[537] Fast File System

Chapter 41
Tyler Harter
11/10/14

File-System Case Studies

Local

- **FFS**: Fast File System
- **LFS**: Log-Structured File System

Network

- **NFS**: Network File System
- **AFS**: Andrew File System

File-System Case Studies

Local

- **FFS**: Fast File System [today]
- **LFS**: Log-Structured File System

Network

- **NFS**: Network File System
- **AFS**: Andrew File System

Review Basic FS

Basic FS

Structures (on disk)

Operations

Structure Overview

Core

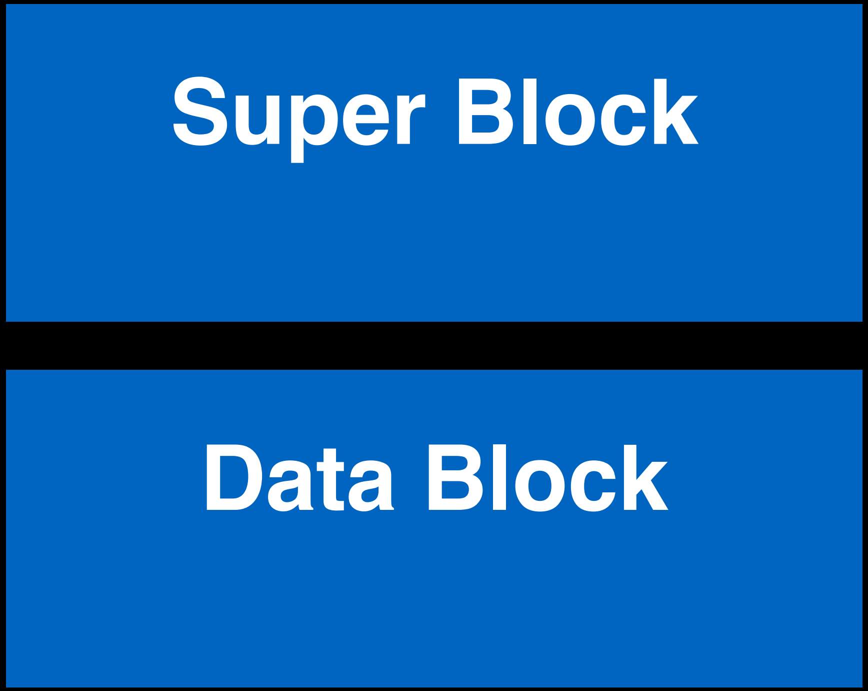
Performance

Super Block

Structure Overview

Core

Performance



Super Block

Data Block

Structure Overview

Core

Super Block

Data Block

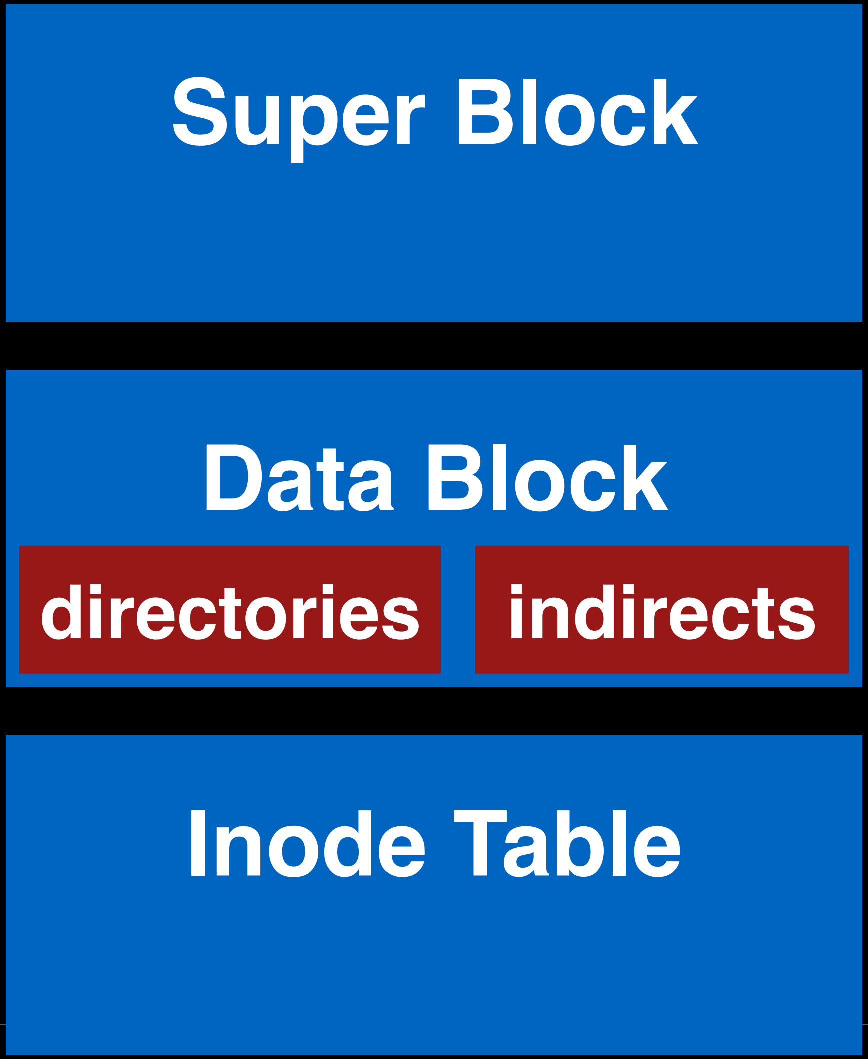
Inode Table

Performance

Structure Overview

Core

Performance



Super Block

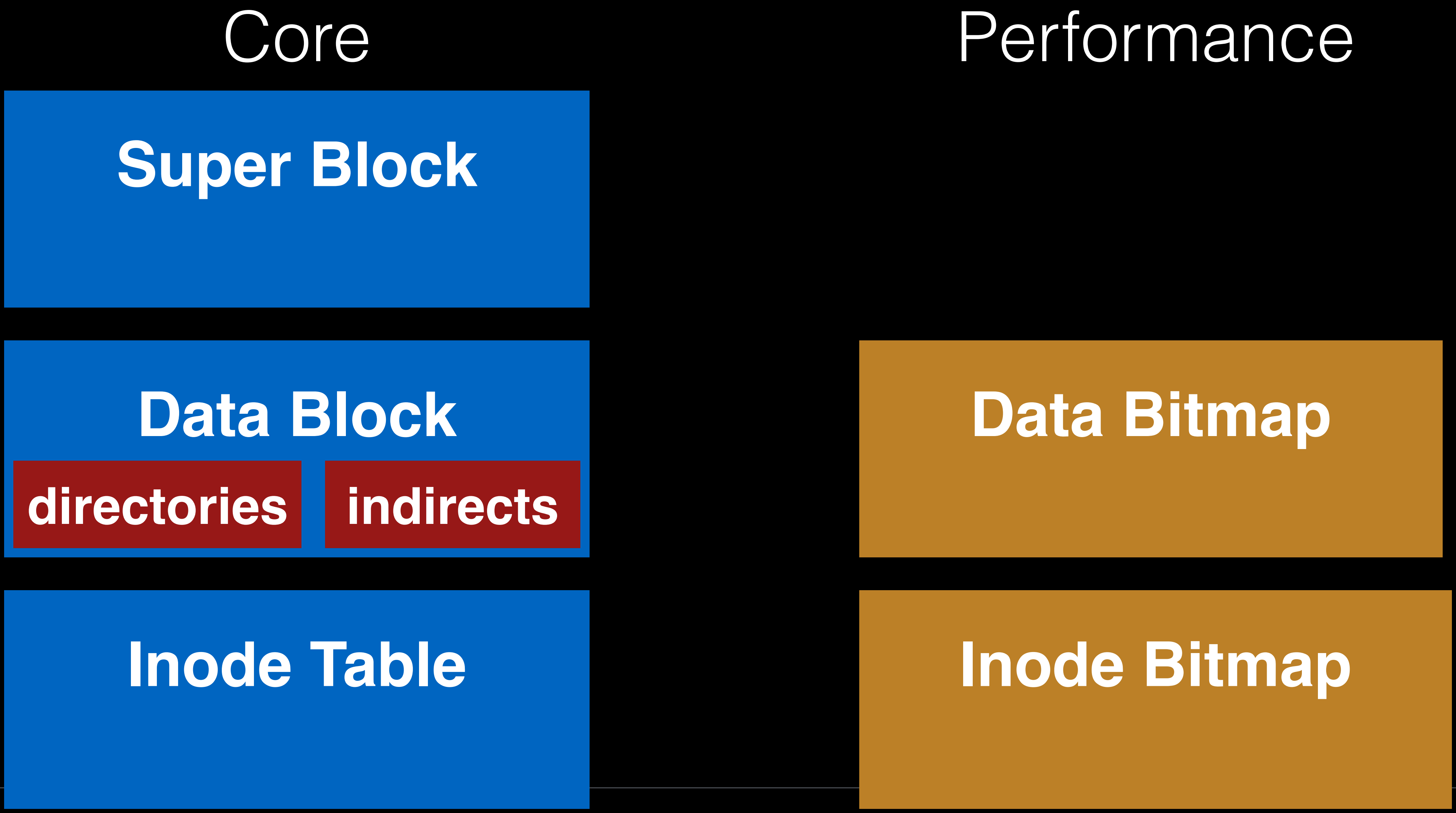
Data Block

directories

indirects

Inode Table

Structure Overview



Core

Performance

Super Block

Data Block

directories

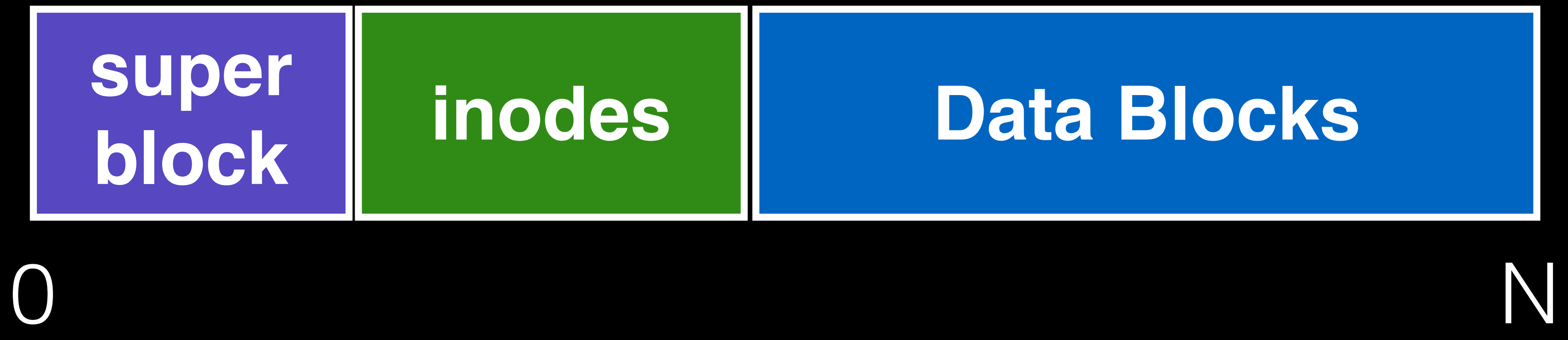
indirects

Inode Table

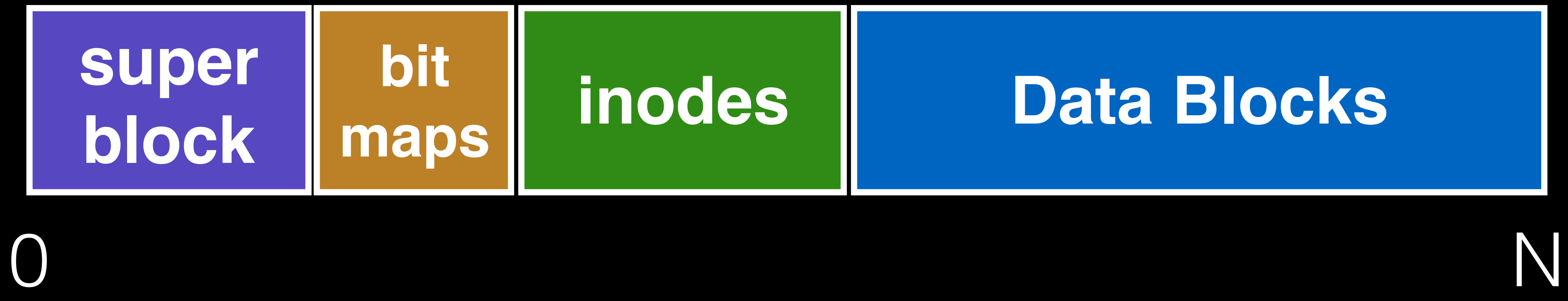
Data Bitmap

Inode Bitmap

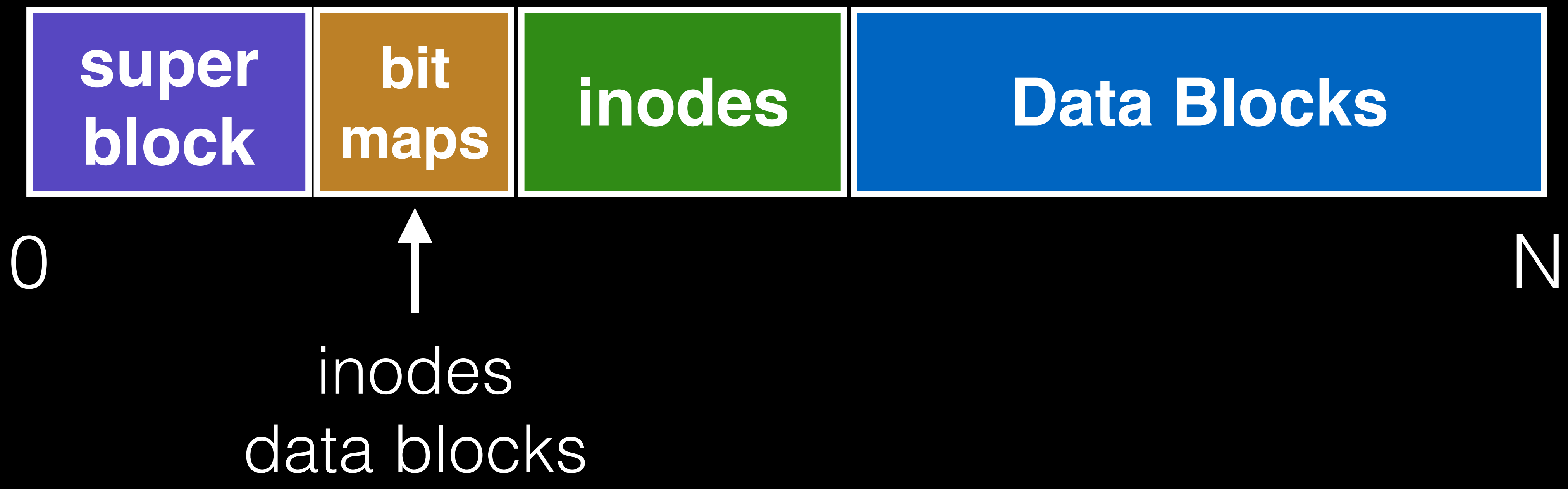
Layout



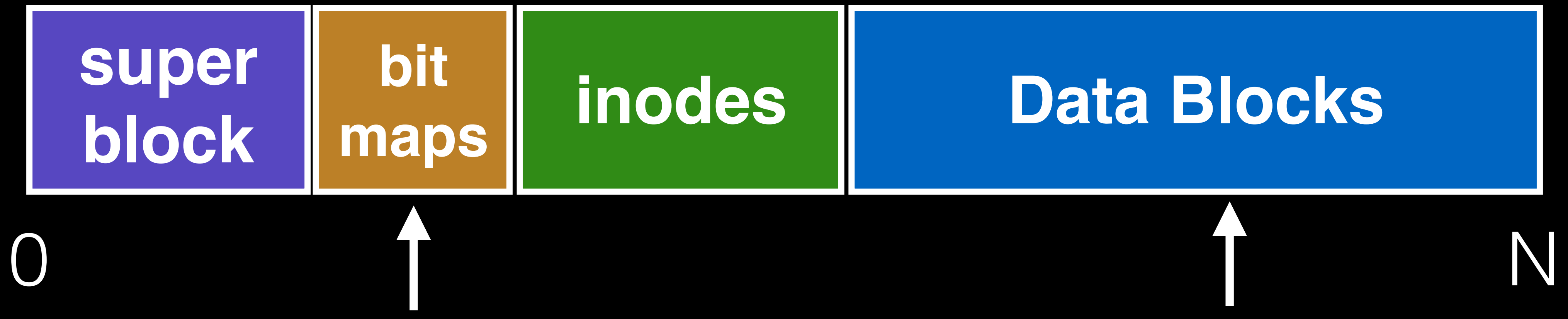
Layout



Layout


super block
bit maps
inodes
Data Blocks
0
N
inodes
data blocks

Layout



inodes
data blocks

regular data
directories
indirect blocks

Basic FS

Structures (on disk)

Operations

create /foo/bar

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

[traverse]

create /foo/bar

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

[traverse]

create /foo/bar

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

[traverse]

create /foo/bar

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

bar does not already exist

create /foo/bar

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

[allocate inode]

create /foo/bar

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

[populate inode]

create /foo/bar

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

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

append to /foo/bar

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

[append? yes]

append to /foo/bar

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

[allocate block]

append to /foo/bar

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			

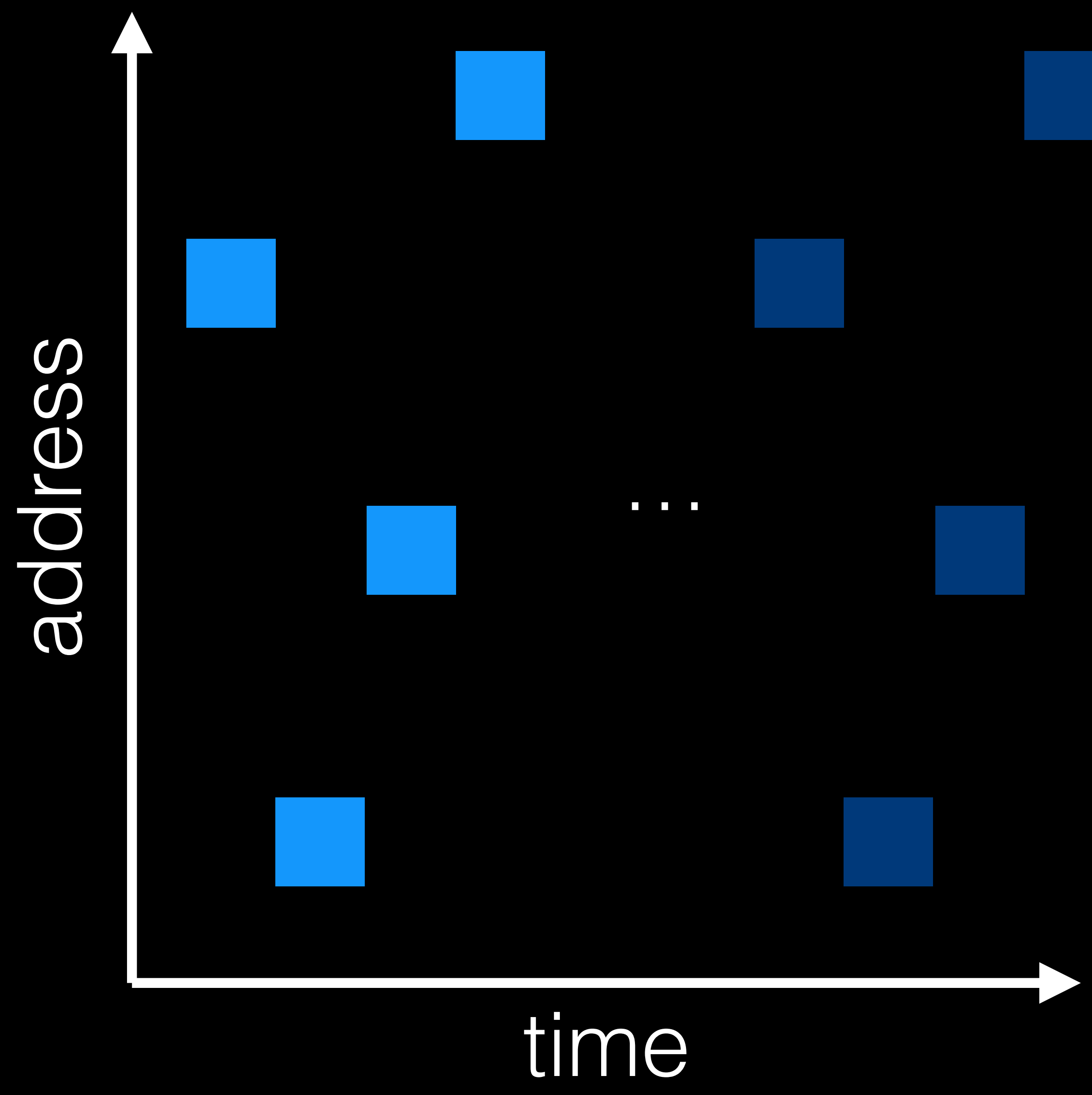
append to /foo/bar

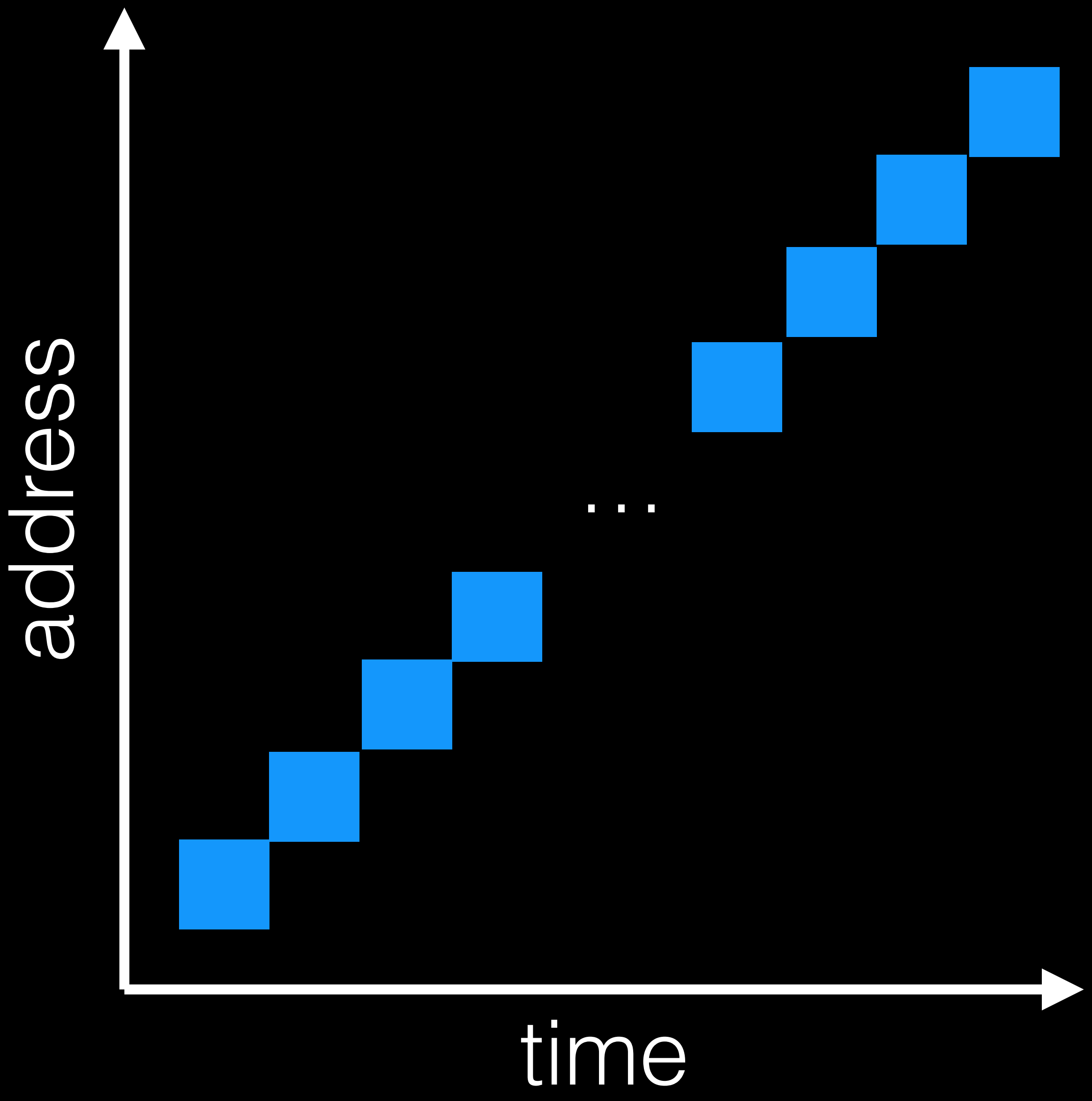
[write to block]

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

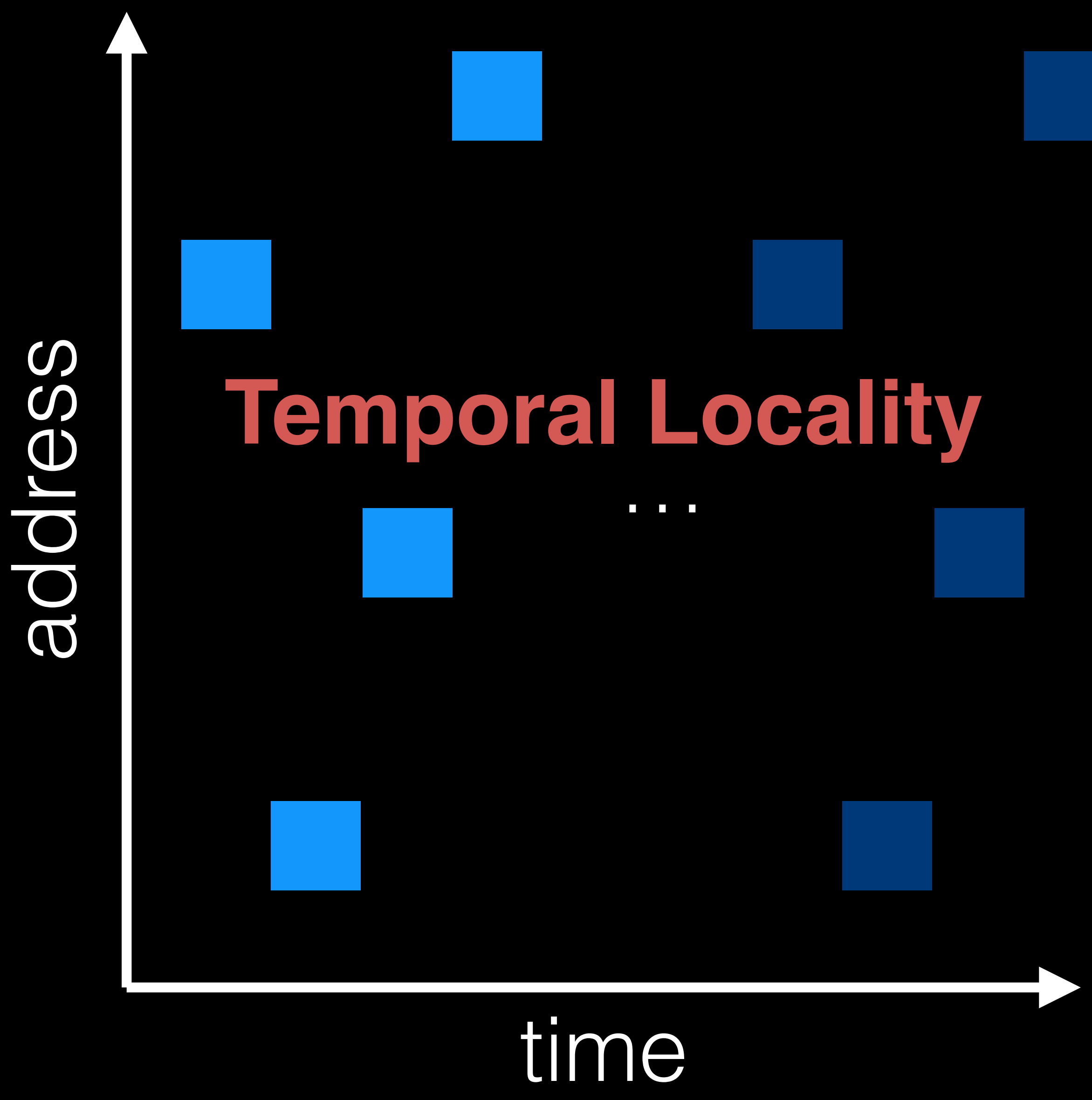
Review Locality

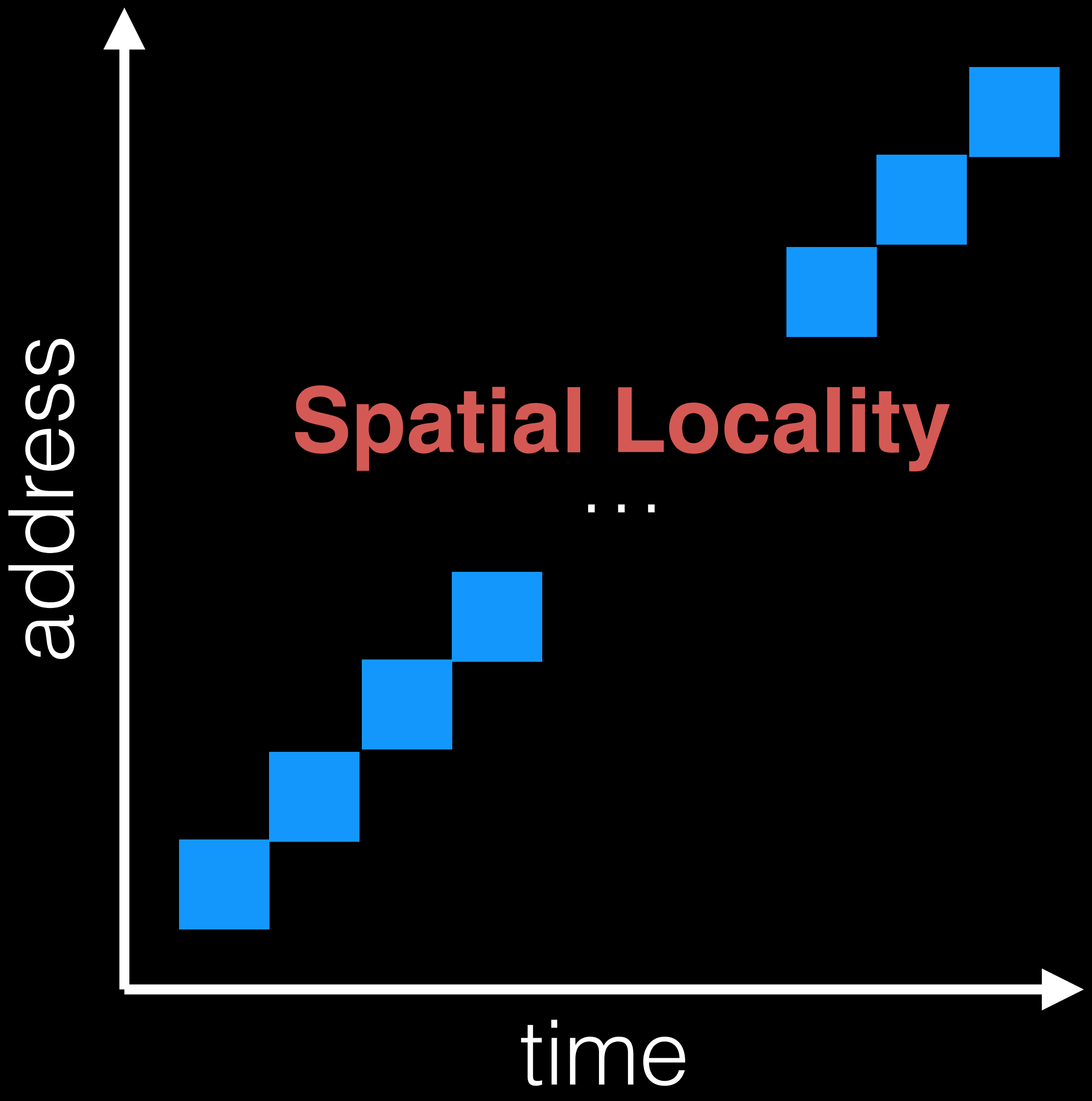
Locality Types





Locality Types



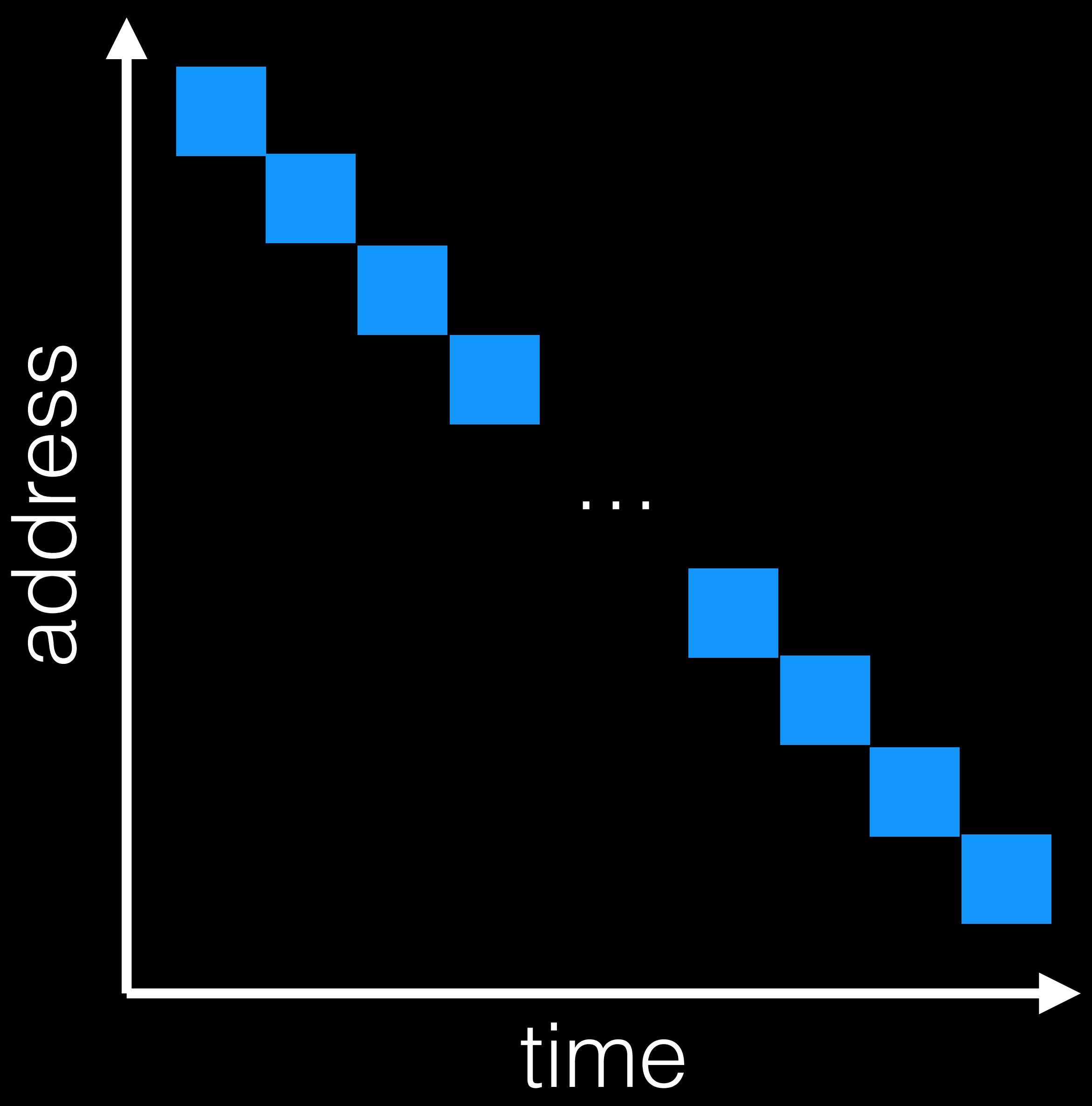


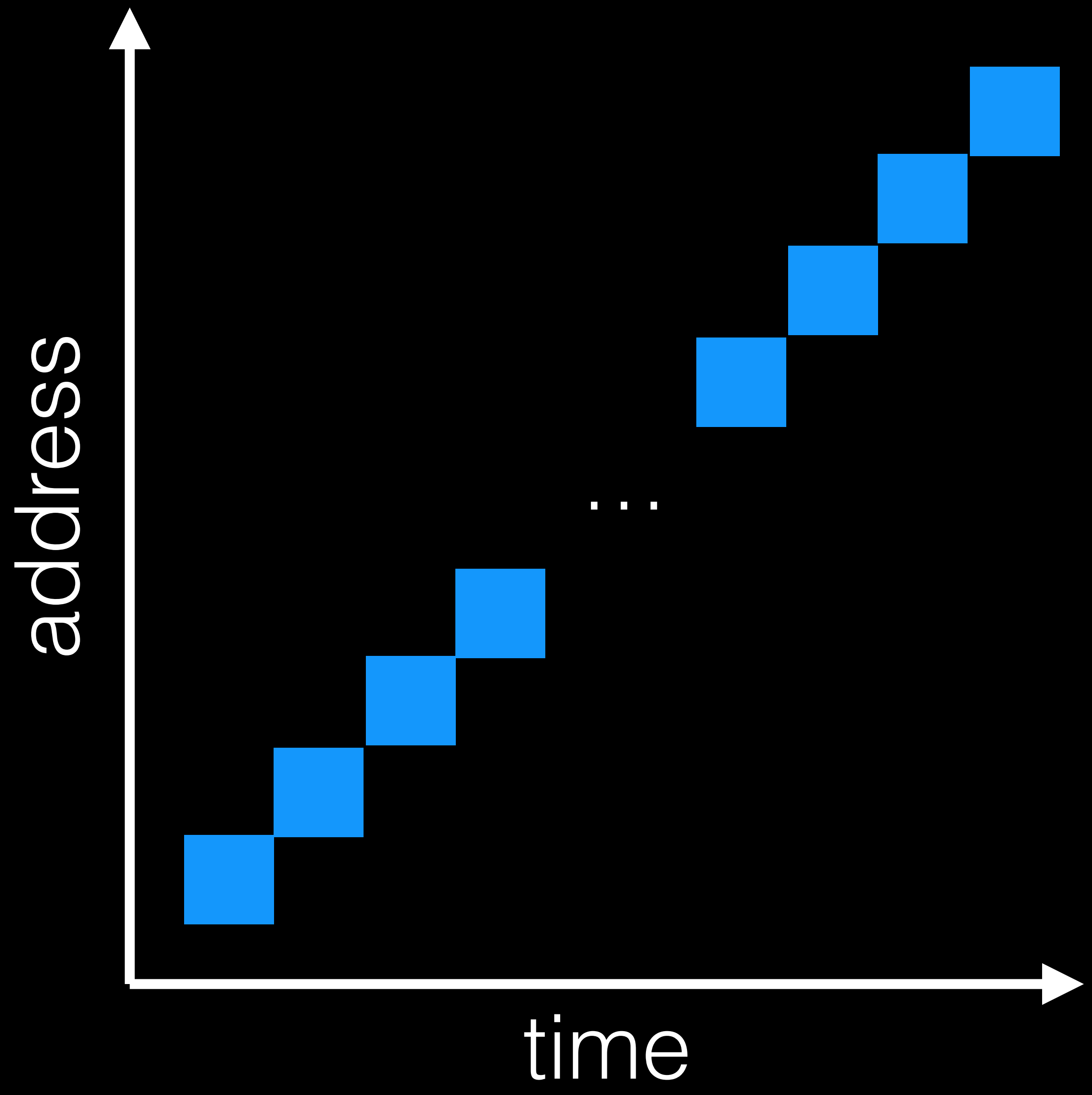
Locality Usefulness

What types of locality are useful for a cache?

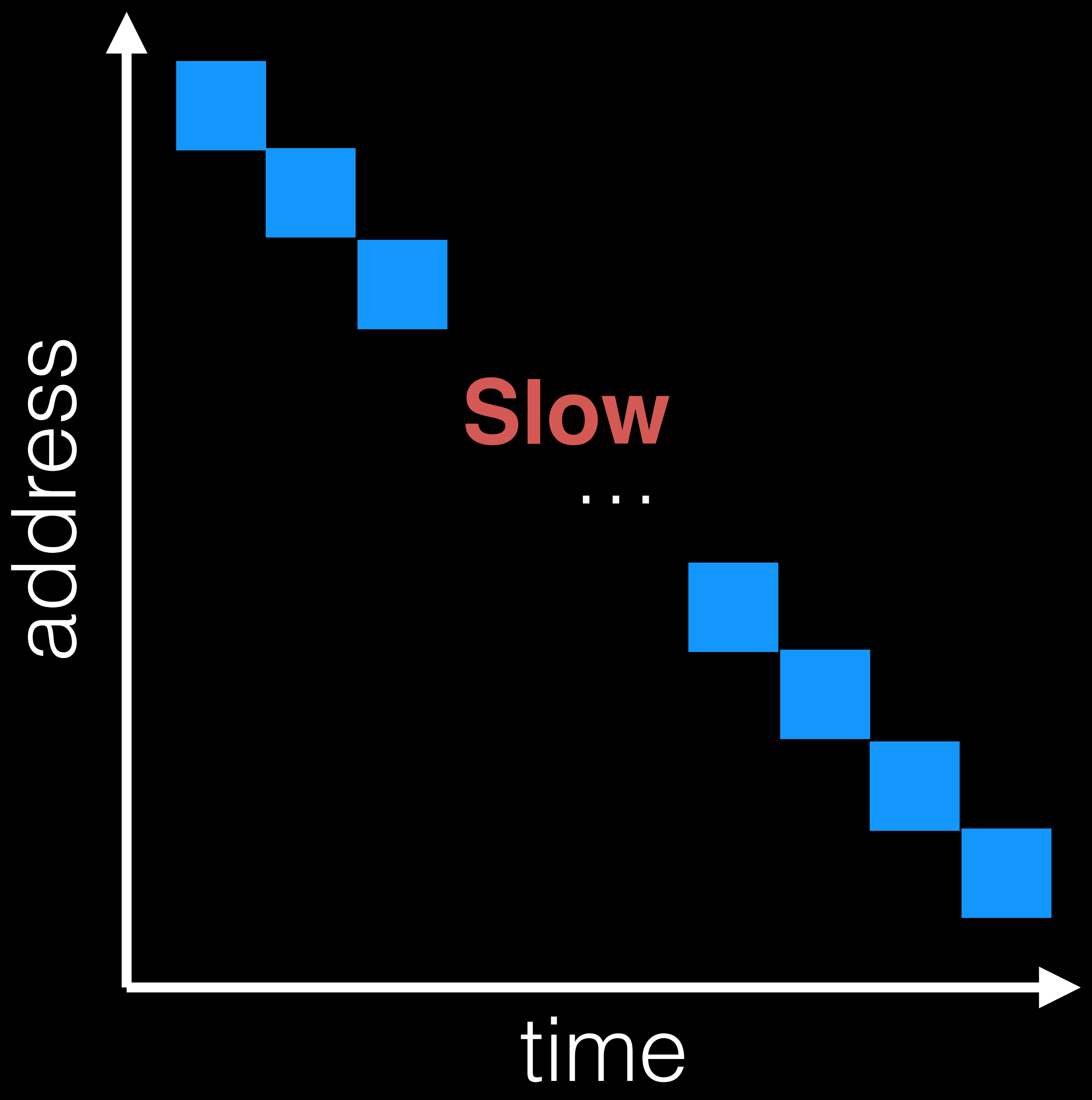
What types of locality are useful for a disk?

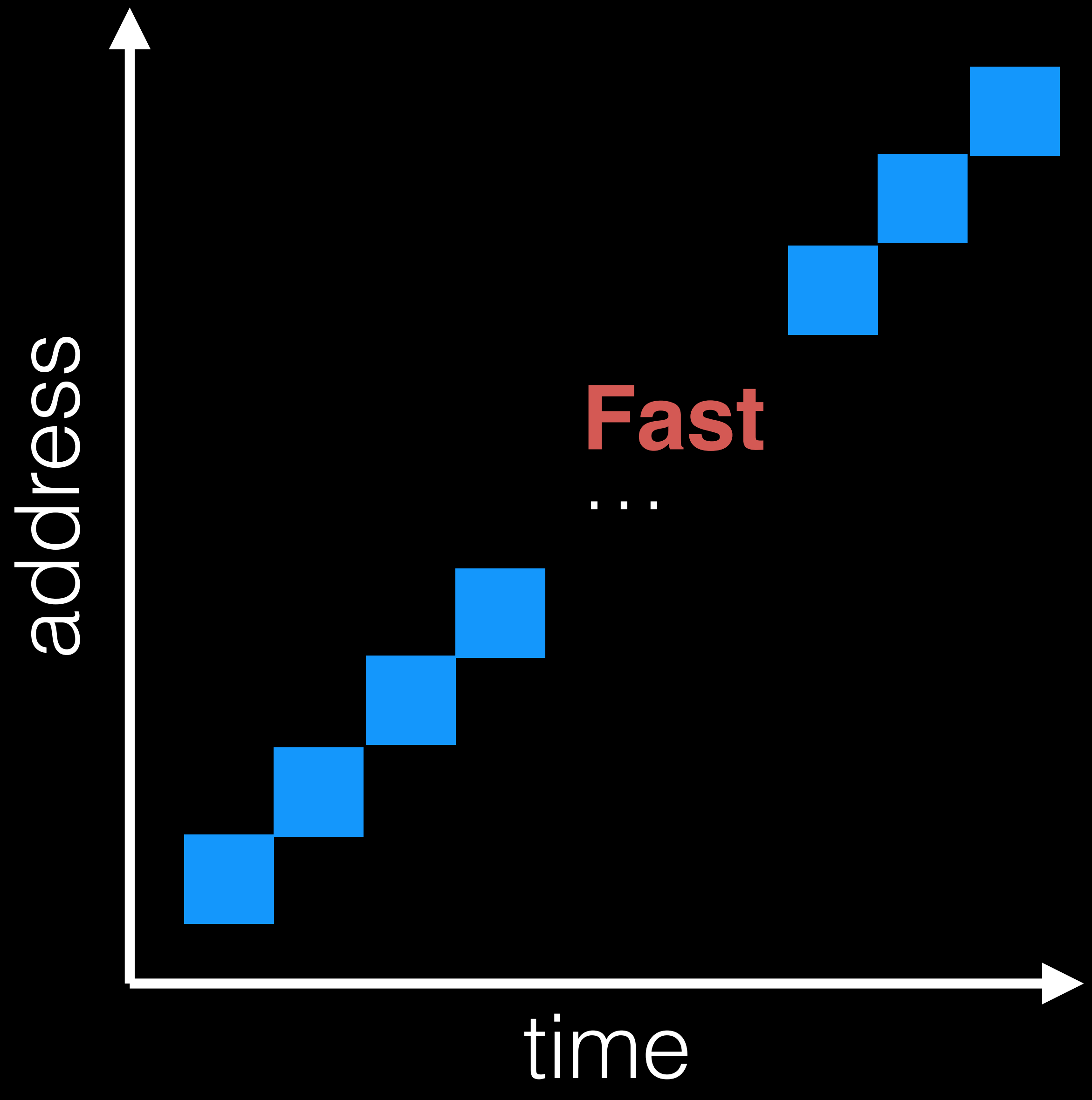
Order Matters Now



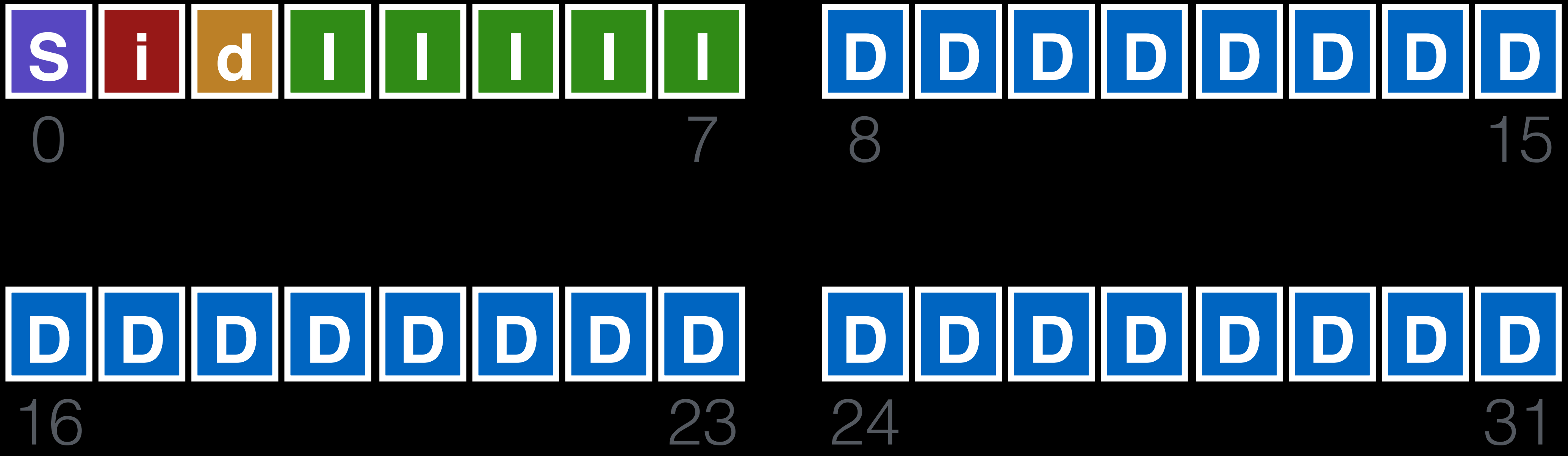


Order Matters Now

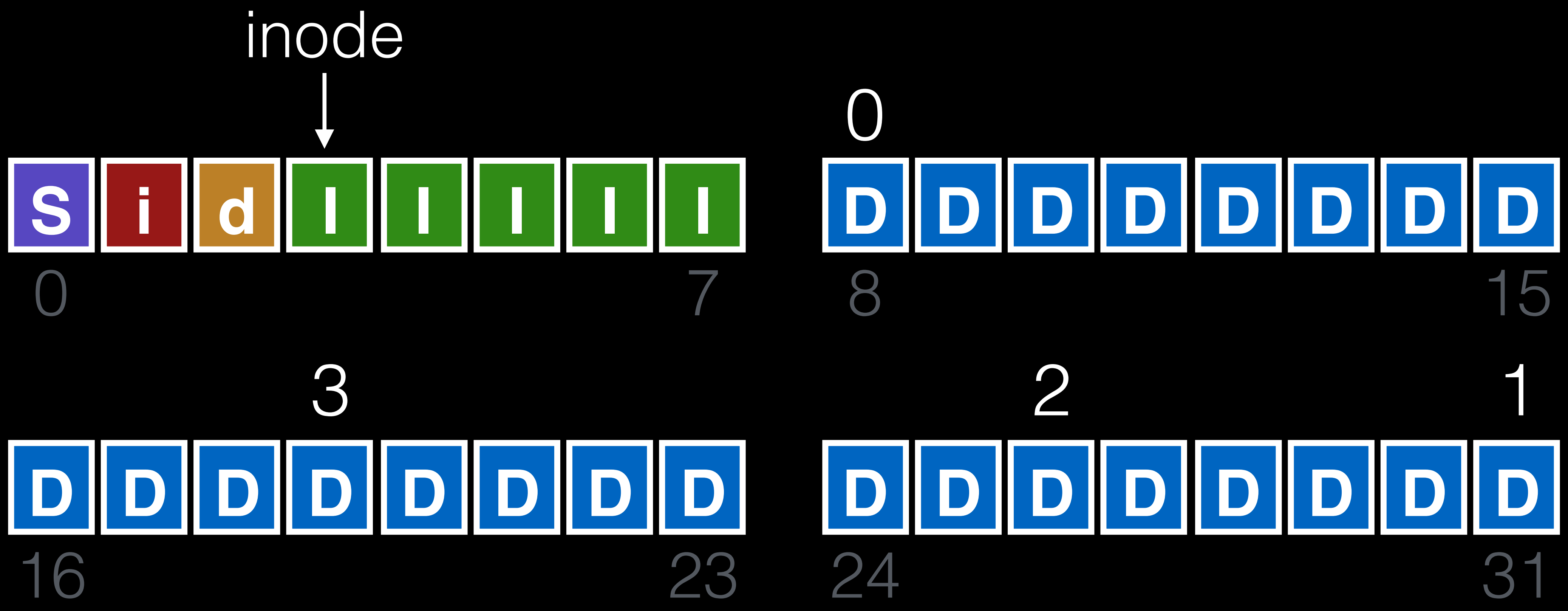




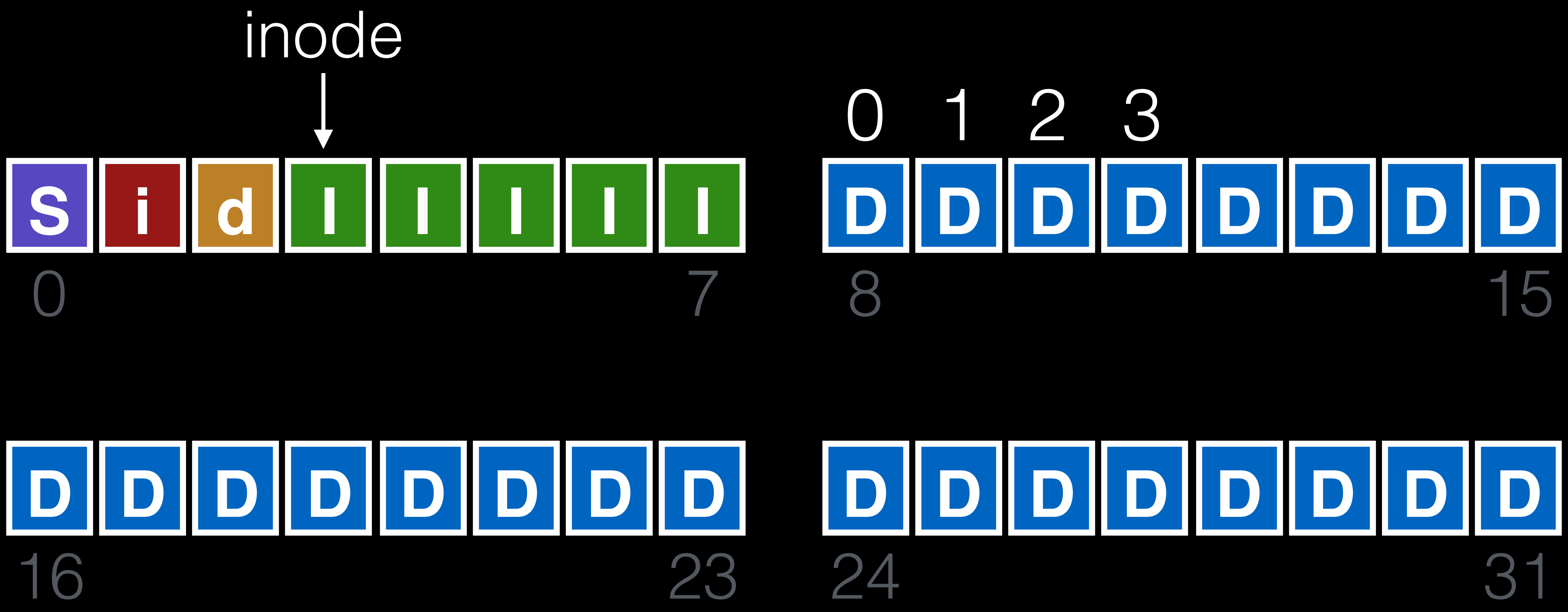
Policy: Choose Inode, Data Blocks



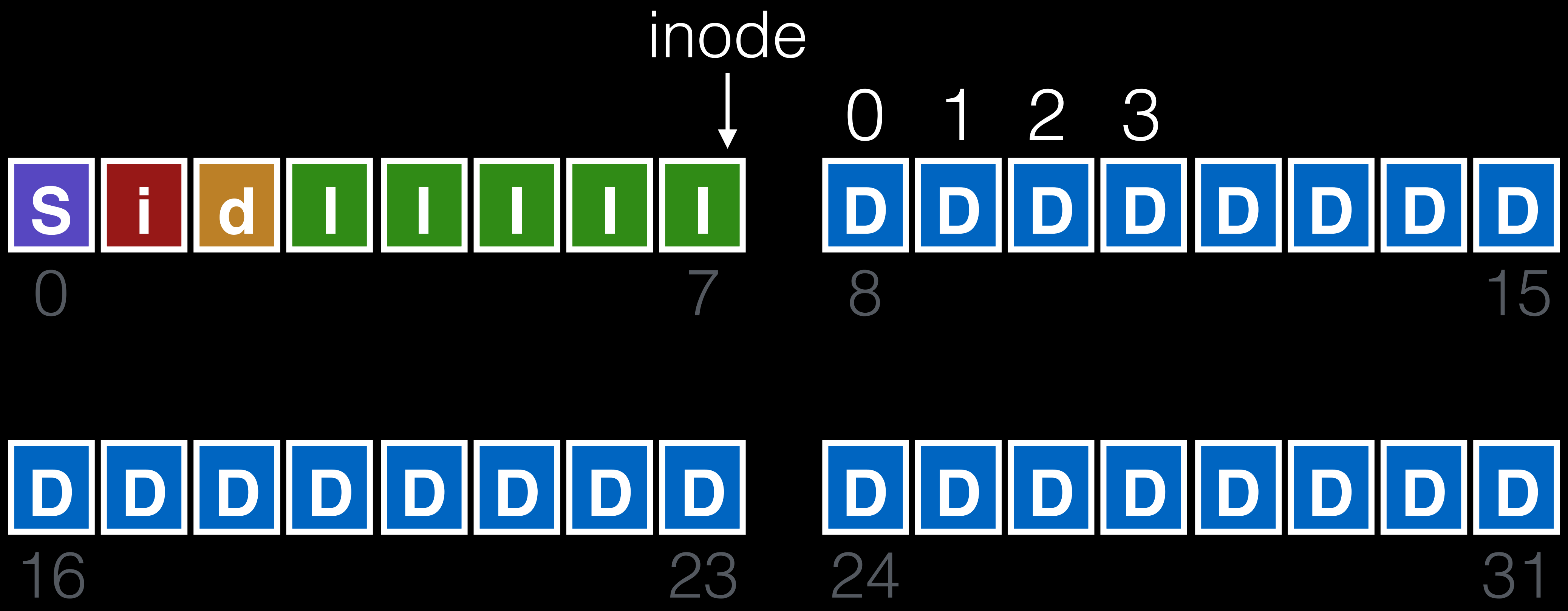
Bad File Layout



Better File Layout



Best File Layout



inode

0 1 2 3

S i d I I I I I D D D D D D D D

0 7 8 15

D D D D D D D D D D D D D D D D

16 23 24 31

Fast File System

System Building

noob approach

1. get idea
2. build it!

System Building

noob approach

1. get idea
2. build it!

pro approach

1. identify state of the art
2. measure it, identify problems
3. get idea
4. build it!

System Building

noob approach

1. get idea
2. build it!

pro approach

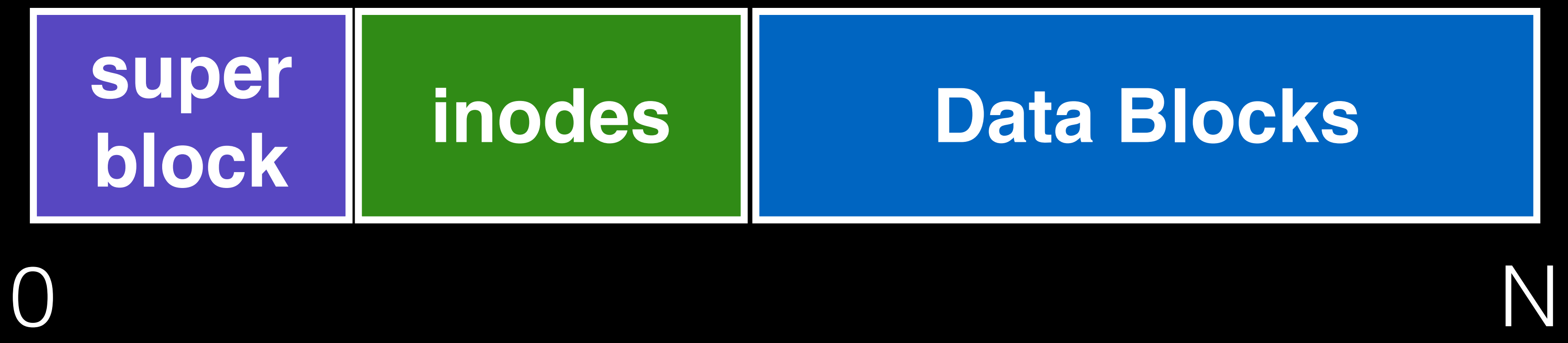
1. identify state of the art
2. measure it, identify problems
3. get idea
4. build it!

measure then build

Old FS

State of the art: original UNIX file system.

Layout



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

Old FS

State of the art: original UNIX file system.

Old FS

State of the art: original UNIX file system.

Measure throughput for file reads/writes.

Compare to theoretical max, which is…

Old FS

State of the art: original UNIX file system.

Measure throughput for file reads/writes.

Compare to theoretical max, which is…
disk bandwidth

Old FS

State of the art: original UNIX file system.

Measure throughput for file reads/writes.

Compare to theoretical max, which is…
disk bandwidth

Old UNIX file system: only **2%** of potential. Why?

Measurement 1

What is performance before/after aging?

Measurement 1

What is performance before/after aging?

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

Measurement 1

What is performance before/after aging?

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

FS is probably becoming fragmented over time.

Free list makes contiguous chunks hard to find.

Measurement 1

What is performance before/after aging?

hacky solution:
occasional defrag

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

FS is probably becoming fragmented over time.

Free list makes contiguous chunks hard to find.

Measurement 2

How does block size affect performance?
Try doubling it!

Measurement 2

How does block size affect performance?
Try doubling it!

Performance **more** than doubled.

Measurement 2

How does block size affect performance?
Try doubling it!

Performance **more** than doubled.

Measurement 2

How does block size affect performance?
Try doubling it!

Performance **more** than doubled.

Logically adjacent blocks are probably not physically adjacent.

Measurement 2

How does block size affect performance?
Try doubling it!

Performance **more** than doubled.

Logically adjacent blocks are probably not physically adjacent.

Measurement 2

How does block size affect performance?
Try doubling it!

Performance **more** than doubled.

Logically adjacent blocks are probably not physically adjacent.

Smaller blocks cause more indirect I/O.

Old FS Summary

Observations:
- long distance between inodes/data
- inodes in single dir not close to one another
- small blocks (512 bytes)
- blocks laid out poorly
- free list becomes scrambled, causes random alloc

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

Problem: old FS treats disk like RAM!

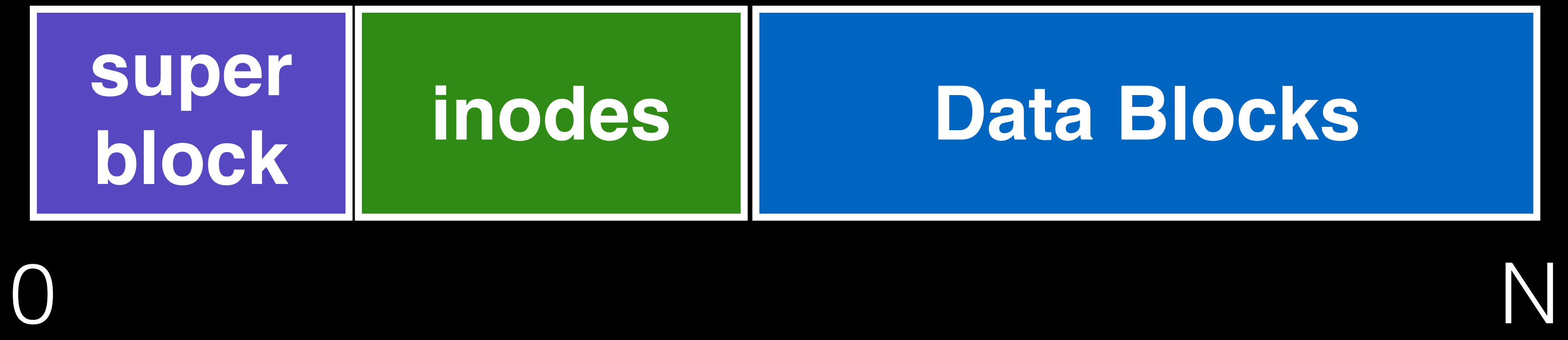
Solution: a disk-aware FS

Design Questions

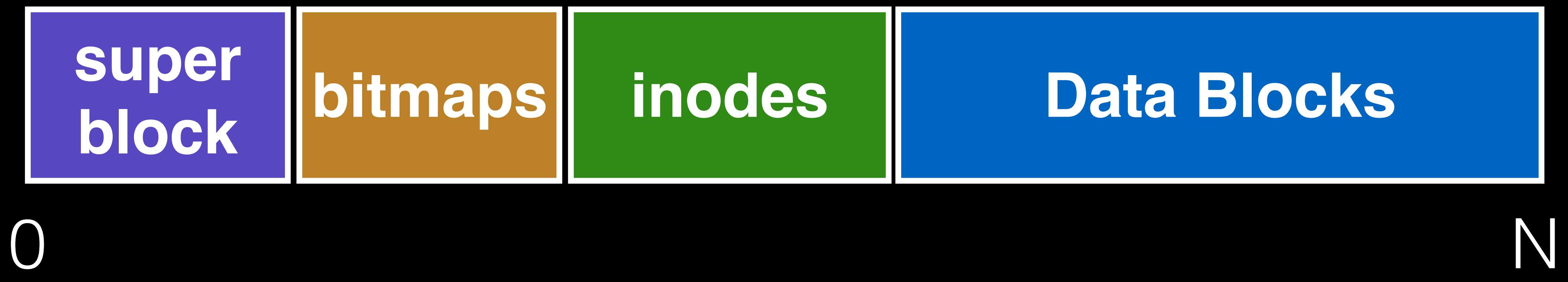
How to use big blocks without wasting space.

How to place data on disk.

Technique 1: Bitmaps



Technique 1: Bitmaps

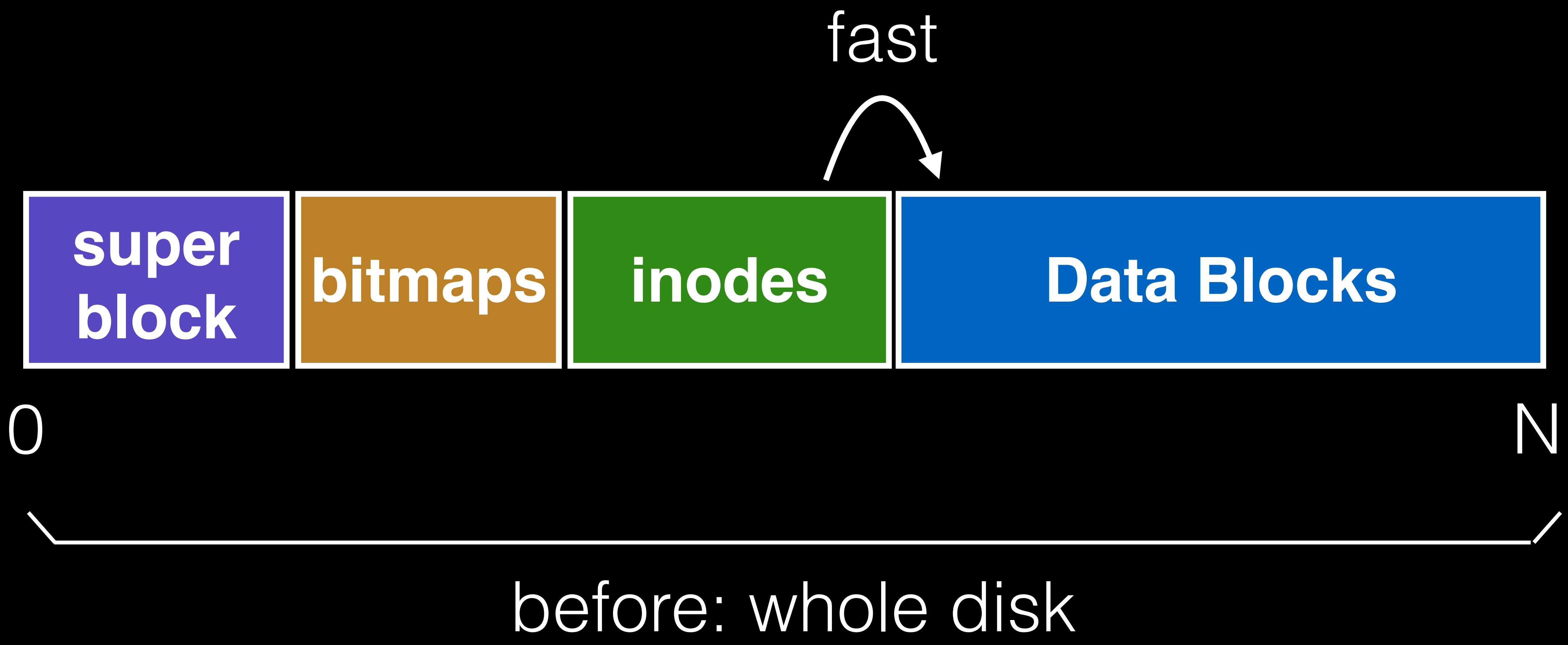


Use bitmaps instead of free list.
Provides more flexibility, with more global view.

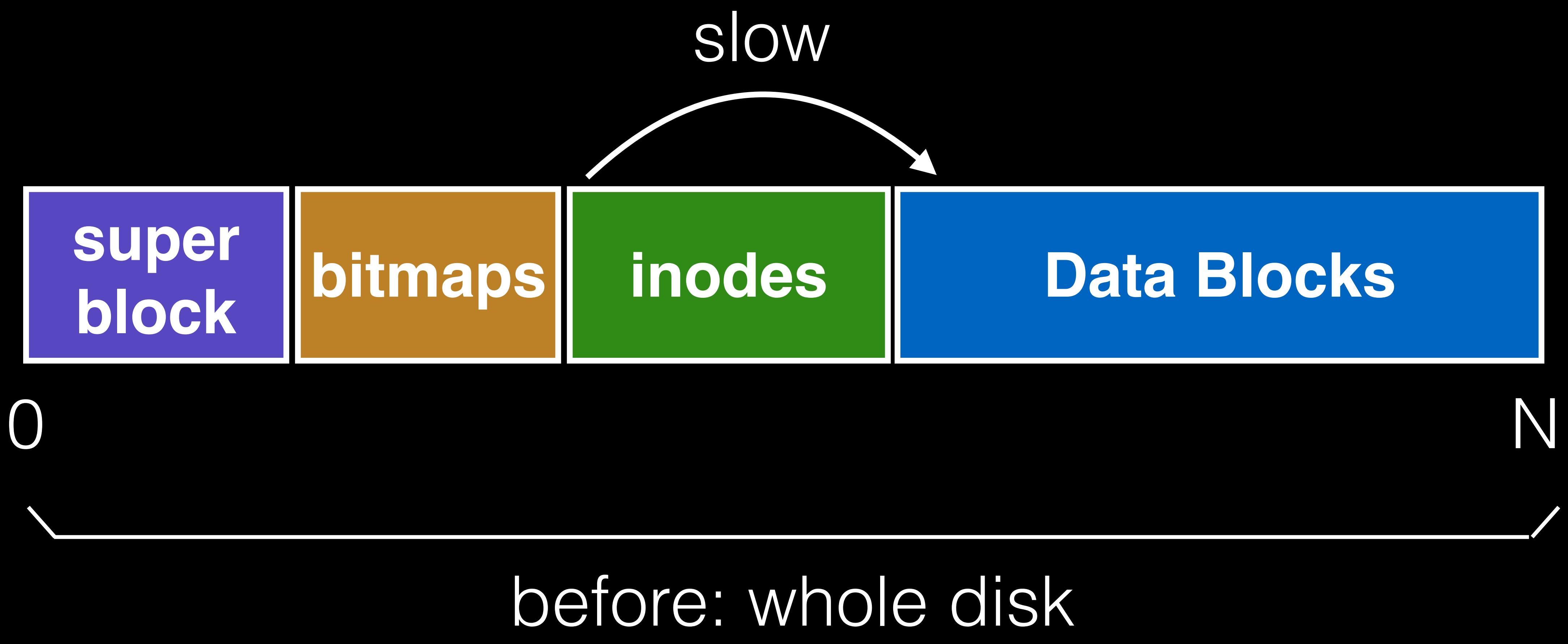
Techniques

Bitmaps

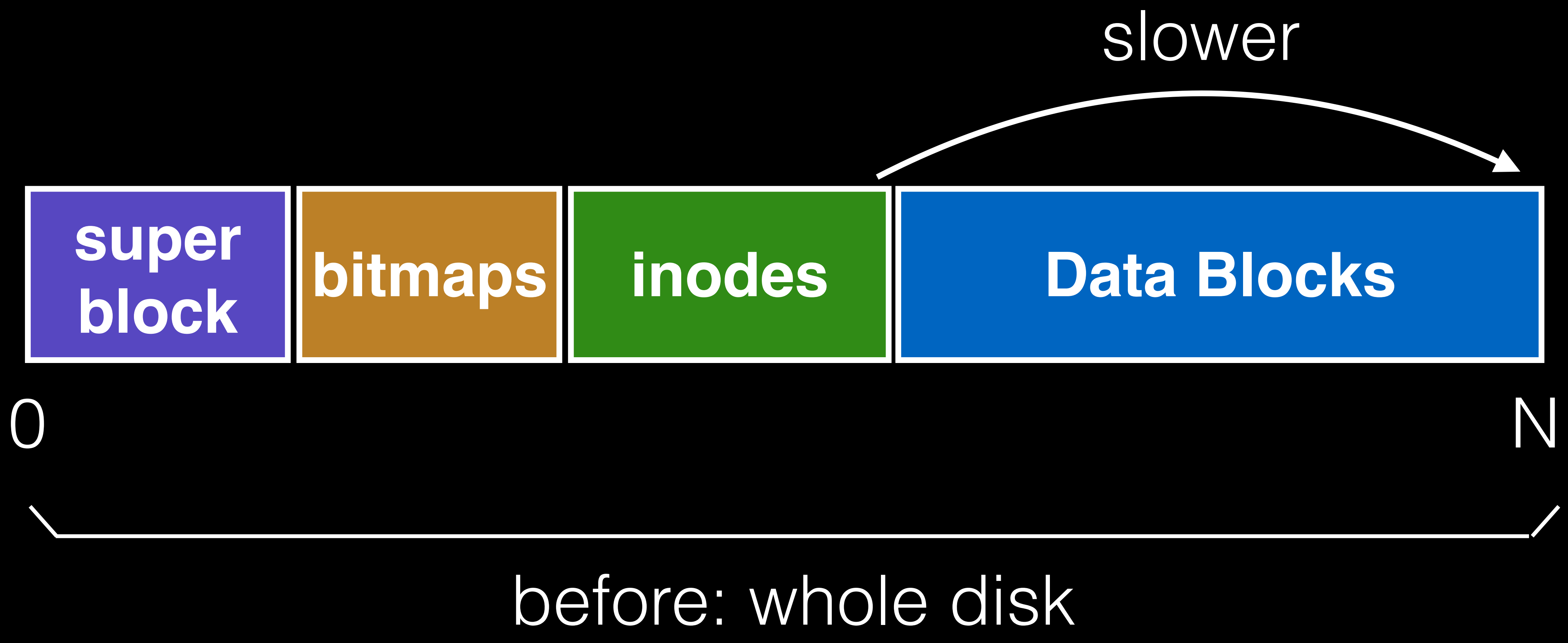
Technique 2: Groups



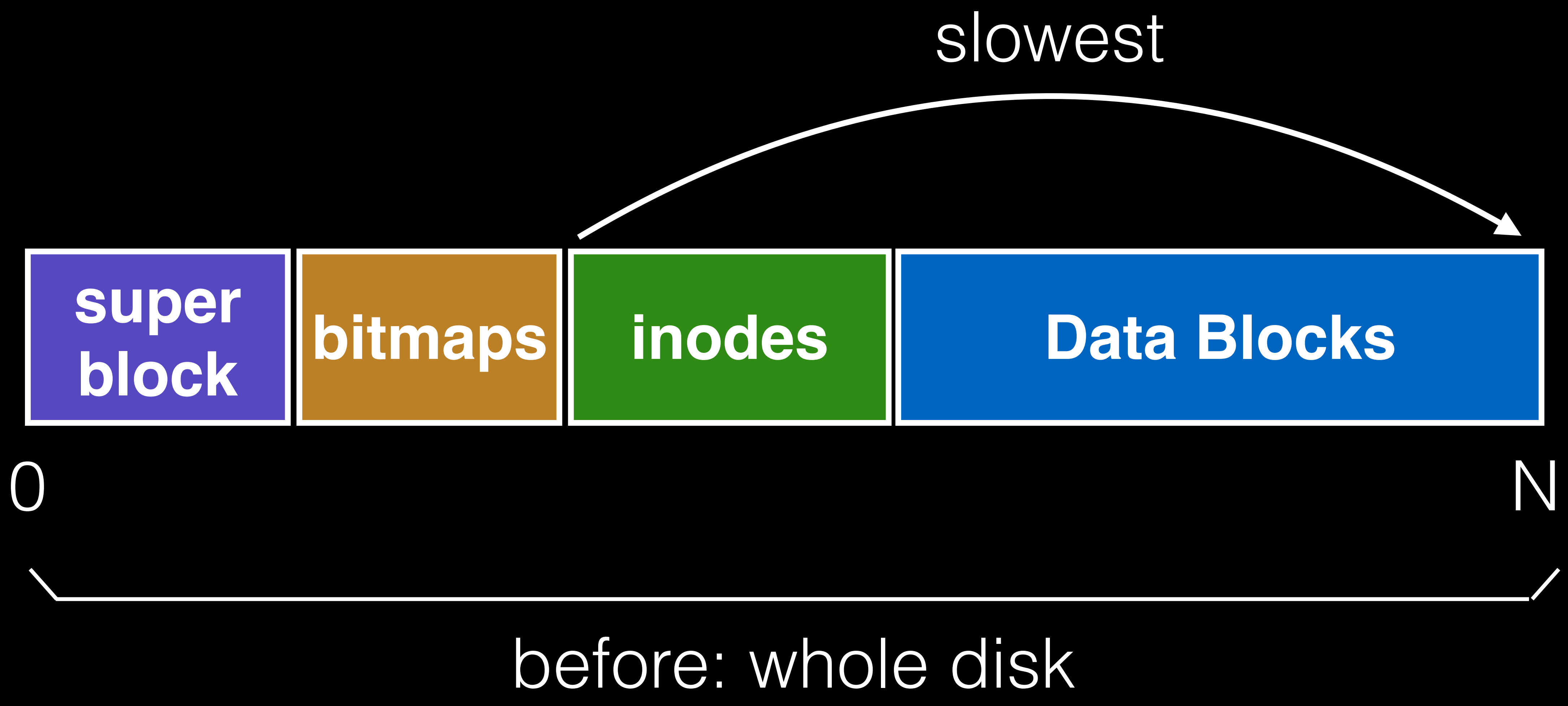
Technique 2: Groups



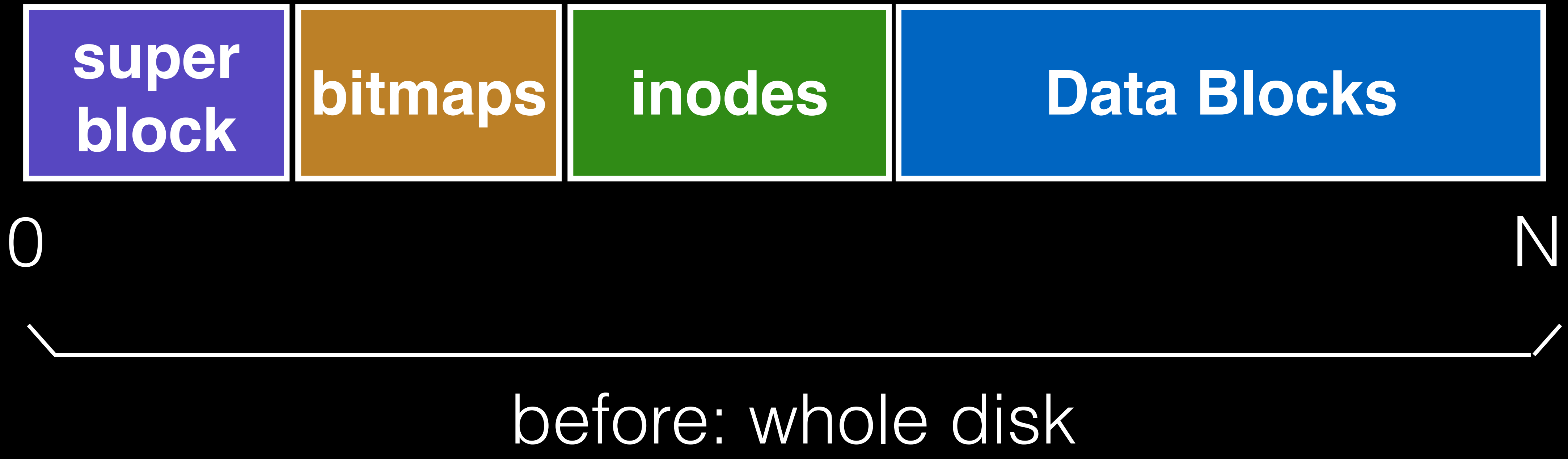
Technique 2: Groups



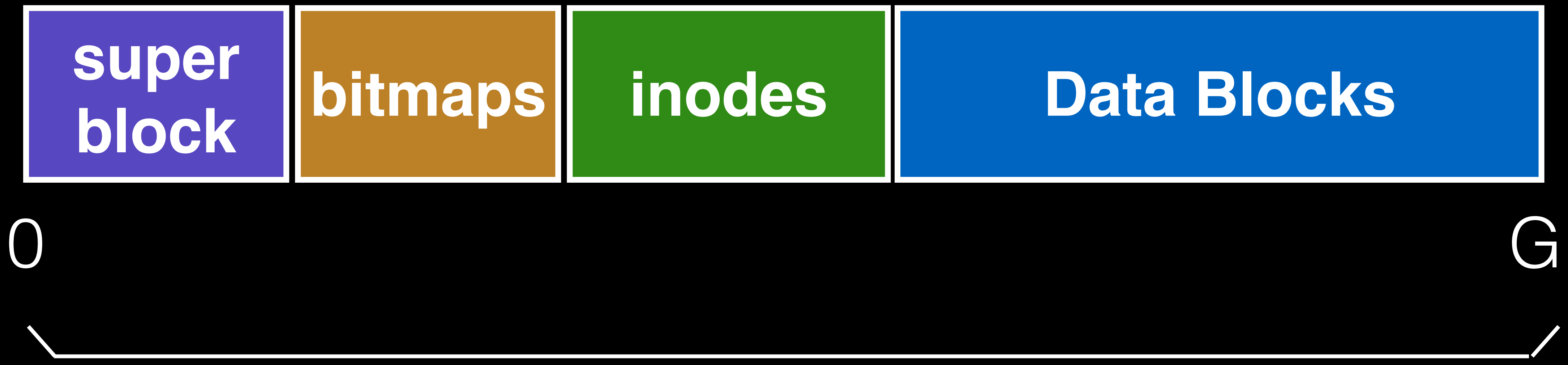
Technique 2: Groups



Technique 2: Groups

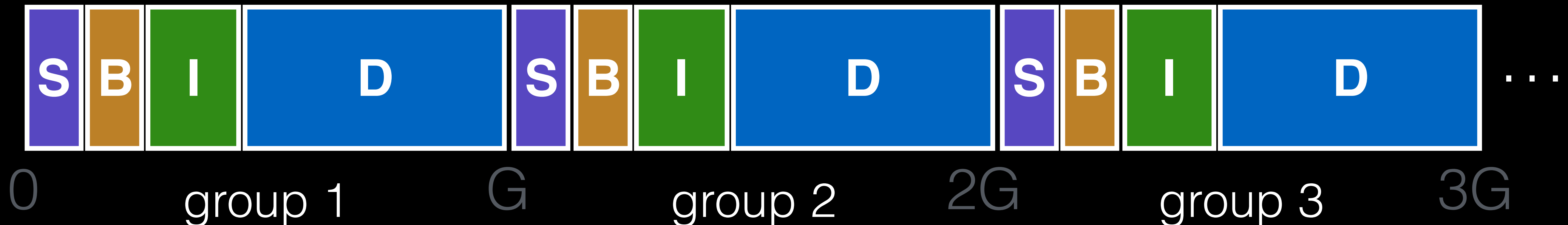


Technique 2: Groups



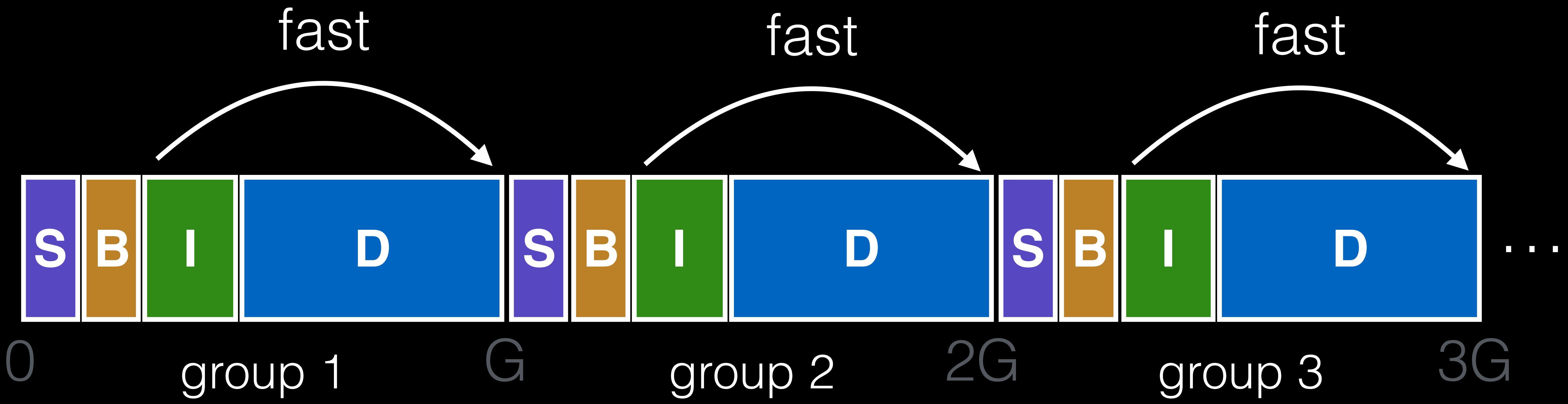
now: one (smallish) group

Technique 2: Groups



zoom out

Technique 2: Groups



strategy: allocate inodes and data blocks in same group.

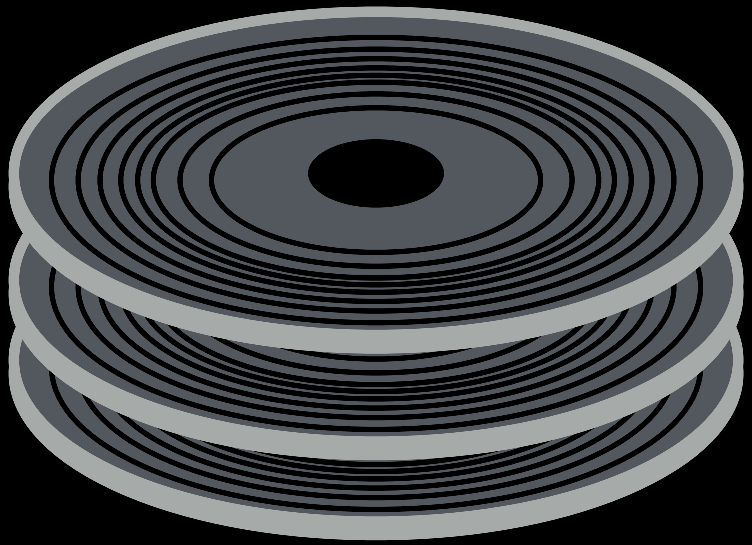
Groups

In FFS, groups were ranges of cylinders

- called <u>cylinder group</u>

In ext2-4, groups are ranges of blocks

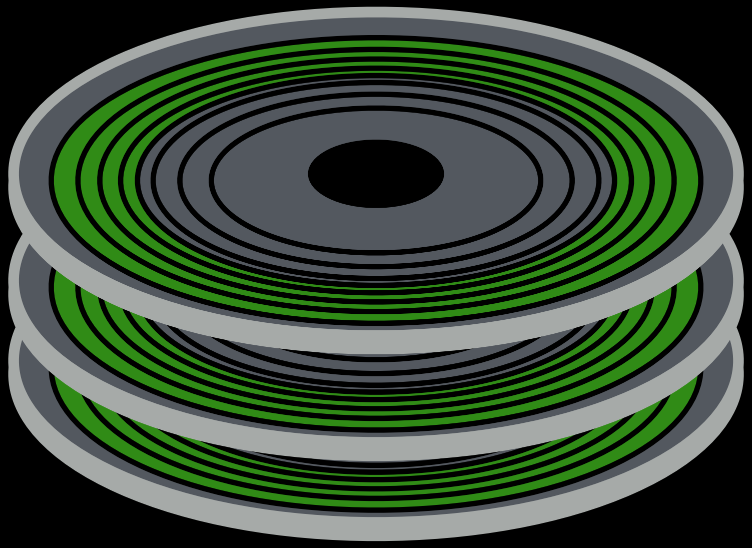
- called <u>block group</u>



Groups

In FFS, groups were ranges of cylinders
- called cylinder group

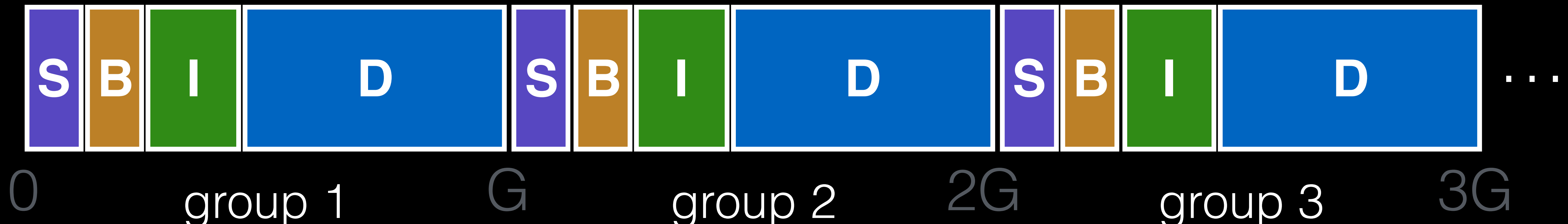
In ext2-4, groups are ranges of blocks
- called block group



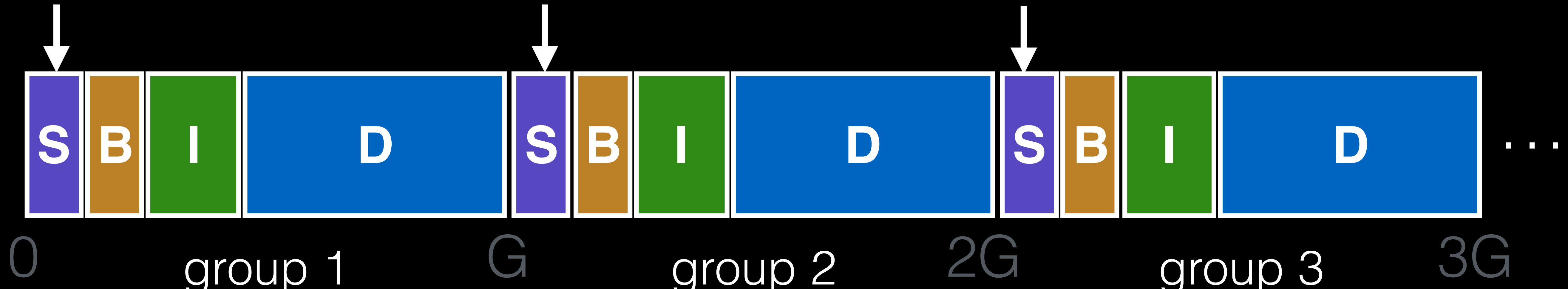
Techniques

Bitmaps
Locality groups

Technique 3: Super Rotation

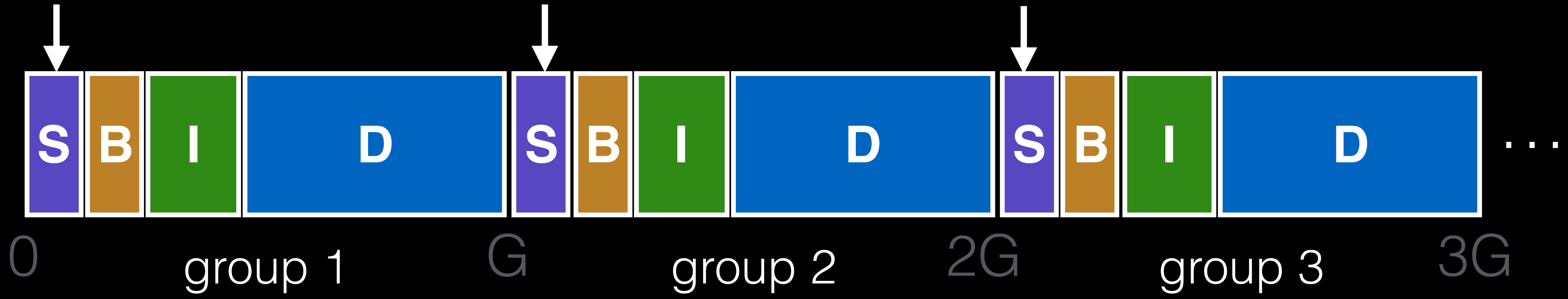


Technique 3: Super Rotation



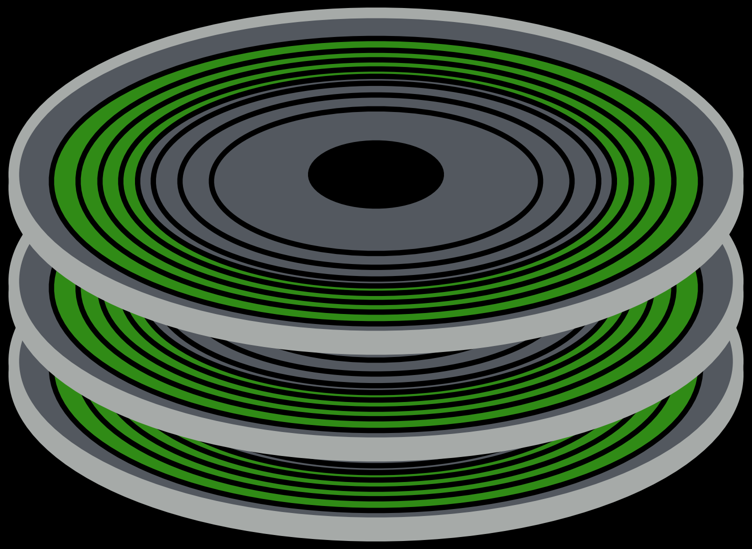
Is it useful to have multiple super blocks?

Technique 3: Super Rotation

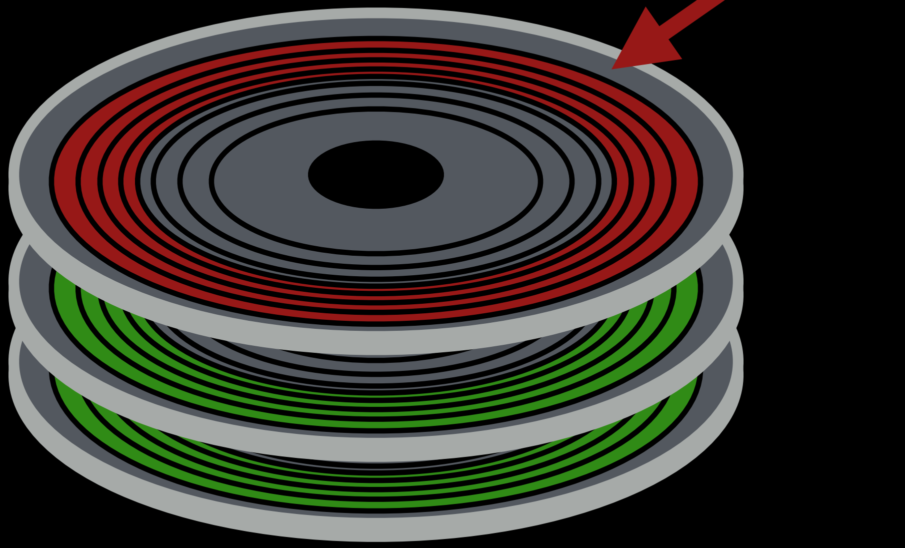


Is it useful to have multiple super blocks?
Yes, if some (but not all) fail.

Problem

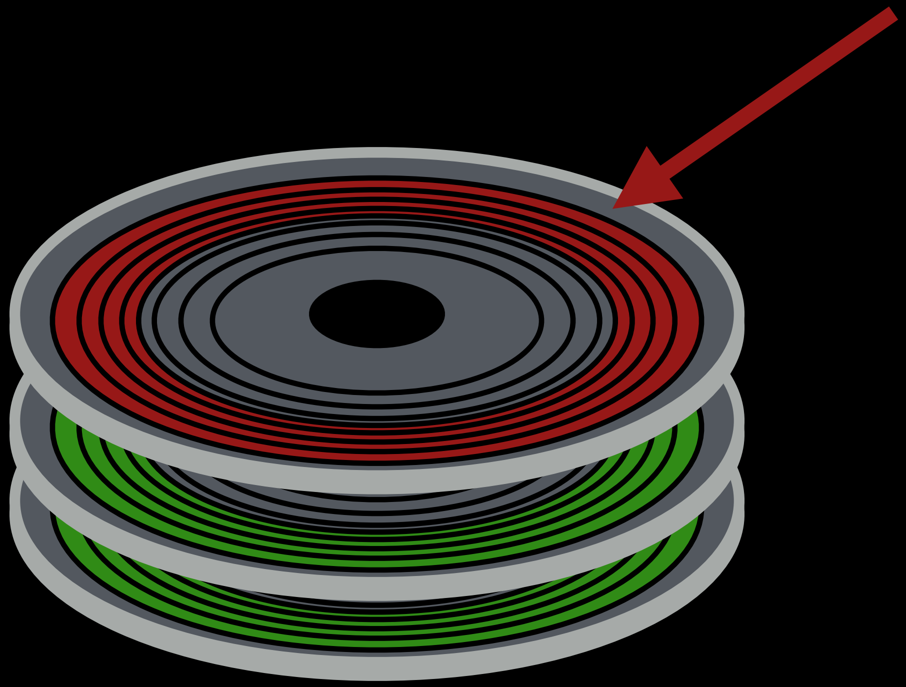


Problem



All super-block copies are on the top platter. What if it dies?

Problem



All super-block copies are on the top platter. What if it dies?

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

Techniques

Bitmaps
Locality groups
Rotated super

Block Size

Doubling the block size for the old FS over doubled performance.

Strategy: choose block size so we never have to read more than two indirect blocks to find a data block (2 levels of indirection max). Want 4GB files.

How large is this?

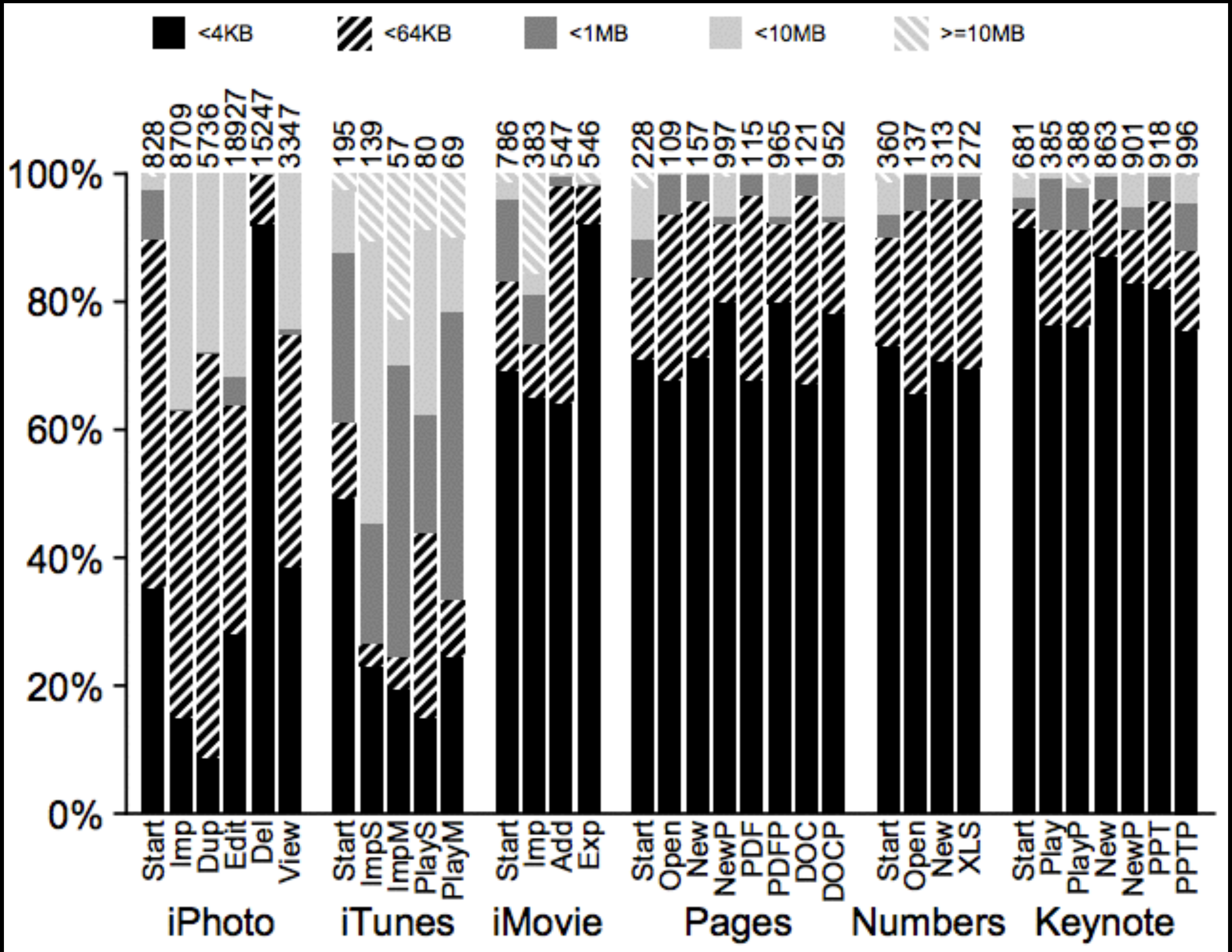
Techniques

Bitmaps
Locality groups
Rotated super
Large blocks

Large Blocks

Why not make blocks huge?

Most file are very small.

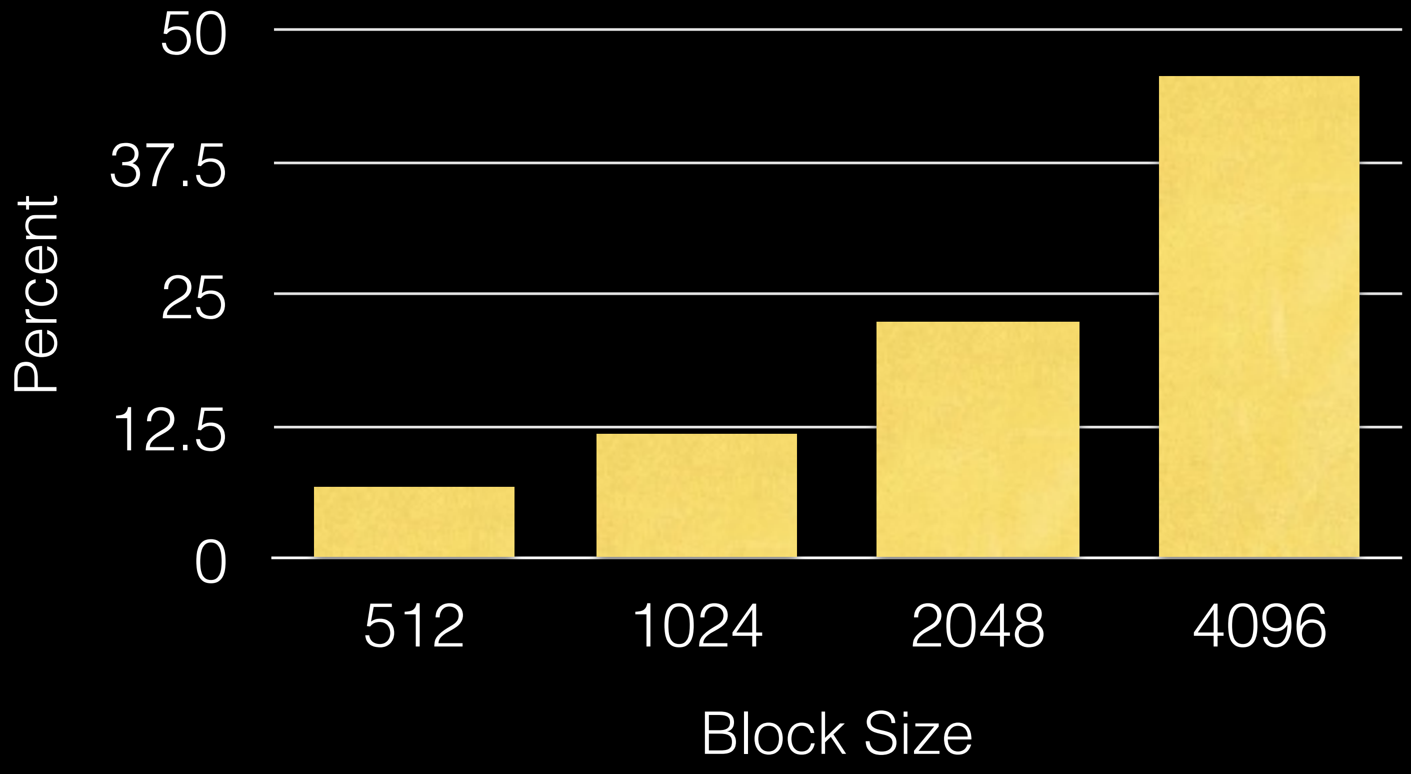


Large Blocks

Why not make blocks huge?
Lots of waste in remainder of blocks.

Large Blocks

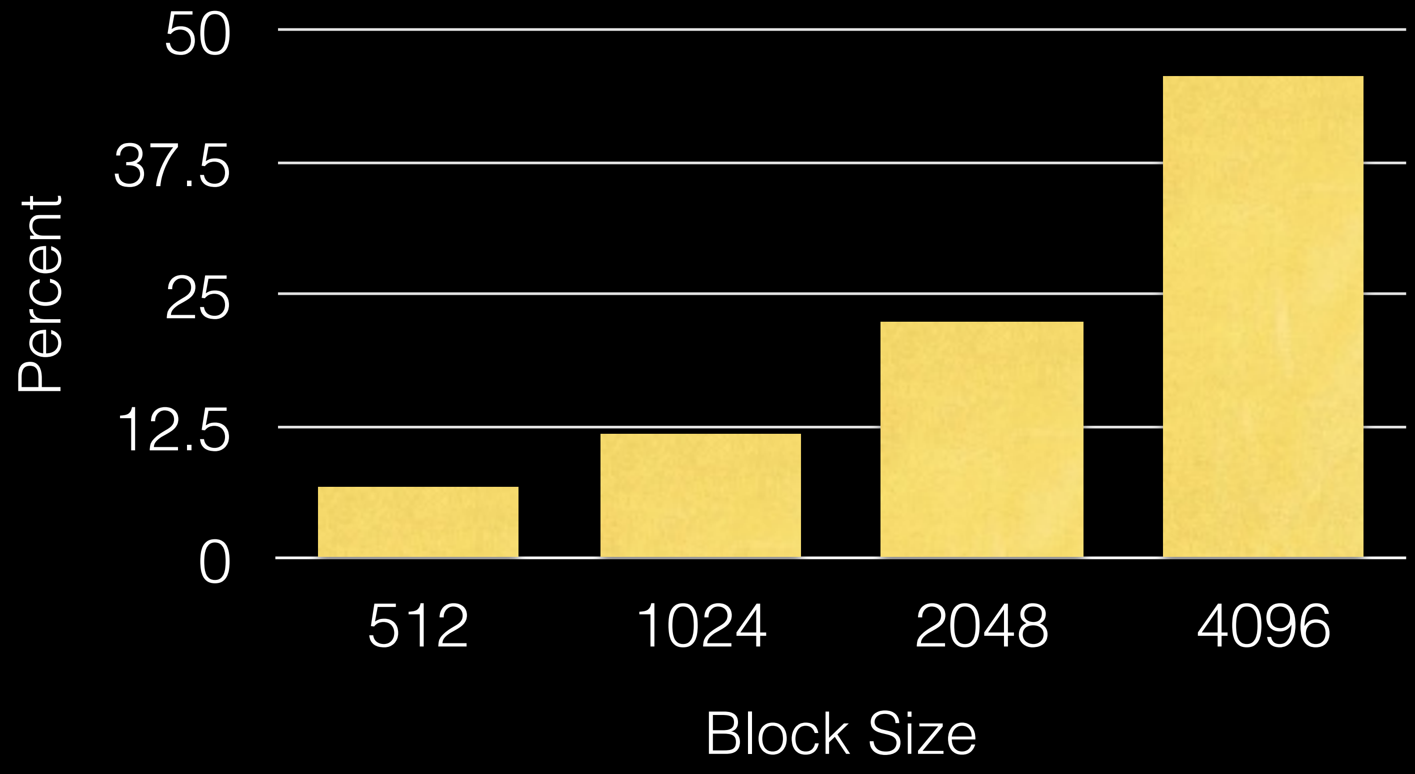
Why not make blocks huge?
Lots of waste in remainder of blocks.



Large Blocks

Why not make blocks huge?
Lots of waste in remainder of blocks.

Time vs. Space
Tradeoffs…



Solution: Fragments

Hybrid!

Introduce “fragment” for files that use parts of blocks.

Only tail of file uses fragments.

Fragment Example

Block size = 4096

Fragment size = 1024

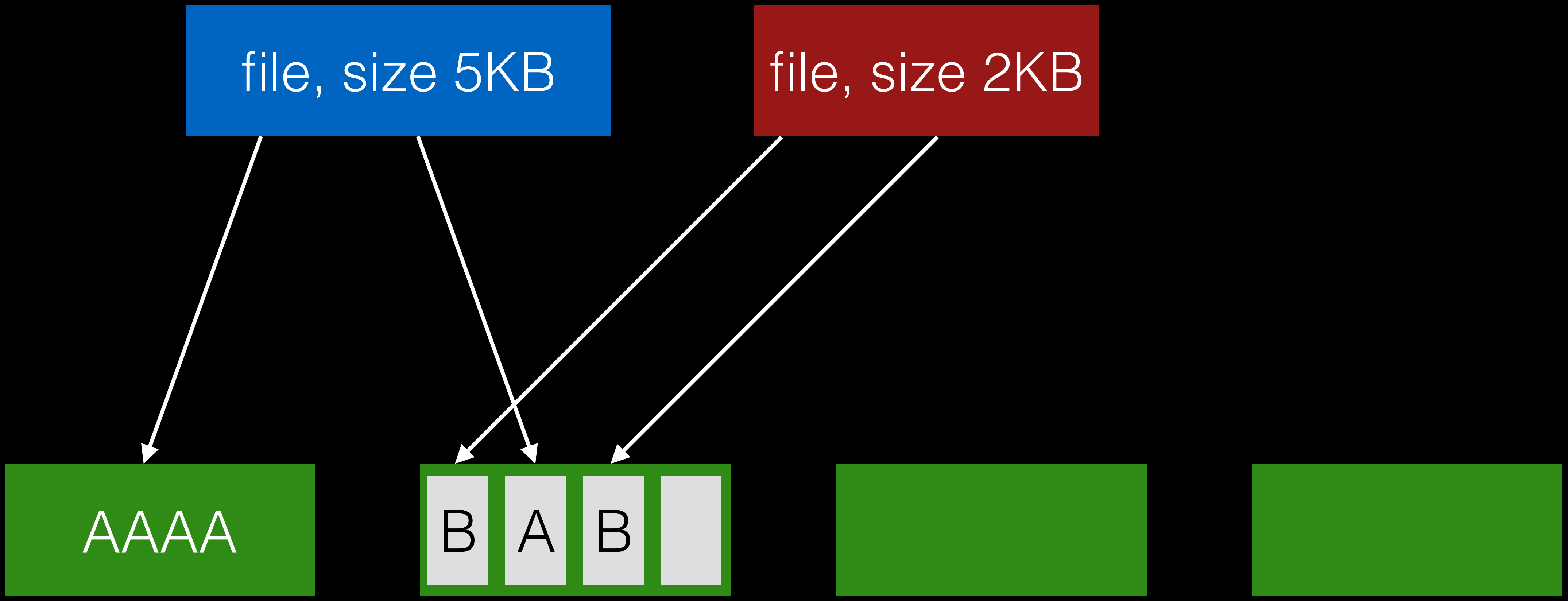
bits:	0000	0000	1111	0010
	blk1	blk2	blk3	blk4

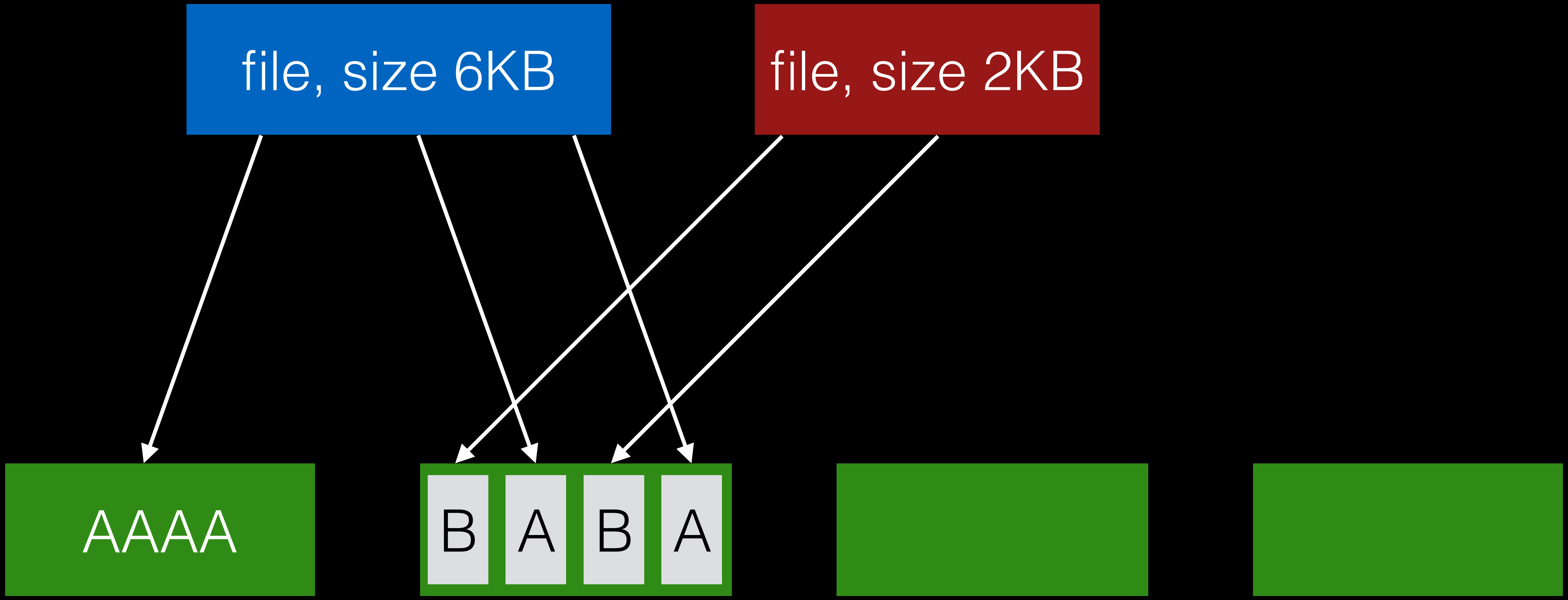
How to Decide

Whether addr refers to block or fragment is inferred by the file size.

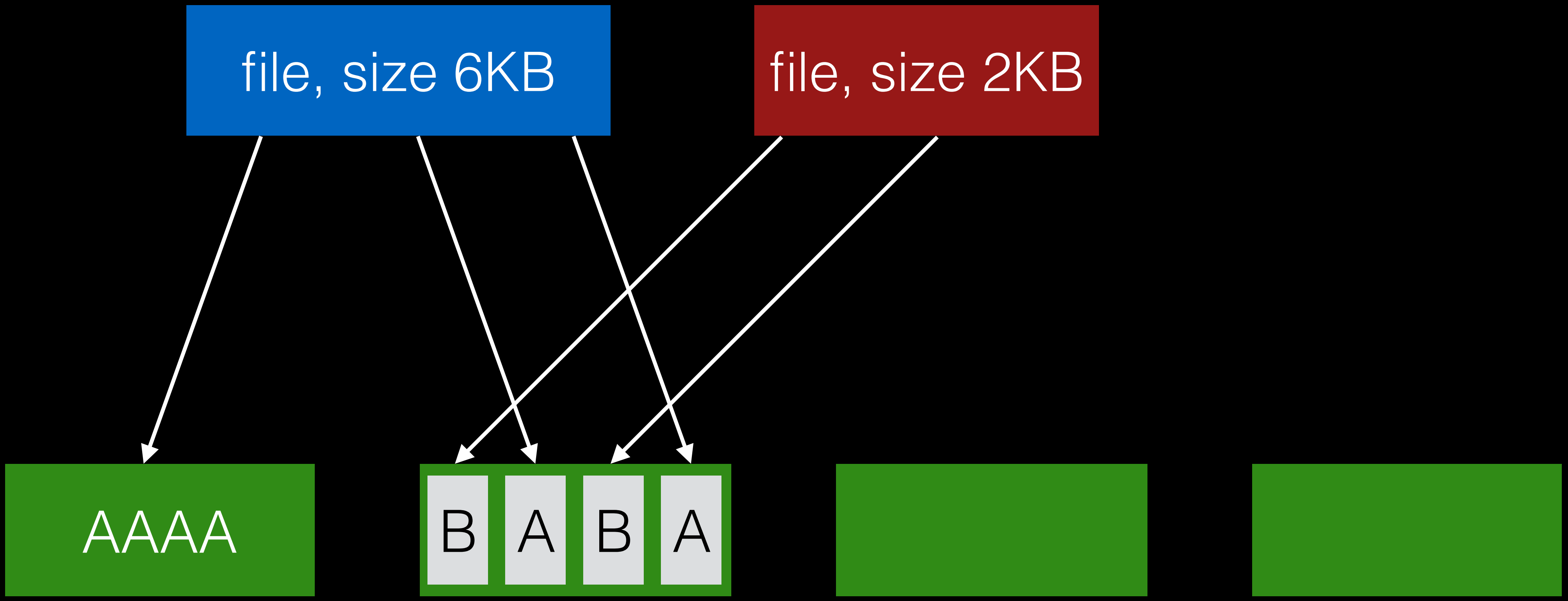
What about when files grow?

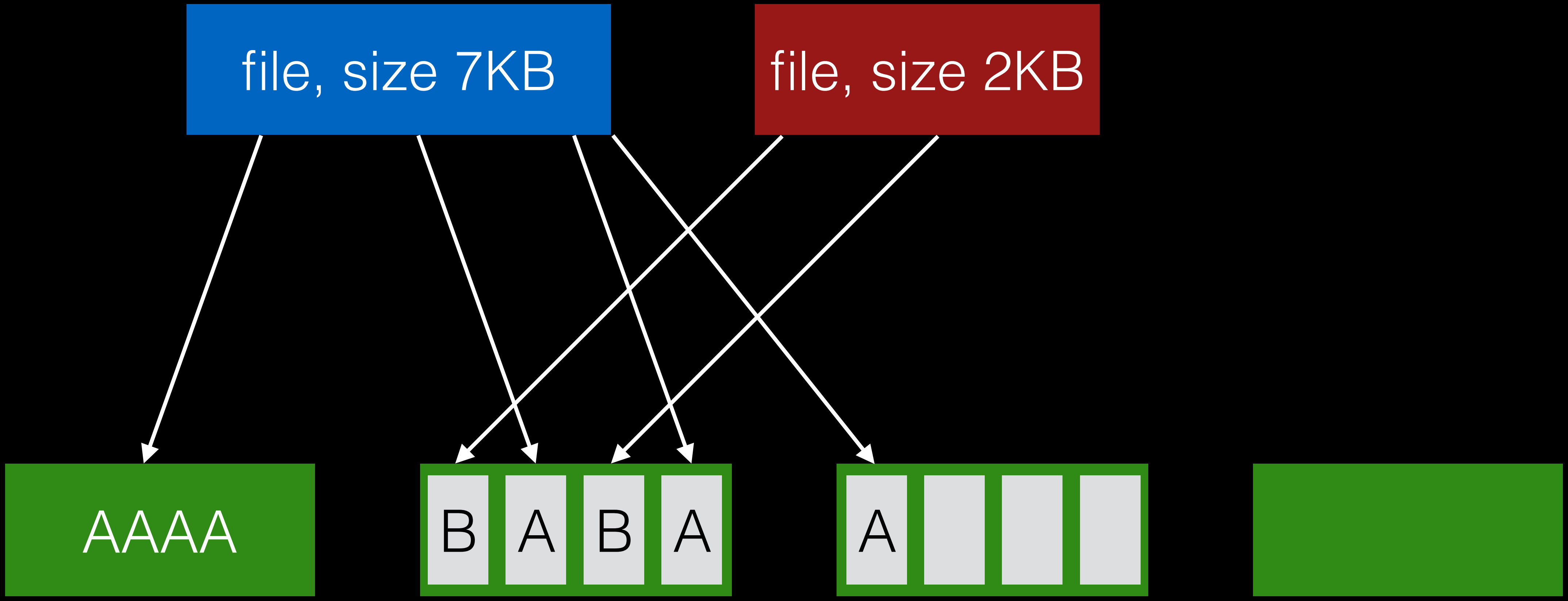
Must copy fragments to new block if there’s not room to grow.


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

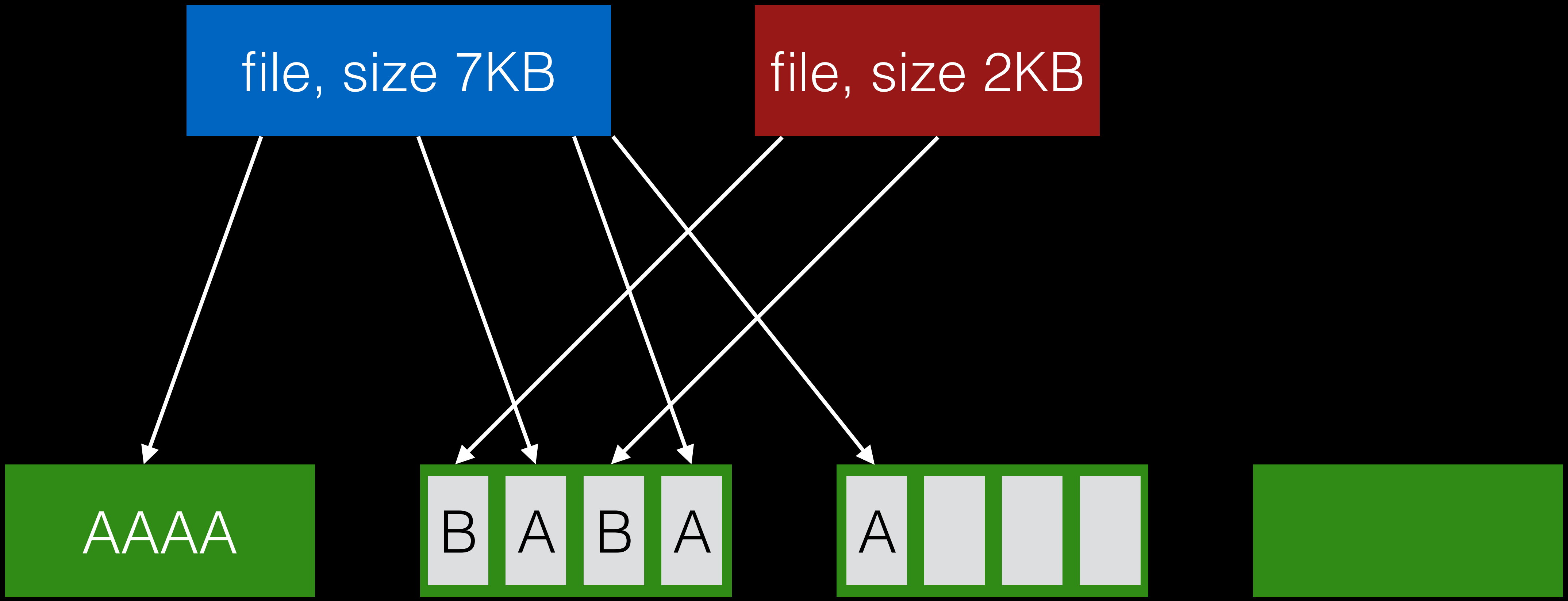


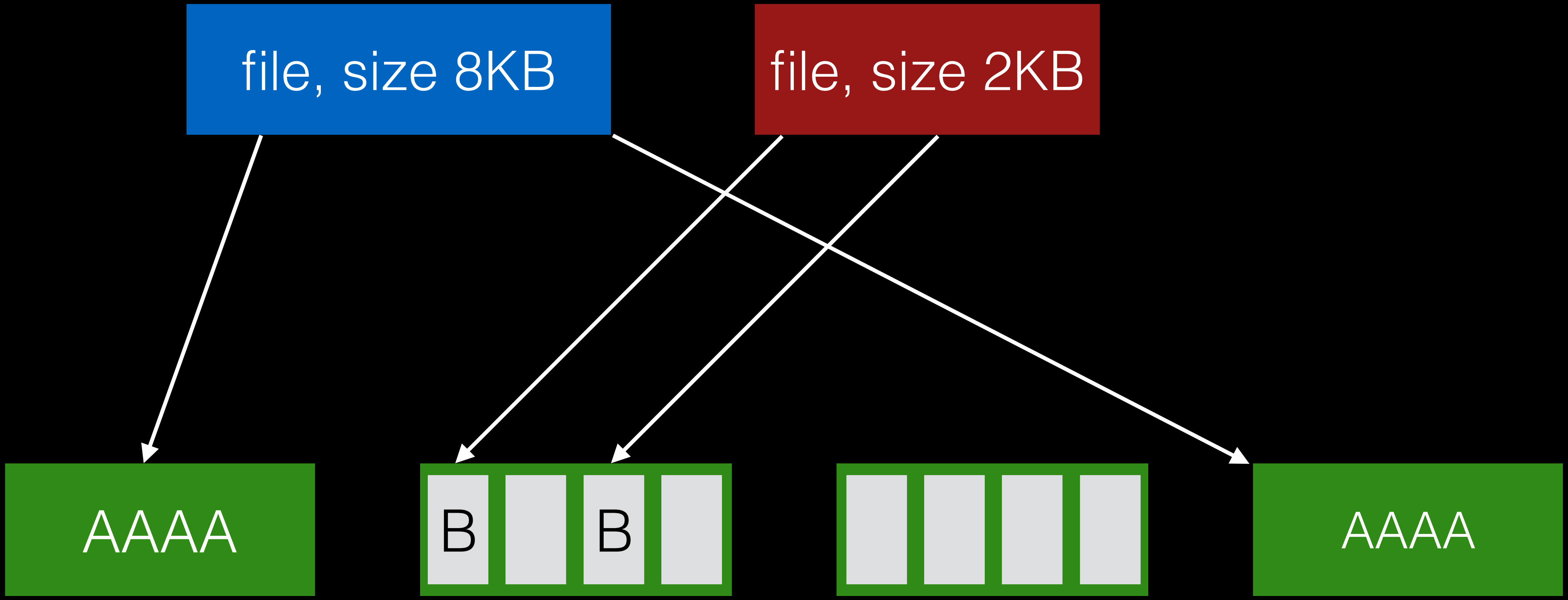
append A to first file


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



append A to first file


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



append A to first file, copy to fragments to new block.

Optimal Write Size

Writing less than a block is inefficient.

Solution: new API exposes optimal write size.

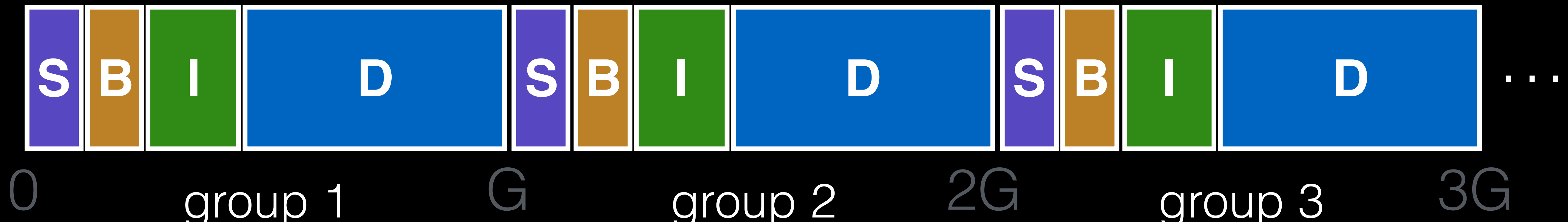
For pipes and sockets, the new call returns the buffer size.

The stdio library uses this call.

Techniques

Bitmaps
Locality groups
Rotated super
Large blocks
Fragments

Smart Policy



Where should new inodes and data blocks go?

Strategy

Put related pieces of data near each other.

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.

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?

Challenge

The file system is one big tree.

All directories and files have a common root.

In some sense, all data in the same FS is related.

Challenge

The file system is one big tree.

All directories and files have a common root.

In some sense, all data in the same FS is related.

Trying to put everything near everything else will leave us with the same mess we started with.

Revised Strategy

Put more-related pieces of data near each other.

Put less-related pieces of data far from each other.

Revised Strategy

Put more-related pieces of data near each other.

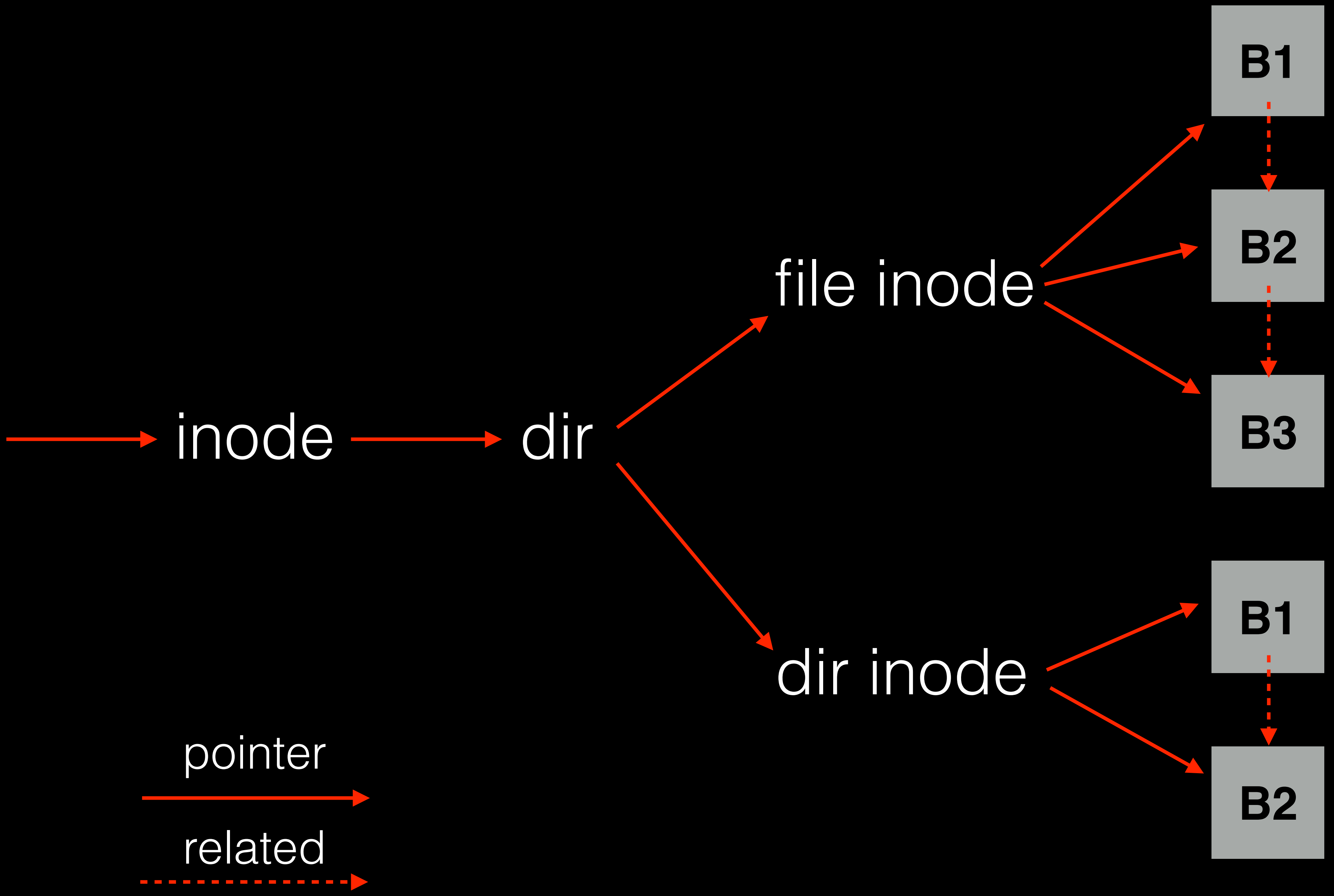
Put less-related pieces of data far from each other.

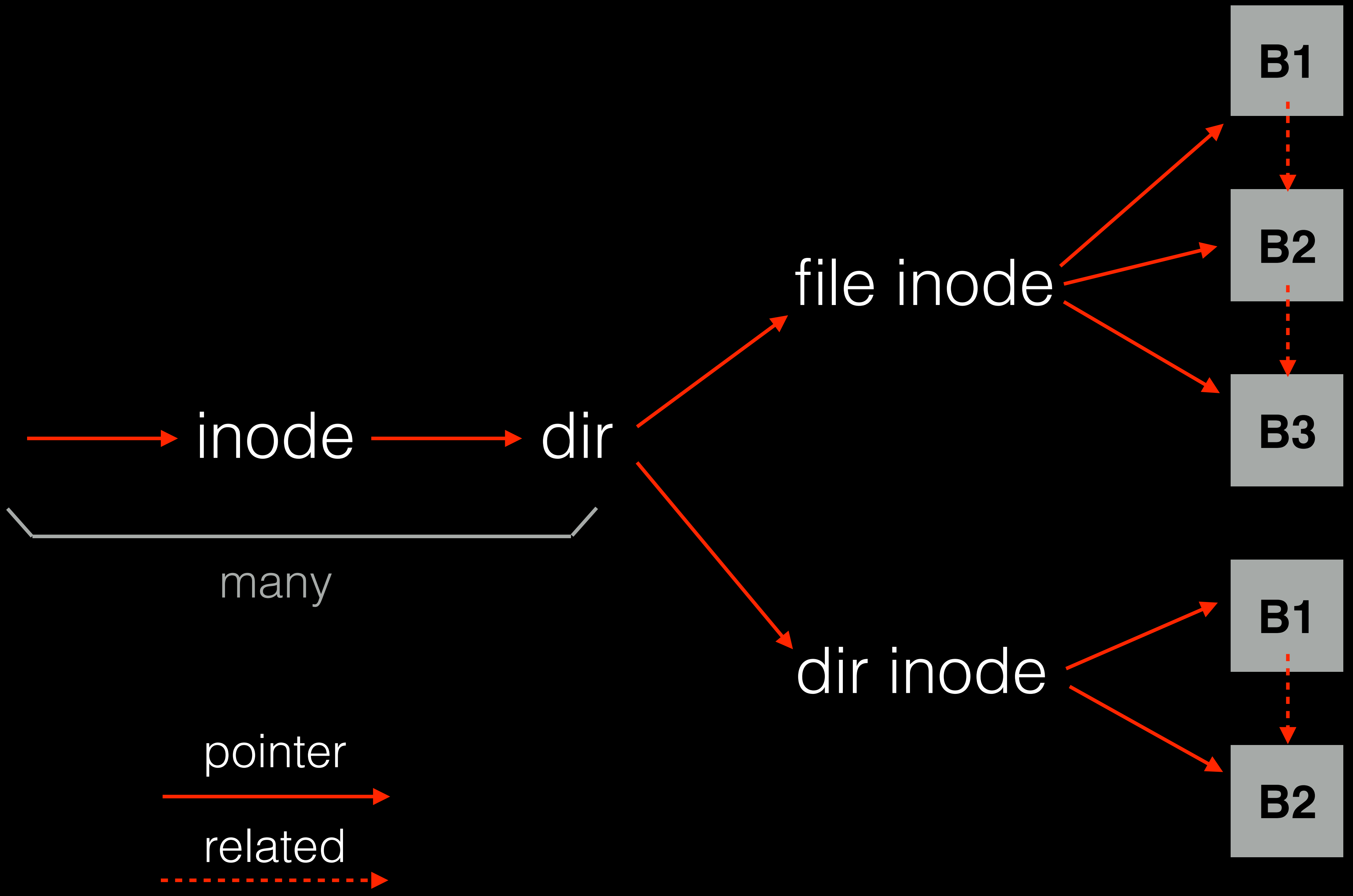
FFS developers used their best judgement.

FFS: Two-Level Allocator

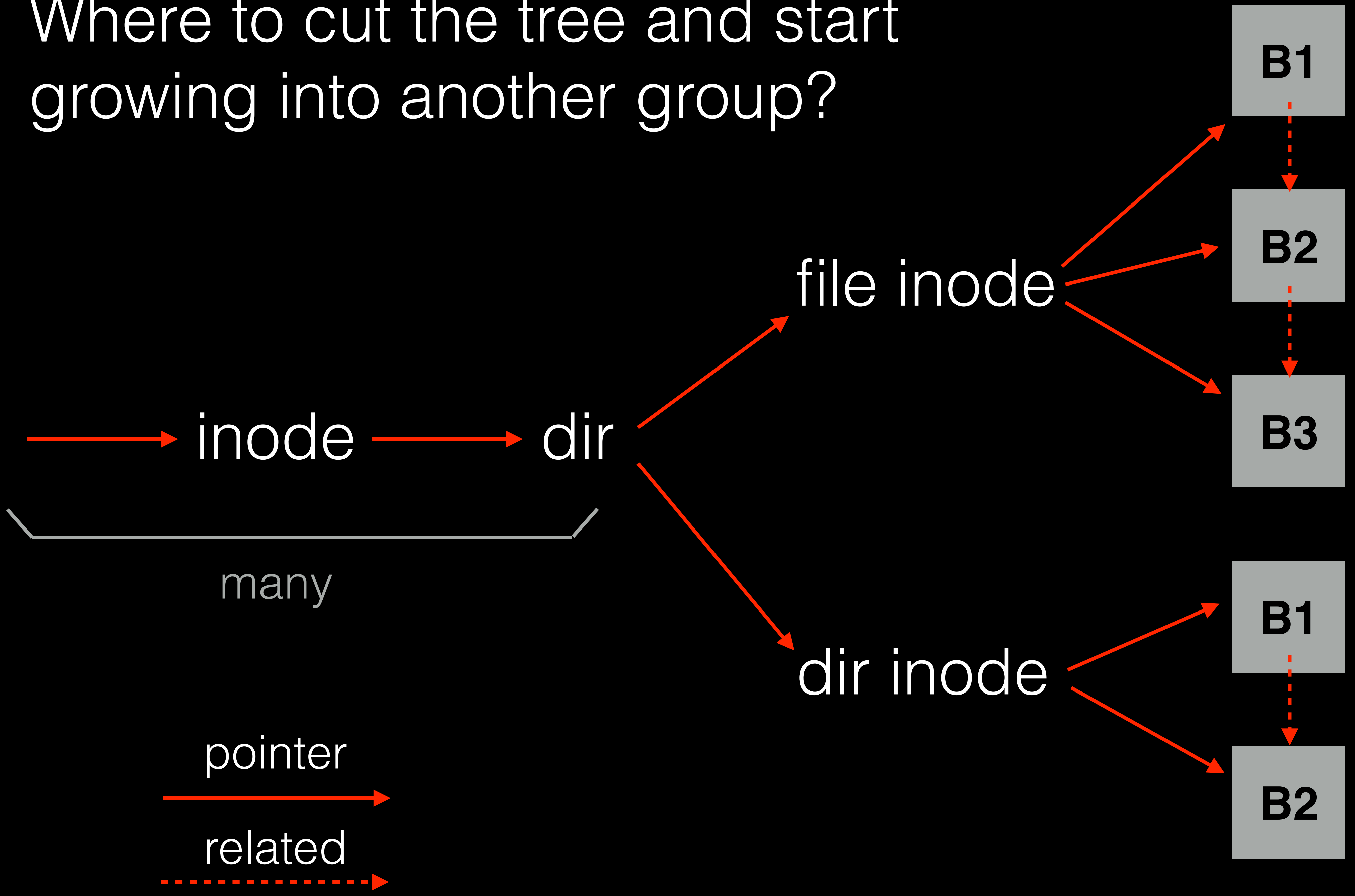
Level 1: decide which group

Level 2: decide where in group

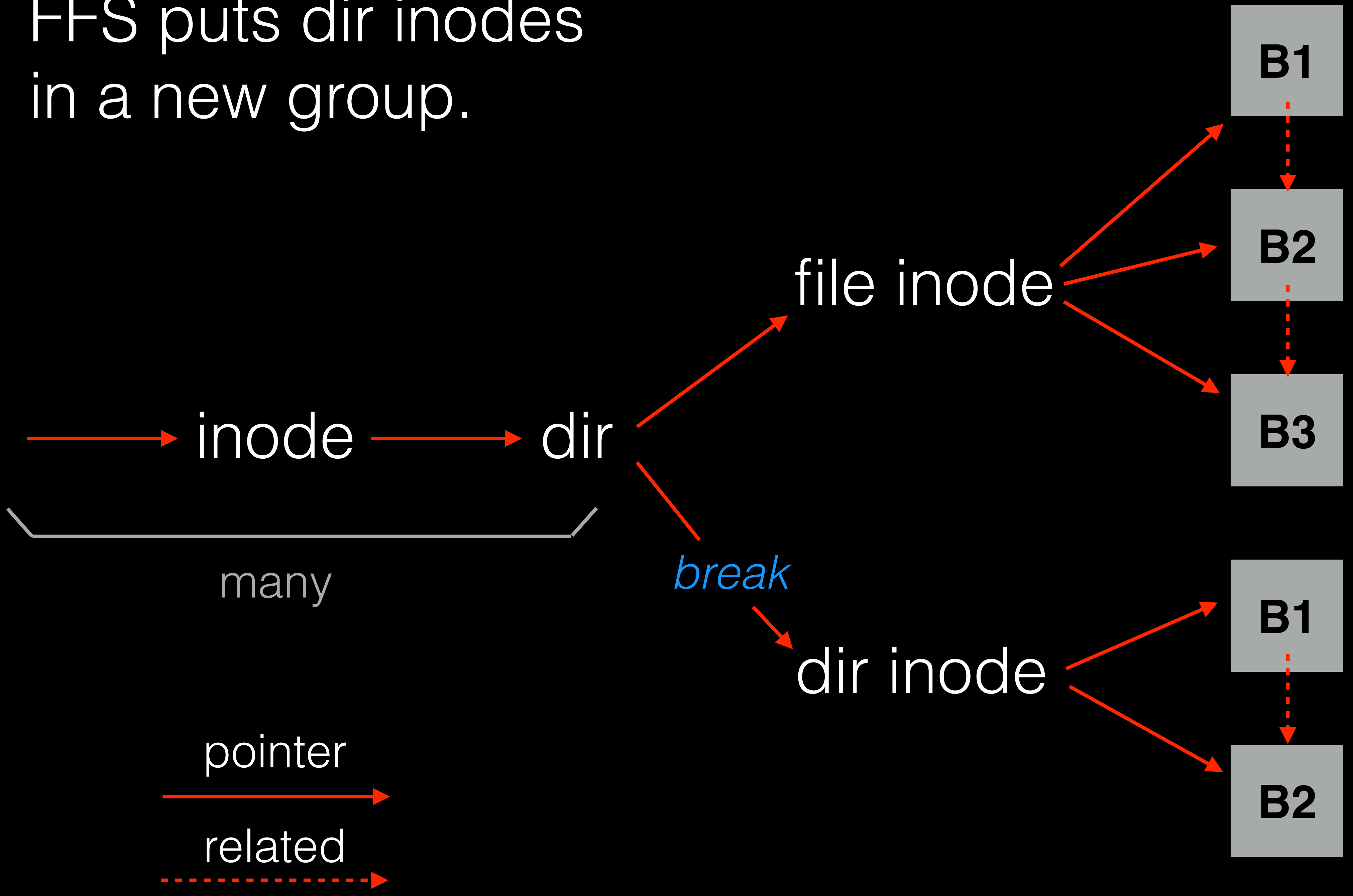

B1
B2
file inode
B3
inode
dir
dir inode
B1
B2
pointer
related


B1
B2
B3
file inode
inode
dir
many
dir inode
B1
B2
pointer
related

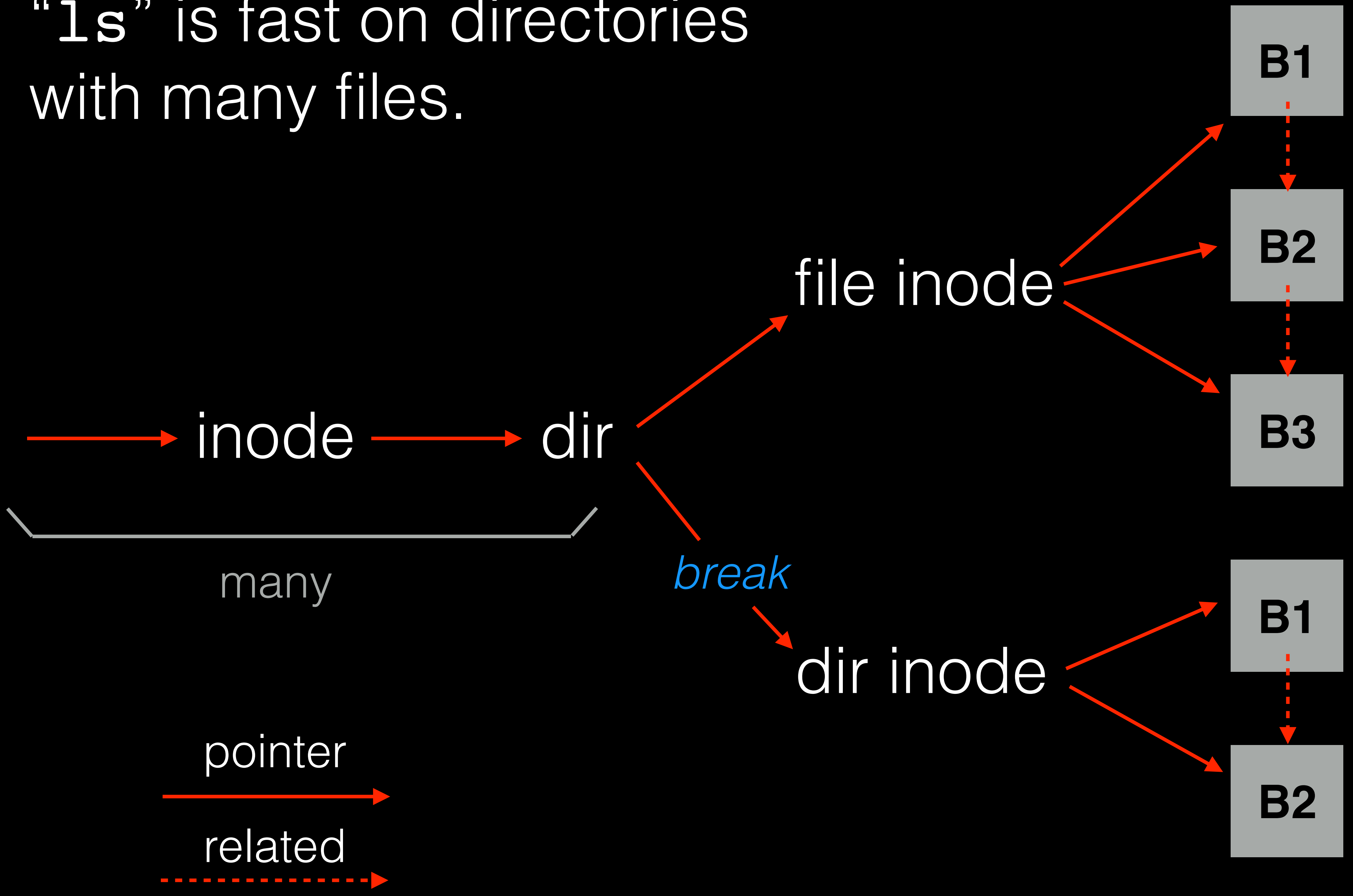
Where to cut the tree and start growing into another group?



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 inodes than the average group

First data block: allocate near inode

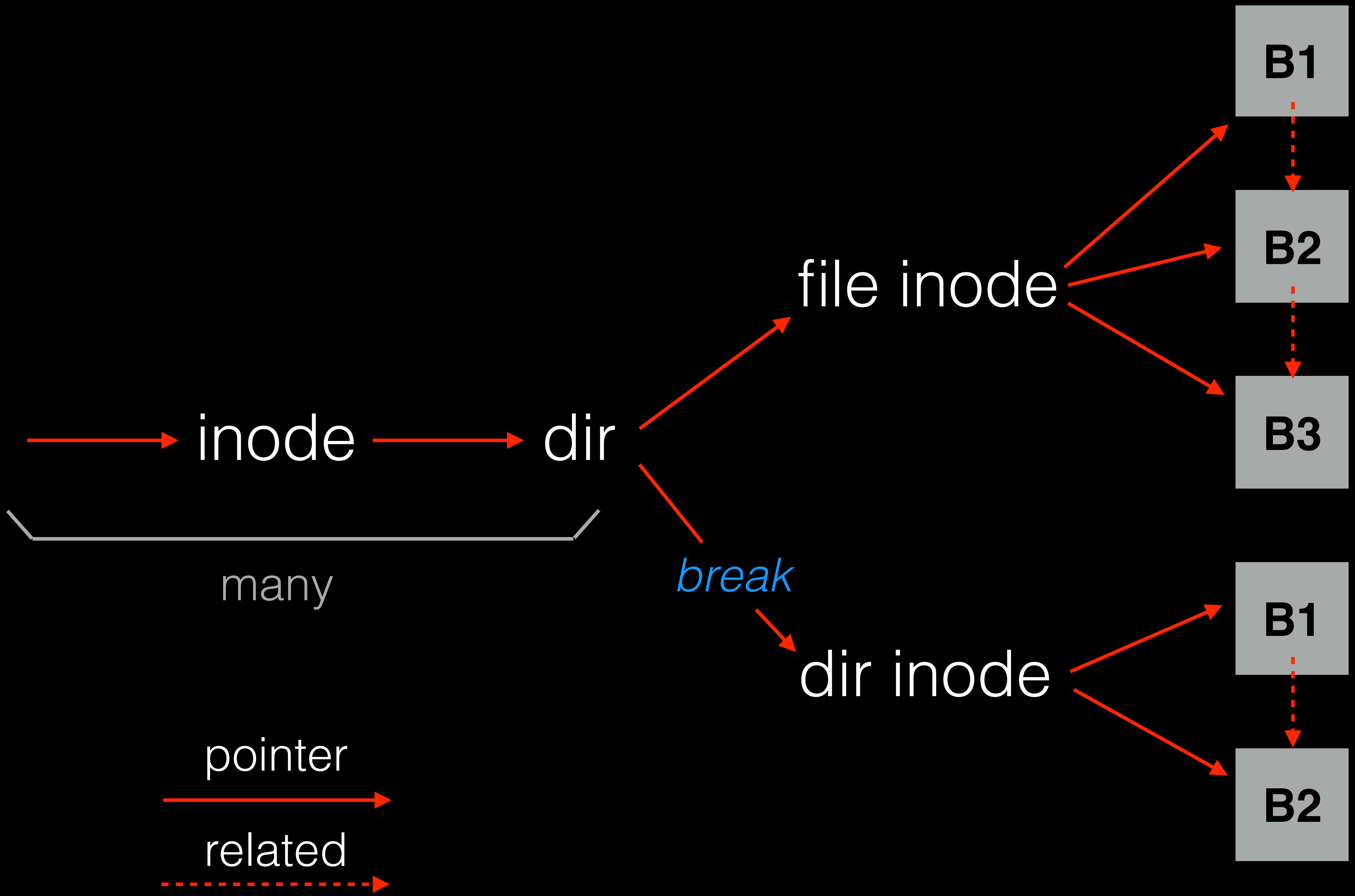
Other data blocks: allocate near previous block

Problem: Large Files

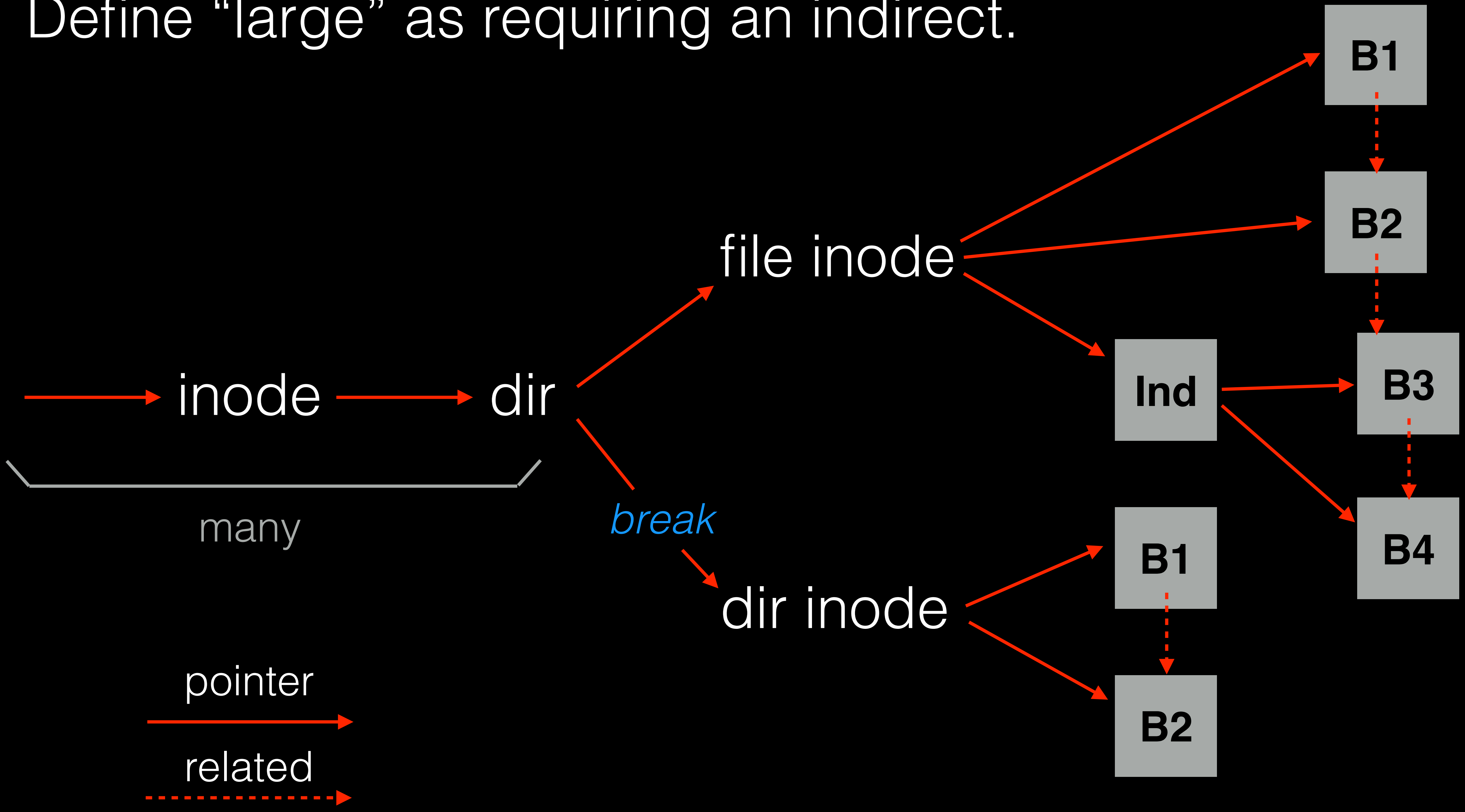
A single large file can use nearly all of a group.

This displaces data for many small files.

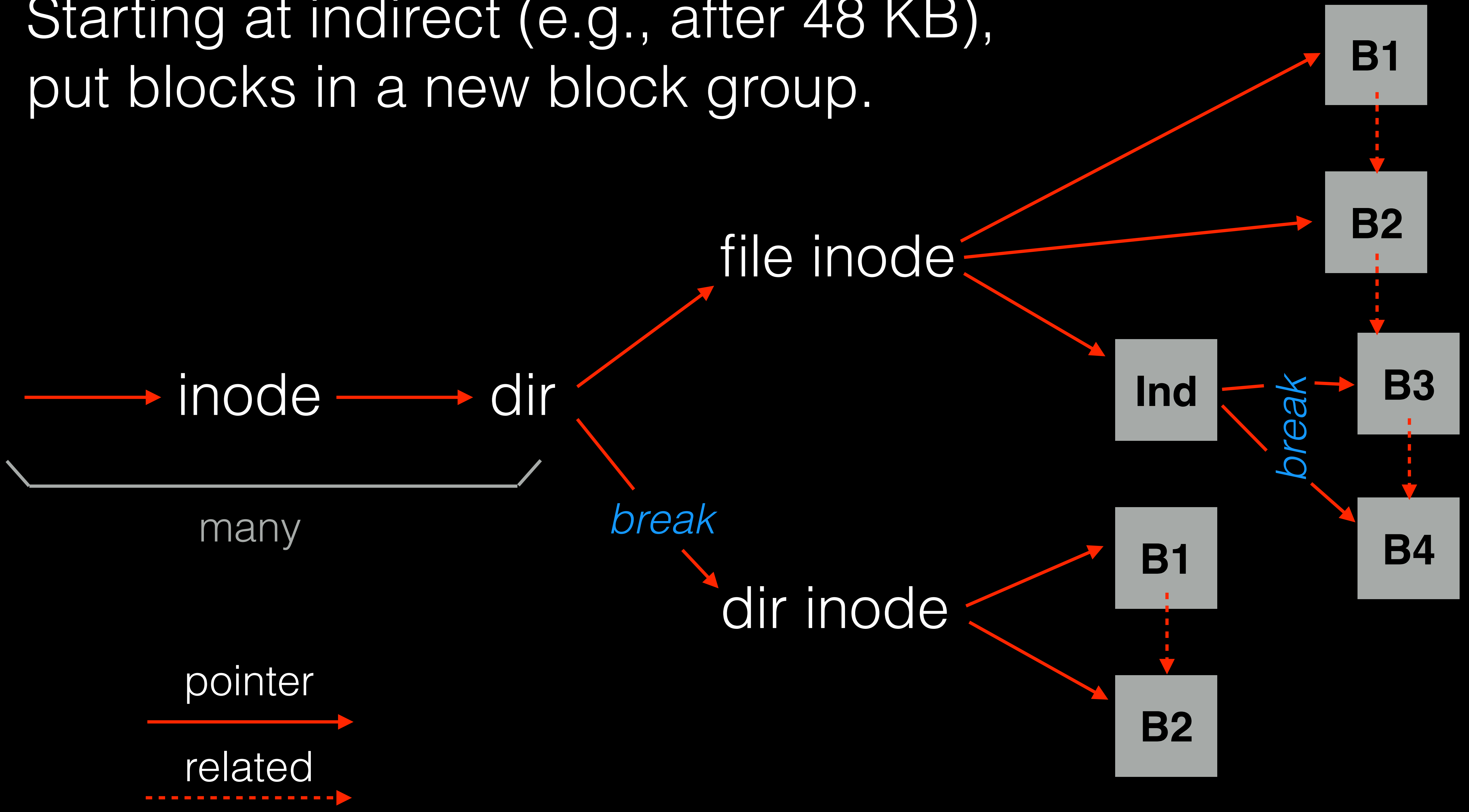
It’s better to do one seek for the large file than one seek for each of many small files.


inode
dir
many
file inode
B1
B2
B3
break
dir inode
B1
B2
pointer
related

Define “large” as requiring an indirect.



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 inodes than the 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.

Preferences

File inodes: allocate in same group with dir

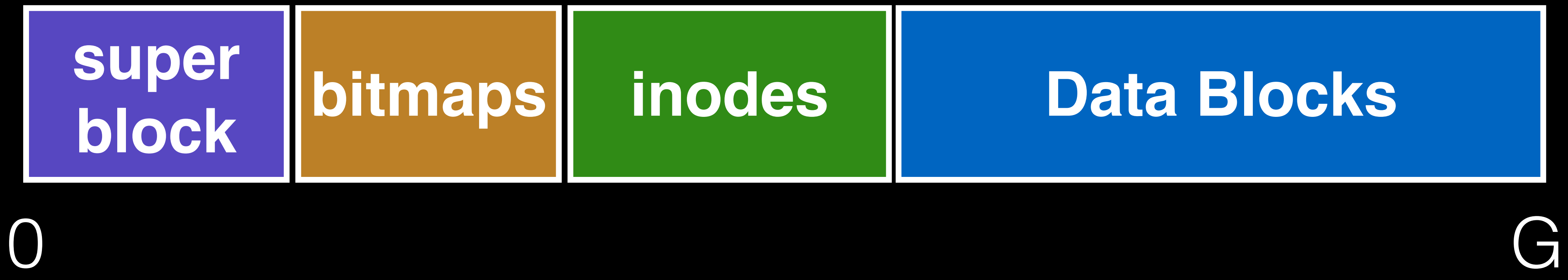
Dir inodes: allocate in new group with fewer inodes than the average group

First data block: allocate near inode

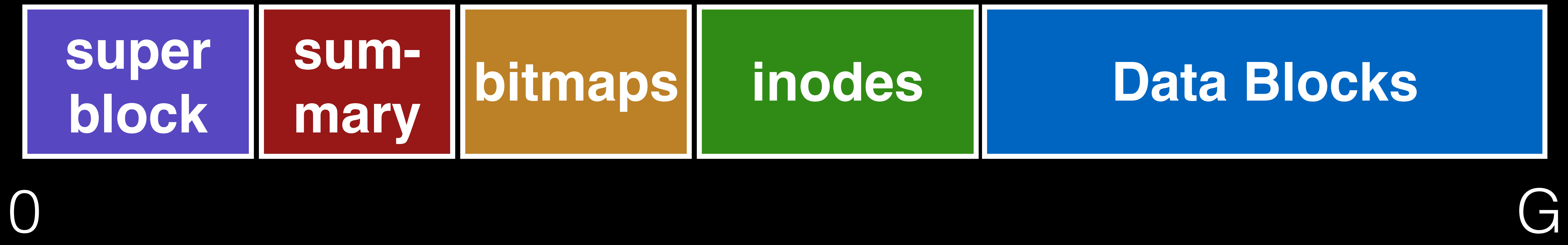
Other data blocks: allocate near previous block

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)



Group Descriptor (aka Summary Block)



how many free inodes, data blocks?

Techniques

Bitmaps
Locality groups
Rotated super
Large blocks
Fragments
Smart allocation

Conclusion

First disk-aware file system.

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!

Advice

All hardware is unique.

Treat disk like disk!

Treat flash like flash!

Treat random-access memory like random-access memory!
(actually don't -- the name is a lie)

Announcements

Exam this Friday
- 7-9pm, CHEM 1351 (same as last time)

Review this Wednesday
- 7-9pm, room TBD. Bring questions.

Office hours Monday (today)
- after class, in lab