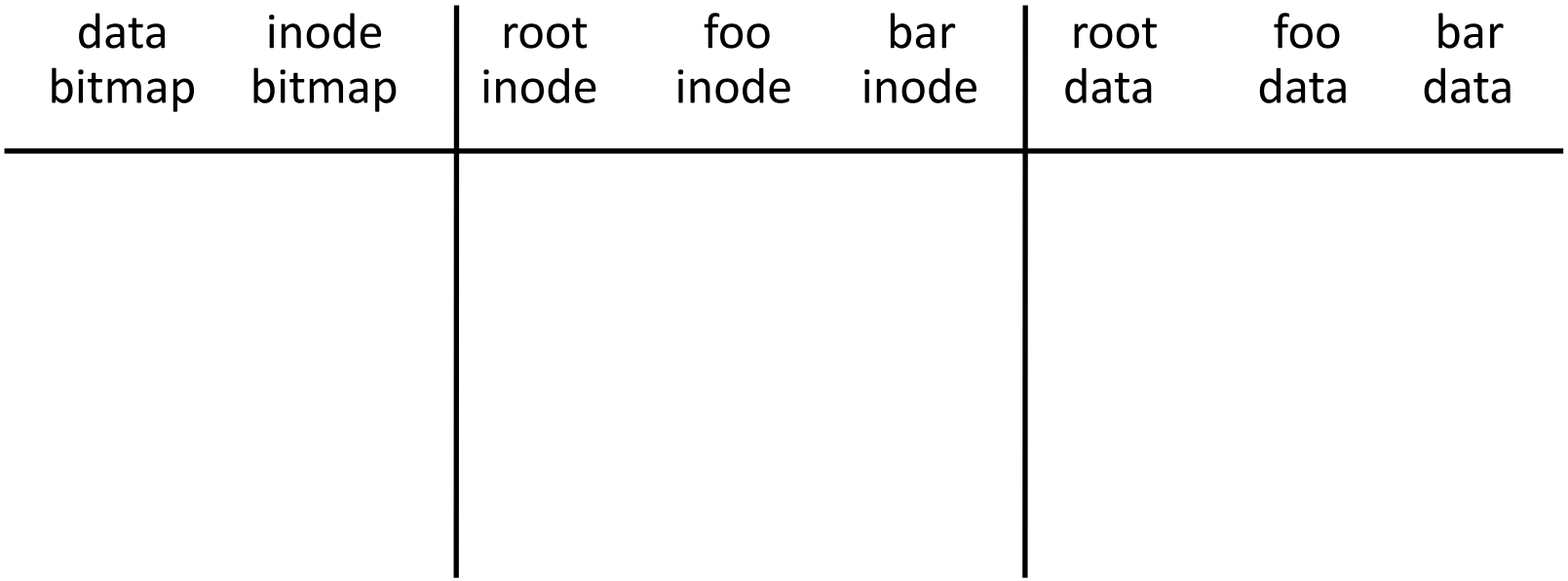
[write to block]

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

read /foo/bar – assume opened

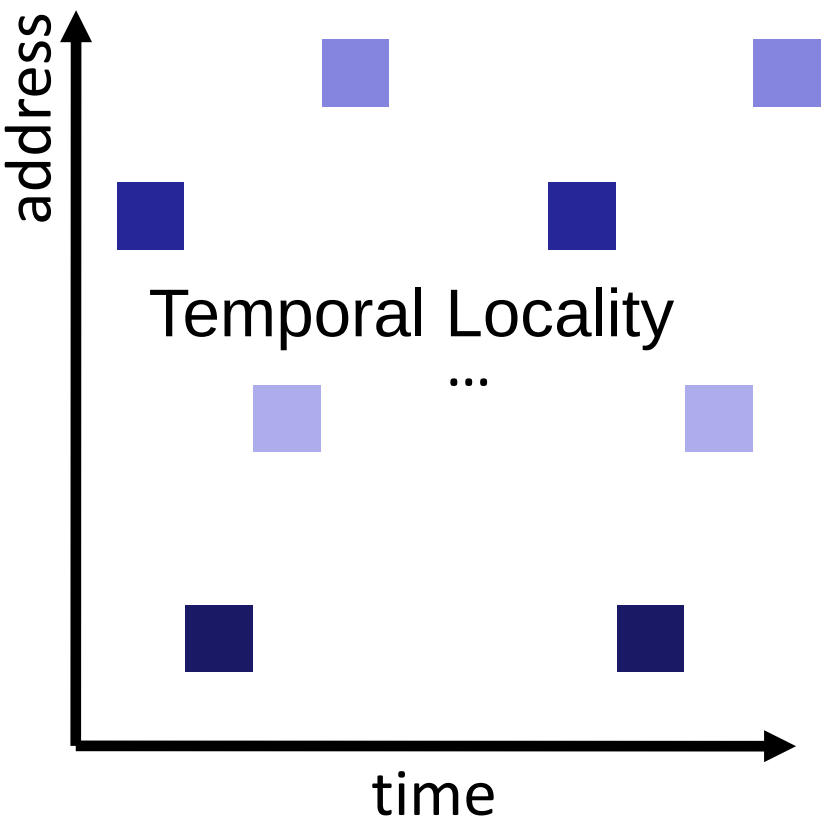
data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data
				read			
							read
				write			

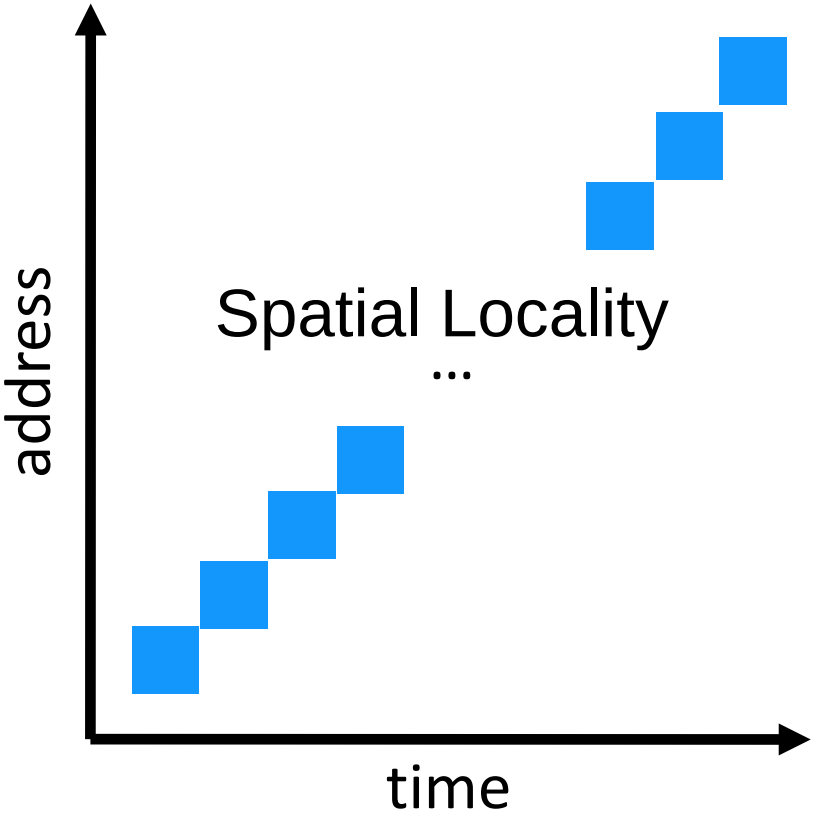
close /foo/bar



nothing to do on disk!

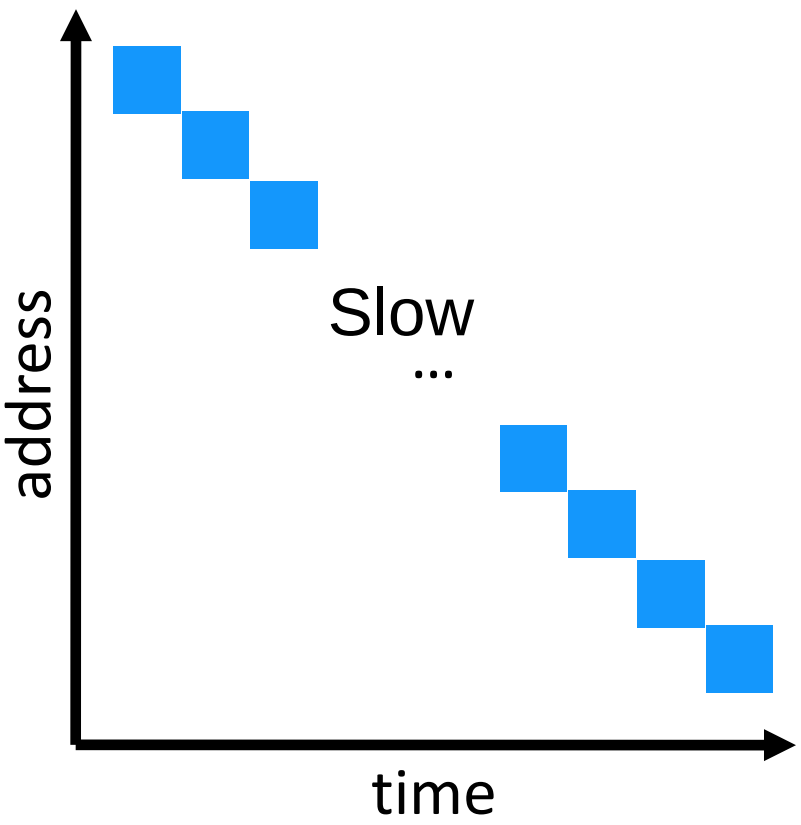
Review: Locality Types

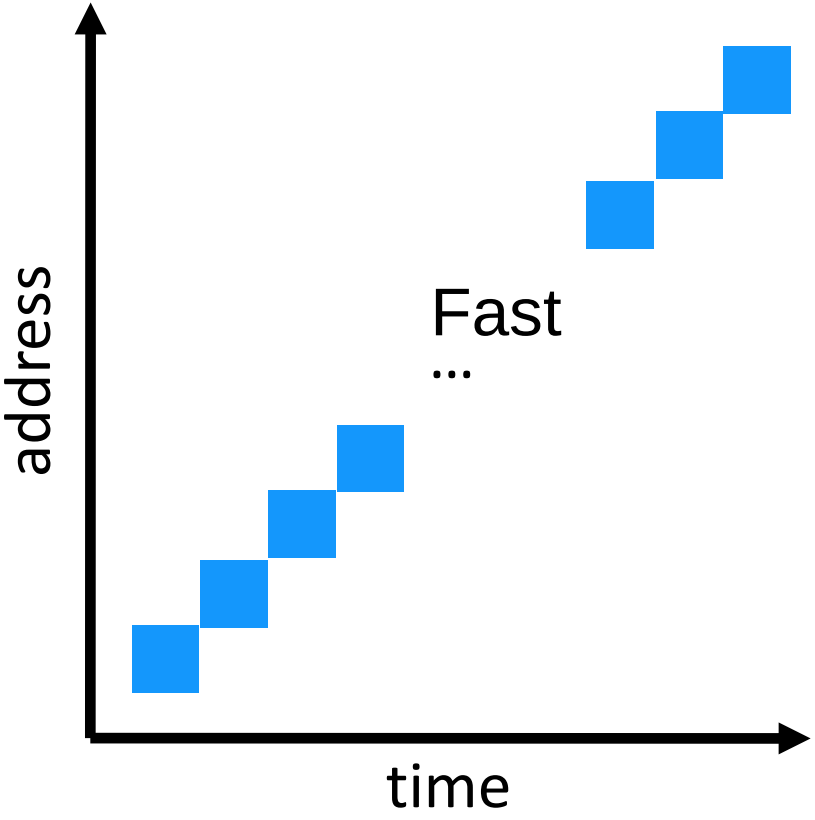




Which type of locality is most interesting with a disk?

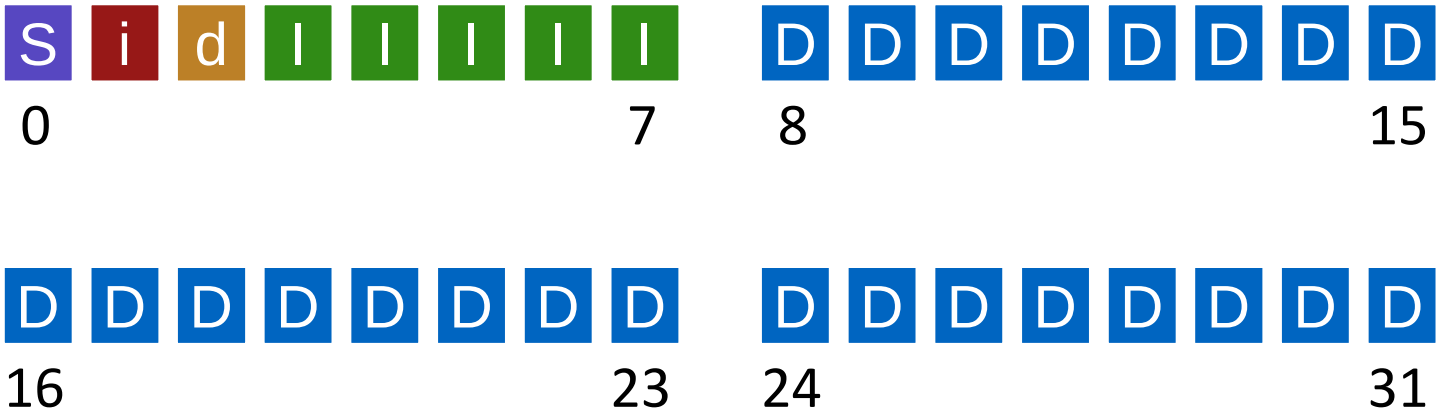
Order Matters





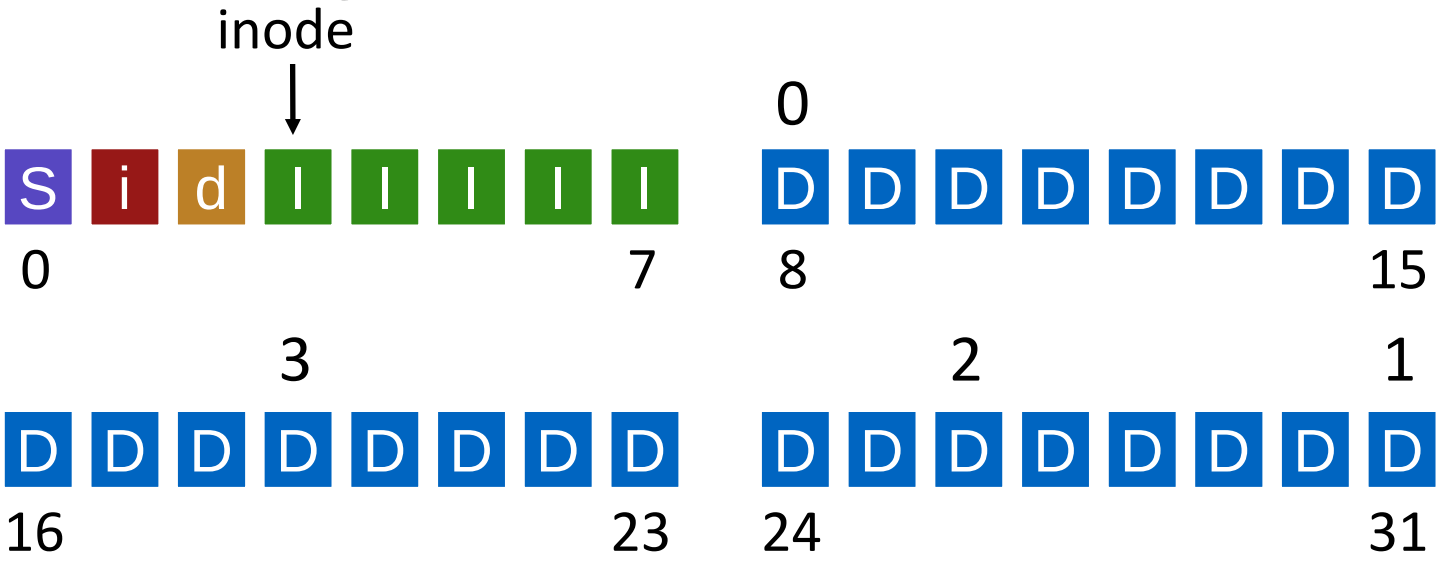
Implication for organizing data on a disk?

Policy: Choose Inode, Data Blocks

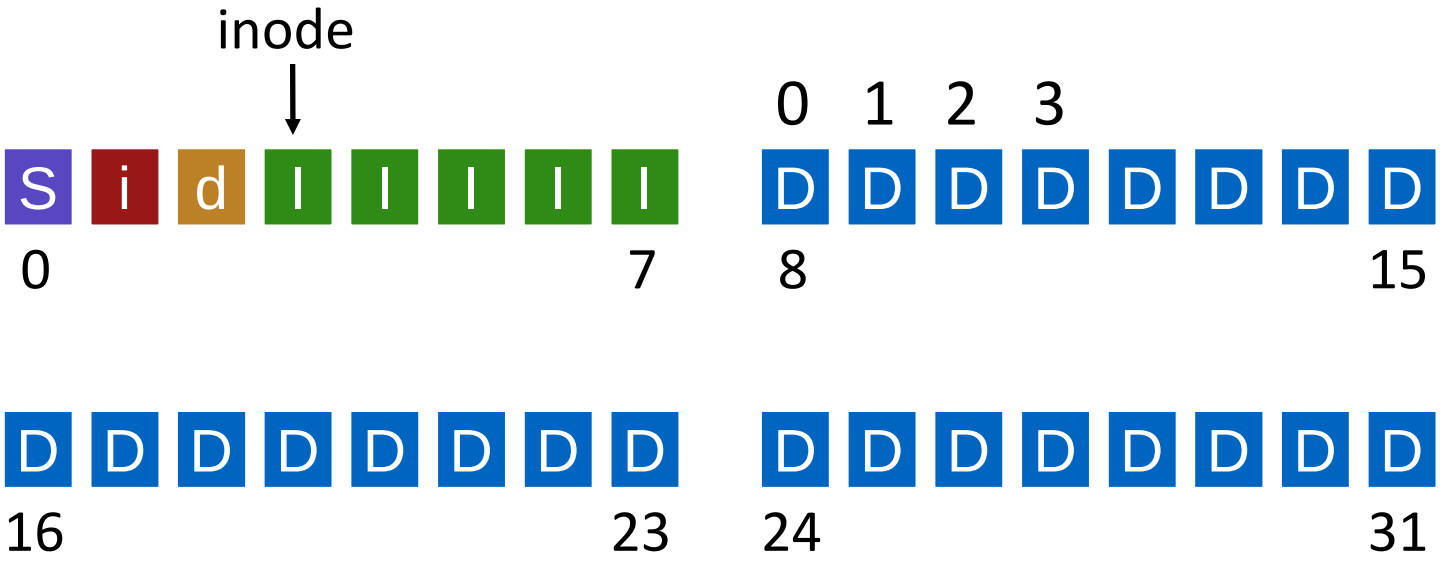


Assuming all free, which should be chosen?

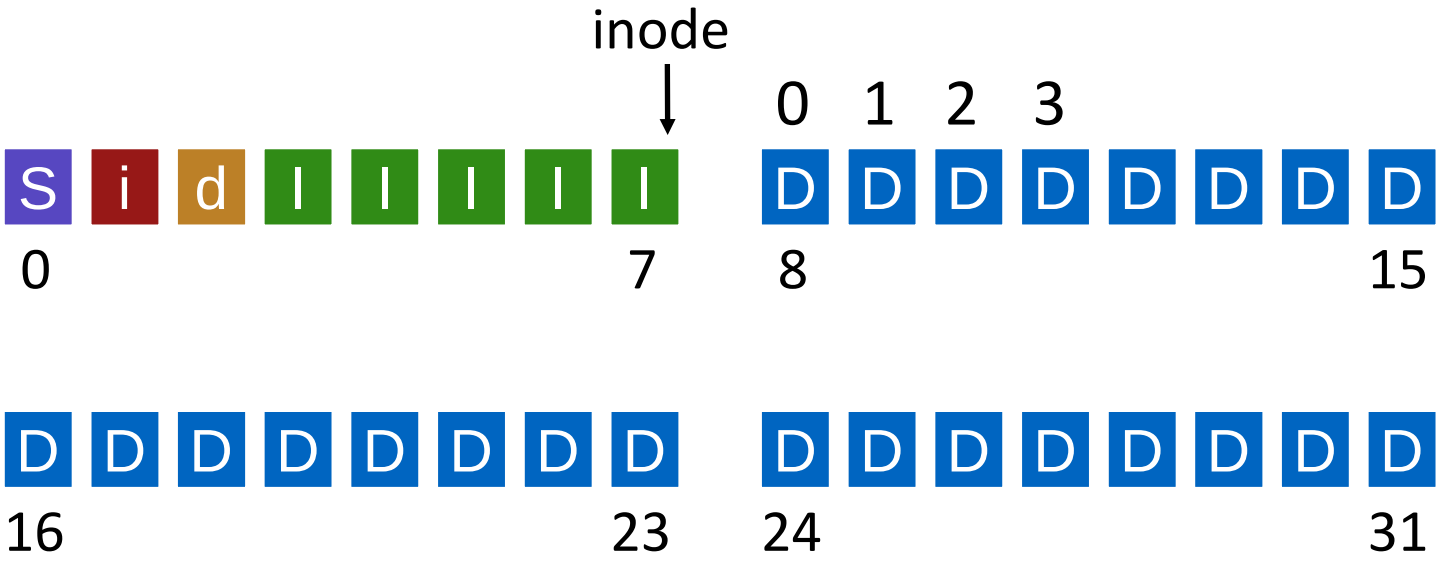
Bad File Layout



Better File Layout



Best File Layout



Can’t do this for all files ☹

Fast File System:
FFS
(1980's)

System Building

- **Beginner's approach**
  - 1. get idea
  - 2. build it!

measure then build

- **Pro approach**
  - 1. identify existing state of the art
  - 2. measure it, identify and understand problems
  - 3. get idea (solutions often flow from deeply understanding problem)
  - 4. build it!

Measure Old FS

- **State of the art: original UNIX file system**

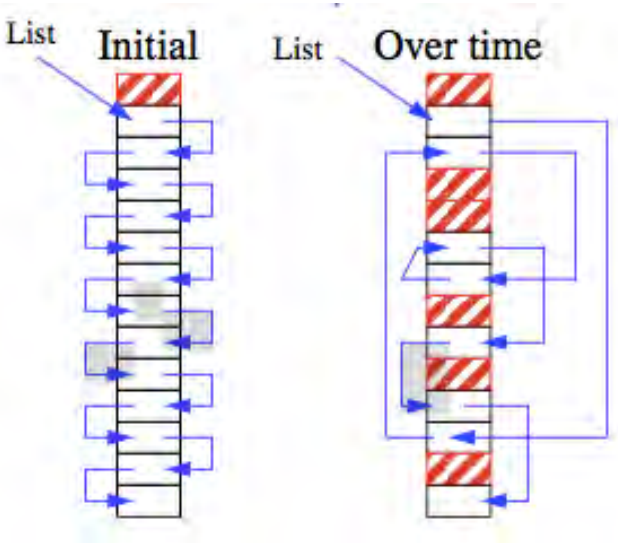
0 N

Free lists are embedded in inodes, data blocks
Data blocks are 512 bytes

- Measure throughput for whole sequential file reads/writes
- Compare to theoretical max, which is… **disk bandwidth**
- Old UNIX file system: achieved only **2%** of potential. Why?

Measurement 1: Aging?

- **What is performance before/after aging?**
  - New FS: **17.5%** of disk bandwidth
  - Few weeks old: **3%** of disk bandwidth
- **Problem: FS is becomes fragmented over time**
  - Free list makes contiguous chunks hard to find
- **Hacky Solutions:**
  - Occassional defrag of disk
  - Keep freelist sorted



Measurement 2: Block Size?

- **How does block size affect performance?**
  - Try doubling it!
- **Result: Performance more than doubled**
- **Why double the performance?**
  - Logically adjacent blocks not physically adjacent
  - Only half as many seeks+rotations now required
- **Why more than double the performance?**
  - Smaller blocks require more indirect blocks

Old FS Summary

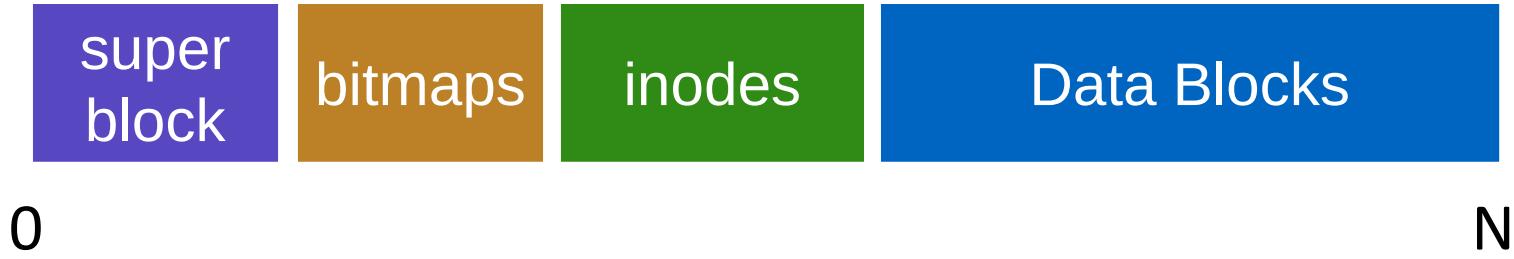
- **Free list becomes scrambled → random allocations**
- **Small blocks (512 bytes)**
- **Blocks laid out poorly**
  - long distance between inodes/data
  - related inodes not close to one another
    - Which inodes related? Inodes in same directory (ls –l)
- **Result: 2% of potential performance! (and worse over time)**

Problem: old FS treats disk like RAM!

Solution: a disk-aware

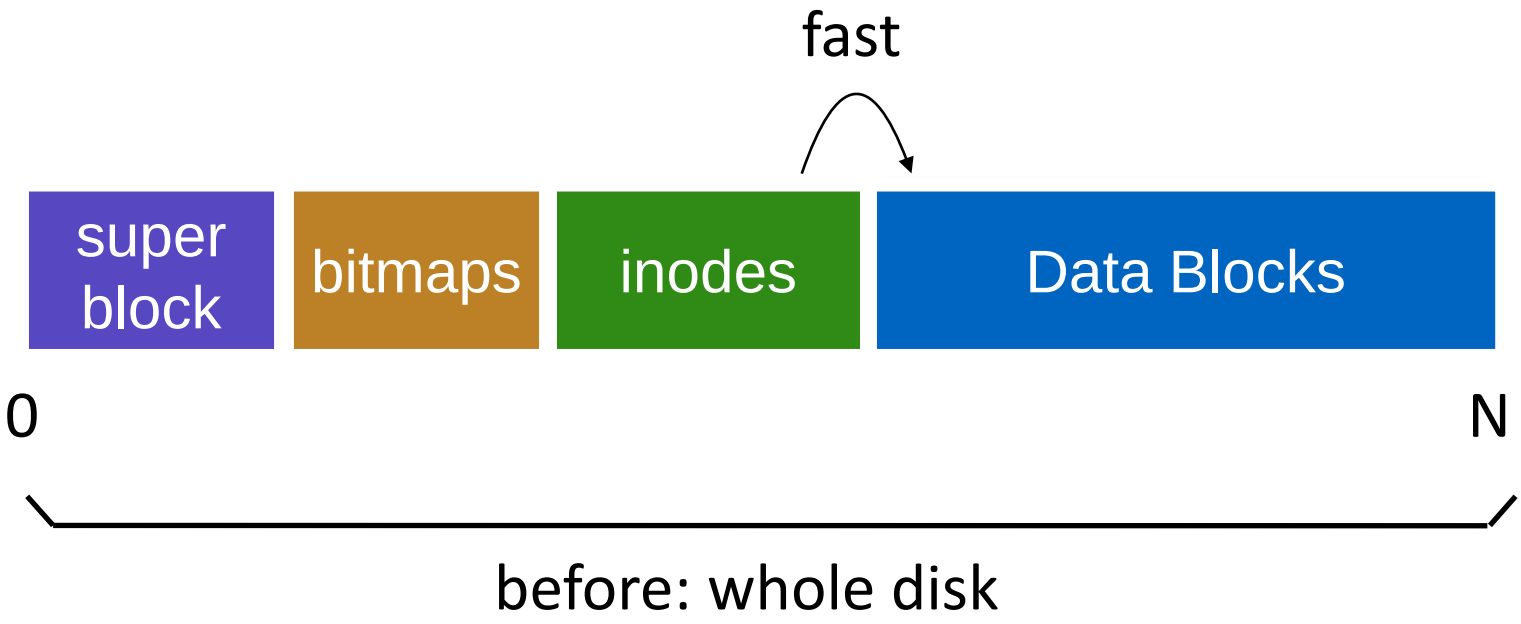
- **Primary File System Design Questions:**
  - Where to place meta-data and data on disk?
  - How to use big blocks without wasting space?

Placement Technique 1: Bitmaps



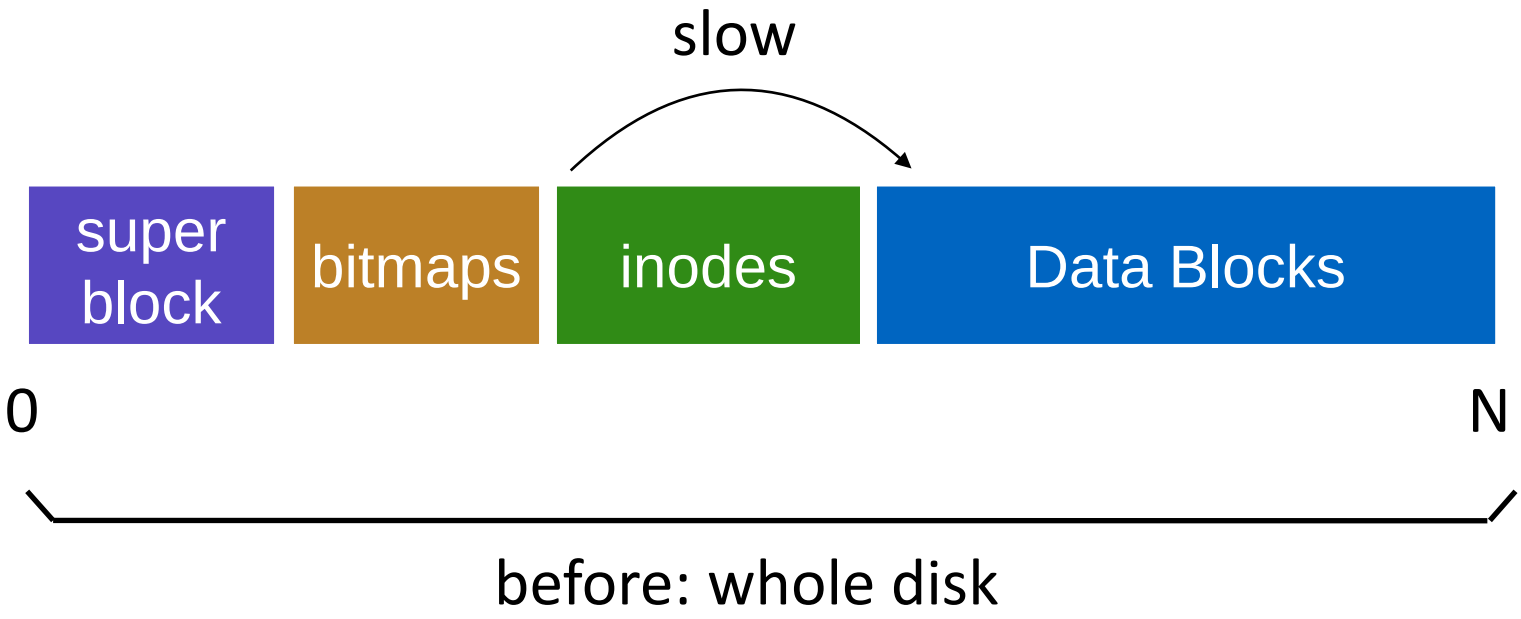
- Use bitmaps instead of free list
- Provides better speed, with more global view
- Faster to find contiguous free blocks

Placement Technique 2: Groups



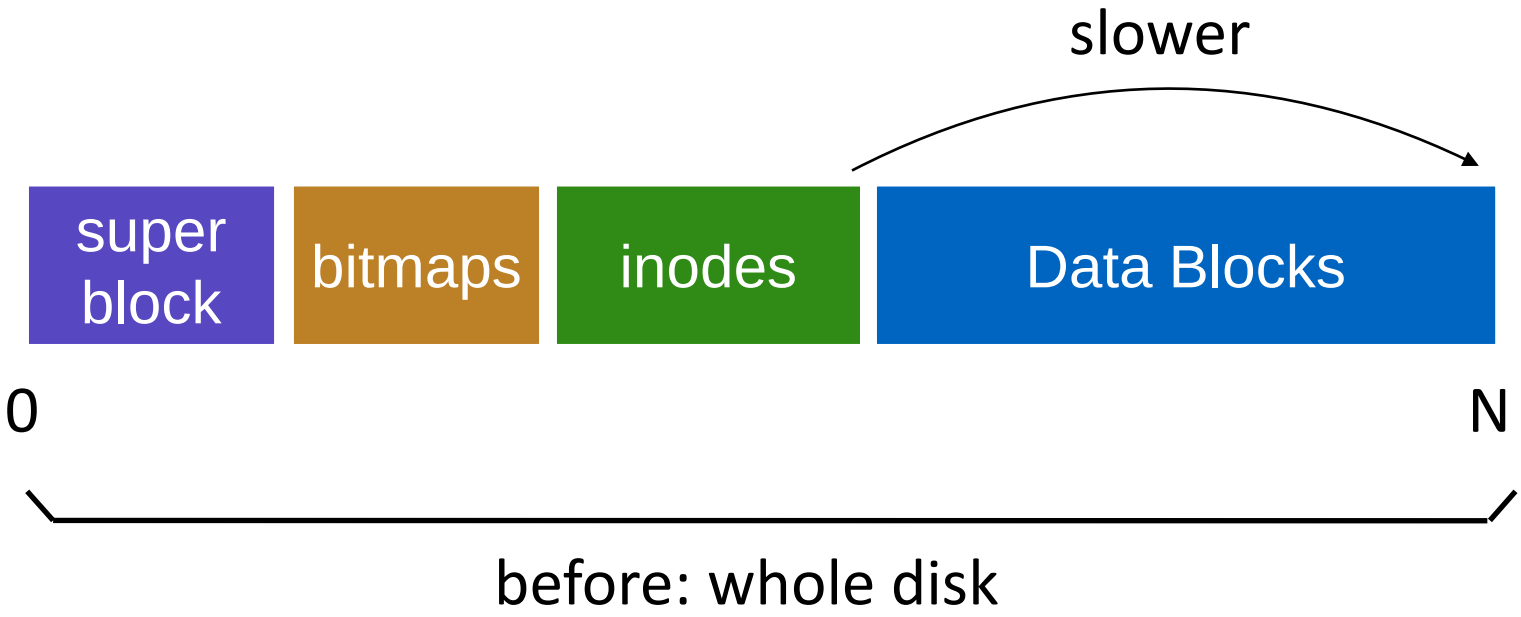
How to keep inode close to data?

Technique 2: Groups



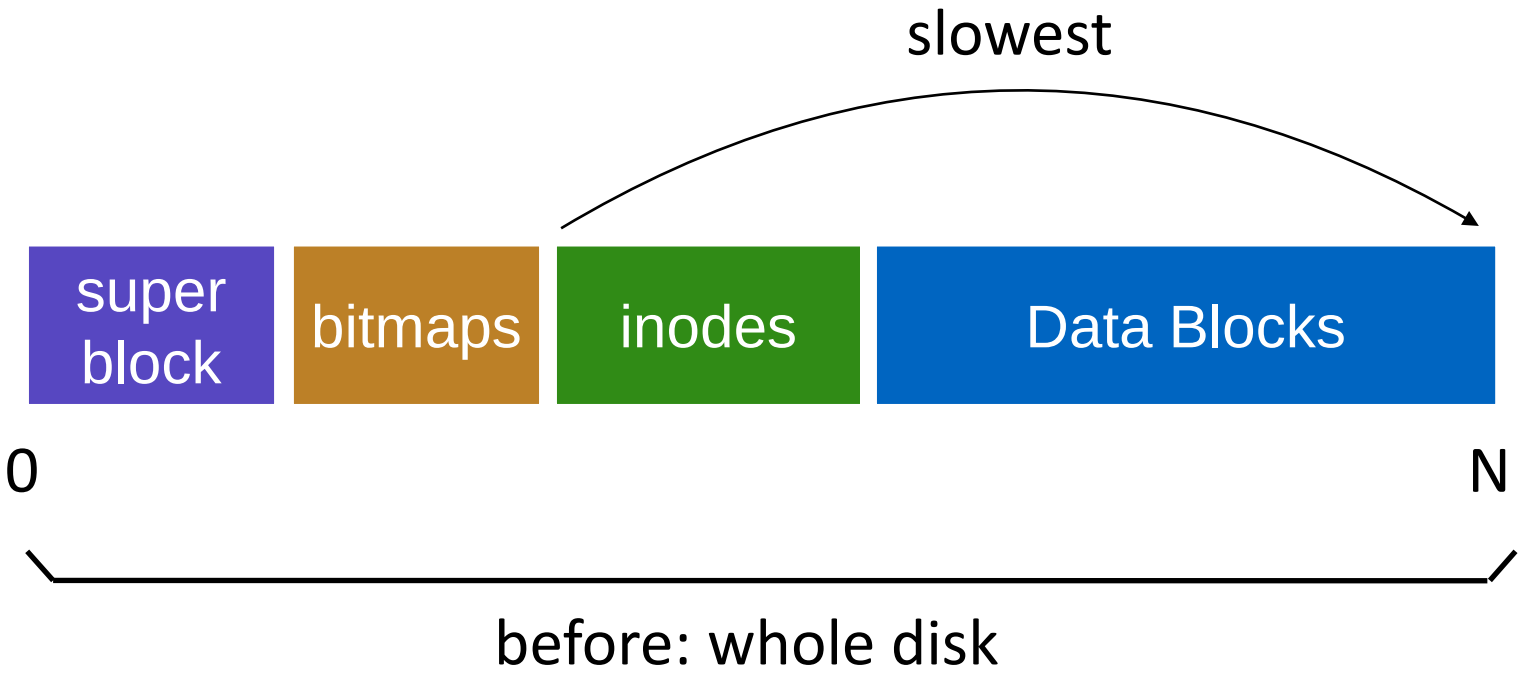
How to keep inode close to data?

Technique 2: Groups



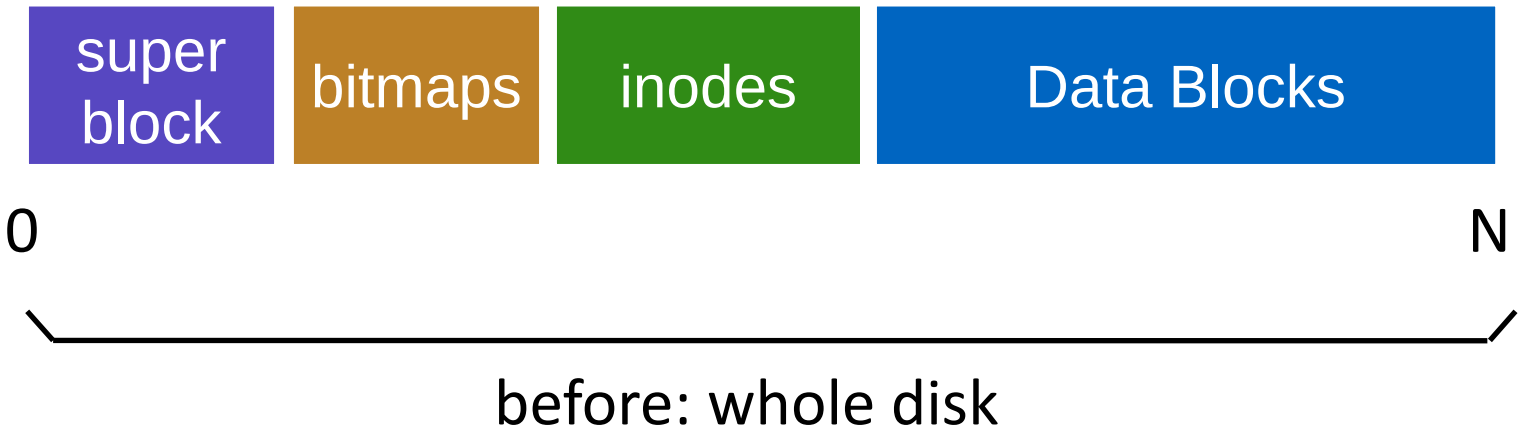
How to keep inode close to data?

Technique 2: Groups



How to keep inode close to data?

Technique 2: Groups



How to keep inode close to data?

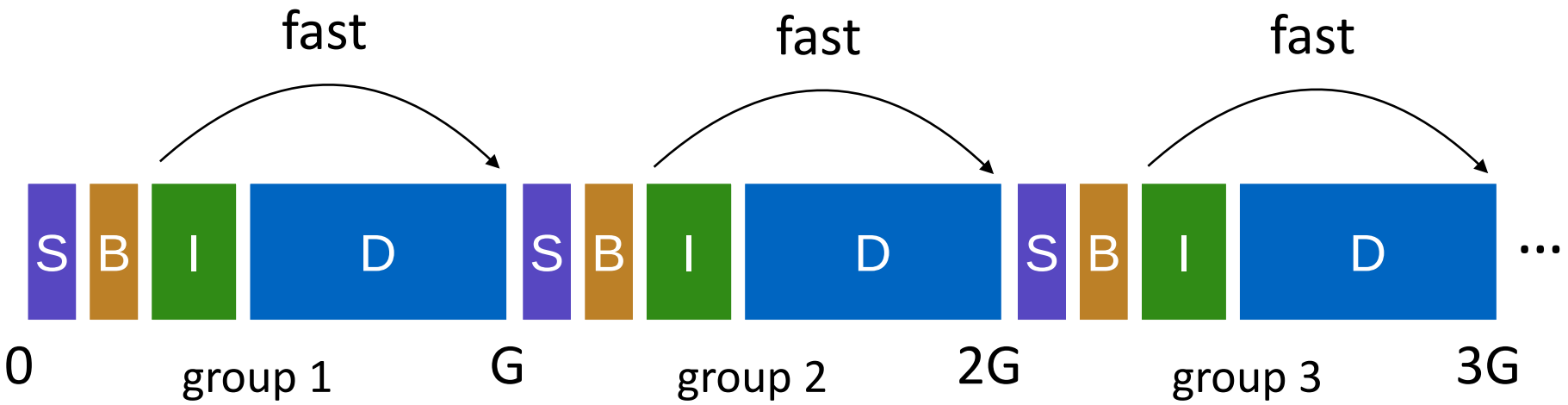
Technique 2: Groups

How to keep inode close to data?

Answer: Use groups across disks;
Try to place inode and data in same group

Minimize seek latency

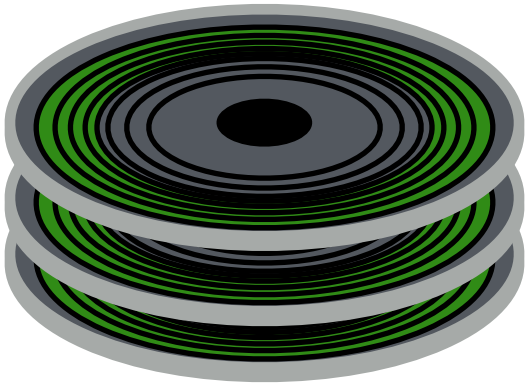
Technique 2: Groups



strategy: allocate inodes and data blocks in same group.

Groups

- **In FFS, groups were ranges of cylinders**
  - called cylinder group
- **In ext2-4, groups are ranges of blocks**
  - called block group

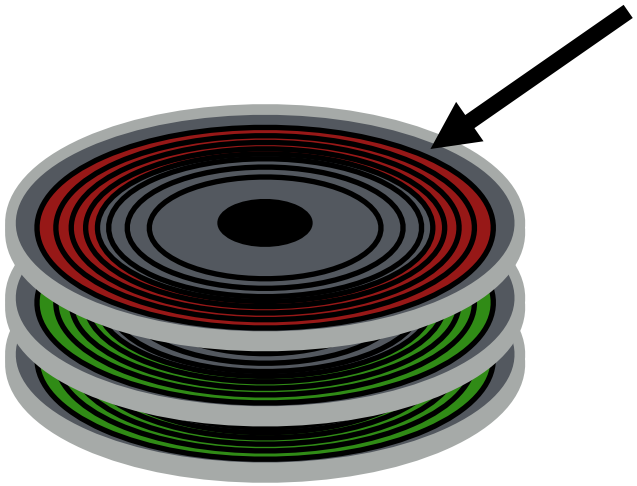


Placement Technique 3: Super Rotation

Is it useful to have multiple super blocks?

Yes, if some (but not all) fail.

Problem



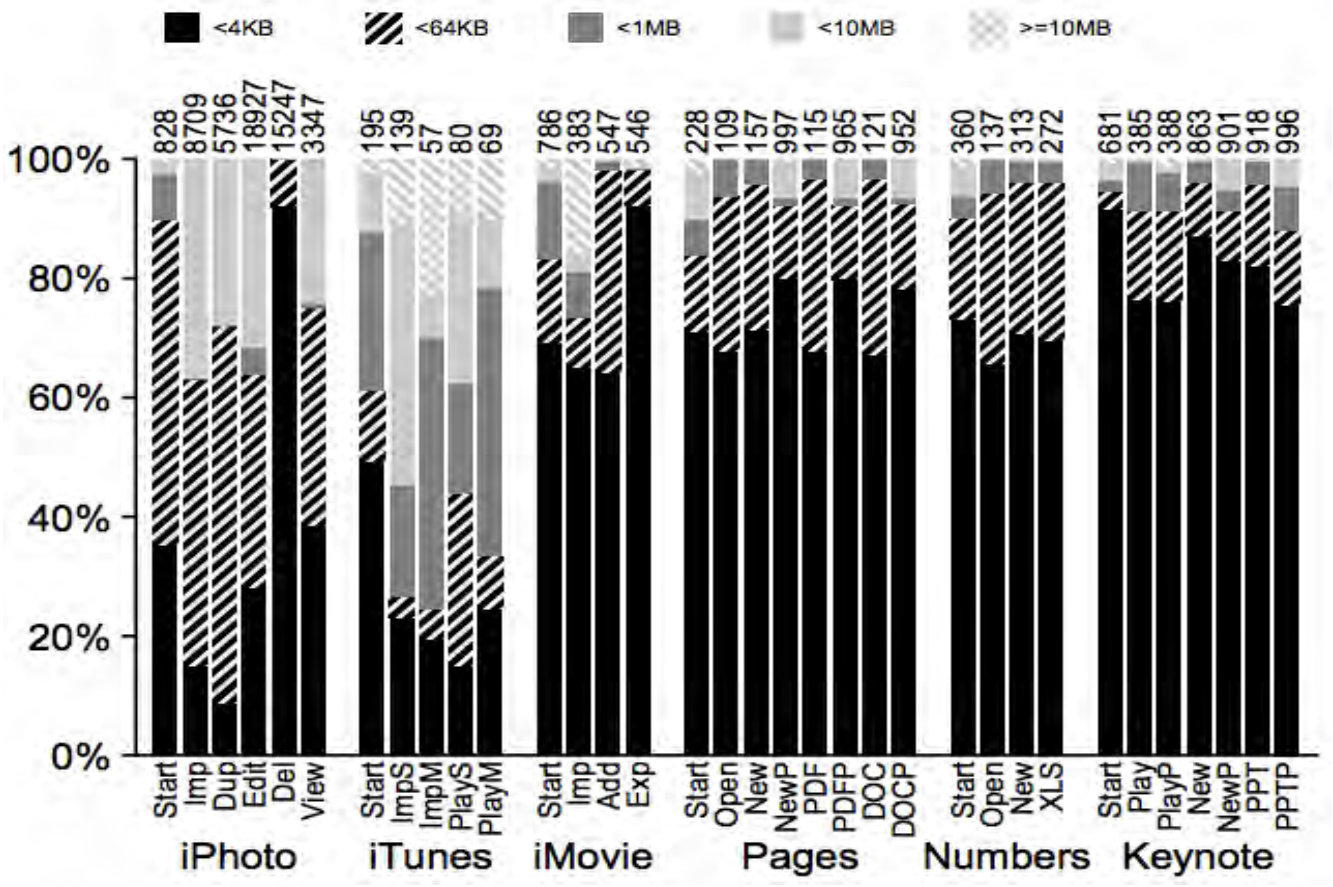
- Old FS: All super-block copies are on the top platter.
- Correlated failures! What if top platter dies?

solution: for each group, store super-block at **different offset**

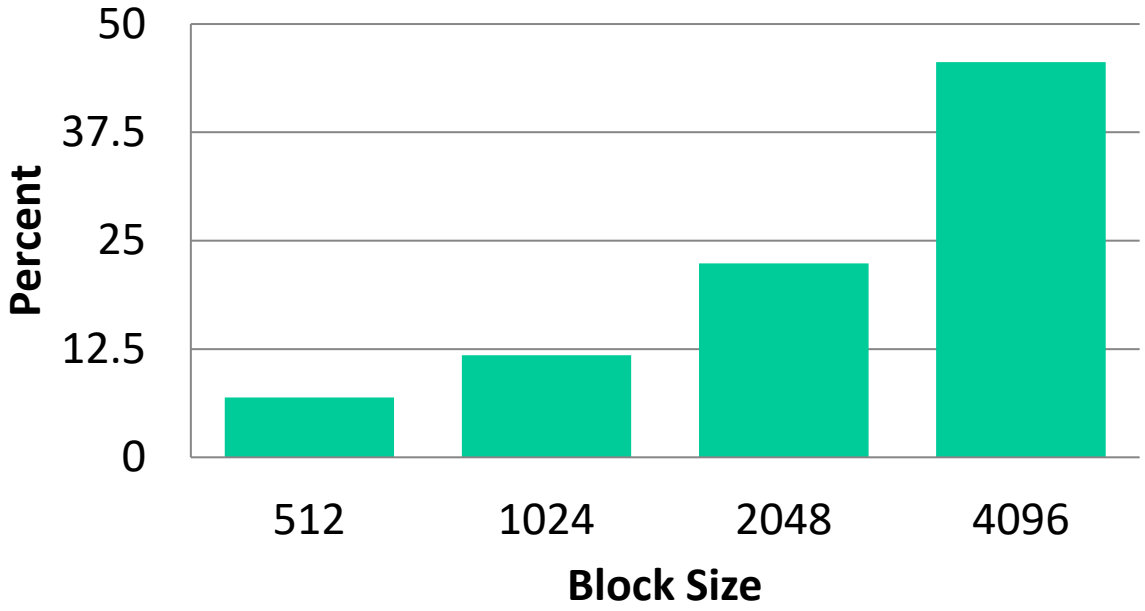
Technique: Larger Blocks

- **Observation**: Doubling block size for old FS over doubled performance
- Why not make blocks huge?

Most file are very small, even today!



Larger Blocks



- Lots of waste due to internal fragment in most blocks
- Time vs. Space tradeoffs…

Solution: Fragments

- **Hybrid – combine best of large blocks and best of small blocks**
- **Use large block when file is large enough**
- **Introduce "fragment" for files that use parts of blocks**
  - Only tail of file uses fragments

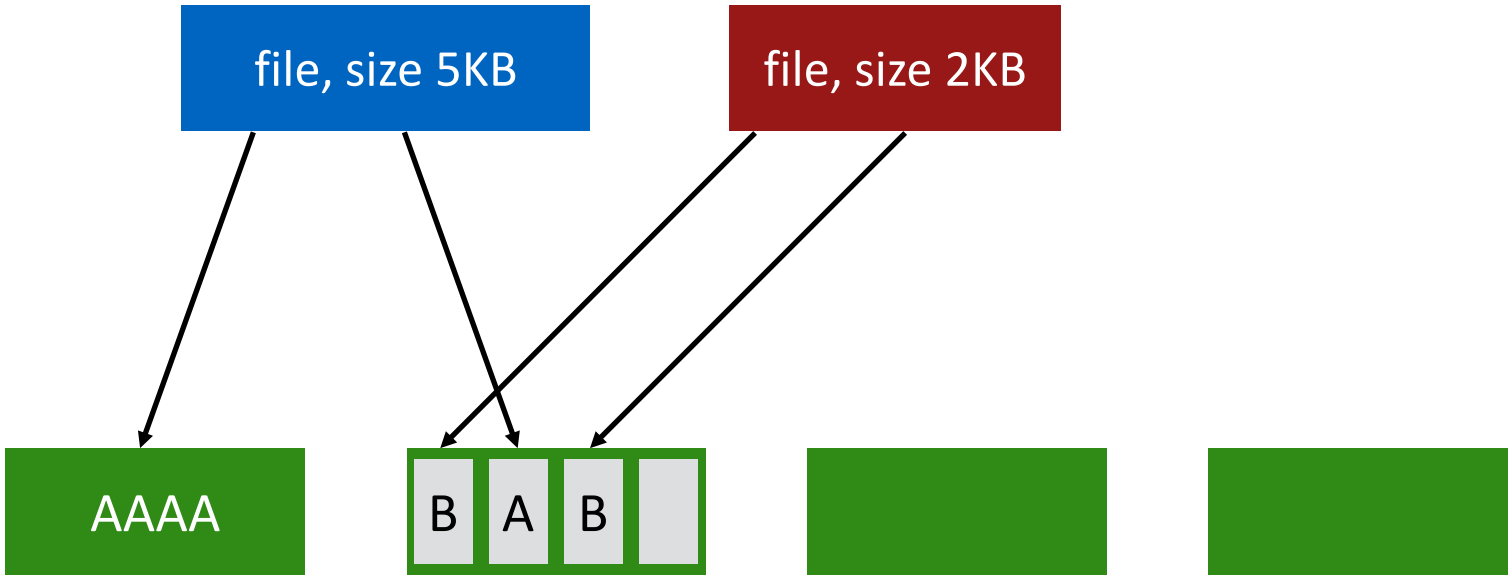
Fragment Example

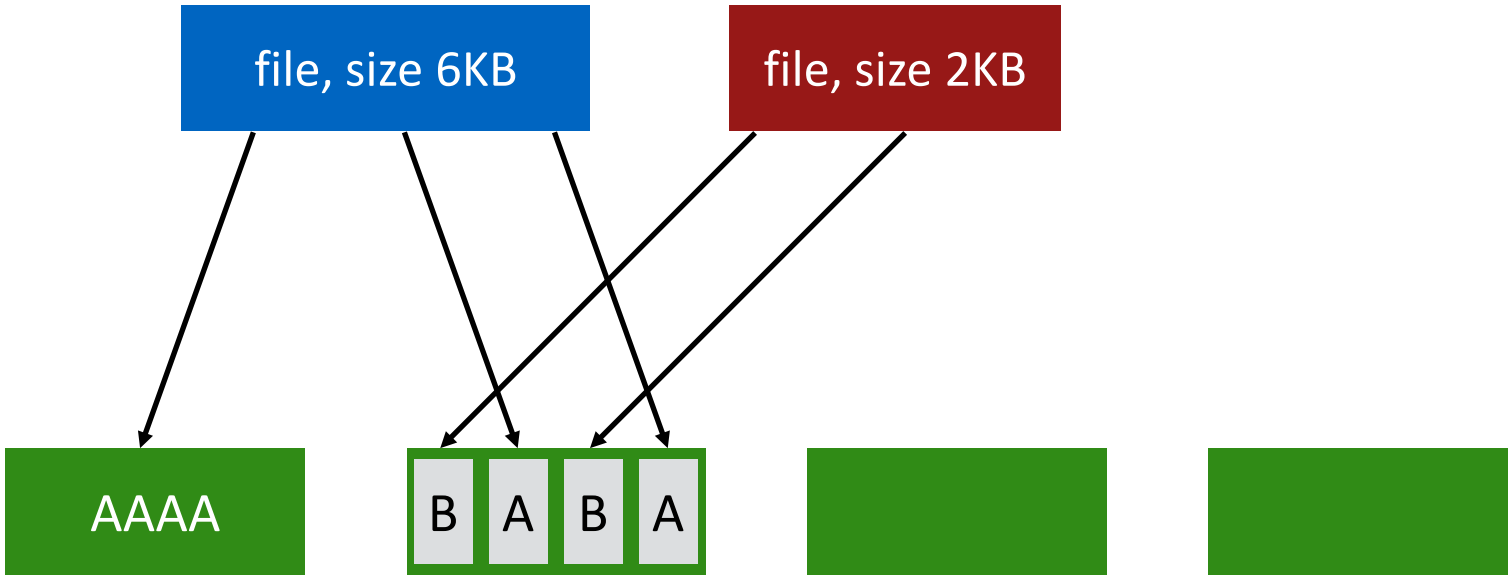
Block size = 4096

Fragment size = 1024

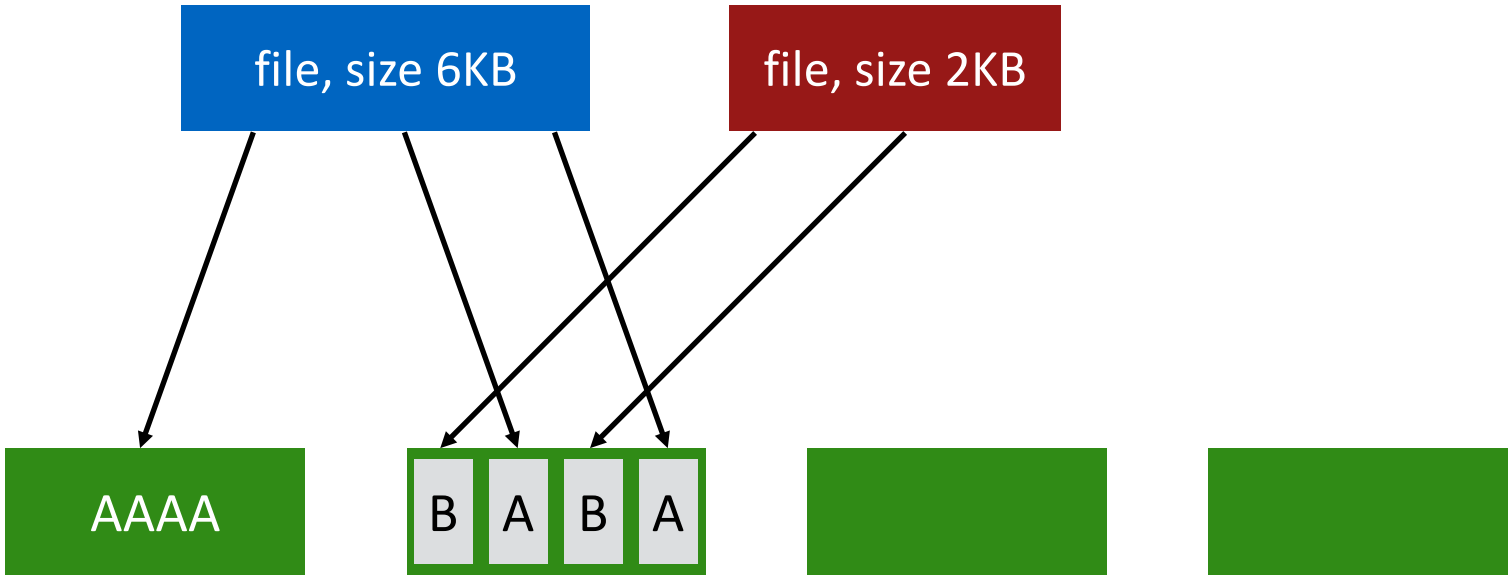
bits: 0000	0000	1111	0010
blk1	blk2	blk3	blk4

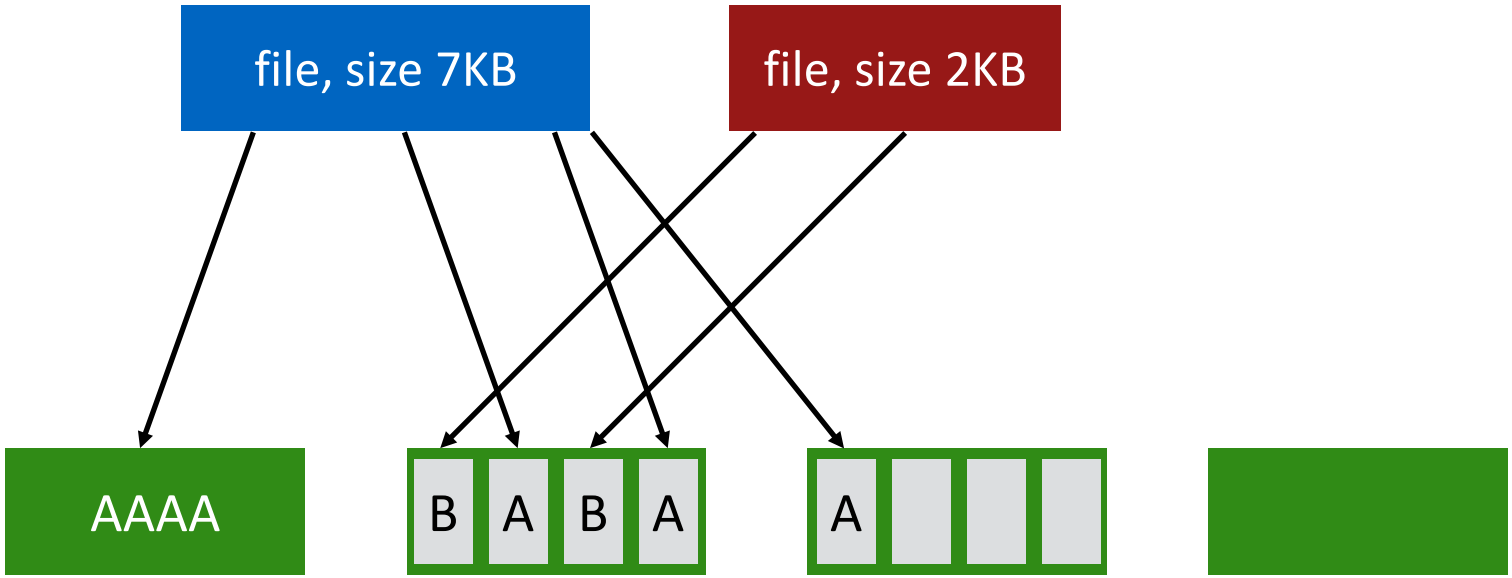
- **Whether addr refers to block or fragment is inferred by file offset**
- **What about when files grow?**
- **Must copy fragments to new block if no room to grow**


file, size 5KB
file, size 2KB
AAAA
B
A
B



append A to first file

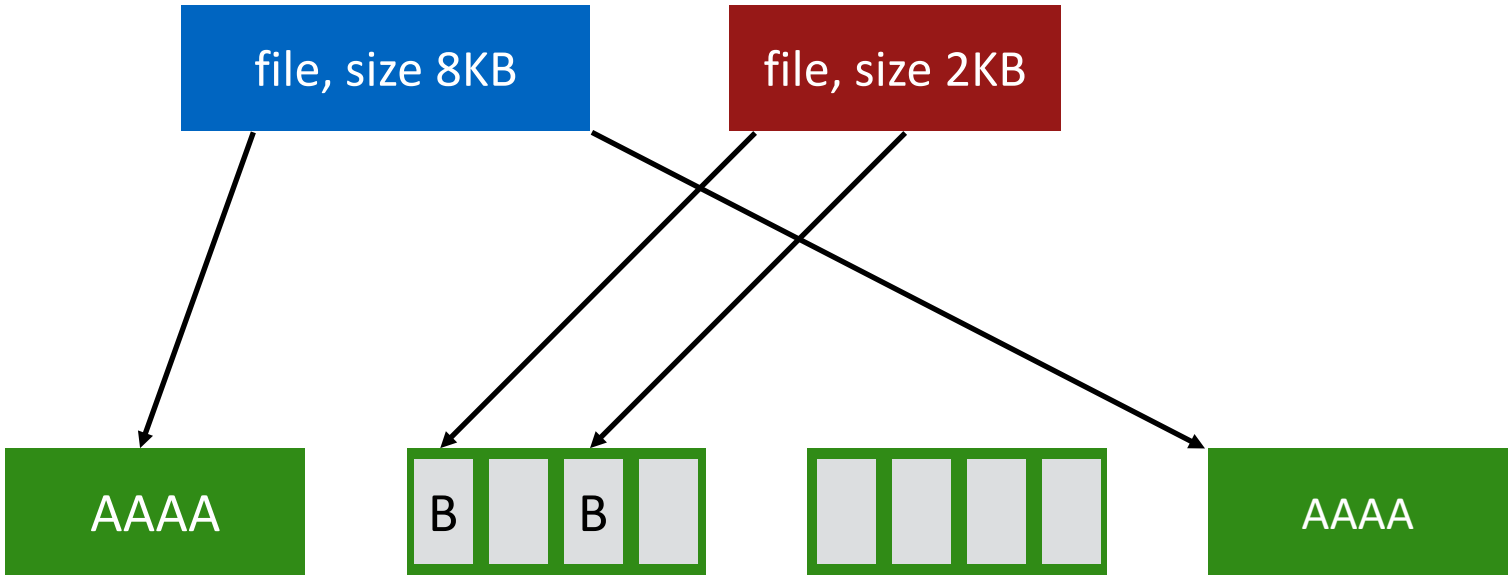

file, size 6KB
file, size 2KB
AAAA
B
A
B
A



append A to first file

Not allowed to use fragments across multiple blocks!

What to do instead?



append A to first file,
copy fragments to new block

Optimal Write Size

- Writing less than a block is inefficient
- Solution: new API exposes optimal write size

Smart Policy

Where should new inodes and data blocks go?

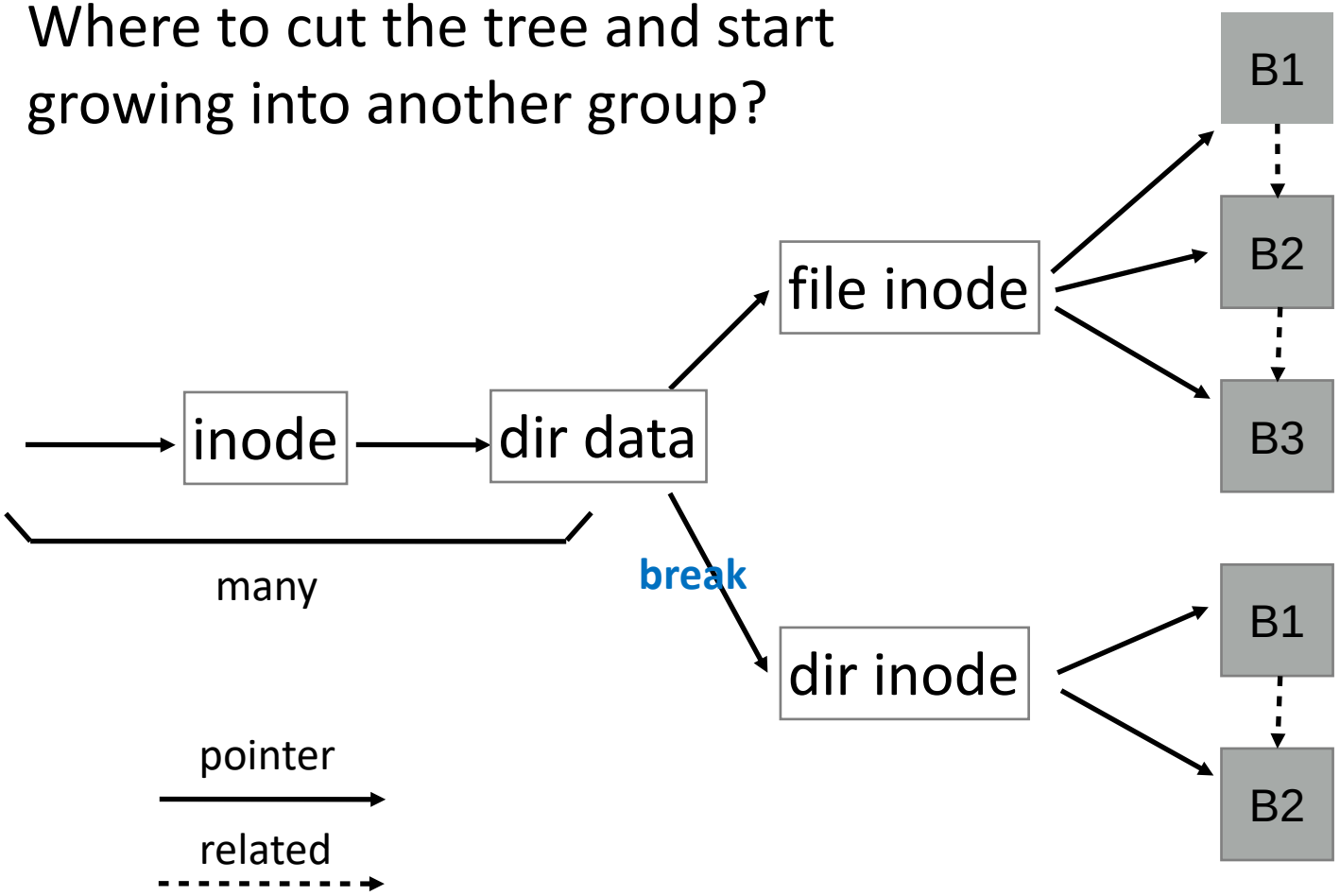
Strategy

- **Put related pieces of data near each other**
- **Rules:**
  - 1. Put directory entries near directory inodes.
  - 2. Put inodes near directory entries.
  - 3. Put data blocks near inodes.
- **Sound good?**
- **Problem: File system is one big tree**
  - All directories and files have a common root.
  - All data in same FS is related in some way
- **Trying to put everything near everything else doesn't make any sense!**

Revised Strategy

- Put more-related pieces of data near each other
- Put less-related pieces of data far from each other
- FFS developers used their best judgement

Where to cut the tree and start growing into another group?



- **FFS puts dir inodes in a new group**
- **"`ls`" is fast on directories with many files.**

Preferences

- File inodes: allocate in same group with dir
- Dir inodes: allocate in new group with fewer used inodes than average group
- First data block: allocate near inode
- Other data blocks: allocate near previous block

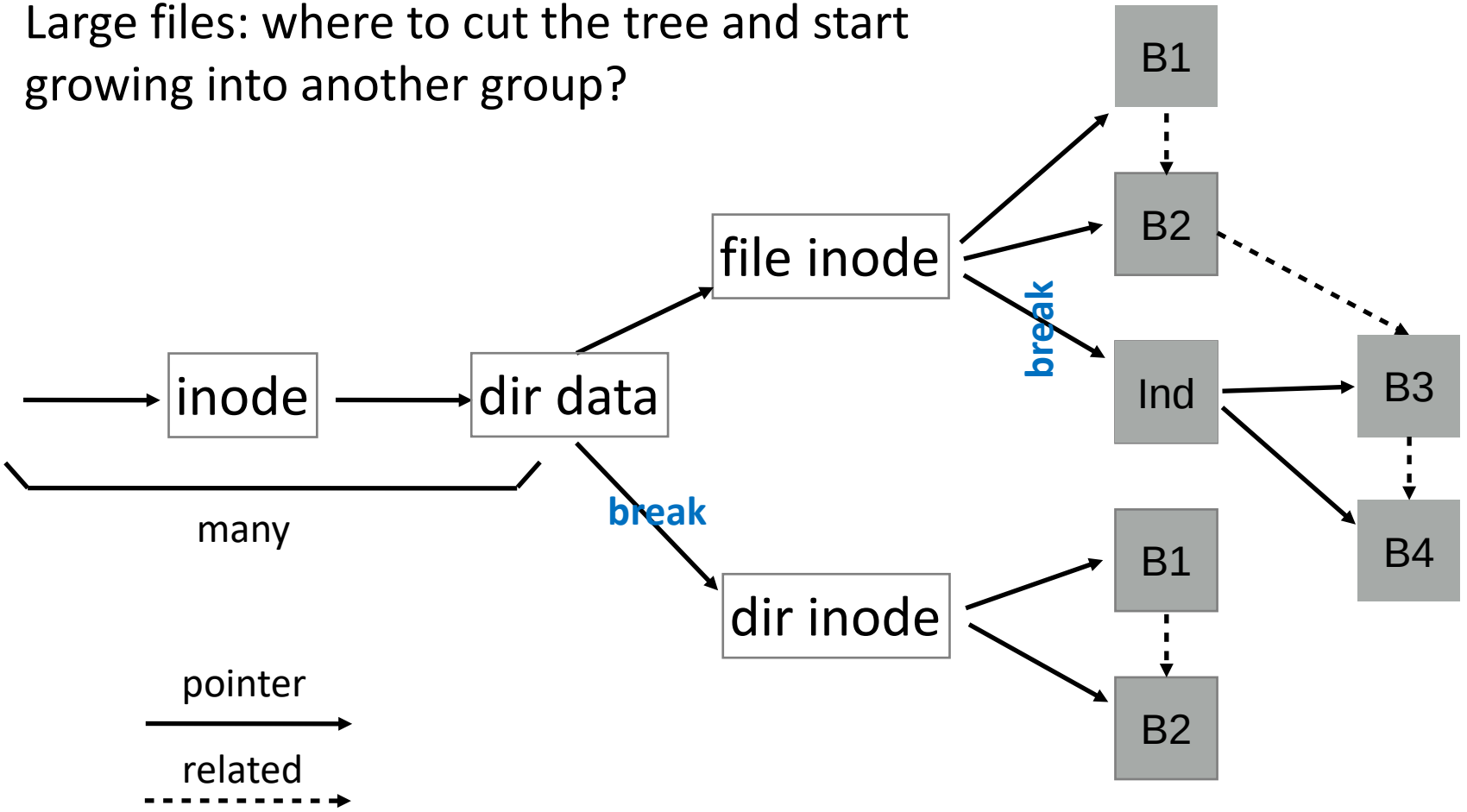
Problem: Large Files

- **Single large file can fill nearly all of a group**

- **Undesirable:**
  - Displaces data for many small files
  - Prevents subsequent related files from being placed within this block group
  - Hurt file-access locality

- **Solution?**
  - It is OK to have multiple seeks in reading a large file

Large files: where to cut the tree and start growing into another group?



- Define “large” as requiring an indirect block
- Starting at indirect (e.g., after 48 KB) put blocks in a new block group.

Preferences

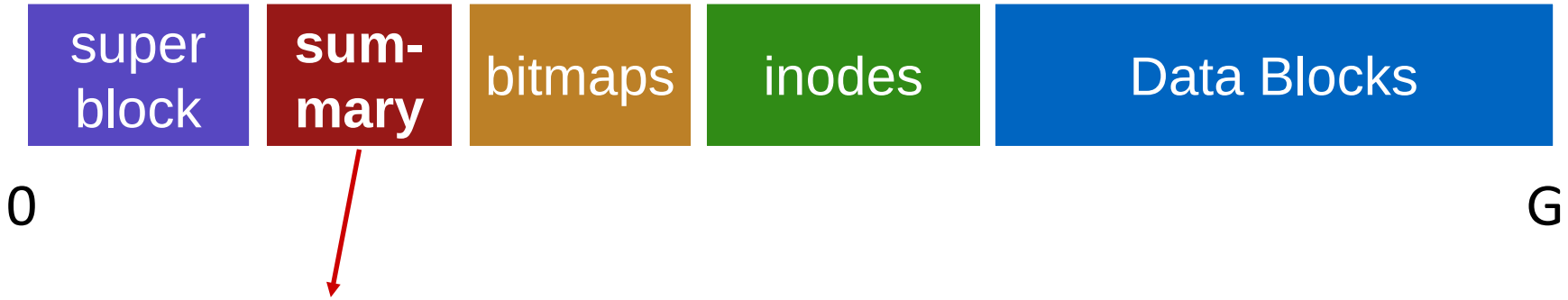
- File inodes: allocate in same group with dir
- Dir inodes: allocate in new group with fewer used inodes than average group
- First data block: allocate near inode
- Other data blocks: allocate near previous block
- Large file data blocks: after 48KB, go to new group. Move to another group (w/ fewer than avg blocks) every subsequent 1MB.

Group Descriptor (aka Summary Block)

- How does file system know which new group to pick?

Group Descriptor (aka Summary Block)

- How does file system know which new group to pick?



Tracks number of free inodes and data blocks in this group

Conclusion

- **First disk-aware file system**
  - Bitmaps
  - Locality groups
  - Rotated superblocks
  - Large blocks
  - Fragments
  - 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

Advice

- All hardware is unique
- Treat disk like disk!
- Treat flash like flash!
- Treat random-access memory like random-access memory!