PERSISTENCE: FILE API AND FILE SYSTEMS

Shivaram Venkataraman

CS 537, Spring 2019

ADMINISTRIVIA

Mid-semester grades: All regrades are done!?

Project 4b: Due next week 4/9

Project 5: One project 9%. Updated due dates on website

Discussion this week: Review worksheet, More Q&A for 4b

AGENDA / LEARNING OUTCOMES

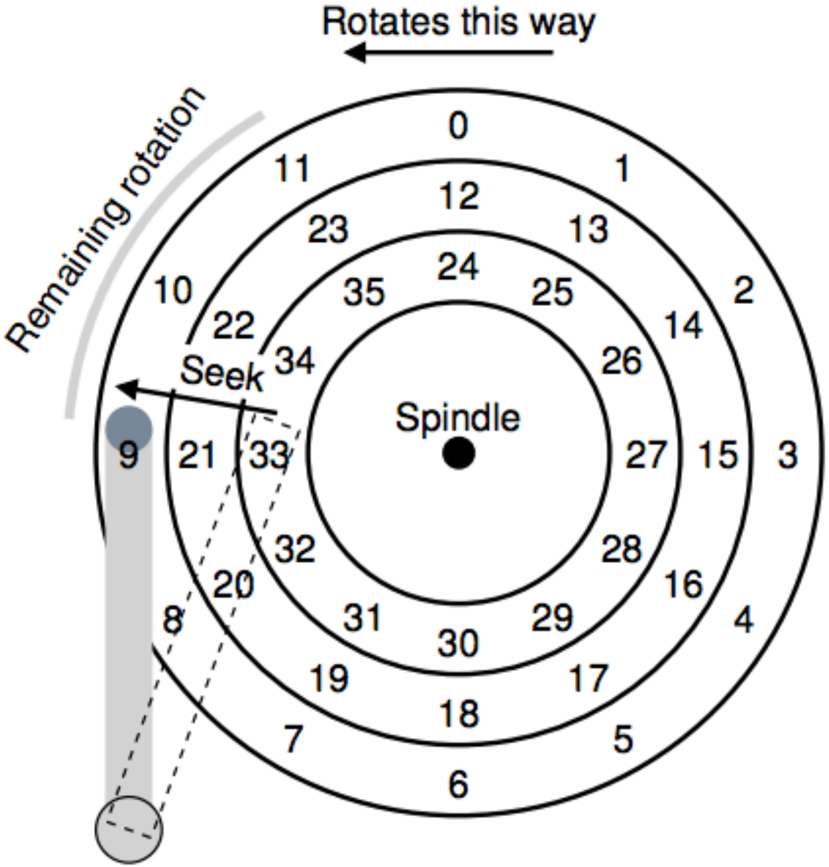
What are the API to create/modify directories?

How does file system represent files, directories?

What steps must reads/writes take?

RECAP

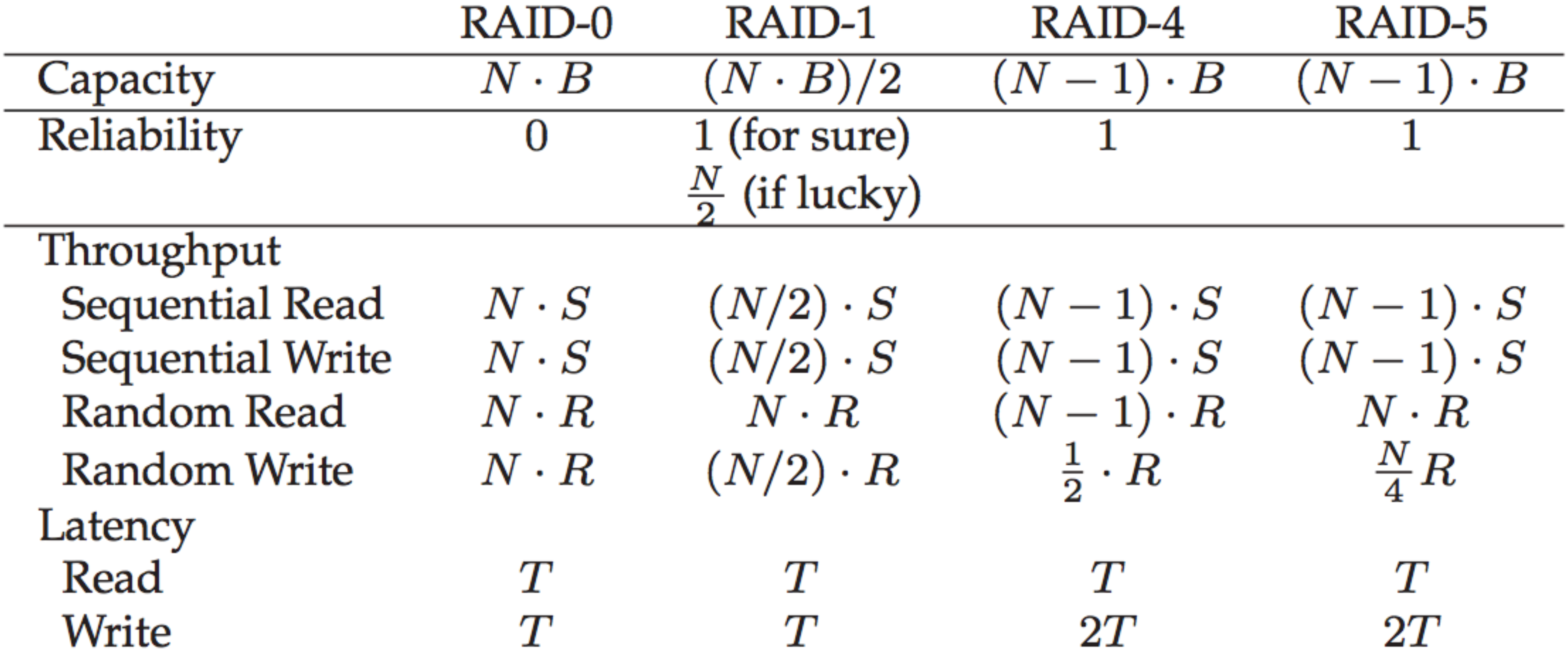
READING DATA FROM DISK



Seek Time

Rotational delay

RAID COMPARISON



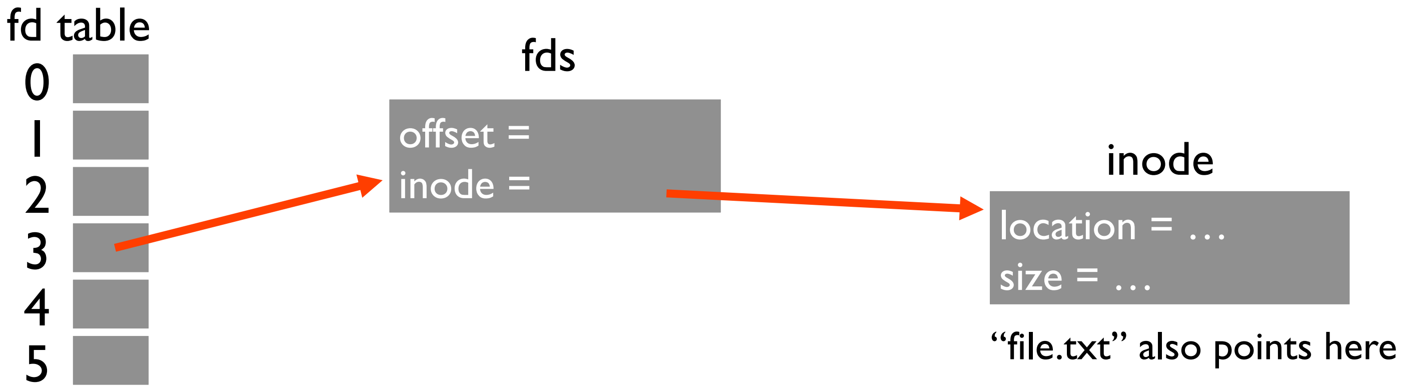
	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	$N \cdot B$	$(N \cdot B)/2$	$(N-1) \cdot B$	$(N-1) \cdot B$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1	1
Throughput				
Sequential Read	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{N}{4}R$
Latency				
Read	T	T	T	T
Write	T	T	$2T$	$2T$

FILE API WITH FILE DESCRIPTORS

```
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)
```

advantages:

- string names
- hierarchical
- traverse once
- offsets precisely defined



```
int fd1 = open(“file.txt”); // returns 3
read(fd1, buf, 12);
int fd2 = open(“file.txt”); // returns 4
int fd3 = dup(fd2);         // returns 5
```

DELETING FILES

There is no system call for deleting files!

Inode (and associated file) is **garbage collected** when there are no references

Paths are deleted when: `unlink()` is called

FDs are deleted when: `close()` or process quits

COMMUNICATING REQUIREMENTS: FSYNC

File system keeps newly written data in memory for awhile

Write buffering improves performance (why?)

But what if system crashes before buffers are flushed?

fsync(int fd) forces buffers to flush to disk, tells disk to flush its write cache

Makes data durable

RENAME

rename(char *old, char *new):

- deletes an old link to a file
- creates a new link to a file

Just changes name of file, does not move data

Even when renaming to new directory

Atomicity guaranteed by OS!

ATOMIC FILE UPDATE

Say application wants to update file.txt atomically

If crash, should see only old contents or only new contents

1. write new data to file.txt.tmp file
2. fsync file.txt.tmp
3. rename file.txt.tmp over file.txt, replacing it

DIRECTORY FUNCTIONS, LINKS

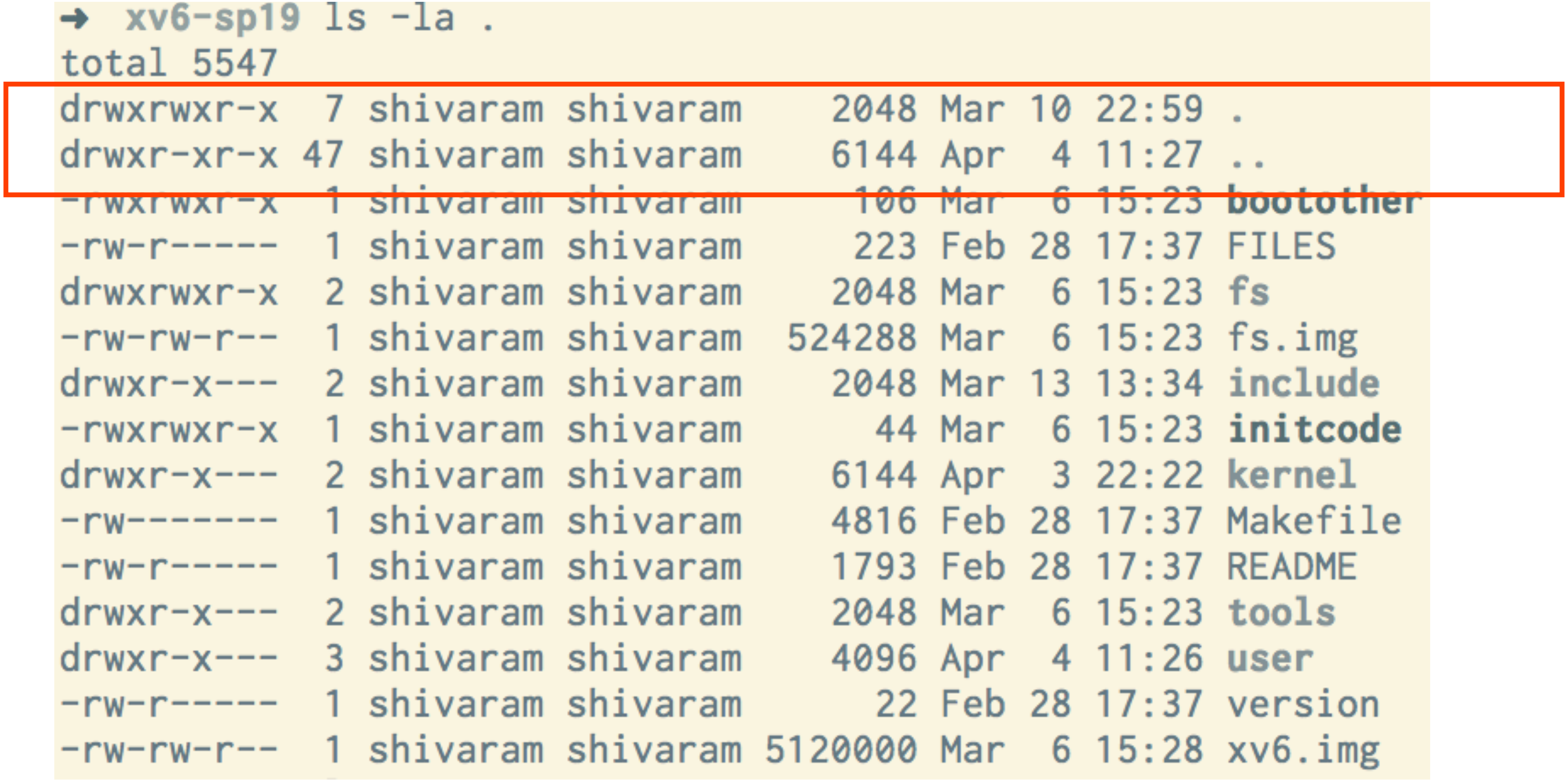
DIRECTORY CALLS

mkdir: create new directory

readdir: read/parse directory entries

Why no writedir?

SPECIAL DIRECTORY ENTRIES



```
➜  xv6-sp19 ls -la .
total 5547
drwxrwxr-x  7 shivaram shivaram    2048 Mar 10 22:59 .
drwxr-xr-x 47 shivaram shivaram    6144 Apr  4 11:27 ..
-rwxrwxr-x  1 shivaram shivaram     106 Mar  6 15:23 bootother
-rw-r-----  1 shivaram shivaram     223 Feb 28 17:37 FILES
drwxrwxr-x  2 shivaram shivaram    2048 Mar  6 15:23 fs
-rw-rw-r--  1 shivaram shivaram  524288 Mar  6 15:23 fs.img
drwxr-x---  2 shivaram shivaram    2048 Mar 13 13:34 include
-rwxrwxr-x  1 shivaram shivaram      44 Mar  6 15:23 initcode
drwxr-x---  2 shivaram shivaram    6144 Apr  3 22:22 kernel
-rw-------  1 shivaram shivaram    4816 Feb 28 17:37 Makefile
-rw-r-----  1 shivaram shivaram    1793 Feb 28 17:37 README
drwxr-x---  2 shivaram shivaram    2048 Mar  6 15:23 tools
drwxr-x---  3 shivaram shivaram    4096 Apr  4 11:26 user
-rw-r-----  1 shivaram shivaram      22 Feb 28 17:37 version
-rw-rw-r--  1 shivaram shivaram 5120000 Mar  6 15:28 xv6.img
```

LINKS

Hard links: Both path names use same inode number

File does not disappear until all removed; cannot link directories

```
echo “Beginning…” > file1
ln file1 link
cat link
ls –li
echo “More info” >> file1
mv file1 file2
rm file2
```

SOFT LINKS

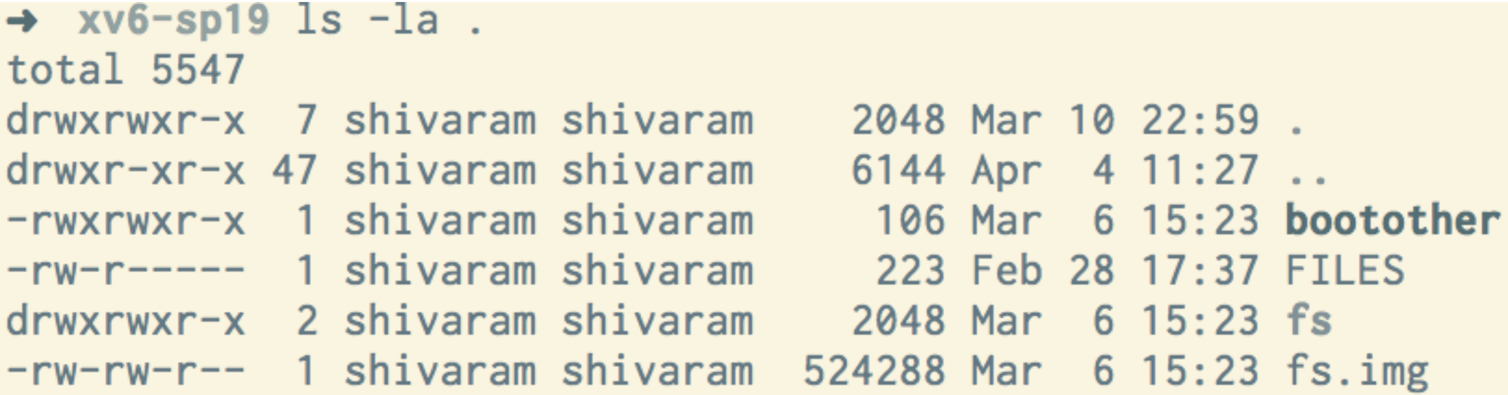
Soft or symbolic links: Point to second path name; can softlink to dirs

```
ln –s oldfile softlink
```

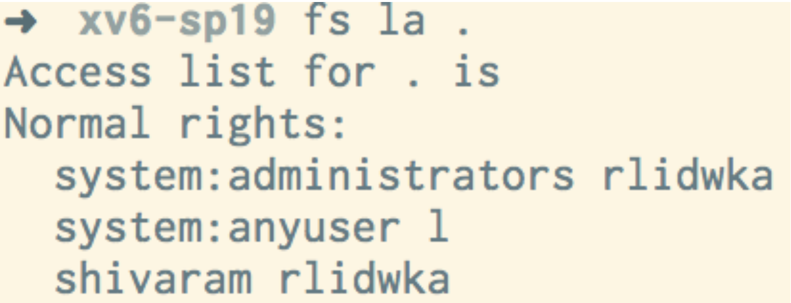
Confusing behavior: “file does not exist”!

Confusing behavior: “cd linked_dir; cd ..; in different parent!

PERMISSIONS, ACCESS CONTROL



```
➜  xv6-sp19 ls -la .
total 5547
drwxrwxr-x  7 shivaram shivaram    2048 Mar 10 22:59 .
drwxr-xr-x 47 shivaram shivaram    6144 Apr  4 11:27 ..
-rwxrwxr-x  1 shivaram shivaram     106 Mar  6 15:23 bootother
-rw-r-----  1 shivaram shivaram     223 Feb 28 17:37 FILES
drwxrwxr-x  2 shivaram shivaram    2048 Mar  6 15:23 fs
-rw-rw-r--  1 shivaram shivaram  524288 Mar  6 15:23 fs.img
```



```
➜  xv6-sp19 fs la .
Access list for . is
Normal rights:
  system:administrators rlidwka
  system:anyuser l
  shivaram rlidwka
```

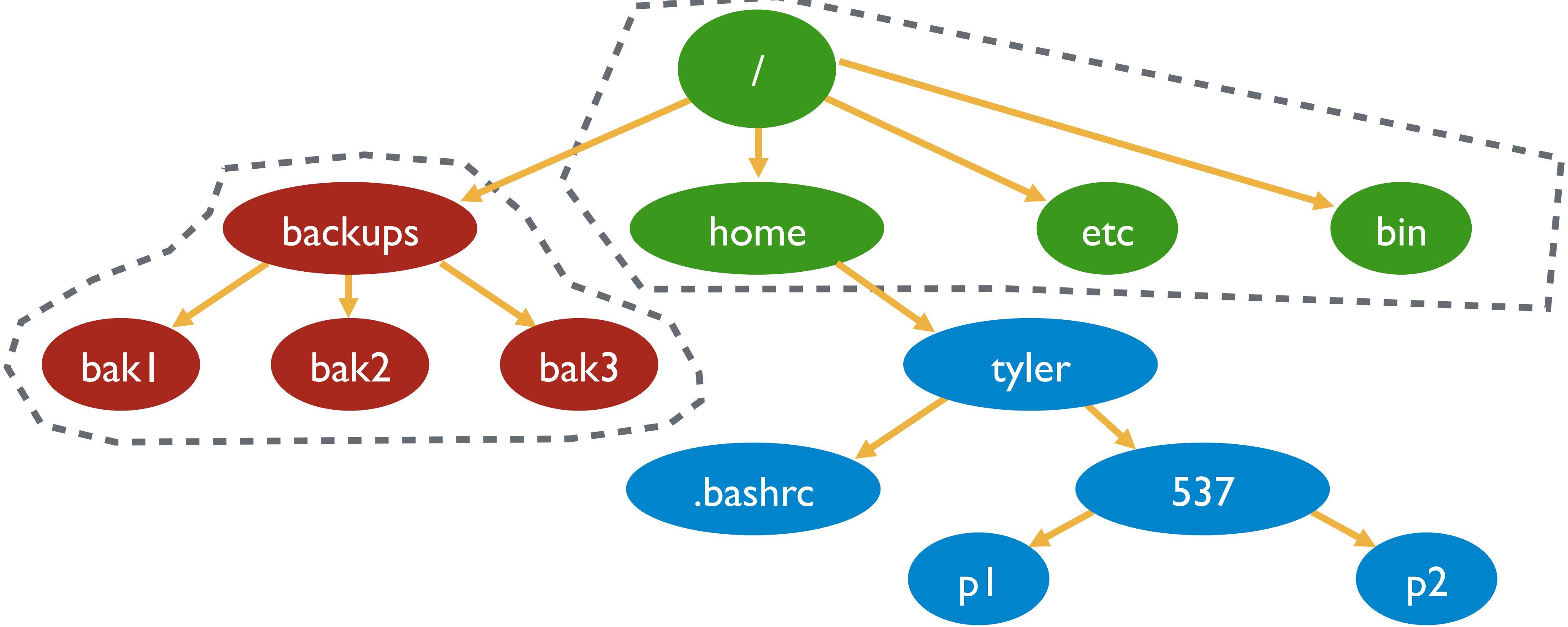
MANY FILE SYSTEMS

Users often want to use many file systems

For example:

- main disk
- backup disk
- AFS
- thumb drives

Idea: stitch all the file systems together into a super file system!



```
sh> mount
/dev/sda1 on / type ext4 (rw)
/dev/sdb1 on /backups type ext4 (rw)
AFS on /home type afs (rw)
```

BUNNY 14

https://tinyurl.com/cs537-sp19-bunny14

BUNNY 14

https://tinyurl.com/cs537-sp19-bunny14

Consider the following code snippet:

```
echo “hello” > oldfile
ln –s oldfile link1
ln oldfile link2
rm oldfile
```

What will be the output of `cat link1`

What will be the output of `cat link2`

What is the file permission to only give current user read, write, execute access?

FILE API SUMMARY

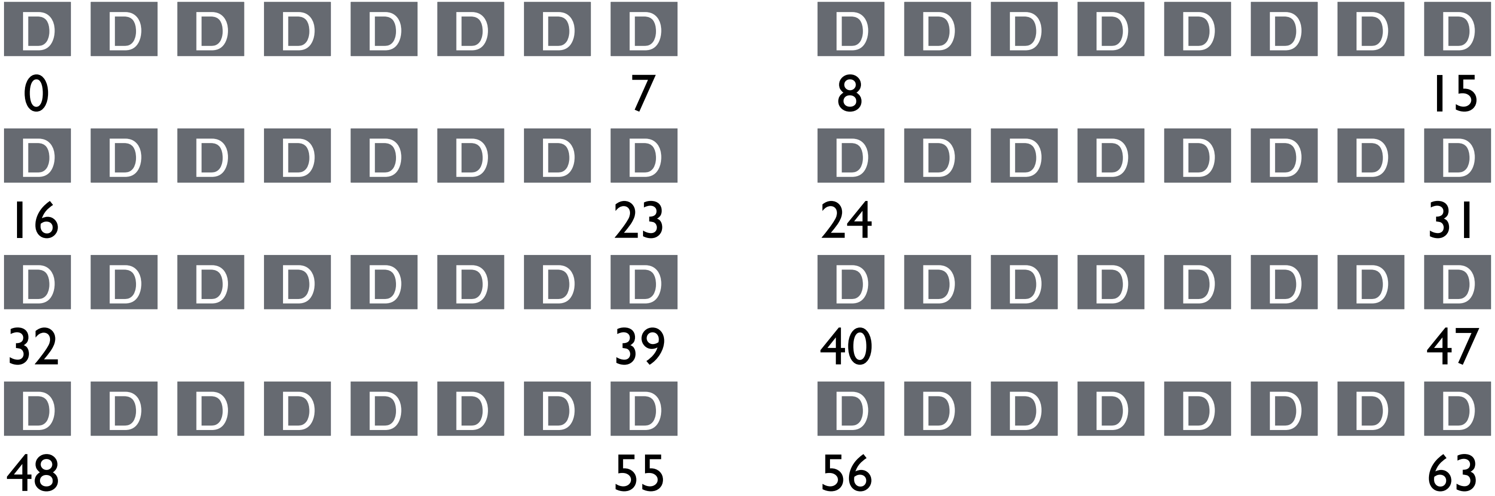
Using multiple types of name provides convenience and efficiency

Mount and link features provide flexibility.

Special calls (fsync, rename) let developers communicate requirements to file system

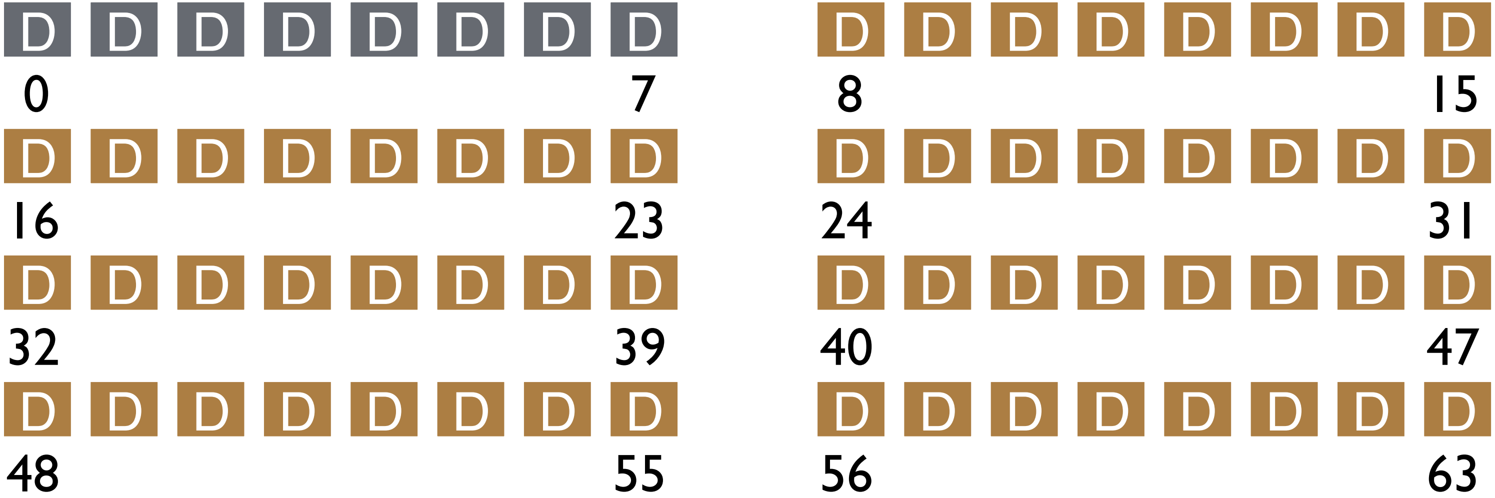
FILESYSTEM DISK STRUCTURES

FS STRUCTS: EMPTY DISK



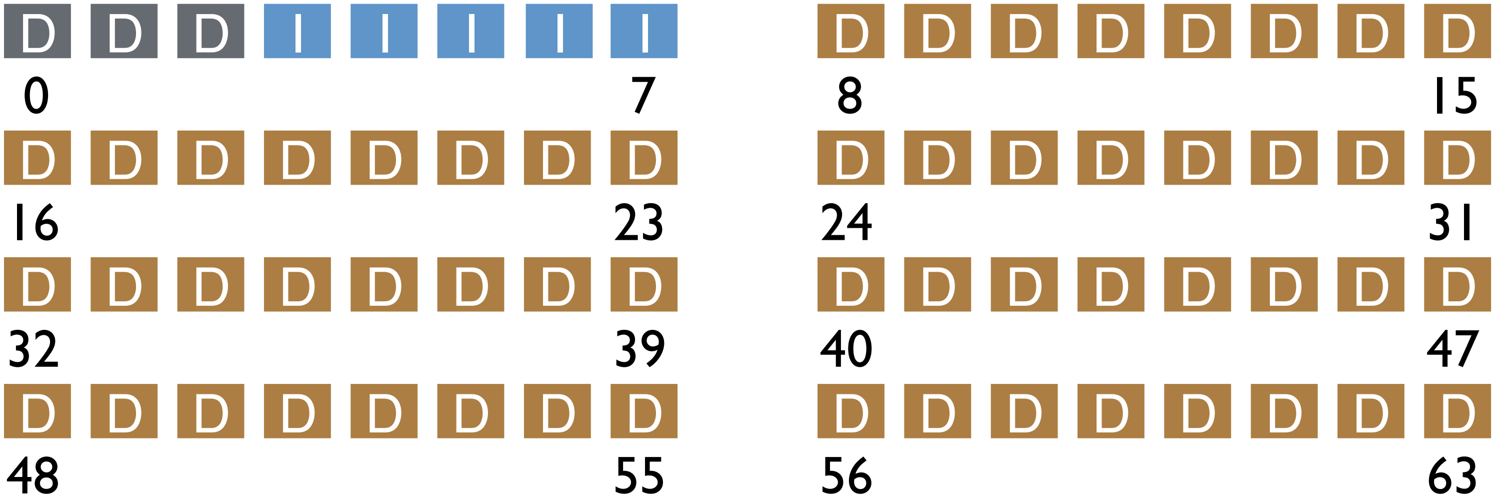
Assume each block is 4KB

FS STRUCTS: DATA BLOCKS



Simple layout → Very Simple File System

INODE POINTERS

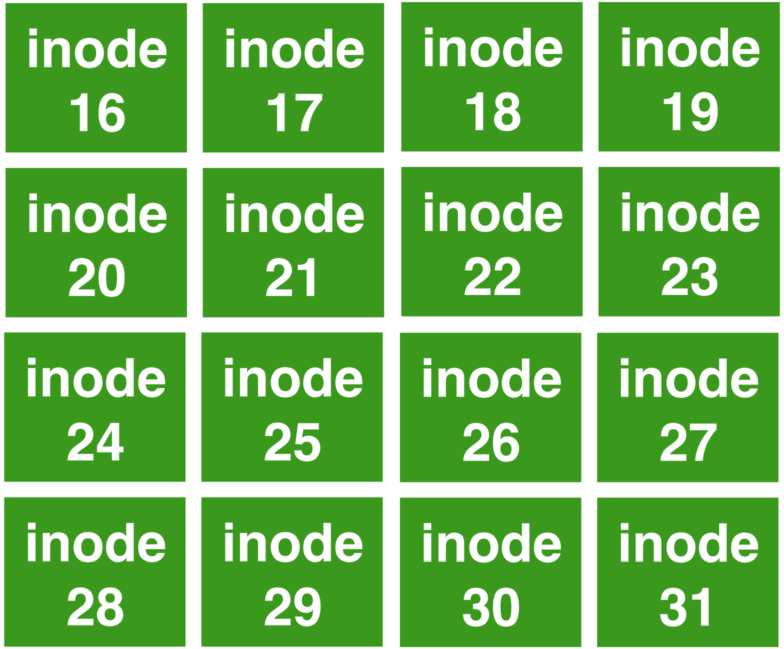


ONE INODE BLOCK

Each inode is typically 256 bytes (depends on the FS, maybe 128 bytes)

4KB disk block

16 inodes per inode block.



inode 16	inode 17	inode 18	inode 19
inode 20	inode 21	inode 22	inode 23
inode 24	inode 25	inode 26	inode 27
inode 28	inode 29	inode 30	inode 31

INODE

type (file or dir?)

uid (owner)

rwx (permissions)

size (in bytes)

Blocks

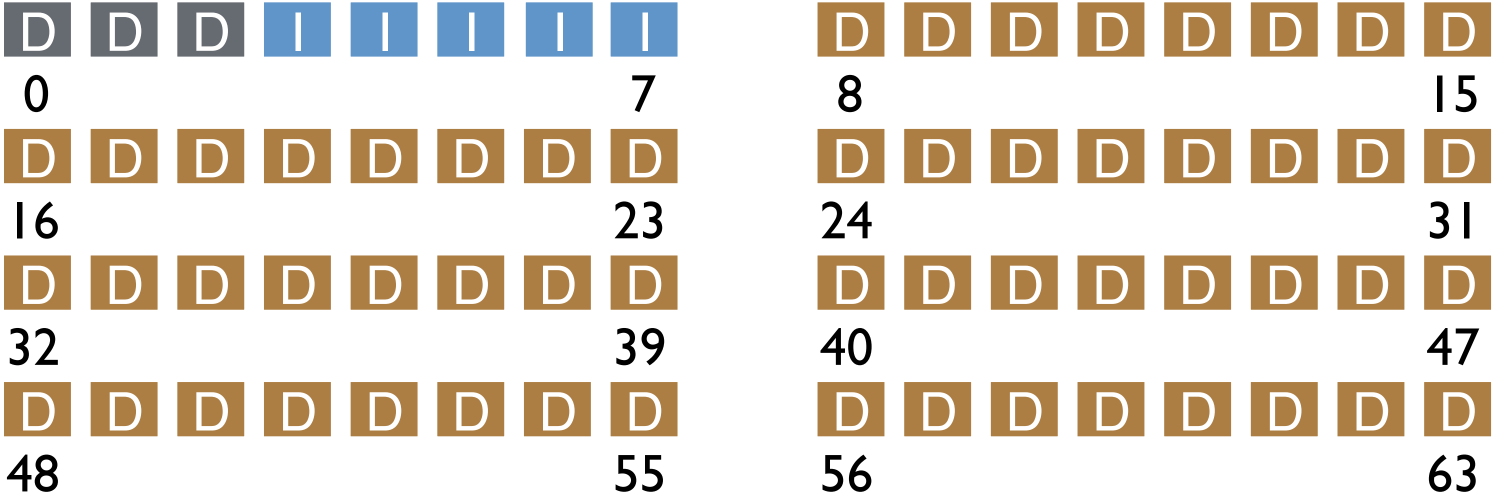
time (access)

ctime (create)

links_count (# paths)

addrs[N] (N data blocks)

FS STRUCTS: INODE DATA POINTERS



INODE

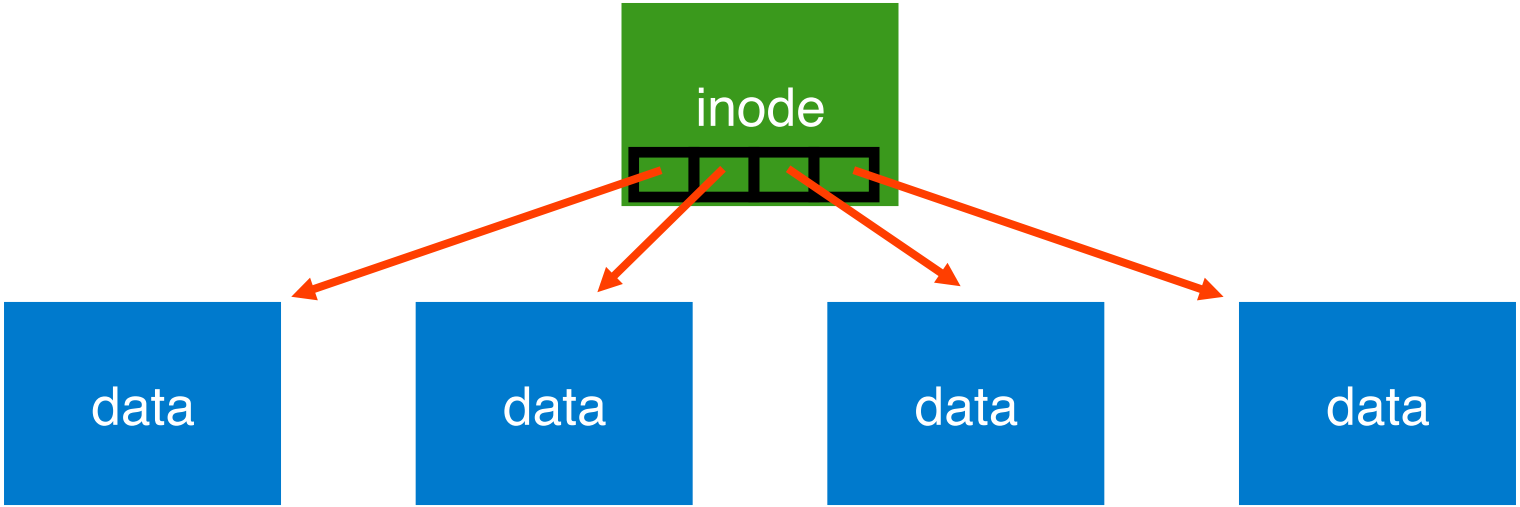
type (file or dir?)
uid (owner)
rwx (permissions)
size (in bytes)
Blocks
time (access)
ctime (create)
links_count (# paths)
addrs[N] (N data blocks)

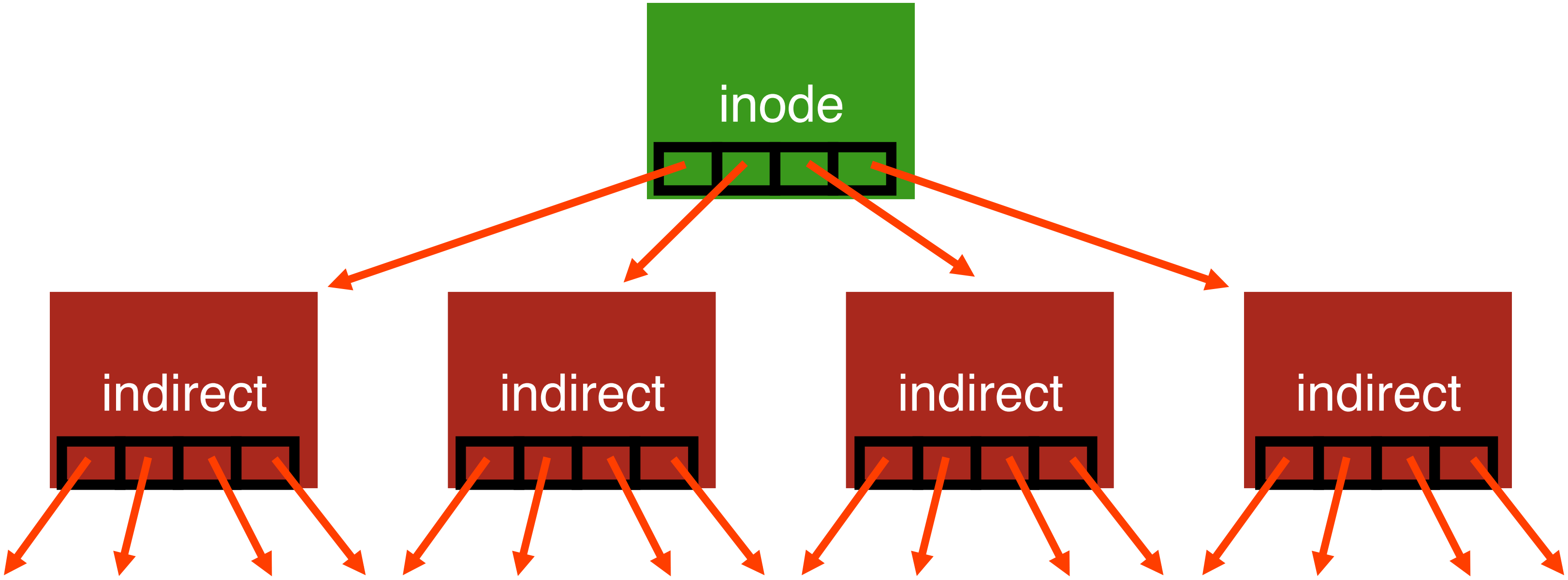
Assume single level (just pointers to data blocks)

What is max file size?

Assume 256-byte inodes
(all can be used for pointers)
Assume 4-byte addrs

How to get larger files?

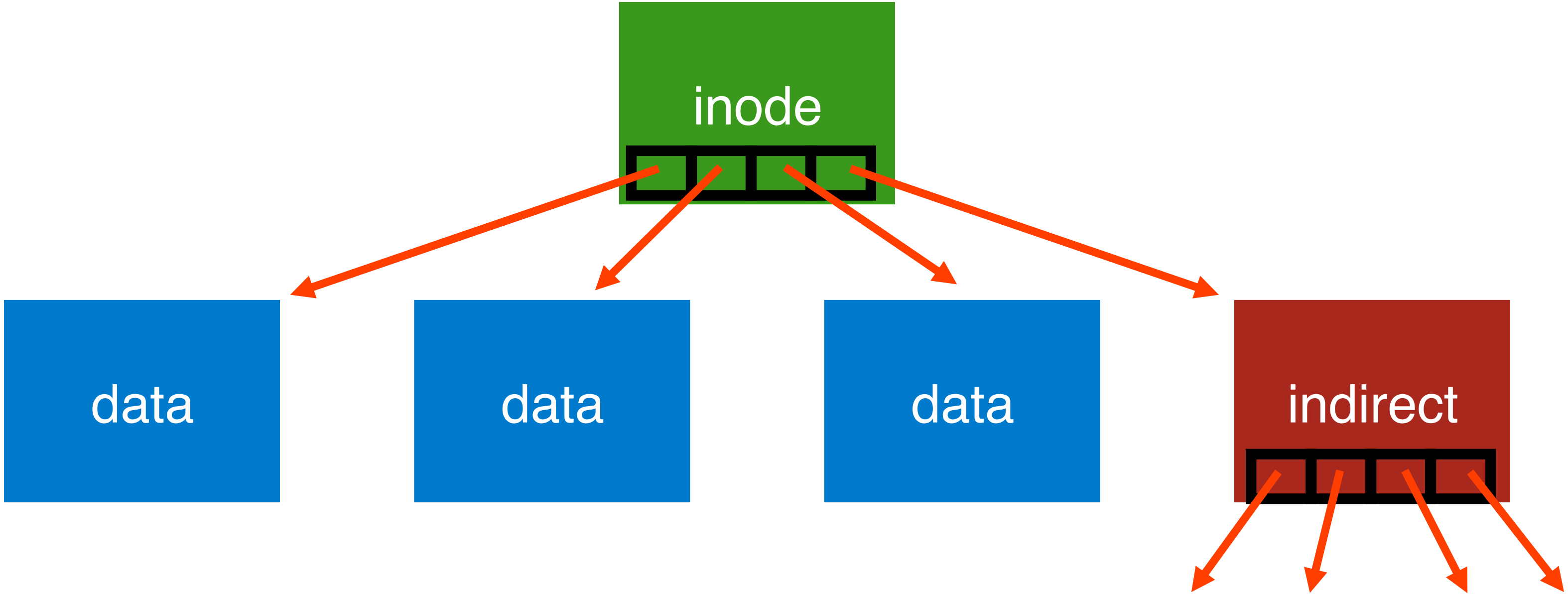

inode
data
data
data
data



Indirect blocks are stored in regular data blocks

Largest file size with 64 indirect blocks?

Any Cons?



Better for small files!

How to handle even larger files?

OTHER APPROACHES

Extent-based

Linked (File-allocation Tables)

Multi-level Indexed

Questions

- Amount of fragmentation (internal and external)
- Ability to grow file over time?
- Performance of sequential accesses (contiguous layout)?
- Speed to find data blocks for random accesses?
- Wasted space for meta-data overhead (everything that isn’t data)?

  Meta-data must be stored persistently too!

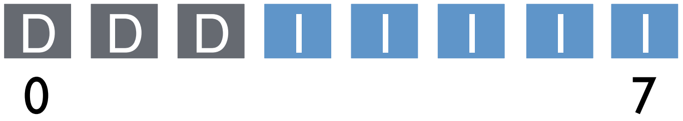
BUNNY 15

https://tinyurl.com/cs537-sp19-bunny15

BUNNY 15

Assume 256 byte inodes (16 inodes/block).
What is the offset for inode with number 0?

https://tinyurl.com/cs537-sp19-bunny15



What is the offset for inode with number 0?

What is the offset for inode with number 0?

Directories

File systems vary

Common design:

- Store directory entries in data blocks
- Large directories just use multiple data blocks
- Use bit in inode to distinguish directories from files

Various formats could be used

- lists
- b-trees

SIMPLE DIRECTORY LIST EXAMPLE

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

unlink("foo")

ALLOCATION

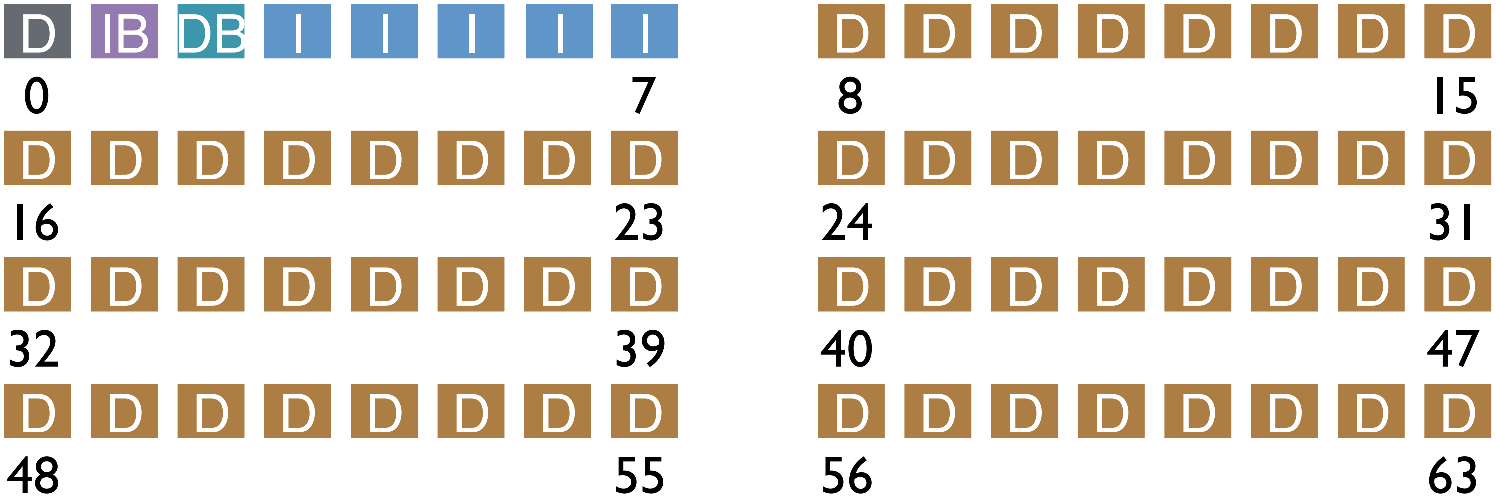
How do we find free data blocks or free inodes?

Free list

Bitmaps

Tradeoffs in next lecture…

FS STRUCTS: BITMAPS



D IB DB 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

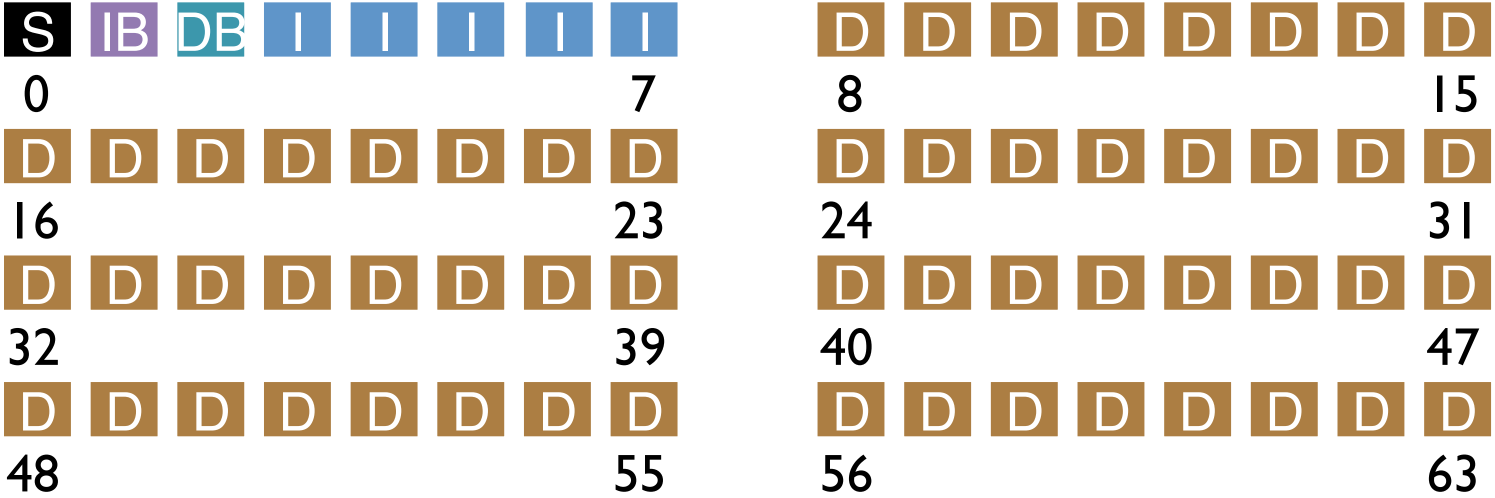
SUPERBLOCK

Need to know basic FS configuration metadata, like:

- block size
- # of inodes

Store this in superblock

FS STRUCTS: SUPERBLOCK



S IB DB 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

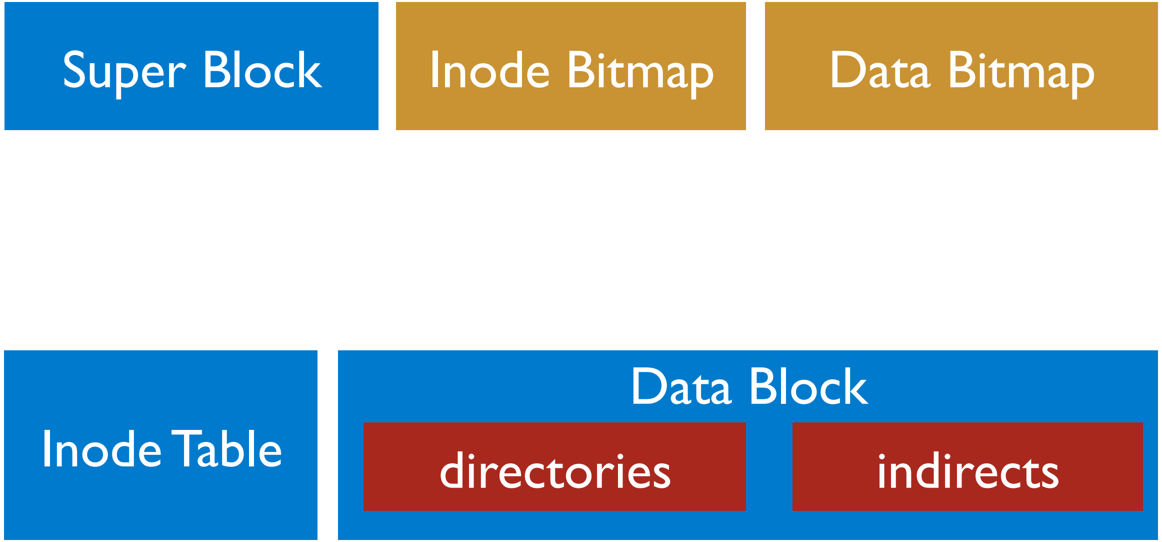
D D D D D D D D
32 39

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

SUMMARY



Super Block

Inode Bitmap

Data Bitmap

Inode Table

Data Block

directories

indirects

PART 2 : OPERATIONS

- create file
- write
- open
- read
- close

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			

What needs to be read and written?

open /foo/bar

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

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

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

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

nothing to do on disk!

EFFICIENCY

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

Cache for:

- reads
- write buffering

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?

NEXT STEPS

Next class: UNIX Fast-File System