PERSISTENCE: FILE SYSTEMS

Shivaram Venkataraman

CS 537, Spring 2020

ADMINISTRIVIA

Midterm grade ranges

Project status going forward

AGENDA / LEARNING OUTCOMES

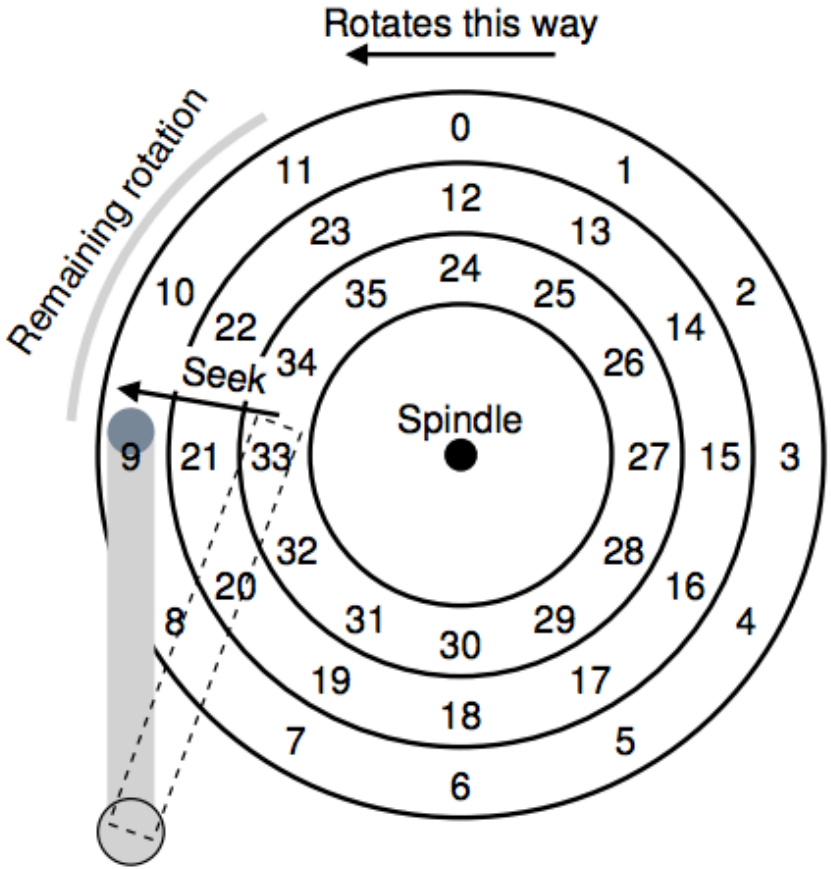
What are the API to create/modify directories?

How does file system represent files, directories?

What steps must reads/writes take?

RECAP

READING DATA FROM DISK



Seek Time

Rotational delay

RAID LEVEL COMPARISONS

	Reliability	Capacity	Read latency	Write Latency	Seq Read	Seq Write	Rand Read	Rand Write
RAID-0	0	C*N	D	D	N * S	N * S	N * R	N * R
RAID-1	1	C*N/2	D	D	N/2 * S	N/2 * S	N * R	N/2 * R
RAID-4	1	(N-1) * C	D	2D	(N-1)*S	(N-1)*S	(N-1)*R	R/2
RAID-5	1	(N-1) * C	D	2D	(N-1)*S	(N-1)*S	N * R	N/4 * R

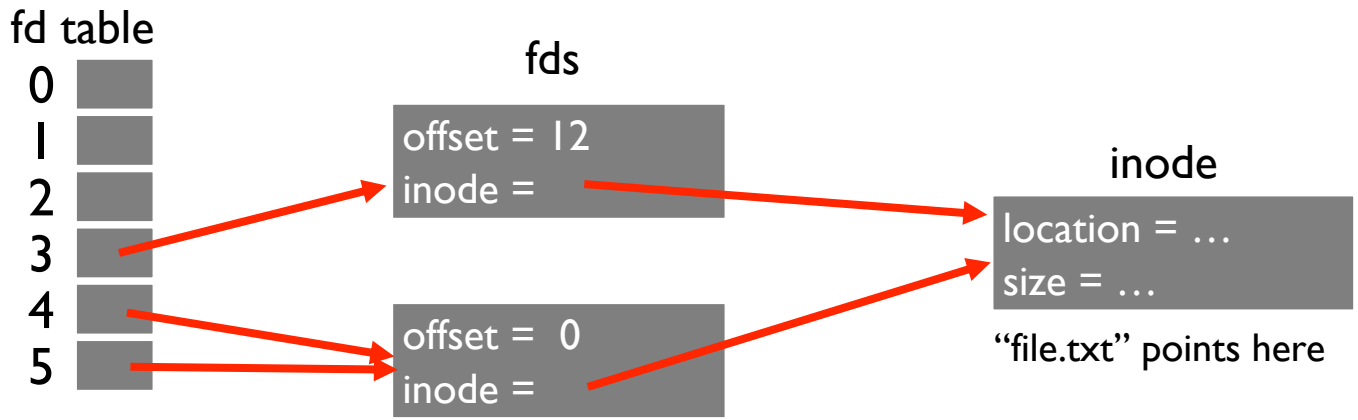
FILE API WITH FILE DESCRIPTORS

```
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

DUP




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

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

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

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)

Renames are atomic!
Either file exists in old path or new one

DIRECTORY CALLS

mkdir: create new directory

readdir: read/parse 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 hard links removed; cannot link directories

```
echo “Beginning…” > file1
ln file1 link
cat link
# “Beginning…”

ls –li
# 18 -rw-rw-r-- 2 shivaram shivaram 10 Apr  6 21:32 file1
# 18 -rw-rw-r-- 2 shivaram shivaram 10 Apr  6 21:32 link
```

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
```

FILE API SUMMARY

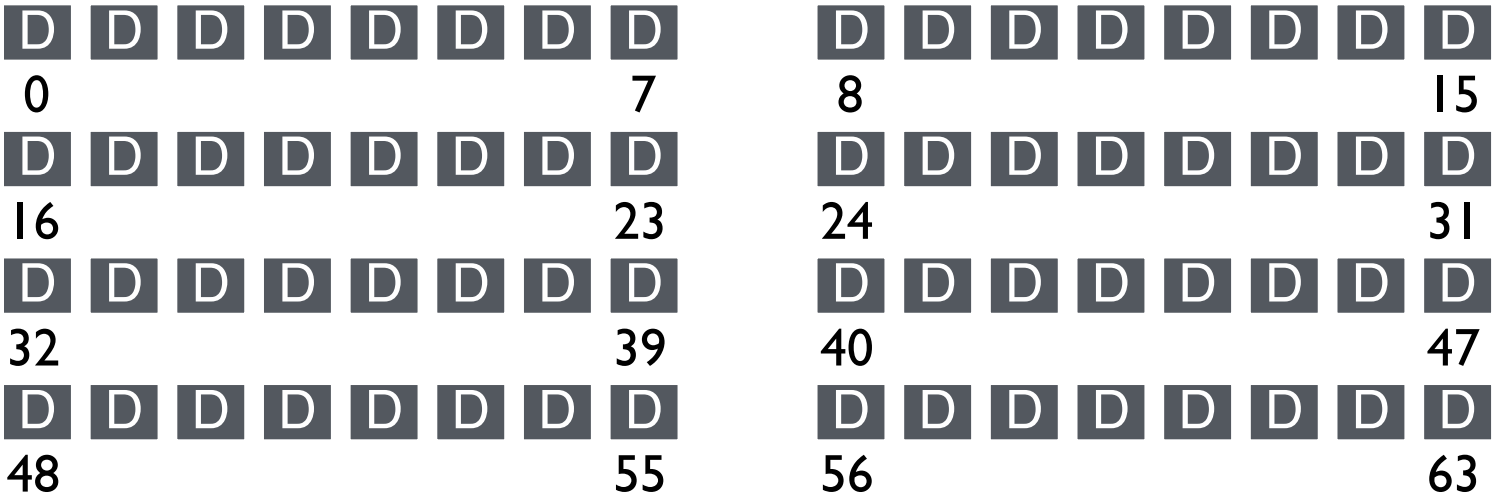
Using multiple types of name provides convenience and efficiency

Hard and soft link features provide flexibility.

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

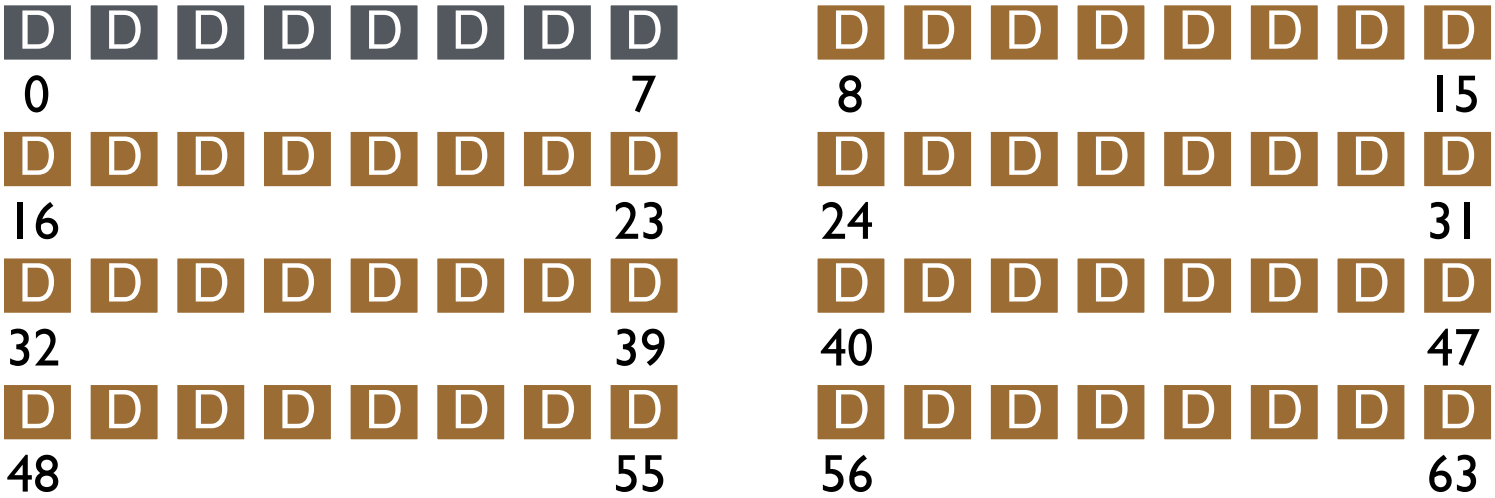
FILESYSTEM DISK STRUCTURES

FS STRUCTS: EMPTY DISK



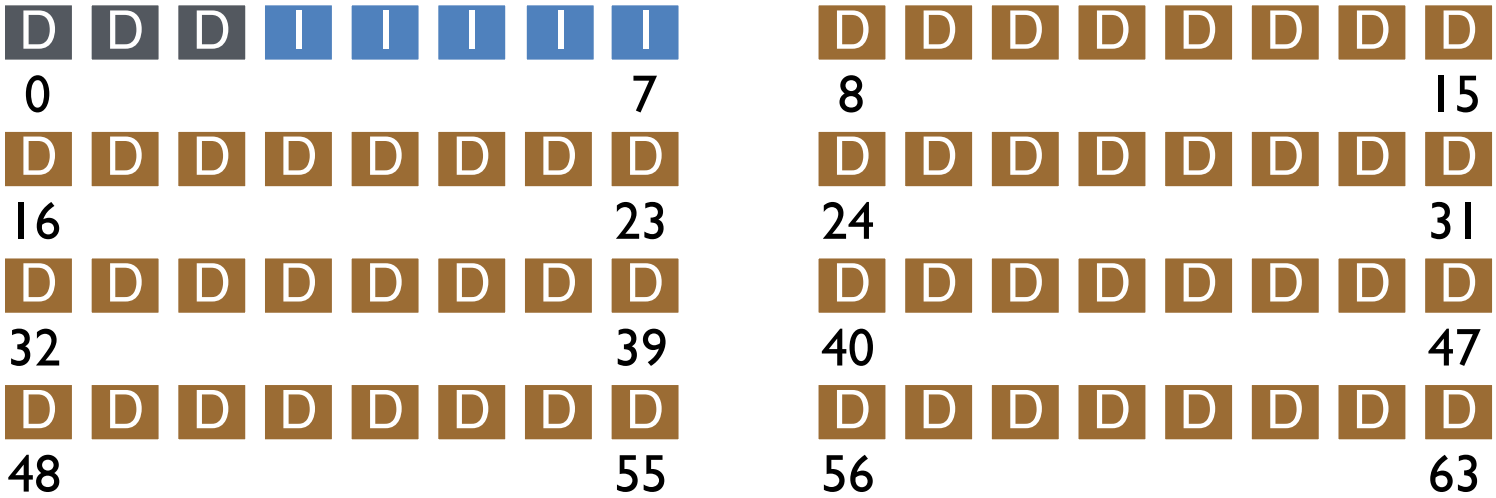
Assume each block is 4KB

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)

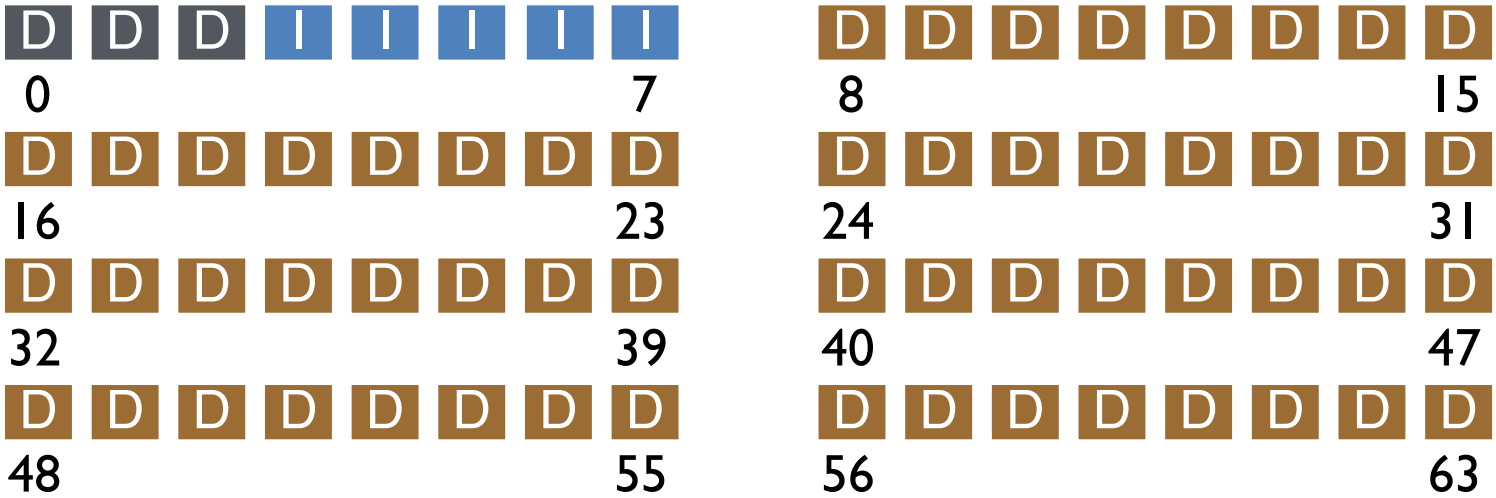
4KB disk block

16 inodes per inode block.

INODE

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

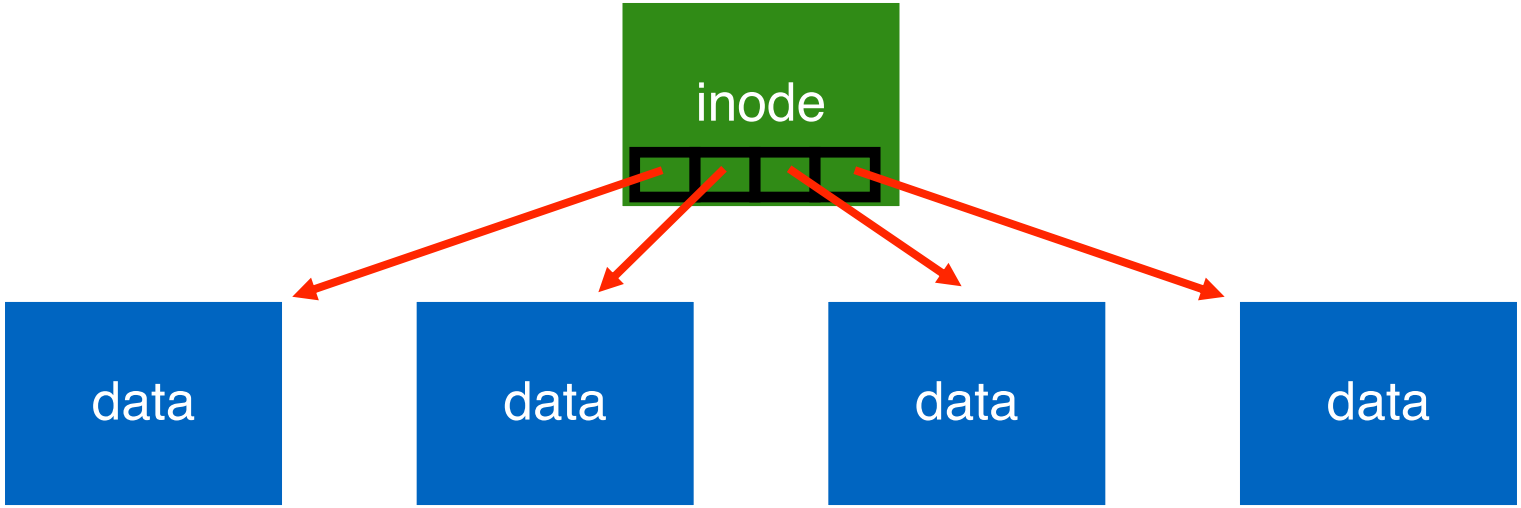
type (file or dir?)
uid (owner)
rwx (permissions)
size (in bytes)
Blocks
time (access)
ctime (create)
links_count (# paths)
addrs[N] (N data blocks)

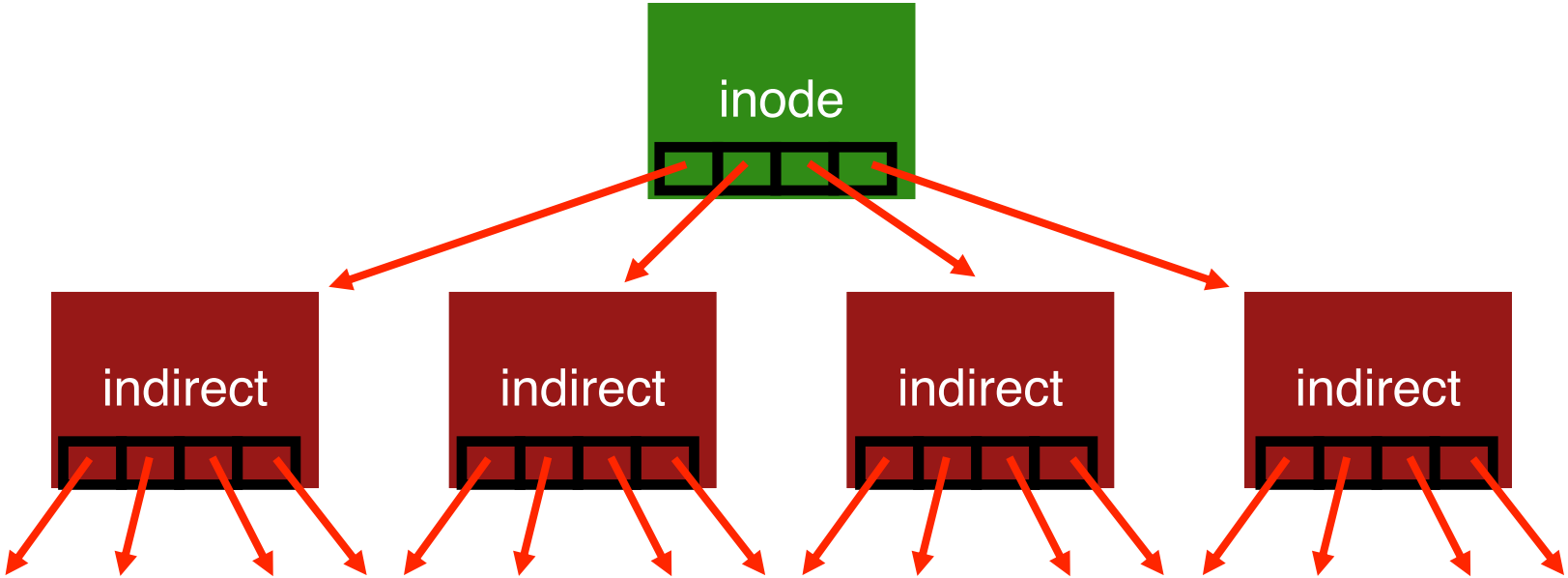
Assume single level (just pointers to data blocks)

What is max file size?

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

How to get larger files?

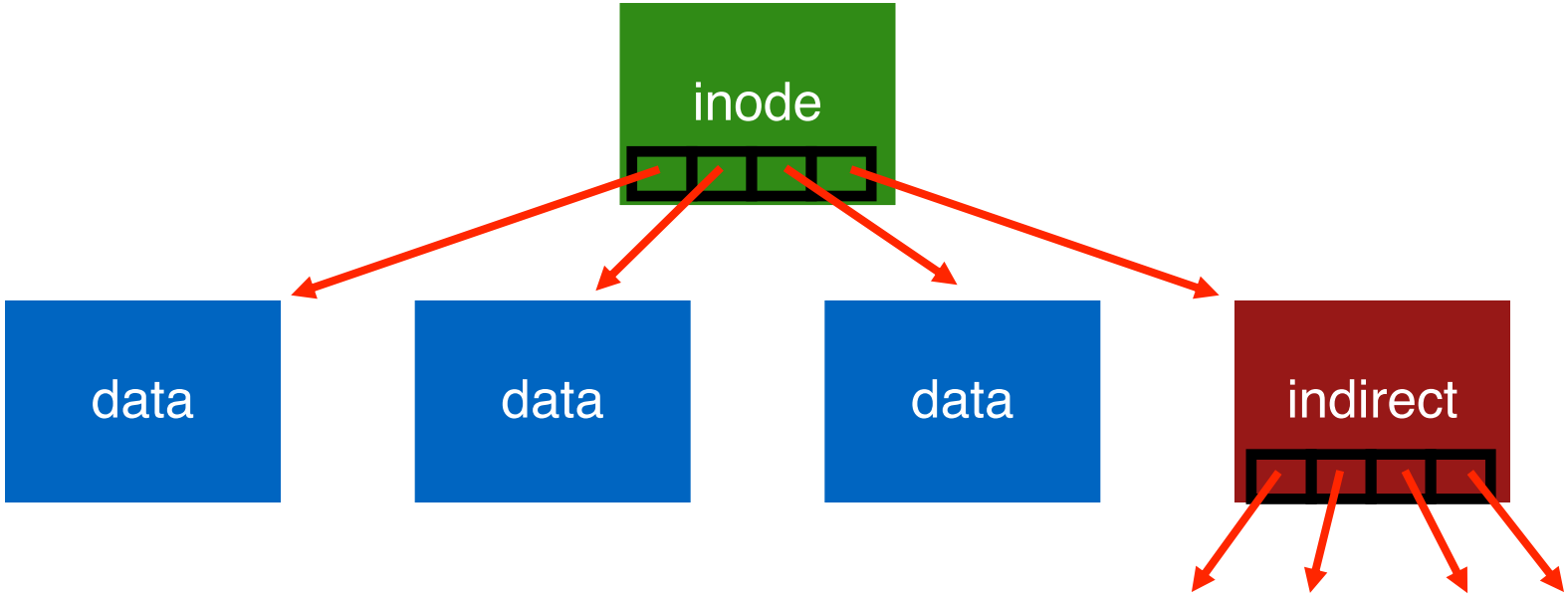

inode
data
data
data
data



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 lists (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!

QUIZ 25

https://tinyurl.com/cs537-sp20-quiz25

Assume 256 byte inodes (16 inodes/block), block size = 4KB.
What is the offset for inode with number 0?

What is the offset for inode with number 4?

What is the offset for inode with number 40?

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

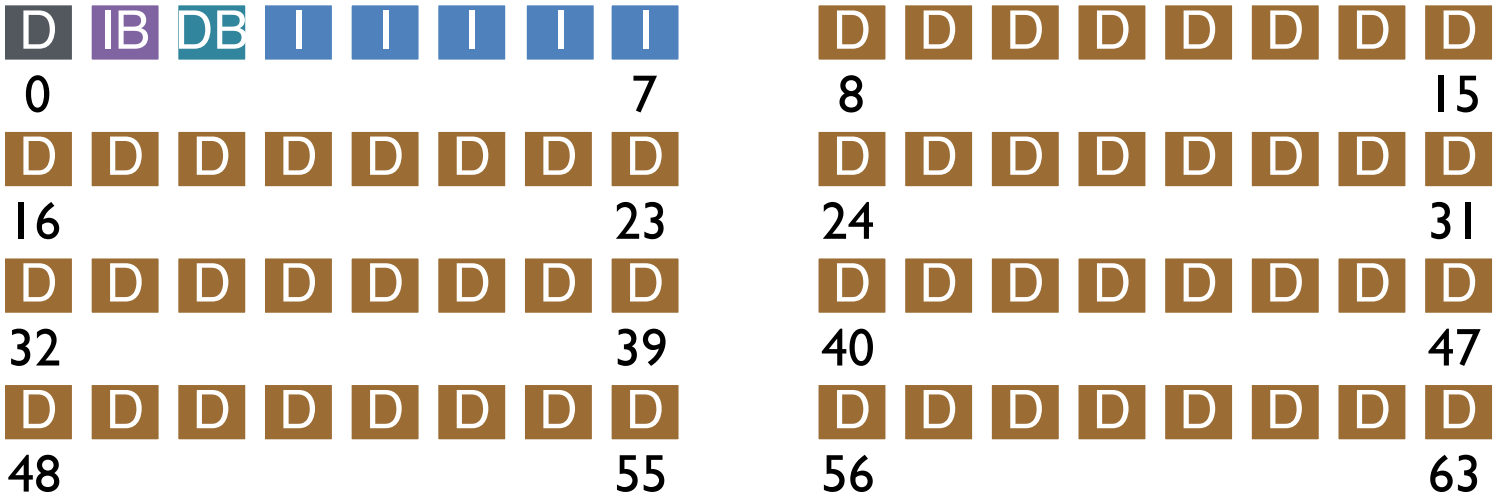
How do we find free data blocks or free inodes?

Free list

Bitmaps

Tradeoffs in next lecture…

FS STRUCTS: BITMAPS



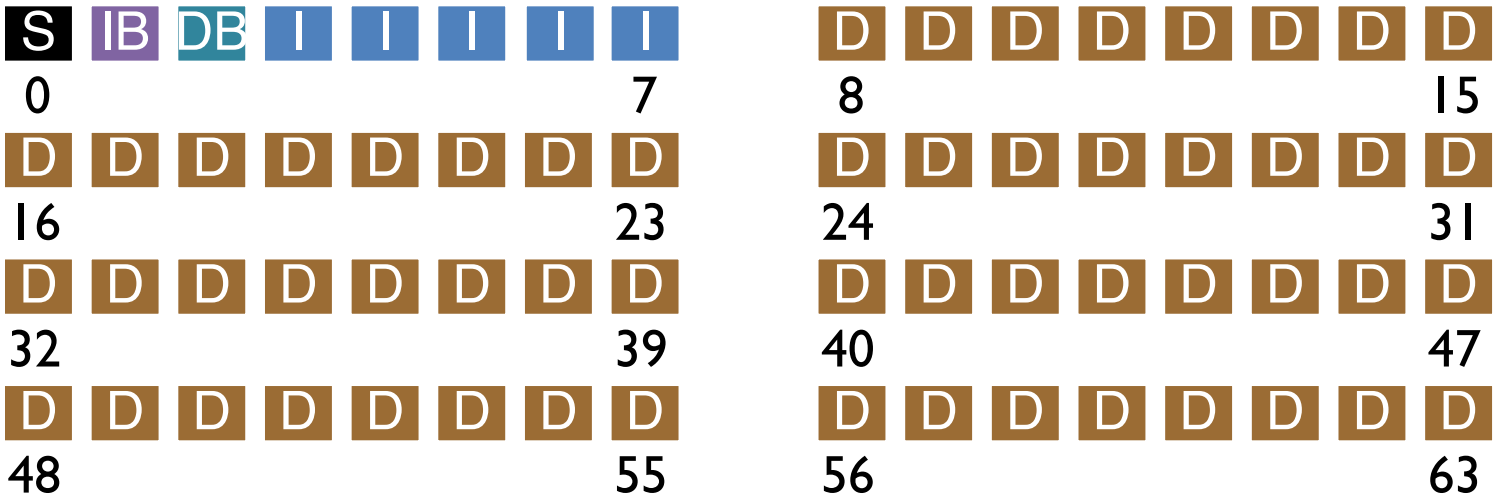
SUPERBLOCK

Need to know basic FS configuration metadata, like:

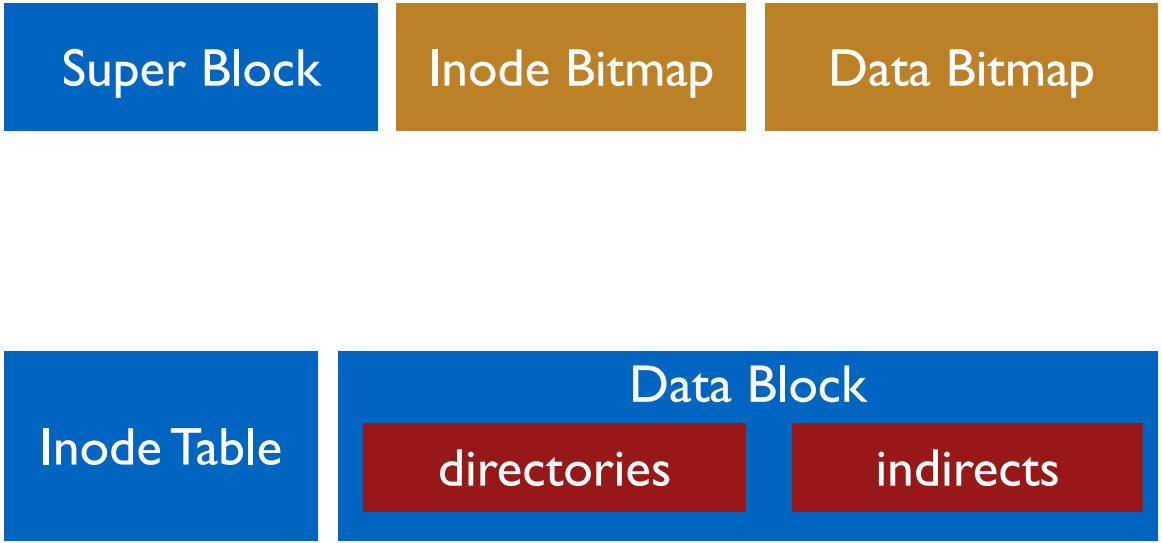
- block size
- # of inodes

Store this in superblock

FS STRUCTS: SUPERBLOCK



SUMMARY


Super Block
Inode Bitmap
Data Bitmap
Inode Table
Data Block
directories
indirects

NEXT STEPS

P5 will be released later this week
Details in the discussion section

Next class: Filesystem operations, FFS!