[537] File-System Implementation

Chapter 40
Tyler Harter
11/05/14

Review File-System API

File Names

Three types of names:
- inode number
- path
- file descriptor

Why?

File Names

inode
- unique name
- remember file size, permissions, etc

path
- easy to remember
- hierarchical

file descriptor
- avoid frequent traversal
- remember multiple offsets

File API

```
int fd = open(char *path, int flag, mode_t mode)

read(int fd, void *buf, size_t nbyte)

write(int fd, void *buf, size_t nbyte)

close(int fd)
```

Special Calls

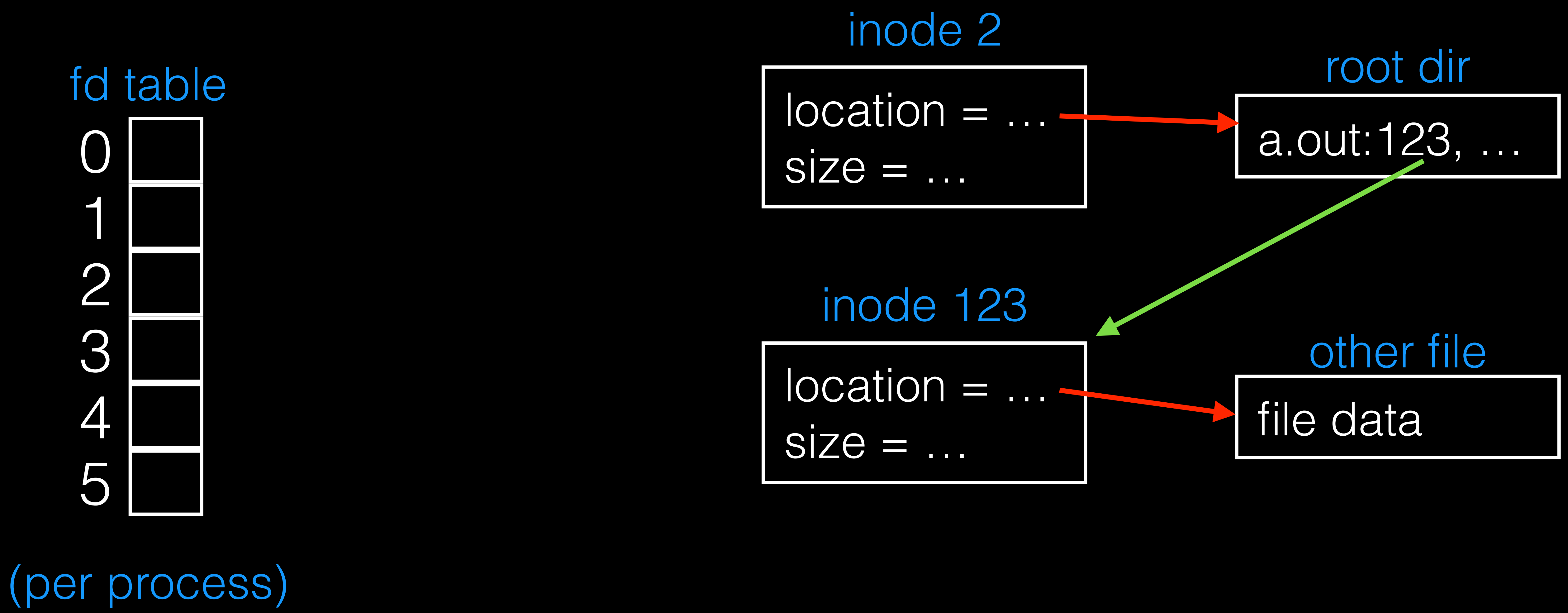
```
fsync(int fd)

rename(char *oldpath, char *newpath)

flock(int fd, int operation)
```

How do you delete a file?

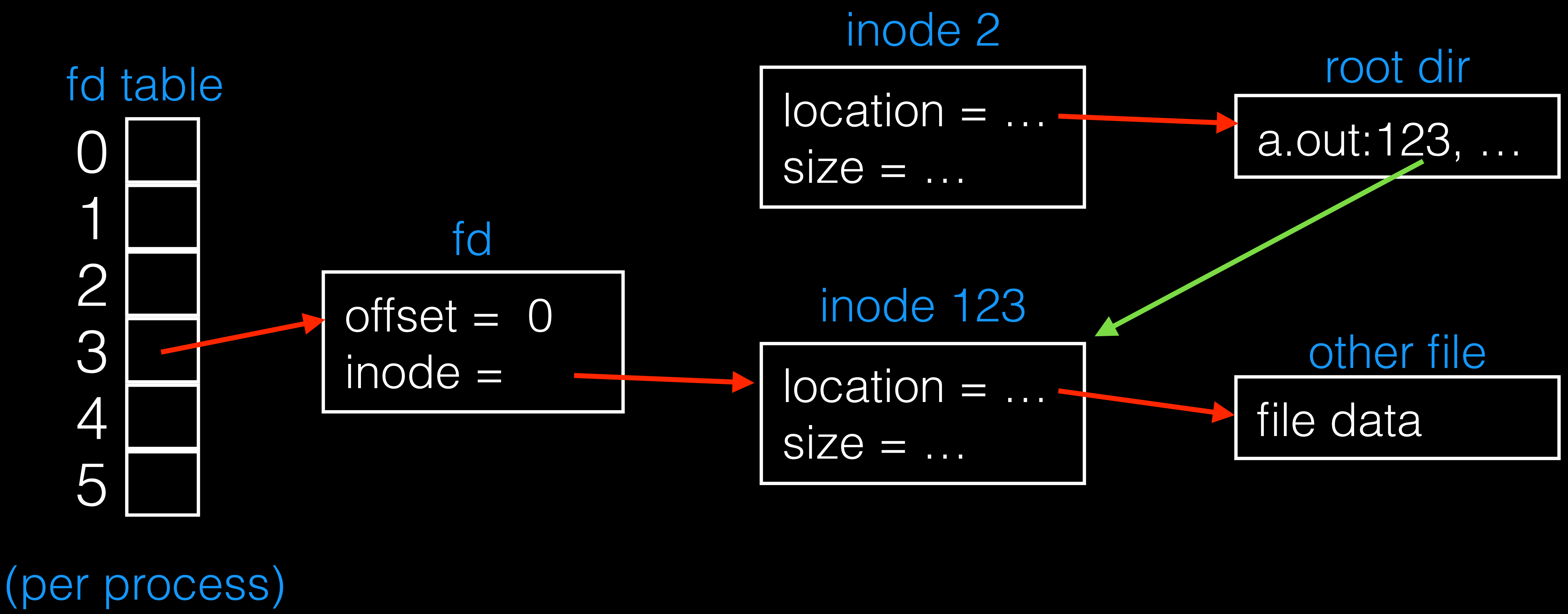
How do you delete a file?

You don’t! It’s garbage collected when there are no more names (fds or paths)

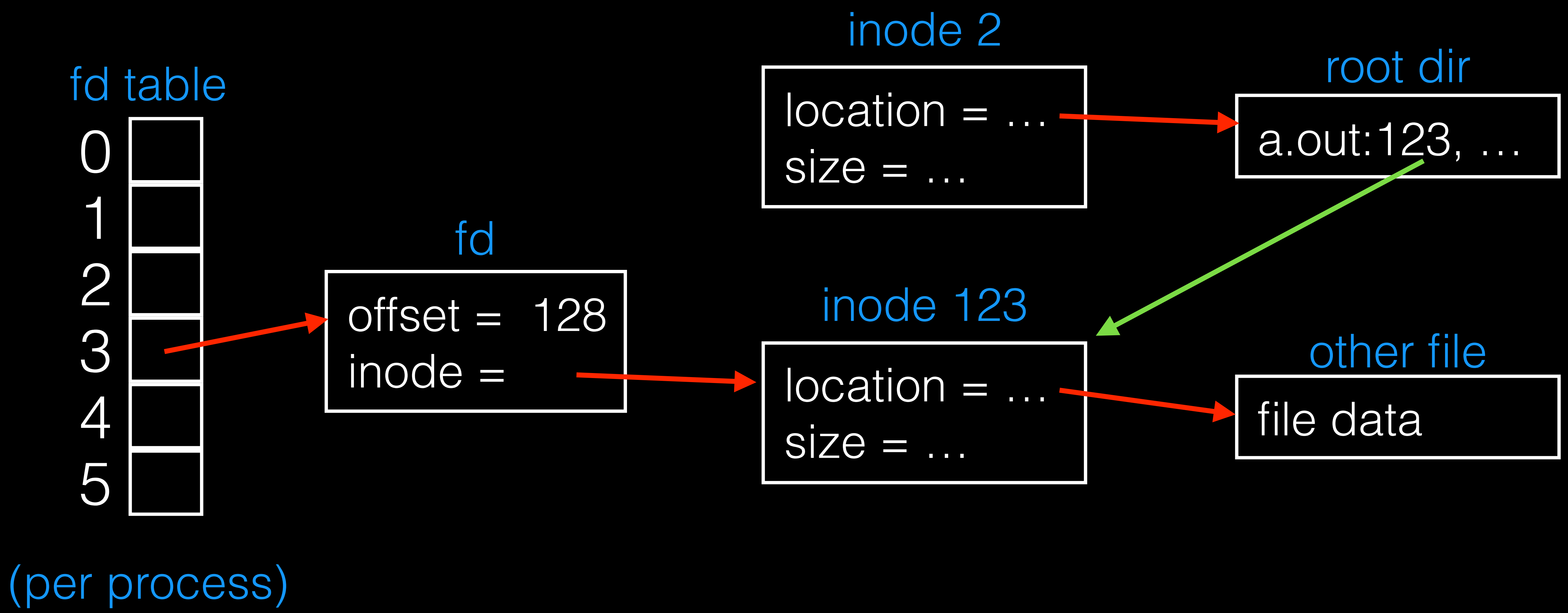
Inodes, Paths, FDs



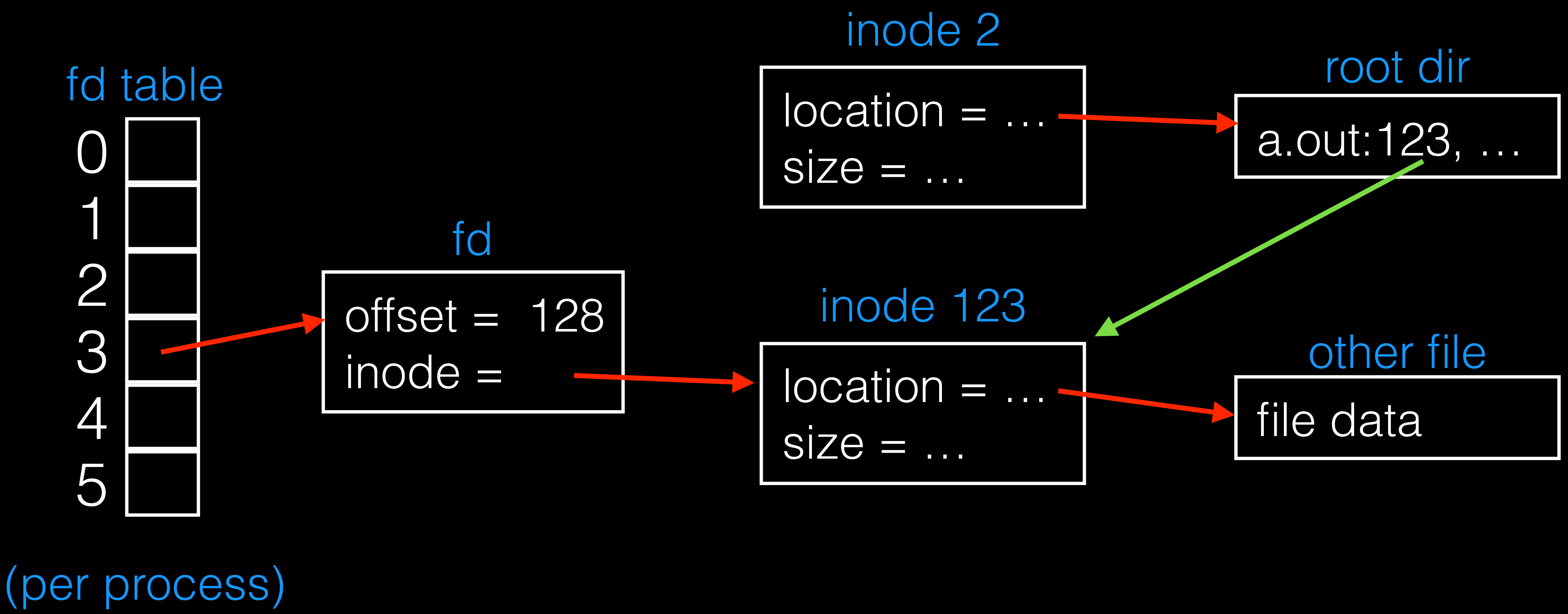
Inodes, Paths, FDs



Inodes, Paths, FDs



Inodes, Paths, FDs



opened /a.out, read 128 bytes

Implementation

Implementation

1. On-disk structures
   - how do we represent files, directories?

2. Access methods
   - what steps must reads/writes take?

Disk Structures

Persistent Store

Given: big array of bytes/blocks.
Want: to add some structure/organization.

What 537 project (so far) is most similar?

Persistent Store

Given: big array of bytes/blocks.
Want: to add some structure/organization.

What 537 project (so far) is most similar?
p3a: malloc.

You could build a persistent malloc that saves to disk (instead of to memory)!
- use offsets instead of ptrs, writes instead of stores

Persistent Malloc vs. FS

What features does a file system provide beyond what a persistent malloc would provide?

Persistent Malloc vs. FS

What features does a file system provide beyond what a persistent malloc would provide?

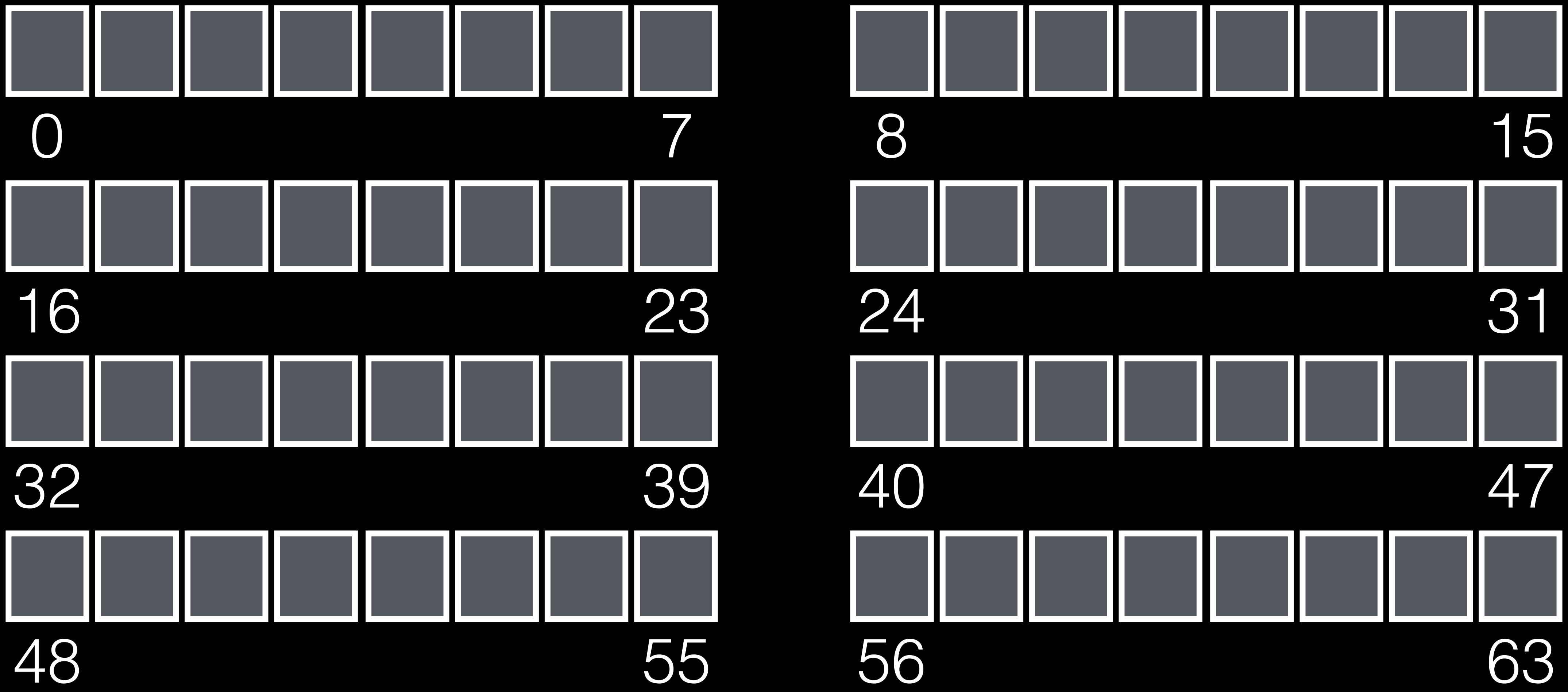
String names
Hierarchy (names within names)
Changeable file sizes
Sharing across processes
…

Structures

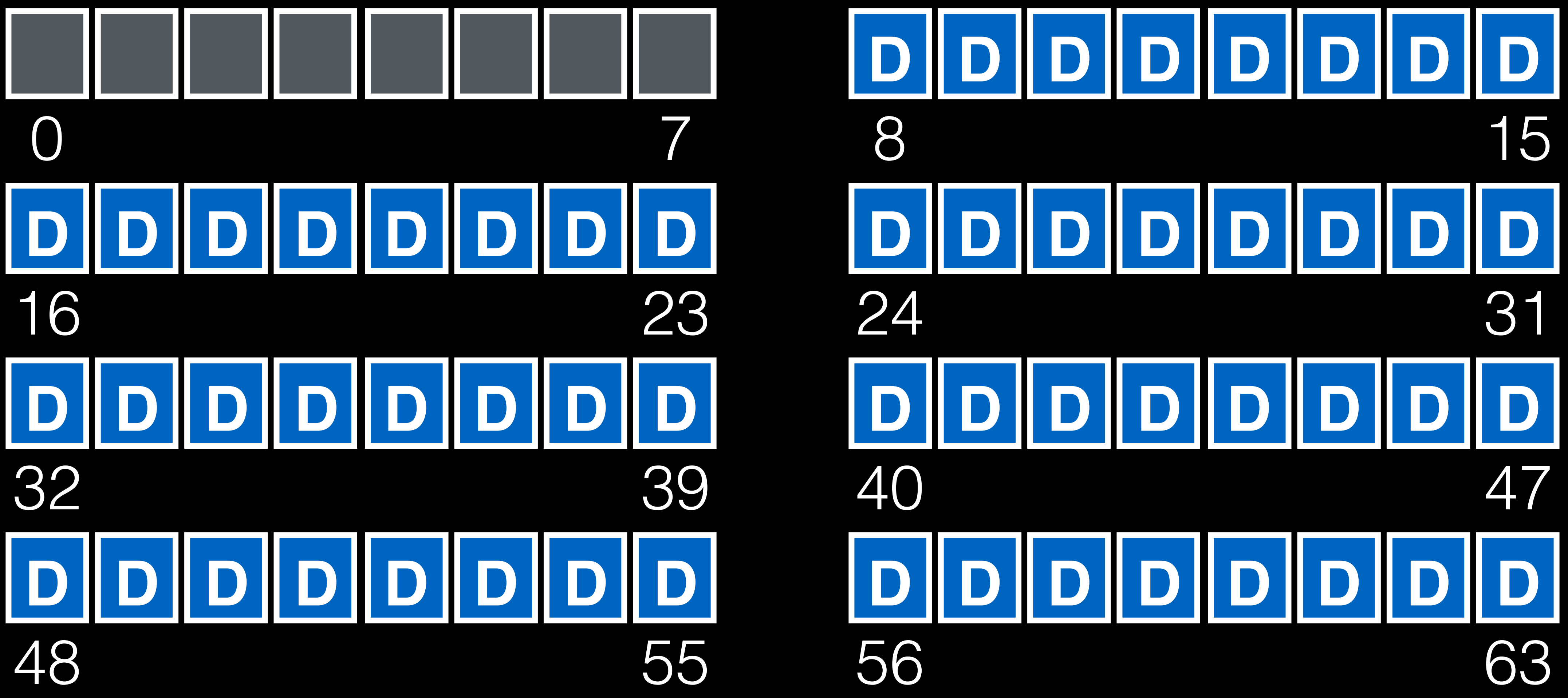
What data is likely to be read frequently?

- data block
- inode table
- indirect block
- directories
- data bitmap
- inode bitmap
- superblock

FS Structs: Empty Disk



Data Blocks



Structures

What data is likely to be read frequently?

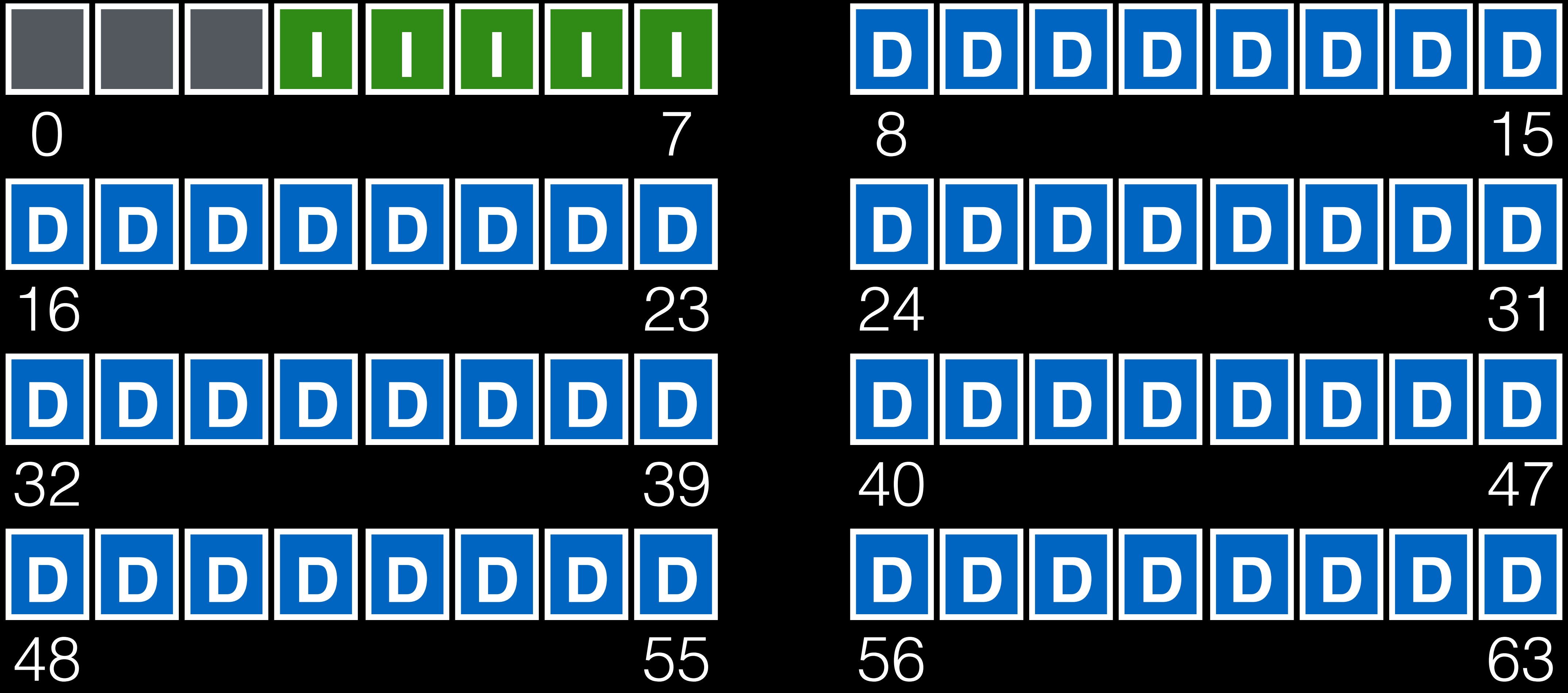
- data block
- inode table
- indirect block
- directories
- data bitmap
- inode bitmap
- superblock

Structures

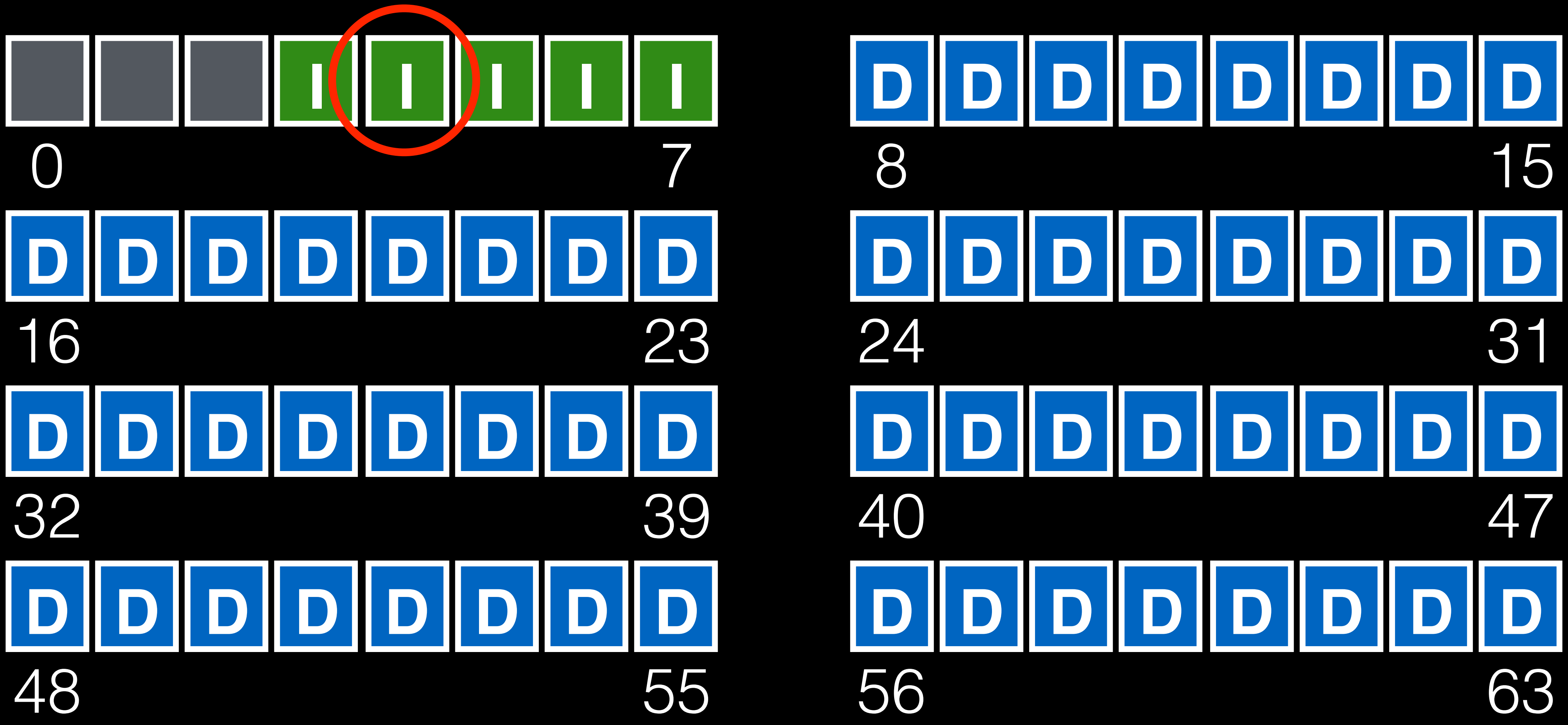
What data is likely to be read frequently?

- data block
- inode table
- indirect block
- directories
- data bitmap
- inode bitmap
- superblock

Inodes



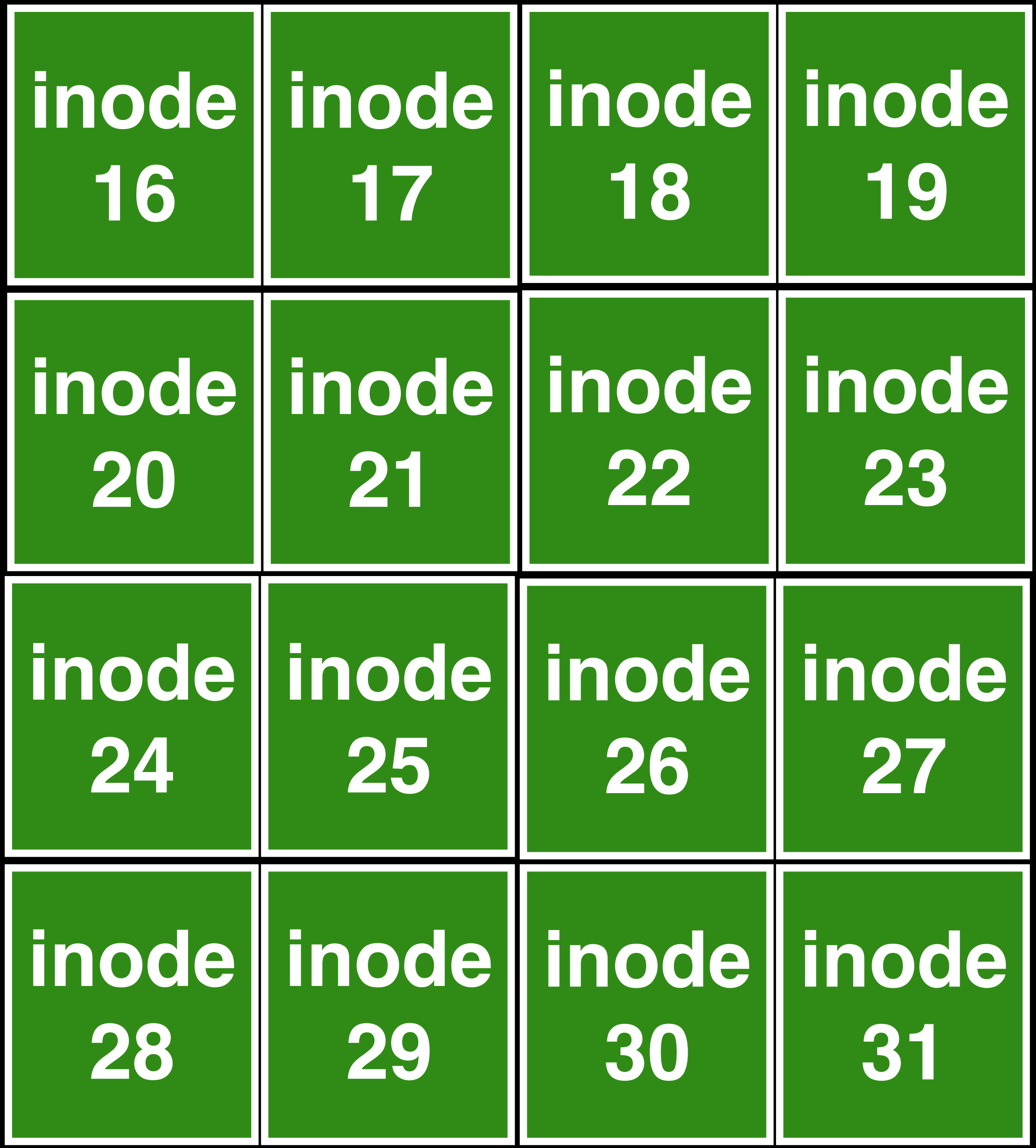
Inodes



Inode Block

Inodes are typically 128 or 256 bytes (depends on the FS).

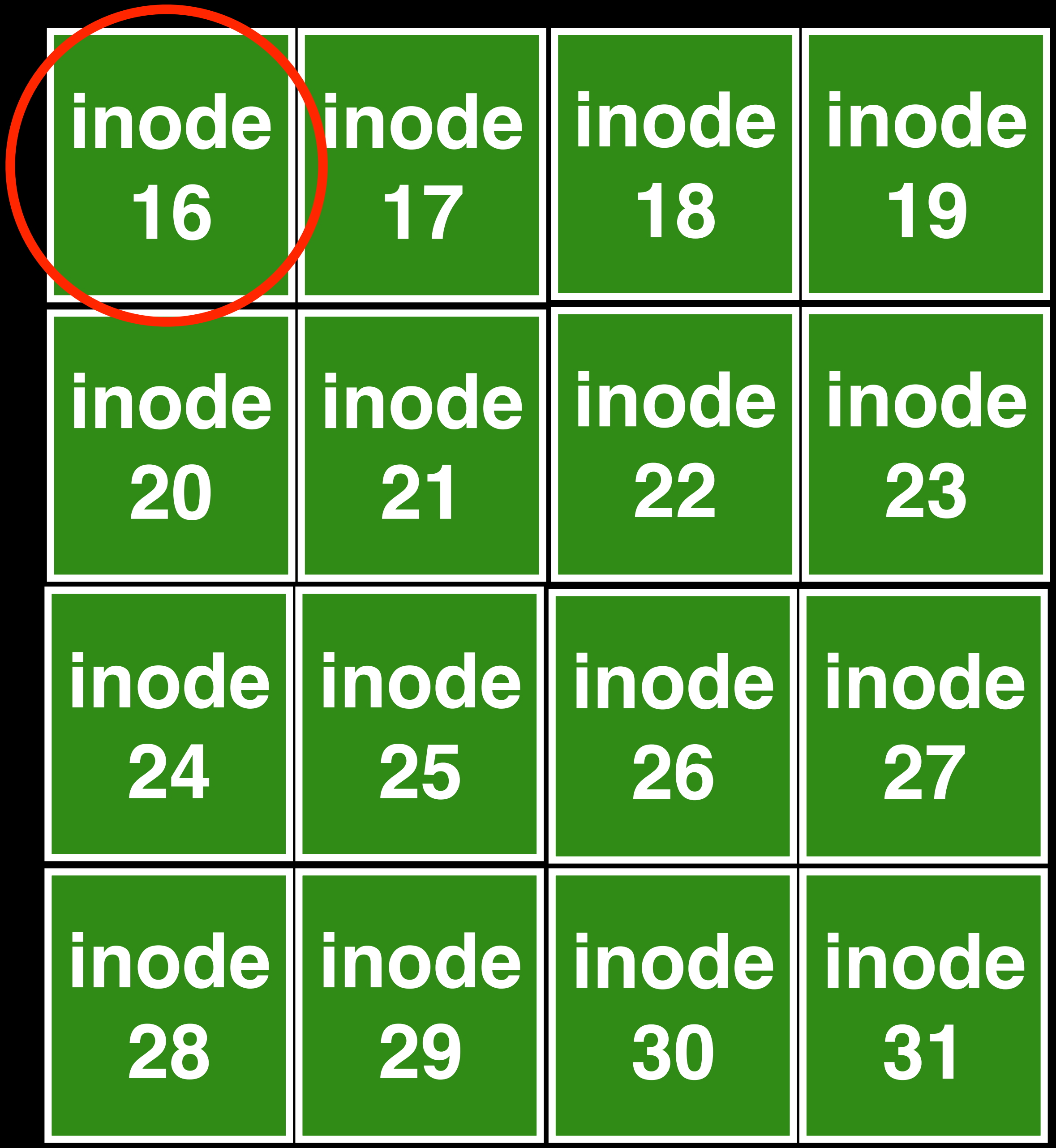
So 16 - 32 inodes per inode block.



Inode Block

Inodes are typically 128 or 256 bytes (depends on the FS).

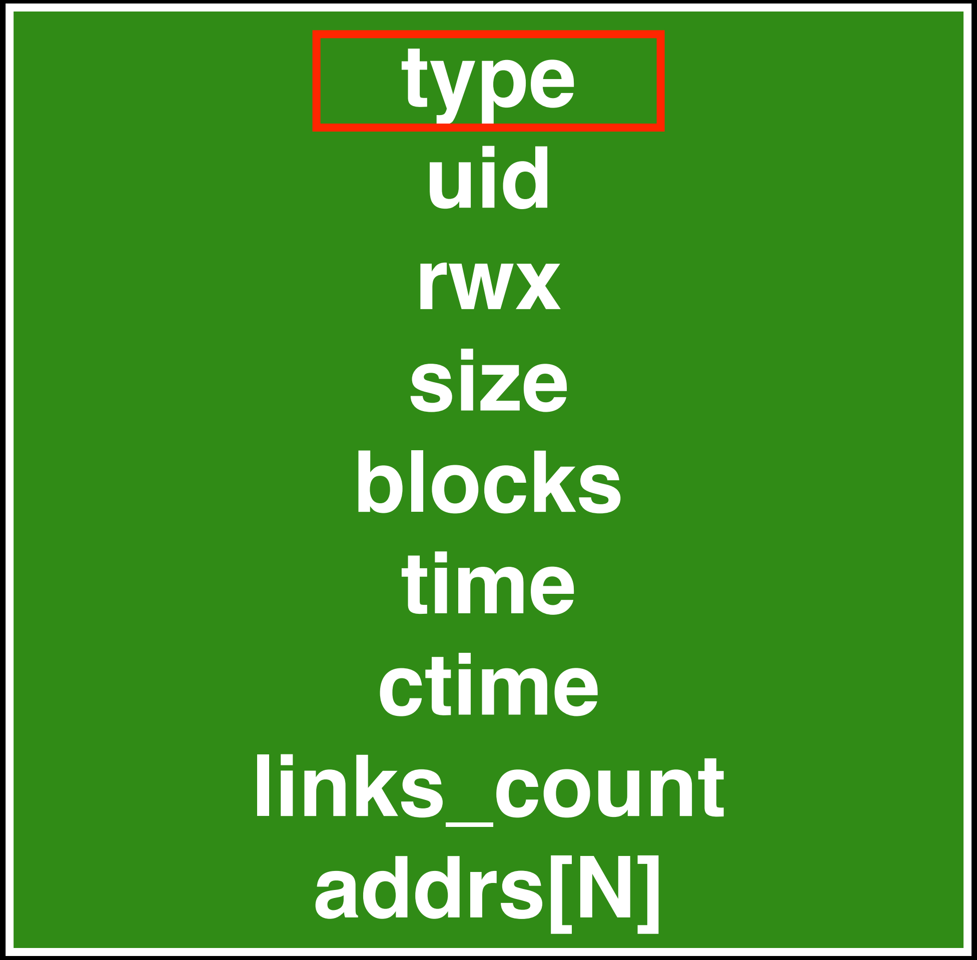
So 16 - 32 inodes per inode block.



Inode

type
uid
rwx
size
blocks
time
ctime
links_count
addrs[N]

Inode



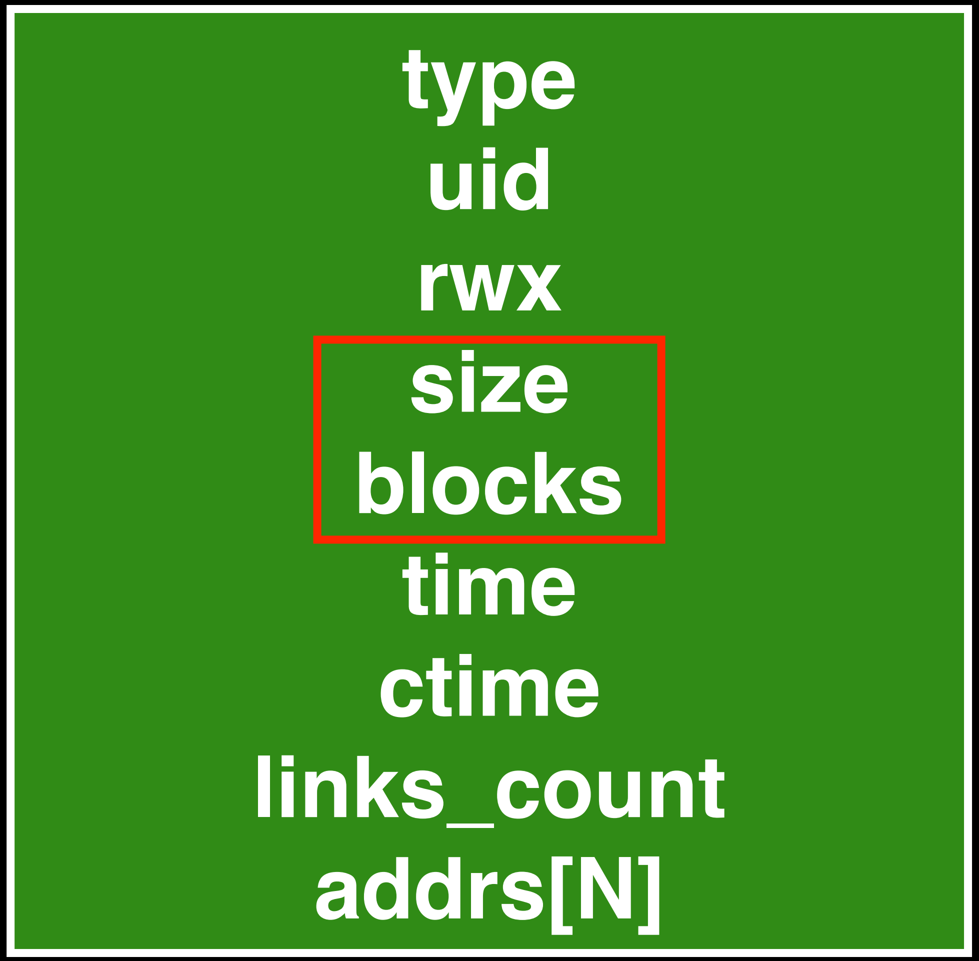
file or directory?

Inode

type
uid
rwx
size
blocks
time
ctime
links_count
addrs[N]

user and permissions

Inode



size in bytes and blocks

Inode

type
uid
rwx
size
blocks
time
ctime
links_count
addrs[N]

access time, create time

Inode

type
uid
rwx
size
blocks
time
ctime
links_count
addrs[N]

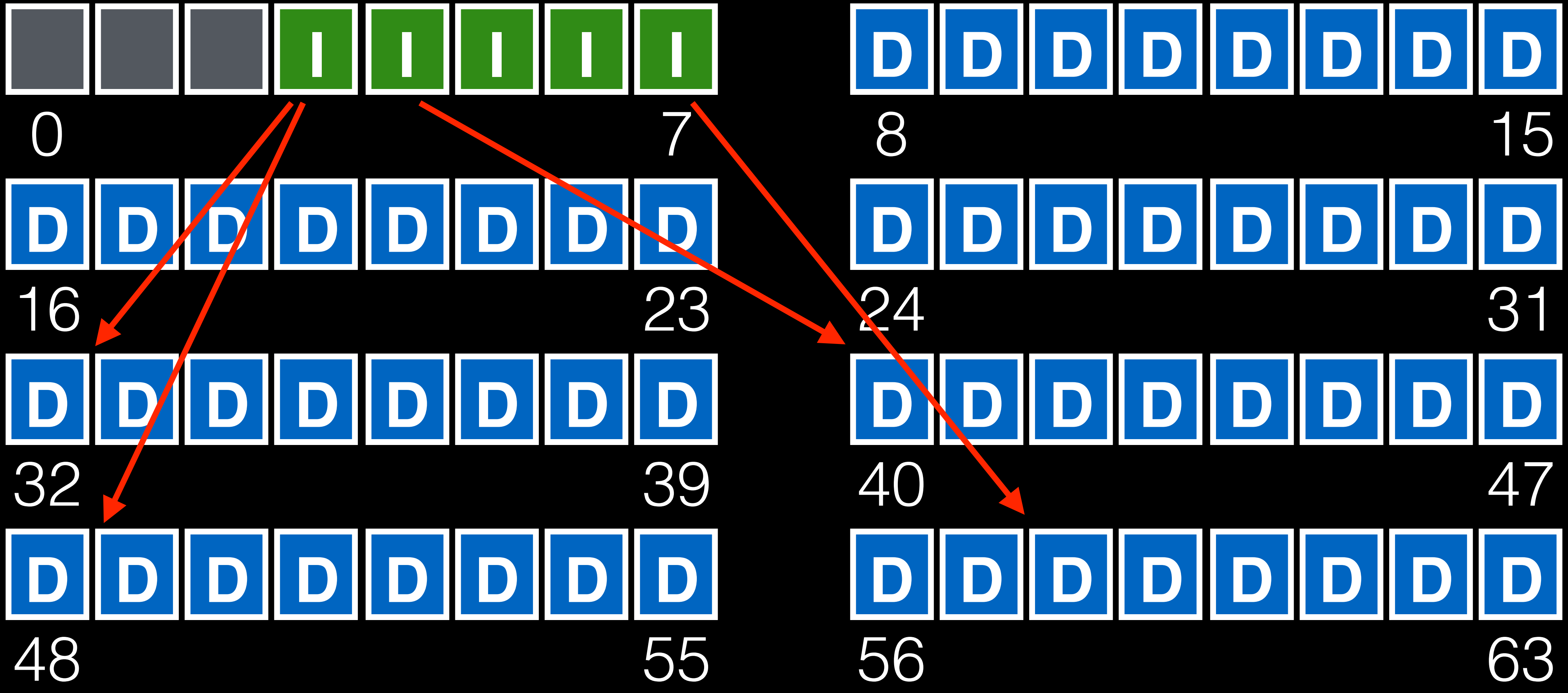
how many paths

Inode

type
uid
rwx
size
blocks
time
ctime
links_count
addrs[N]

N data blocks

Inodes



Inode

type
uid
rwx
size
blocks
time
ctime
links_count
addrs[N]

Assume 4-byte addrs.
What is an upper bound on the file size?
(assume 256-byte inodes)

Inode

type
uid
rwx
size
blocks
time
ctime
links_count
addrs[N]

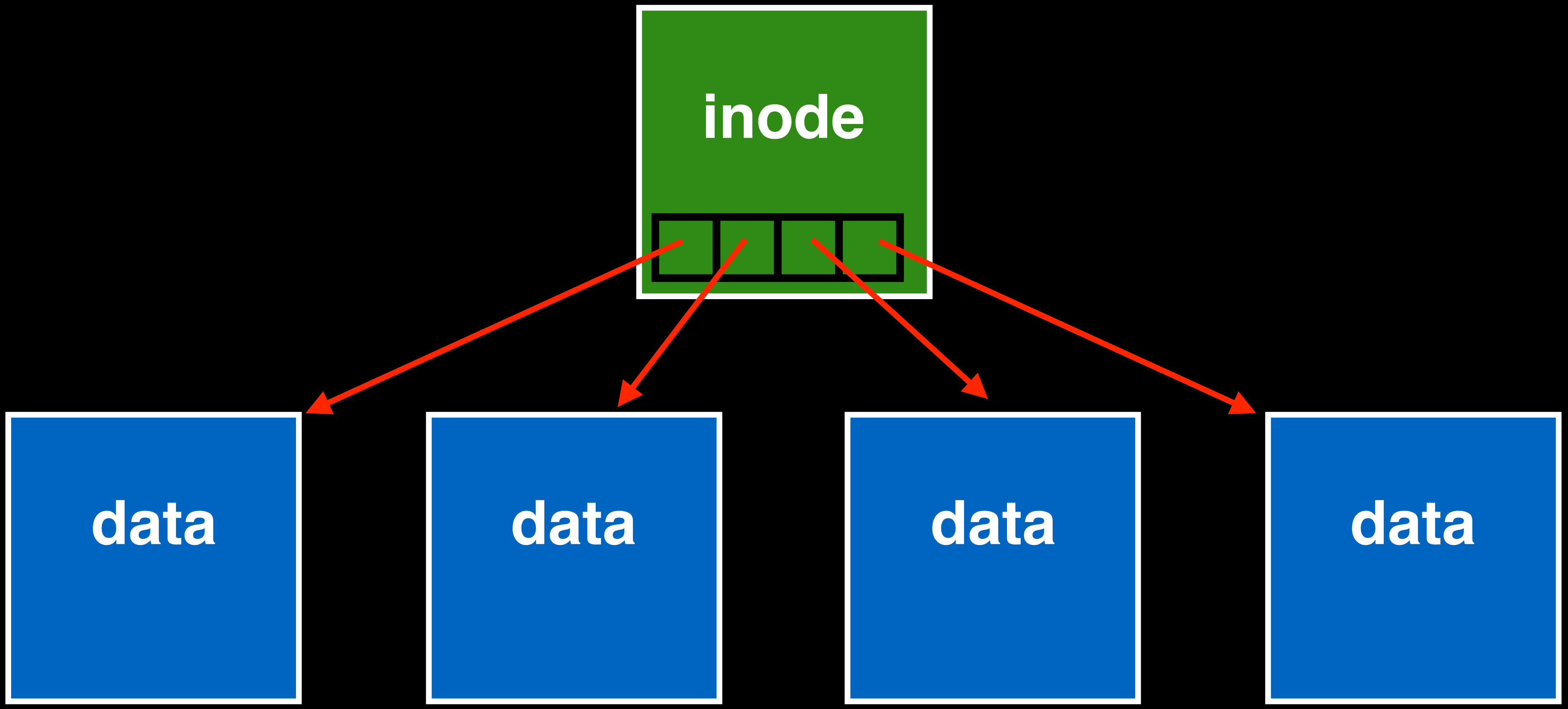
Assume 4-byte addrs.
What is an upper bound on the file size?
(assume 256-byte inodes)

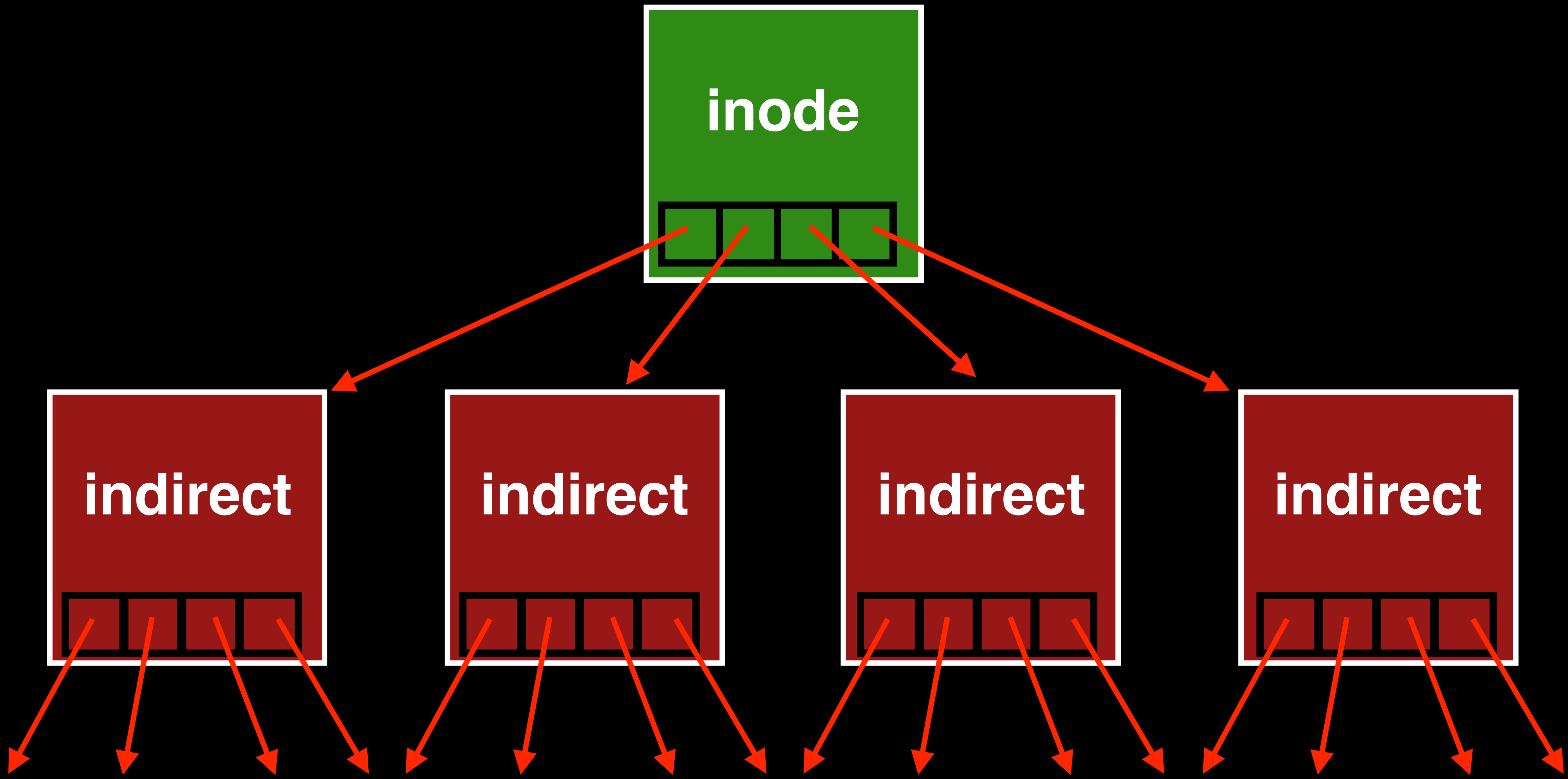
How to get larger files?

Structures

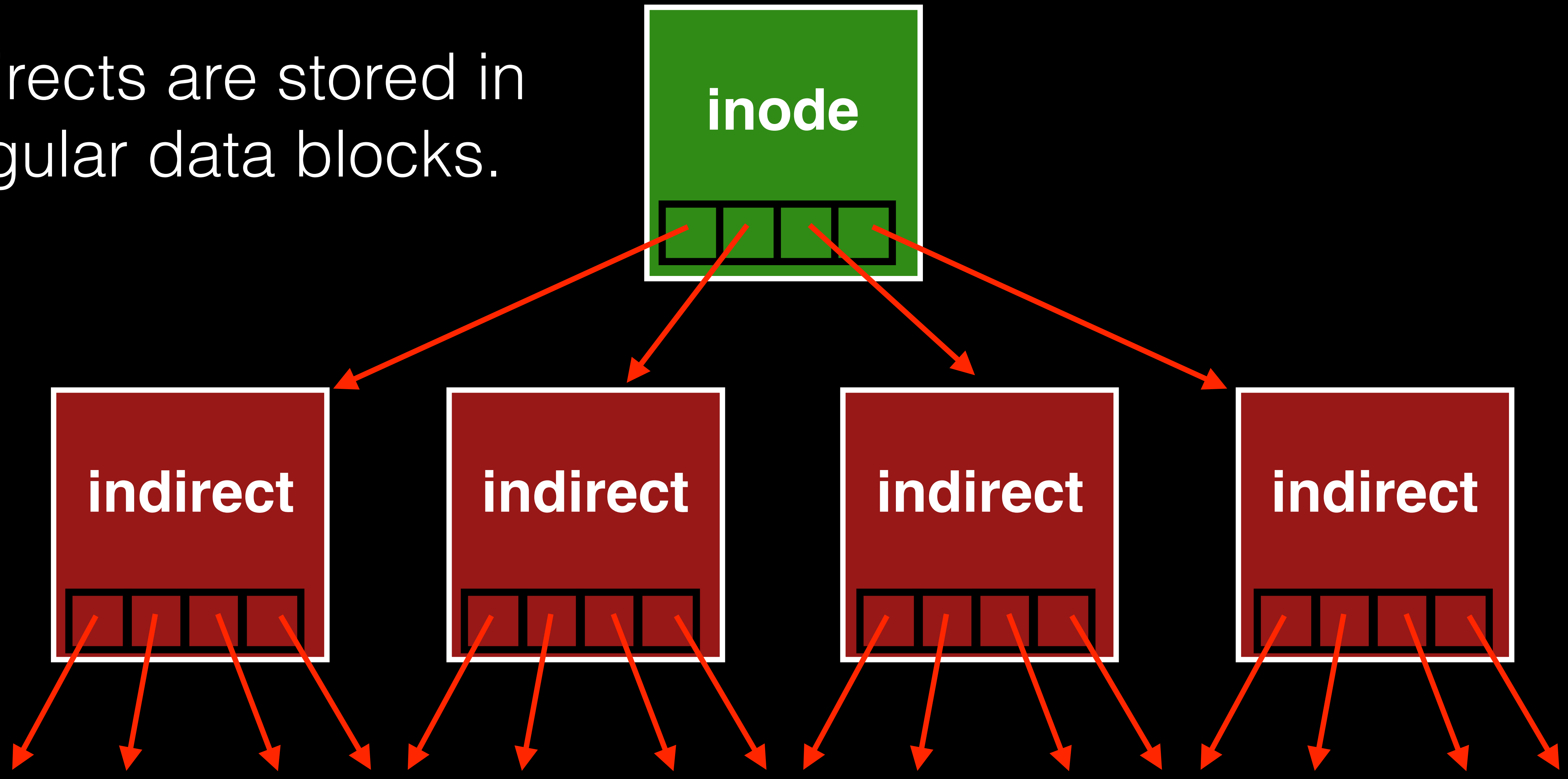
What data is likely to be read frequently?

- data block
- inode table
- indirect block
- directories
- data bitmap
- inode bitmap
- superblock

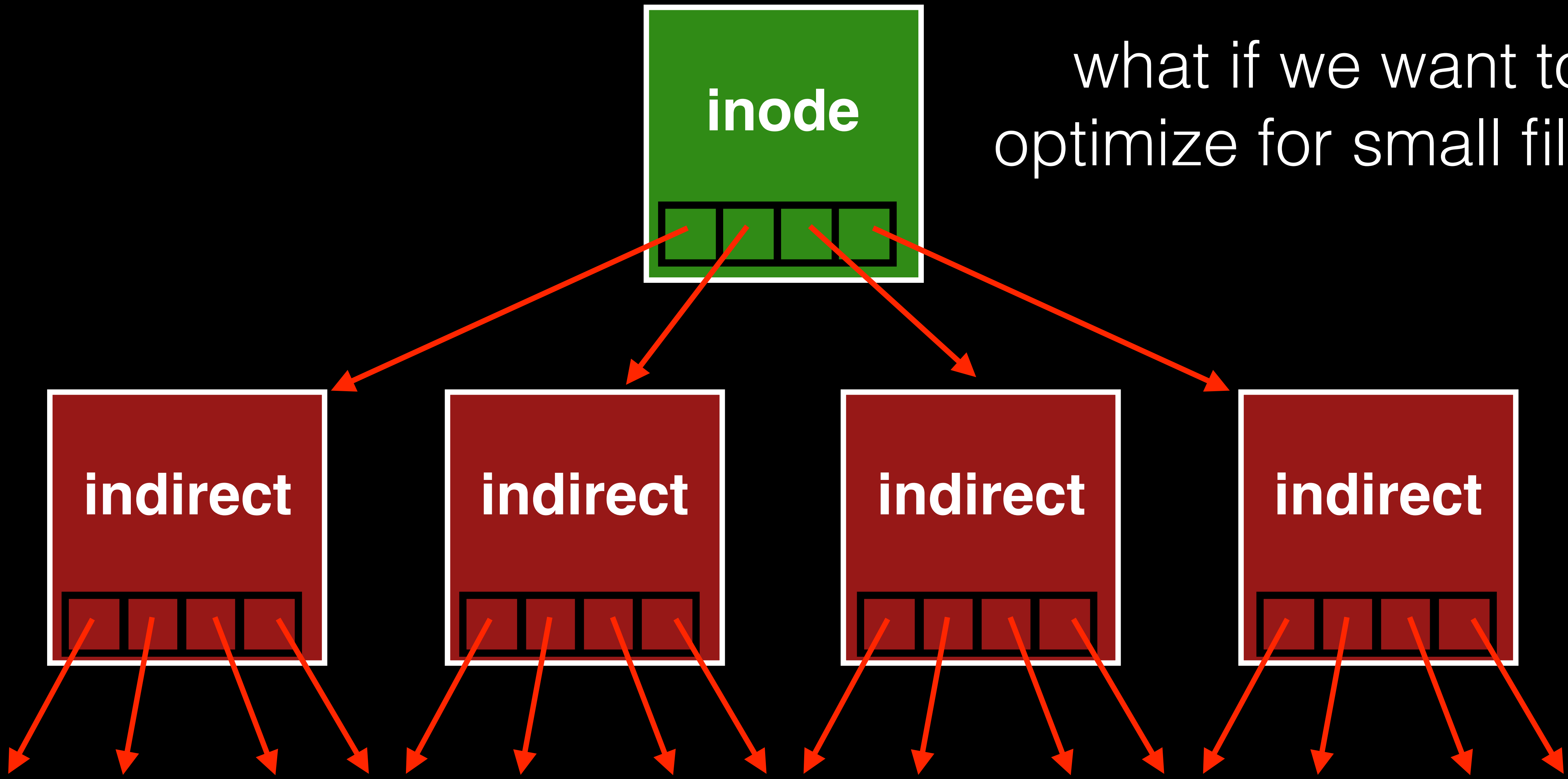

inode
data
data
data
data


inode
indirect
indirect
indirect
indirect

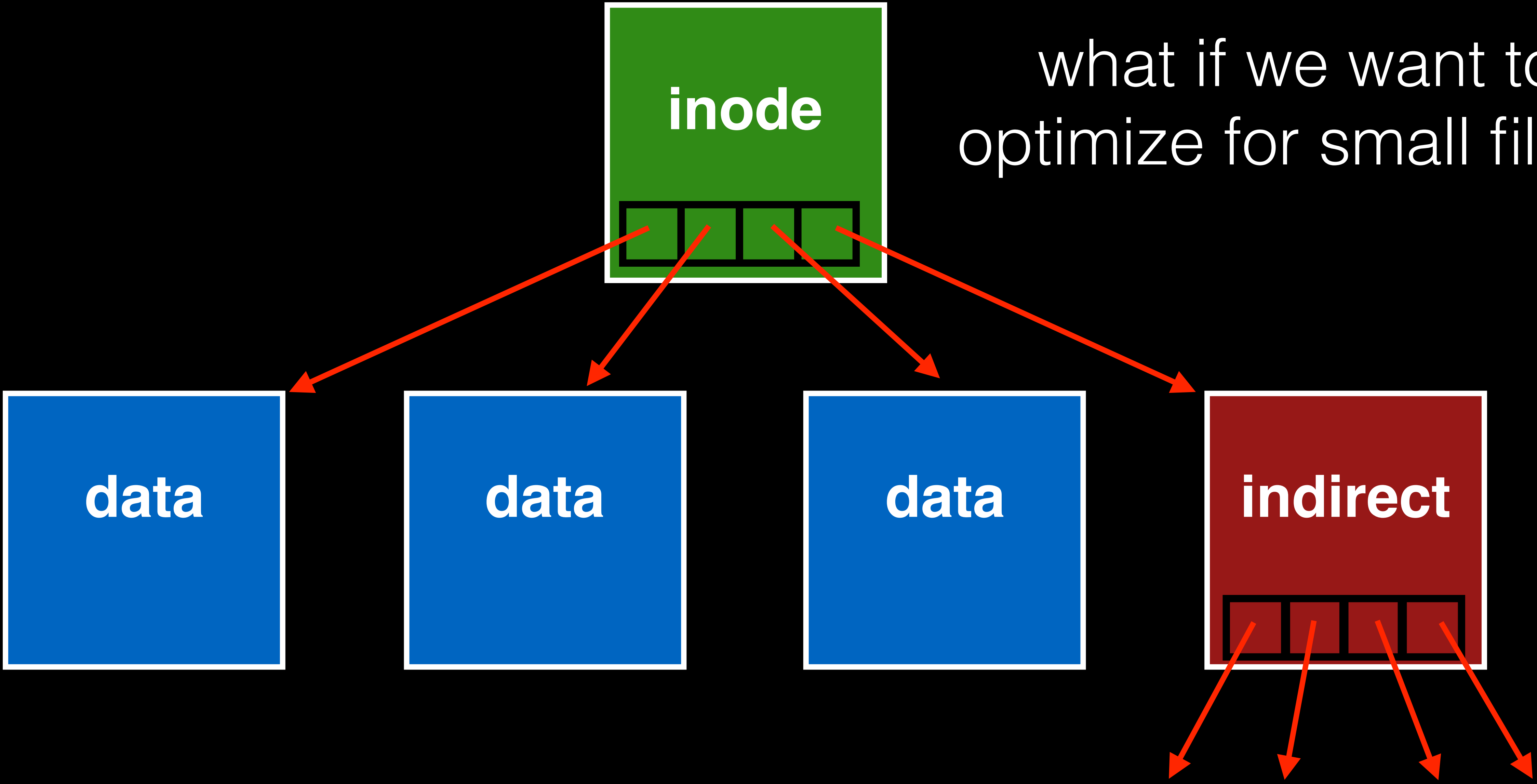
indirects are stored in regular data blocks.



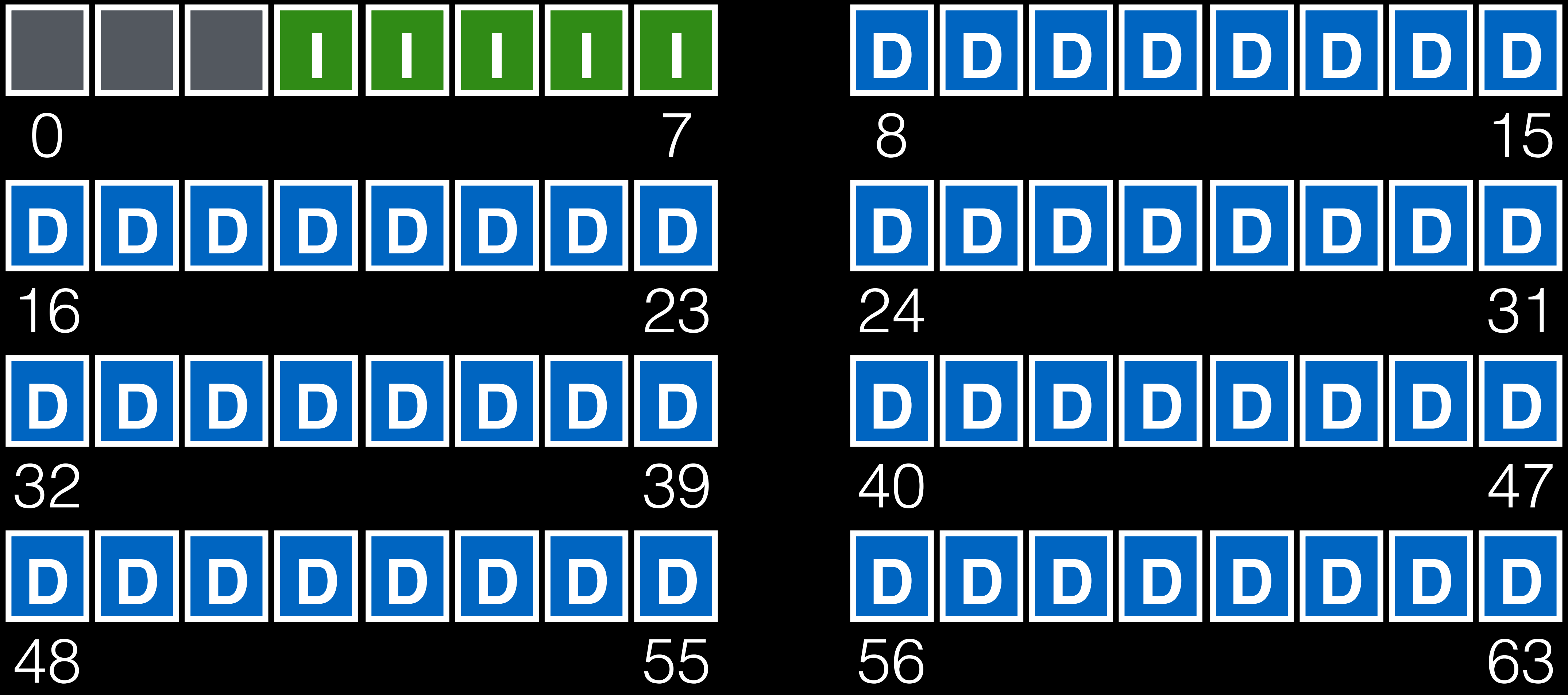
what if we want to optimize for small files?



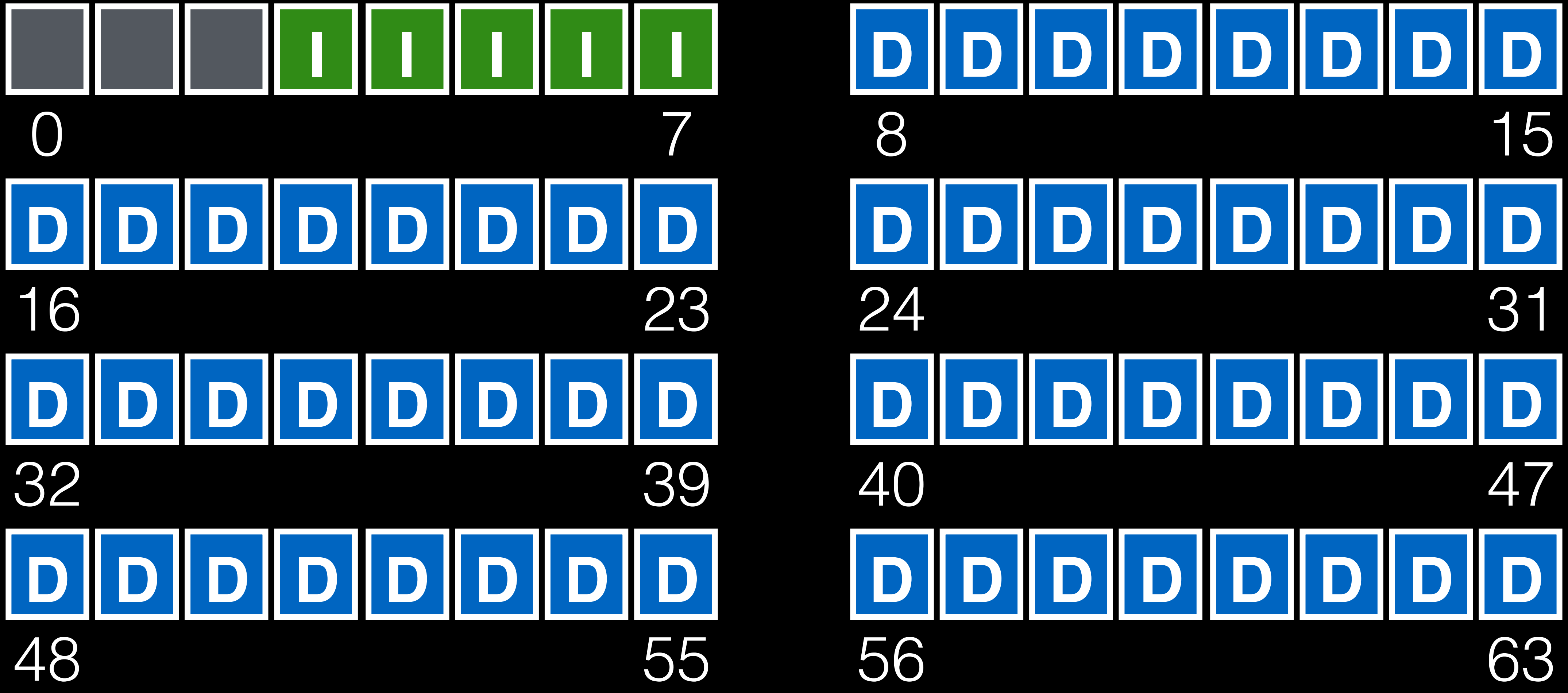
what if we want to optimize for small files?



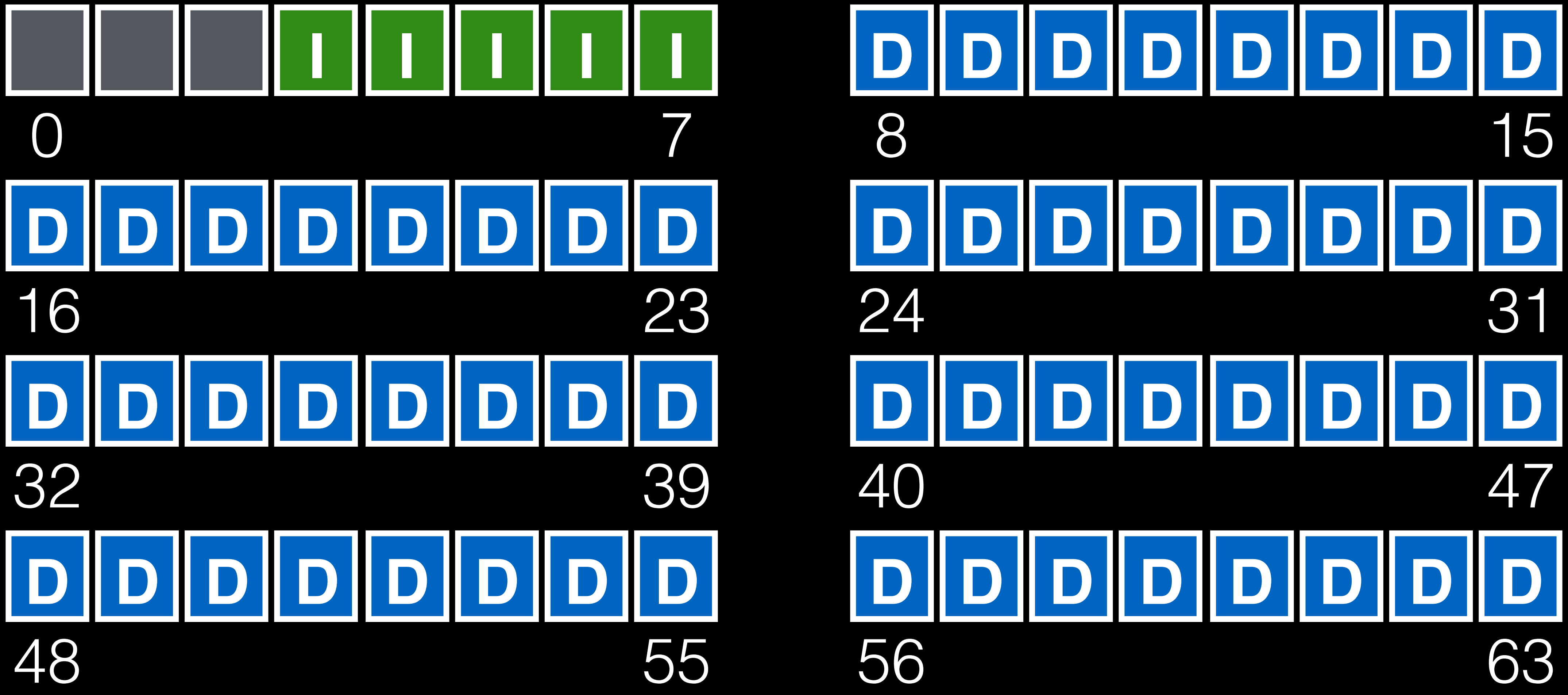
Assume 256 byte sectors. What is offset for inode with number 0?



Assume 256 byte sectors. What is offset for inode with number 4?



Assume 256 byte sectors. What is offset for inode with number 40?



Various Link Structures

Tree (usually unbalanced)
- with indirect blocks
- e.g., ext3

Extents
- store offset+size pairs
- e.g., ext4

Linked list
- each data block points to the next
- e.g., FAT

Structures

What data is likely to be read frequently?

- data block
- inode table
- indirect block
- directories
- data bitmap
- inode bitmap
- superblock

Directories

File systems vary.

Common design: just store directory entries in files.

Various formats could be used
- lists
- b-trees

Simple List Example

valid	name	inode
1	.	134
1	..	35
1	foo	80
1	bar	23

Simple List Example

valid	name	inode
1	.	134
1	..	35
0	foo	80
1	bar	23

unlink("foo")

Structures

What data is likely to be read frequently?

- data block
- inode table
- indirect block
- directories
- data bitmap
- inode bitmap
- superblock

Allocation

How do we find free data blocks or free inodes?

Allocation

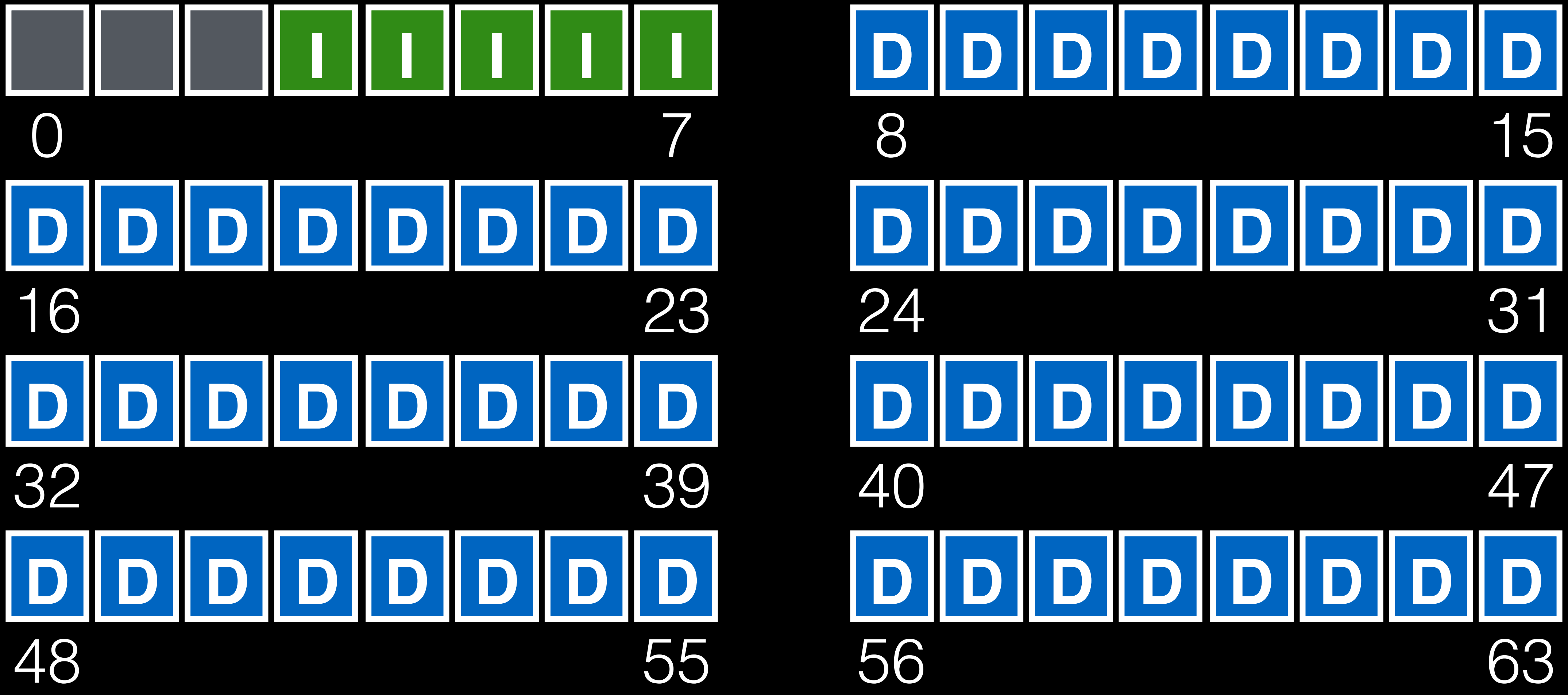
How do we find free data blocks or free inodes?

Free list.

Bitmaps.

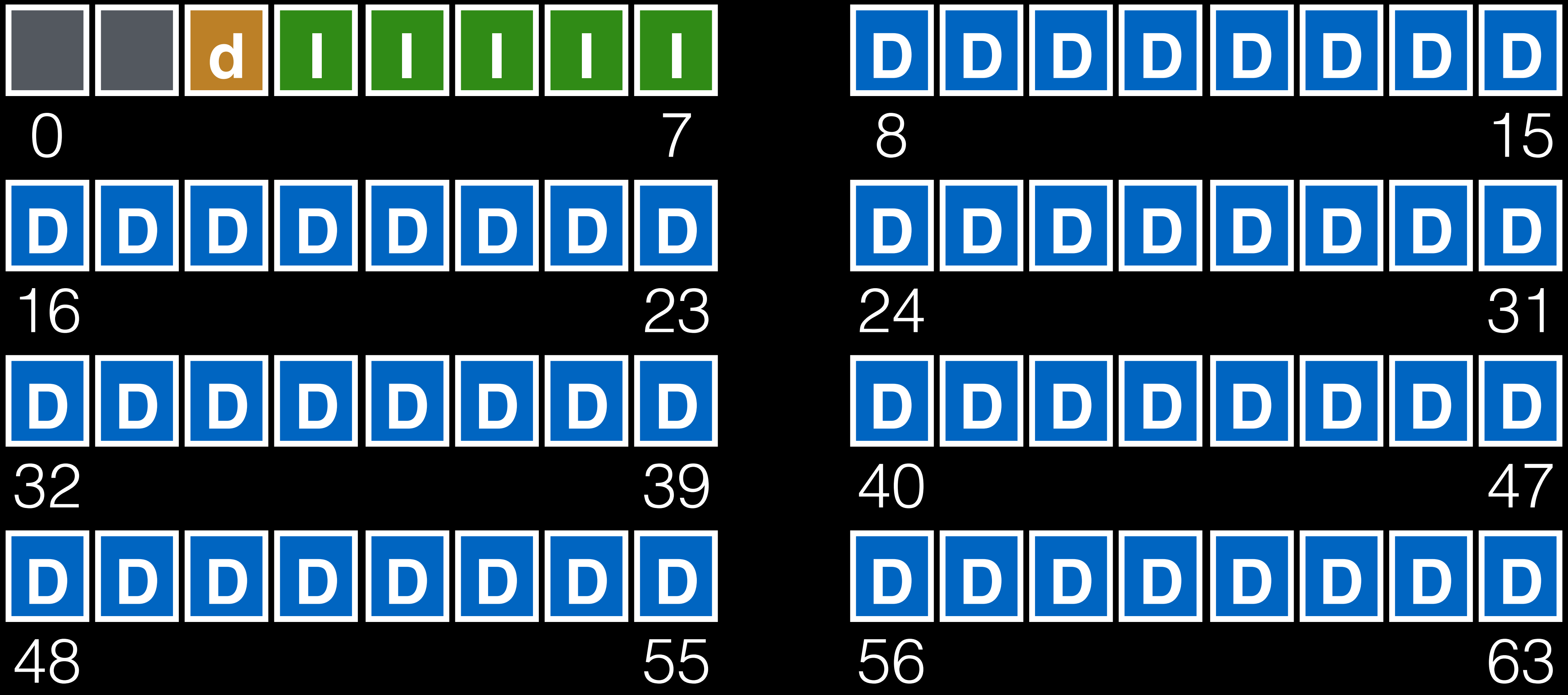
Tradeoffs?

Bitmaps

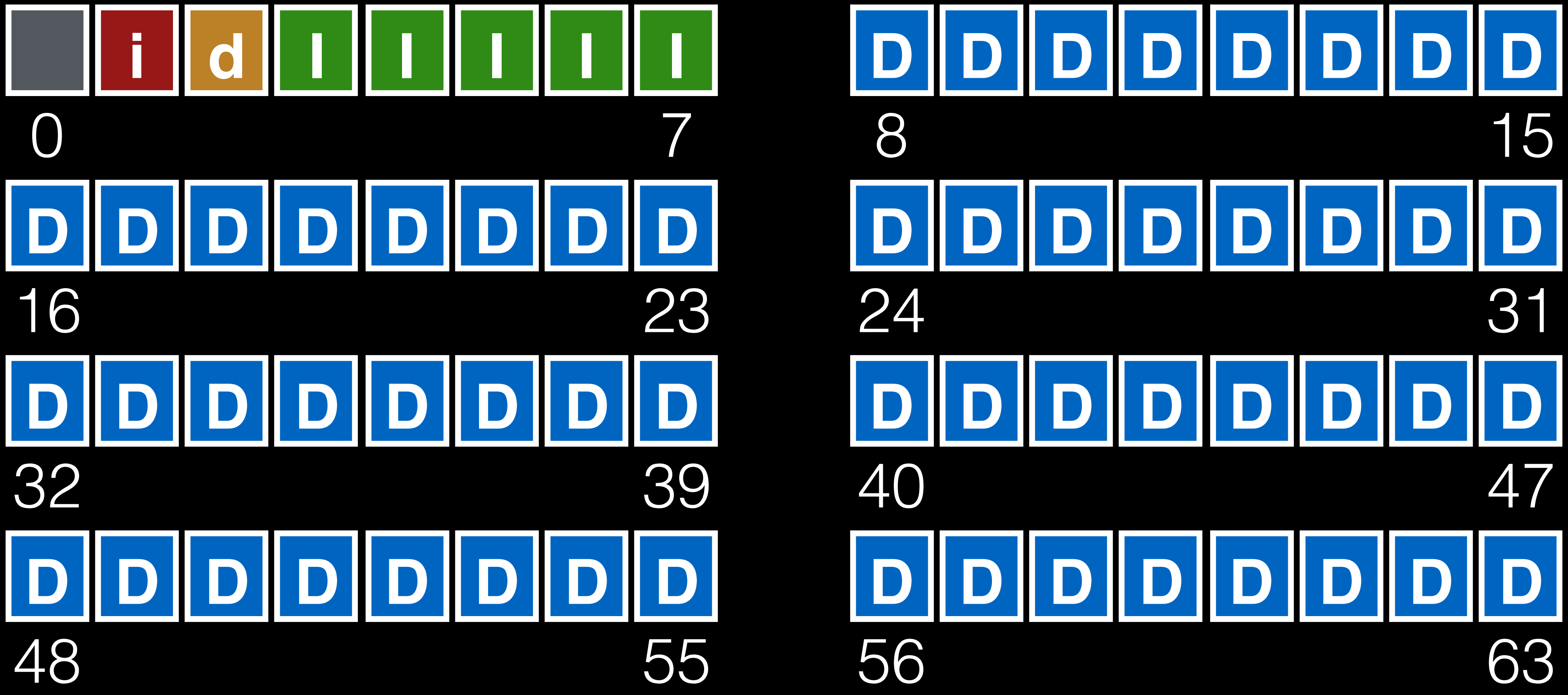


I I I I I
0
7
D D D D D D D D
8
15
D D D D D D D D
16
23
D D D D D D D D
24
31
D D D D D D D D
32
39
D D D D D D D D
40
47
D D D D D D D D
48
55
D D D D D D D D
56
63

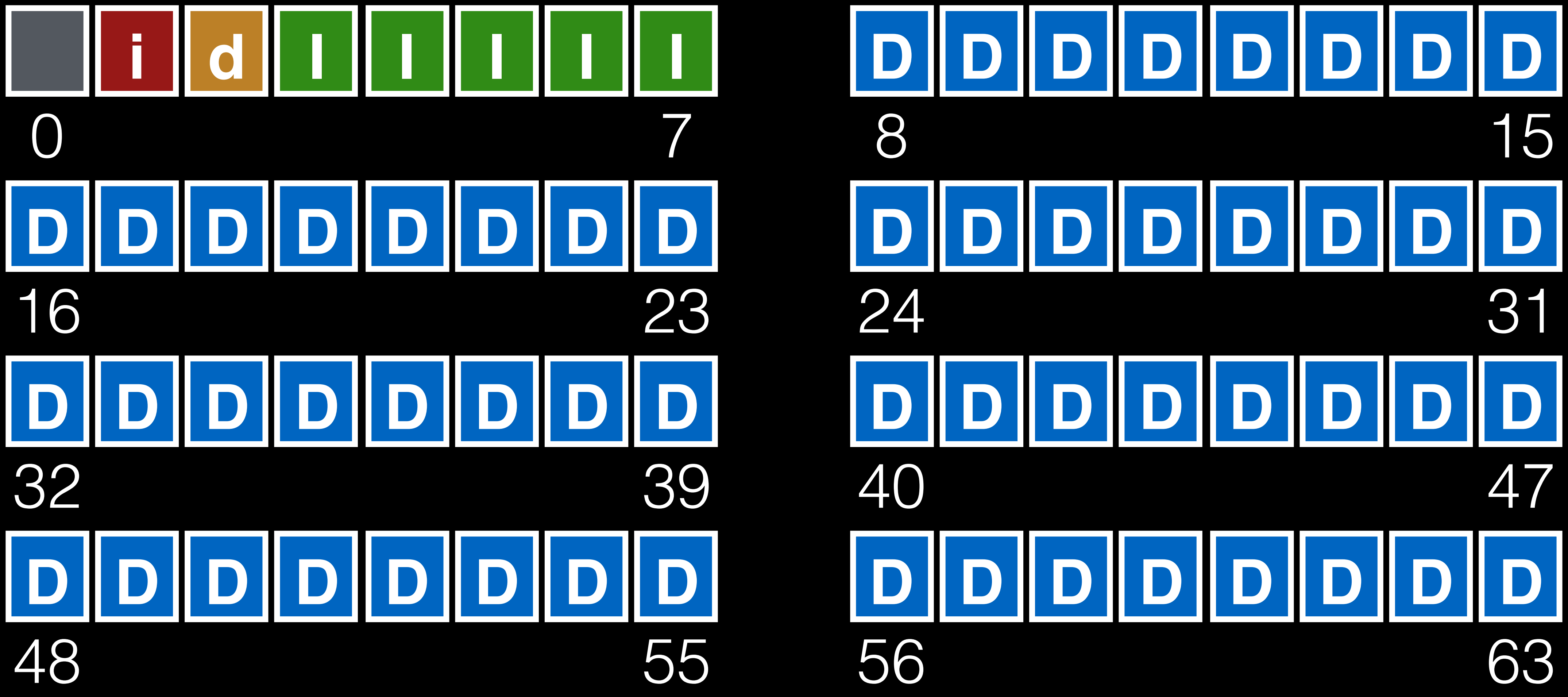
Data Bitmap



Inode Bitmap



Opportunity for Inconsistency (fsck)



Structures

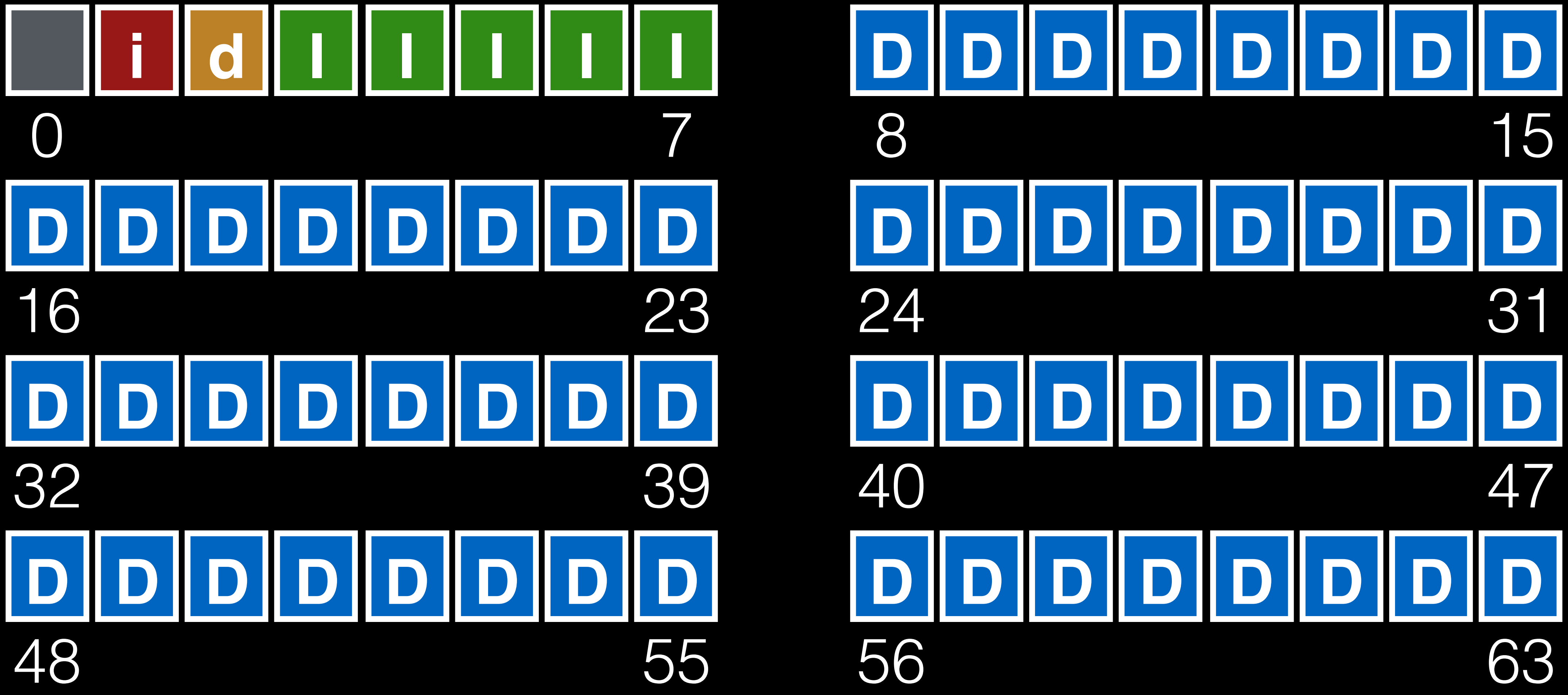
What data is likely to be read frequently?
- data block
- inode table
- indirect block
- directories
- data bitmap
- inode bitmap
- superblock

Superblock

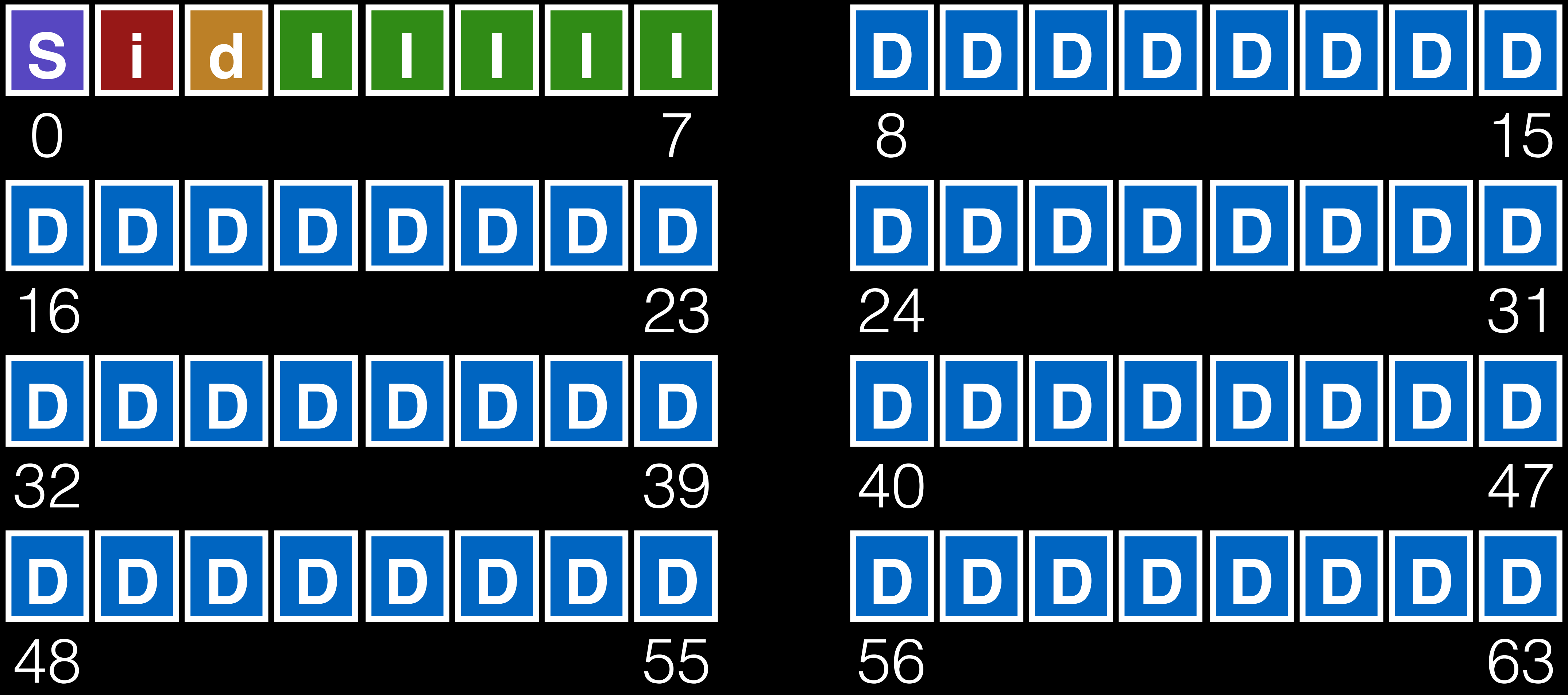
Need to know basic FS metadata, like:
- block size
- how many inodes are there
- how much free data

Store this in a superblock

Super Block



Super Block



Structure Overview

Structures:

- superblock
- data block
- data bitmap
- inode table
- inode bitmap
- indirect block
- directories

Structure Overview

Core

Performance

Super Block

Structure Overview

Core

Performance

Super Block

Data Block

Structure Overview

Core

Super Block

Data Block

Performance

Data Bitmap

Structure Overview

Core

Super Block

Data Block

Inode Table

Performance

Data Bitmap

Structure Overview

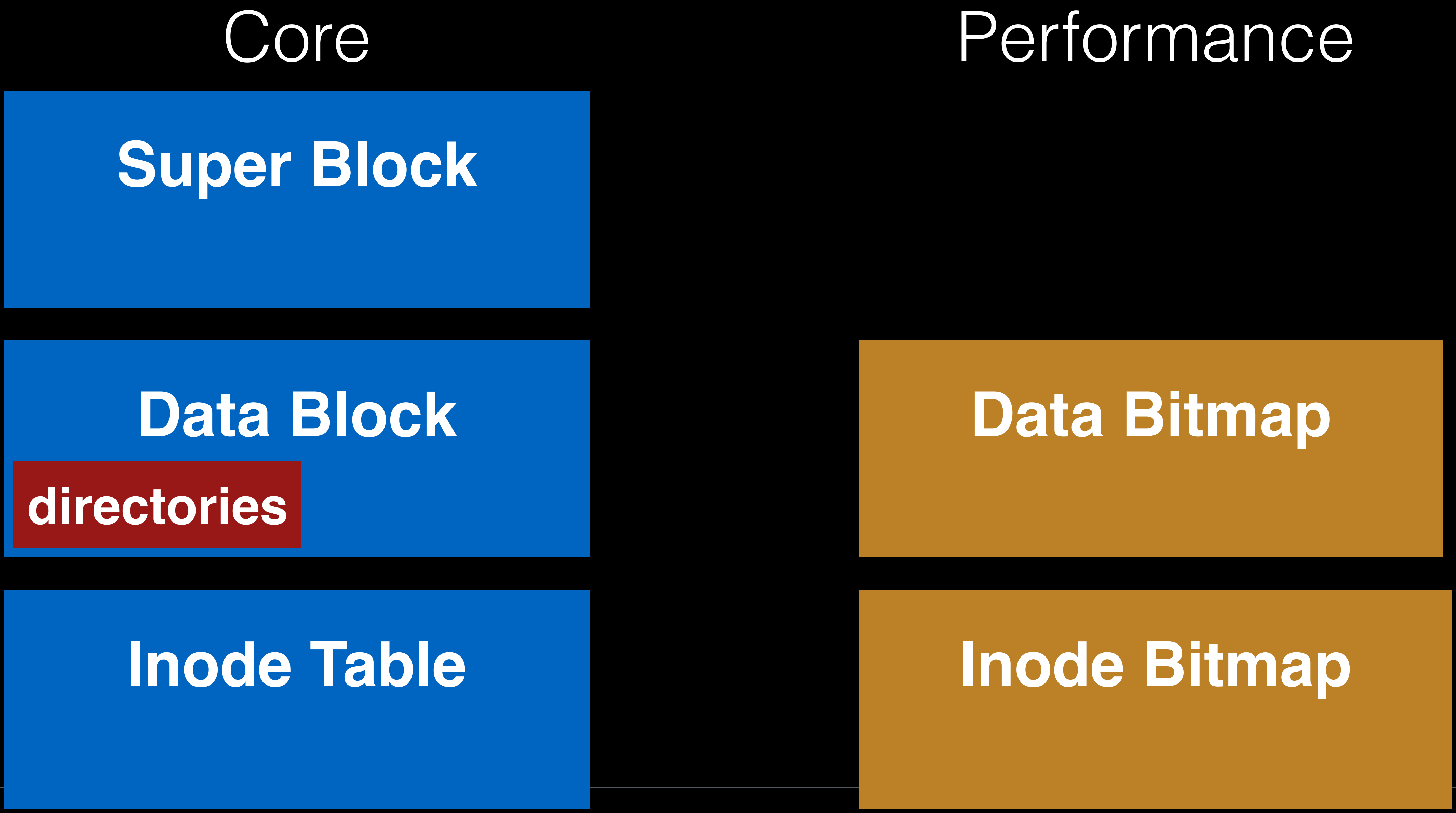
Core

- Super Block
- Data Block
- Inode Table

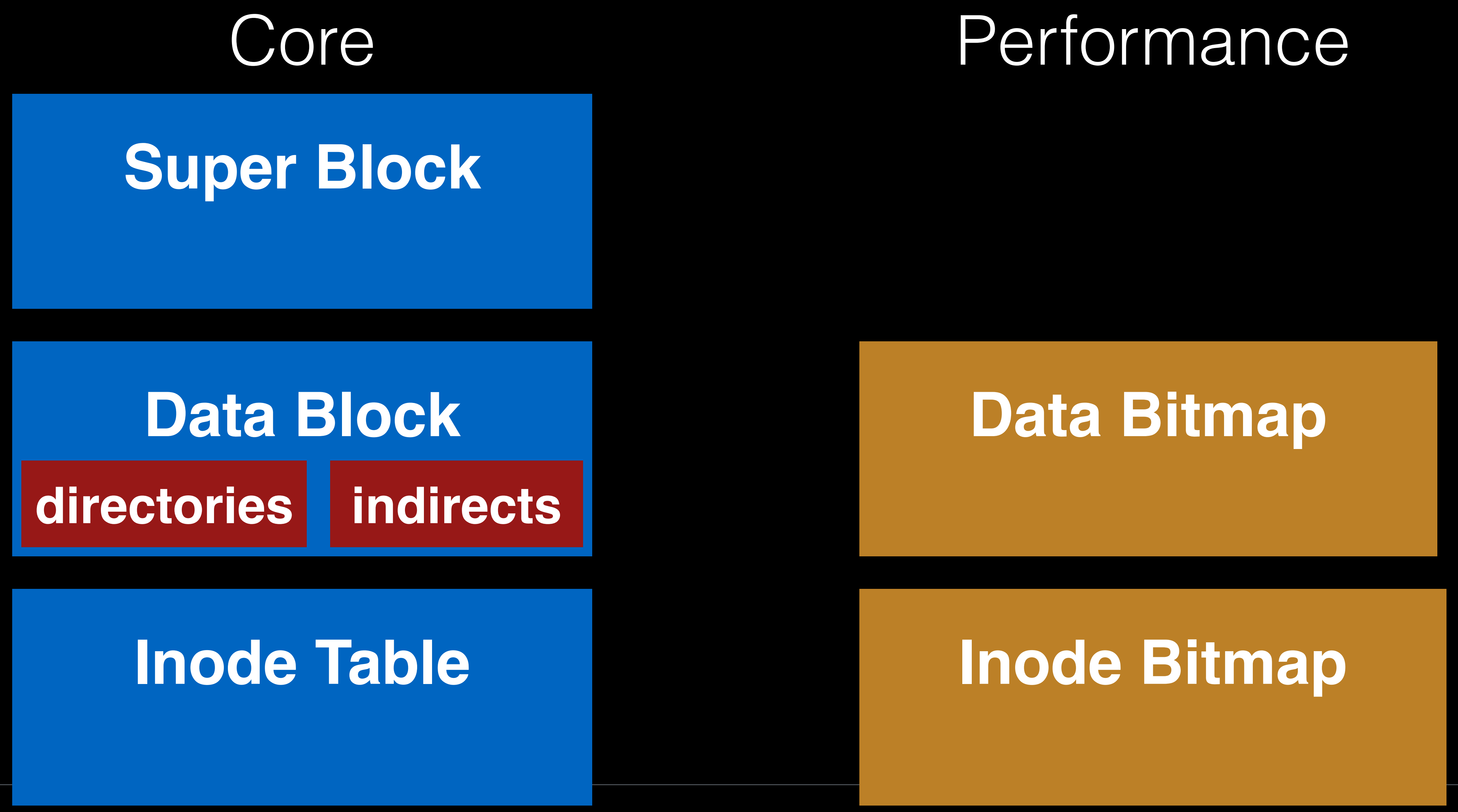
Performance

- Data Bitmap
- Inode Bitmap

Structure Overview



Structure Overview



Operations

Operations

FS
- mkfs
- mount

File
- create
- write
- open
- read
- close

Operations

FS
- mkfs
- mount

File
- create
- write
- open
- read
- close

mkfs

Different version for each file system
(e.g., mkfs.ext4, mkfs.xfs, mkfs.btrfs, etc)

Initialize metadata (bitmaps, inode table).

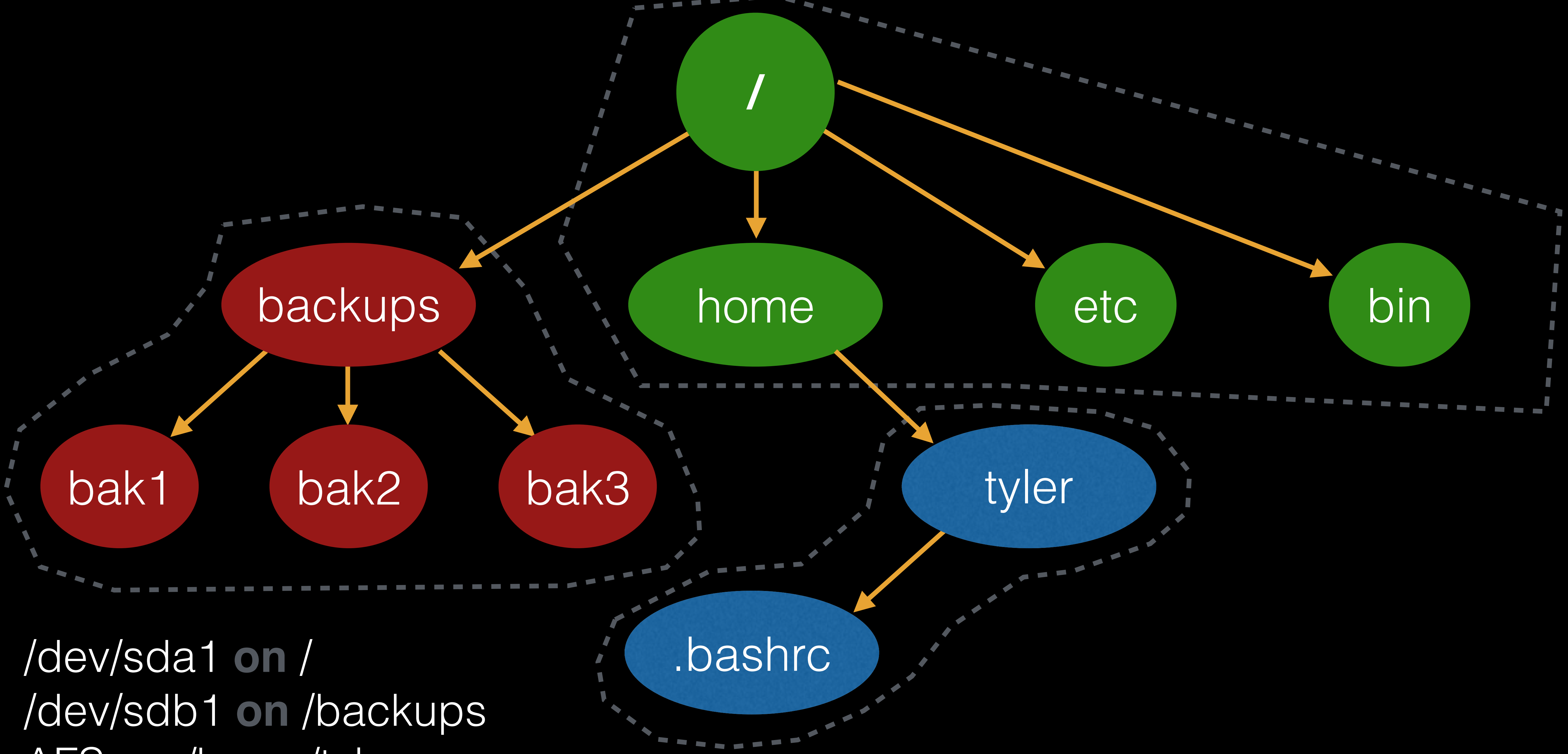
Create empty root directory.

Demo…

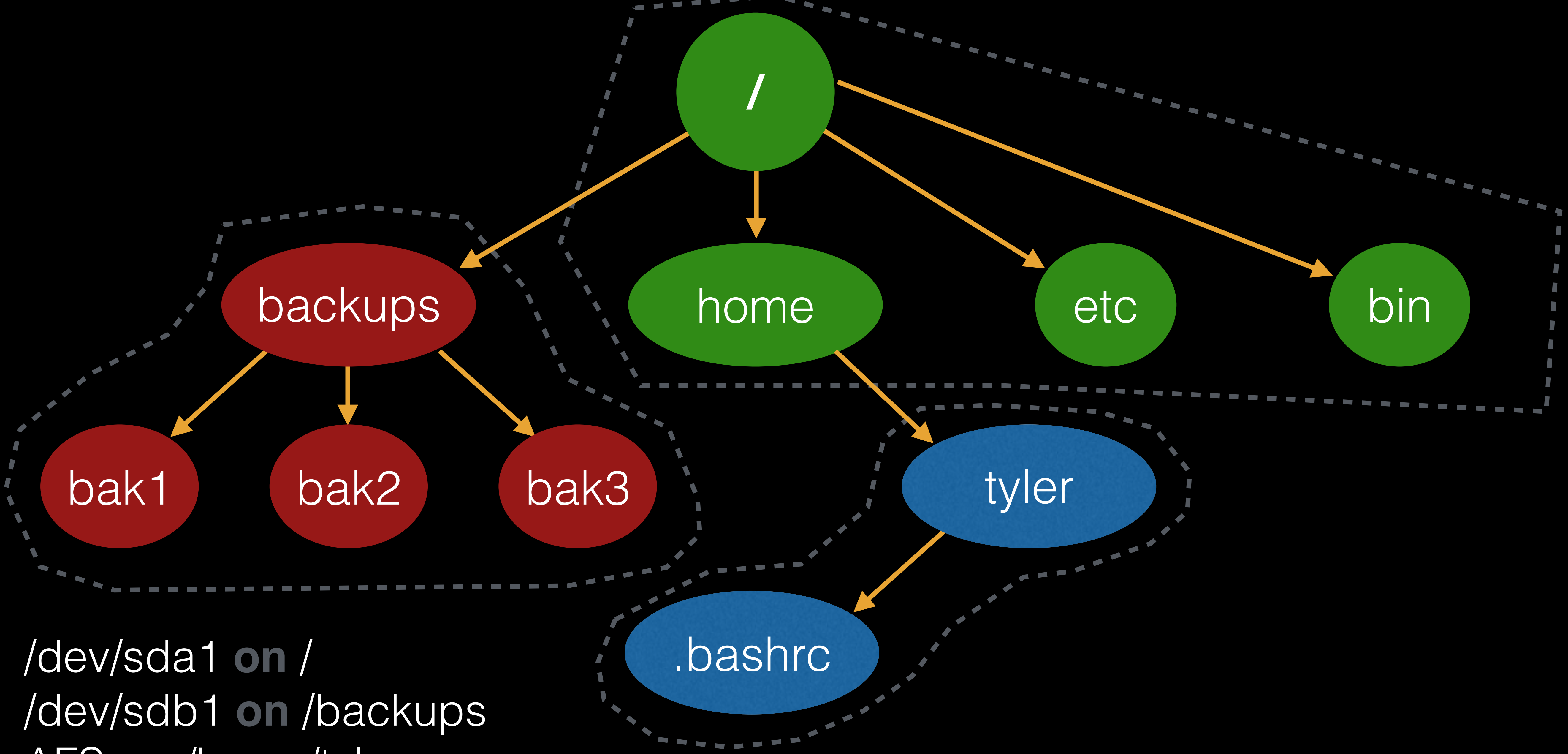
Operations

FS
- mkfs
- mount

File
- create
- write
- open
- read
- close

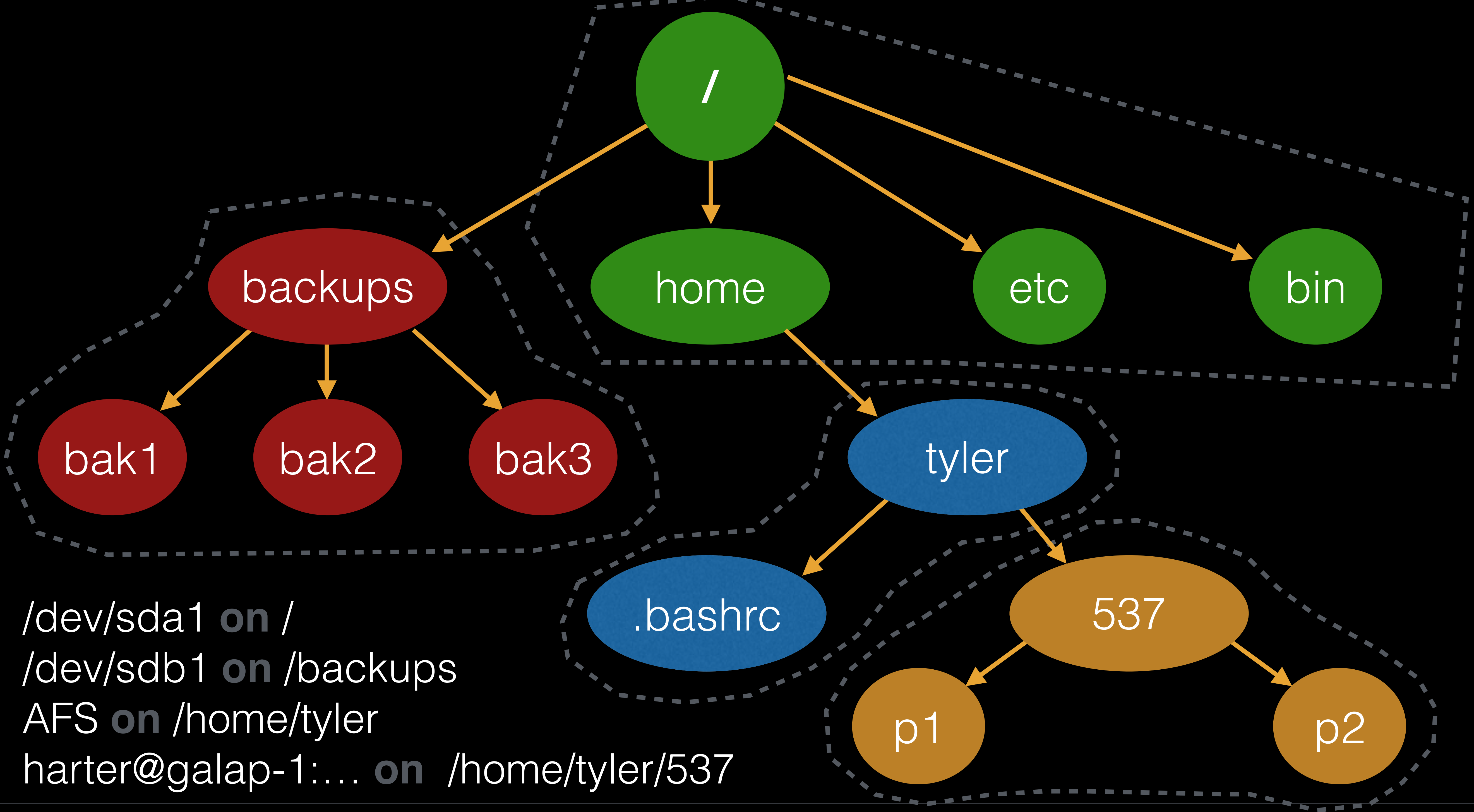


/dev/sda1 **on** /
/dev/sdb1 **on** /backups
AFS **on** /home/tyler



/dev/sda1 **on** /
/dev/sdb1 **on** /backups
AFS **on** /home/tyler

mount harter@galap-1:… /home/tyler/537



/dev/sda1 **on** /
/dev/sdb1 **on** /backups
AFS **on** /home/tyler
harter@galap-1:… **on** /home/tyler/537

mount

Add the file system to the FS tree.

Minimally requires reading superblock.

Demo…

Operations

FS
- mkfs
- mount

File
- create
- write
- open
- read
- close

create /foo/bar

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

create /foo/bar

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

create /foo/bar

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

create /foo/bar

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

create /foo/bar

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

create /foo/bar

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

create /foo/bar

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

Operations

FS
- mkfs
- mount

File
- create
- write
- open
- read
- close

write to /foo/bar

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

write to /foo/bar

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

write to /foo/bar

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

write to /foo/bar

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

write to /foo/bar

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

write to /foo/bar

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

Operations

FS
- mkfs
- mount

File
- create
- write
- open
- read
- close

open /foo/bar

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

open /foo/bar

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

open /foo/bar

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

open /foo/bar

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

open /foo/bar

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

open /foo/bar

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

Operations

FS
- mkfs
- mount

File
- create
- write
- open
- read
- close

read /foo/bar

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

read /foo/bar

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

read /foo/bar

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

read /foo/bar

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

Operations

FS
- mkfs
- mount

File
- create
- write
- open
- read
- close

close /foo/bar

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

close /foo/bar

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

nothing to do on disk!

Efficiency

Efficiency

How can we avoid this excessive I/O for basic ops?

Efficiency

How can we avoid this excessive I/O for basic ops?

Cache for:
- reads
- write buffering

Structures

What data is likely to be read frequently?

- superblock
- data block
- data bitmap
- inode table
- inode bitmap
- indirect block
- directories

Unified Page Cache

Instead of a dedicated file-system cache, draw pages from a common pool for FS and processes.

API change:
- read
- shrink_cache (Linux)

LRU Example

Ops	Hits	State
read 1	miss	1
read 2	miss	1,2
read 3	miss	1,2,3
read 4	miss	1,2,3,4
shrink	-	2,3,4
shrink	-	3,4
read 1	miss	1,3,4
read 2	miss	1,2,3,4
read 3	hit	1,2,3,4
read 4	hit	1,2,3,4

Write Buffering

Why does procrastination help?

Write Buffering

Why does procrastination help?

Overwrites, deletes, scheduling.

Shared structs (e.g., bitmaps+dirs) often overwritten.

Write Buffering

Why does procrastination help?

Overwrites, deletes, scheduling.

Shared structs (e.g., bitmaps+dirs) often overwritten.

We decide: how much to buffer, how long to buffer…
 - tradeoffs?

Structure Review

Structure Review

Core

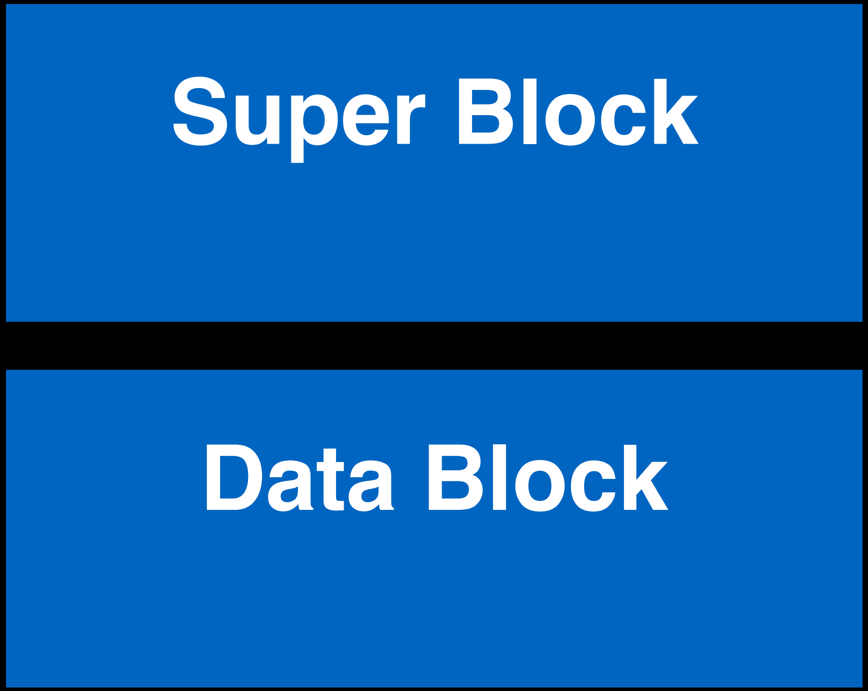
Performance

Super Block

Structure Review

Core

Performance



Structure Review

Core	Performance
Super Block	
Data Block	**Data Bitmap**

Structure Review

Core

Super Block

Data Block

Inode Table

Performance

Data Bitmap

Structure Review

Core

Super Block

Data Block

Inode Table

Performance

Data Bitmap

Inode Bitmap

Structure Review

Core

Super Block

Data Block

directories

Inode Table

Performance

Data Bitmap

Inode Bitmap

Structure Review

Core

Super Block

Data Block

directories **indirects**

Inode Table

Performance

Data Bitmap

Inode Bitmap

Summary/Future

We've described a very simple FS.
- basic on-disk structures
- the basic ops

Future questions:
- how to allocate efficiently?
- how to handle crashes?

Announcement

Office hours today at 1pm, in office.

Discussion tomorrow. p4b.