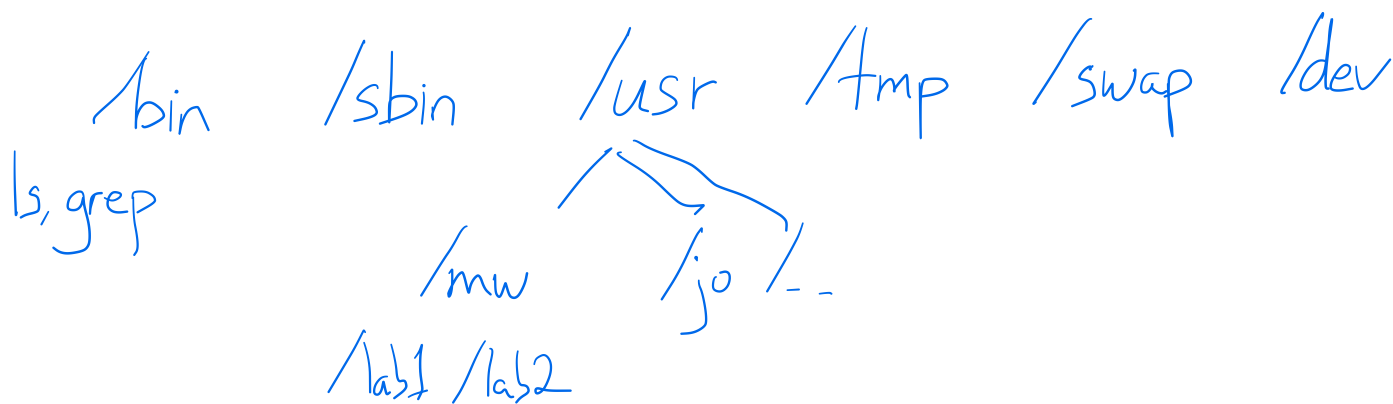


- ☑ 1. Last time
- ☑ 2. Directories
- ☑ 3. Performance

## 2. Directories

(see sheet at the end)

- Intro to directories
- Hierarchical Unix



Directory is a special kind of file:

struct dirent	name	inode #
	bin	1021
	dev	1001
	sbin	2011
	⋮	

← can be another directory.  
This turns the FS into a hierarchical tree.

This data (the table) can either live in the data blocks of a file (as in lab 5) or else in the inode of the directory (as in many of the examples that we will go over).

bootstrapping: root dir (/) always inode # 2

special names: /, ., ..

can navigate the name space with:

\$ cd name

\$ ls

Example: two files

/a/foo.c

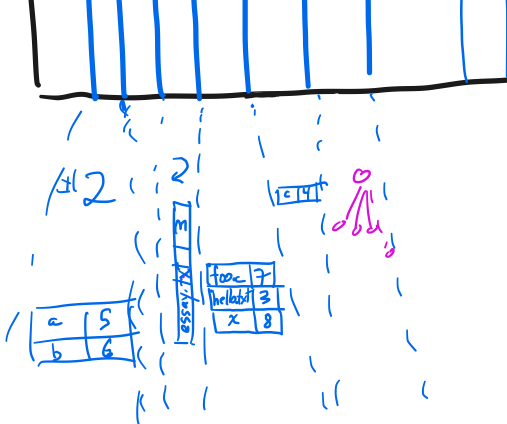
/b/c/essay.txt

what does the file system look like?

inodes



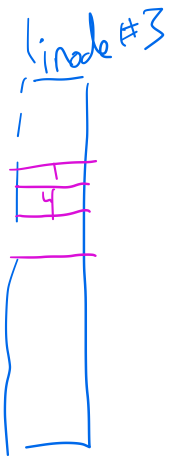
ks



links: (see also below)

/a/hello.txt ≡ /b/c/essay.txt

unlink();



hard link:

→ \$ ln /b/c/essay.txt /a/hello.txt

→ \$ ln -s /b/c/essay.txt /a/x

soft link:

# Links

Hard: \$ ln x y creates a synonym ("y") for ("x")

Question: what is the result of:

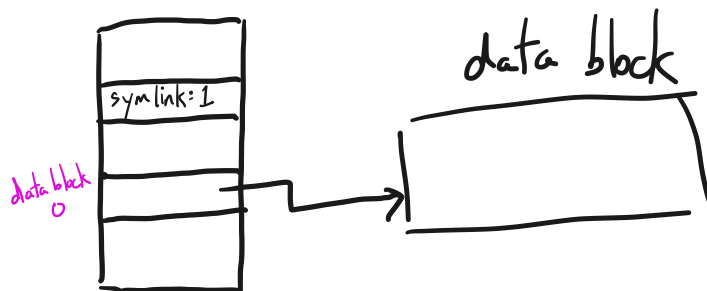
```
$ ln /b/c/essay.txt /a/hello.txt
```

?

Soft: 

```
$ ln -s x y
```

creates new inode. new file named "y". Its contents are "x".



### 3. Performance

case study: FFS (1984)

problems w/ the original:

- blocks too small (512B)
- inode array at the beginning of the disk
- free blocks stored in linked list on disk
- poor clustering of related objects

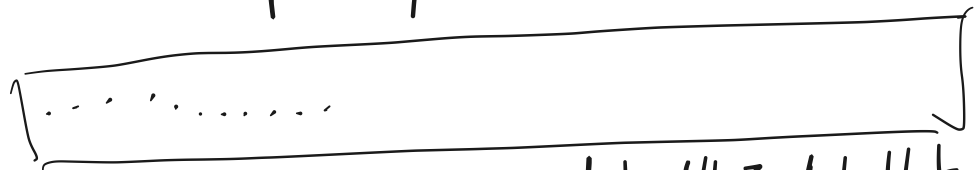
consecutive file blocks  
inodes relative to pointed-to disk blocks  
inodes for a given directory

result:

```
ls -l  
grep <path> *.c } ⇒ slow
```

Improvements?

- Make data blocks/inodes close to each other
- Cluster files in the same directory
- make data blocks bigger (4KB, 8KB, 16KB)
- free blocks: store separately (bitmap)

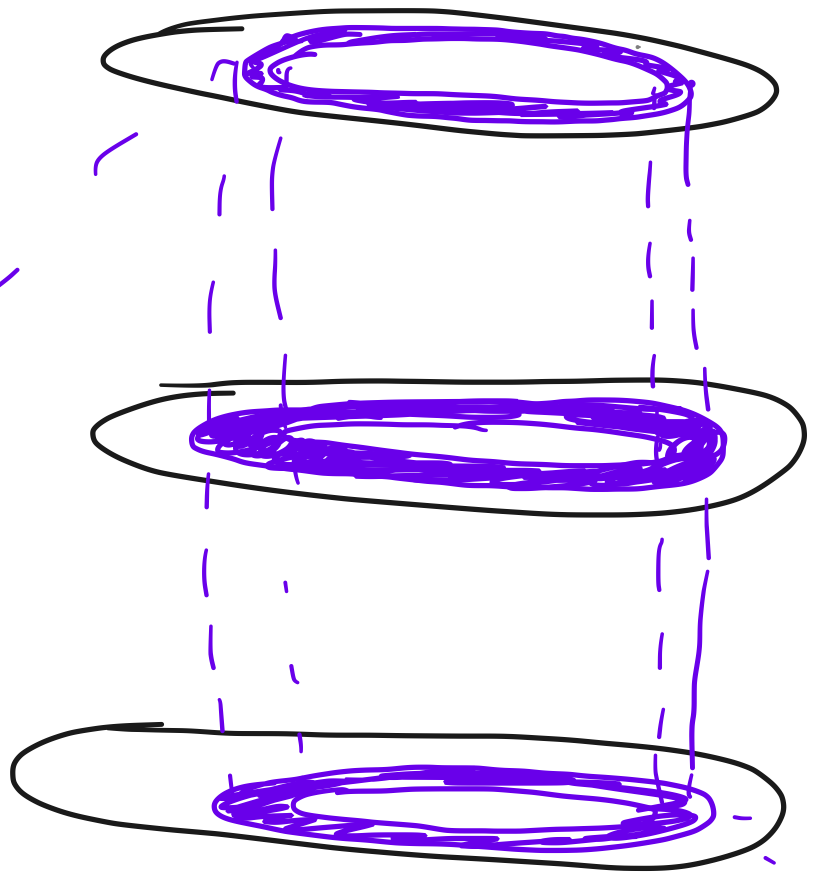


Assume 2TB disk, 4KB disk blocks  
How much space does the bitmap take up?

$$\frac{1 \text{ bit}}{\text{block}} \times \frac{2 \text{ TB}}{\text{block}} \times \frac{1 \text{ block}}{4 \text{ KB}} \times \frac{1 \text{ byte}}{8 \text{ bits}} = \frac{2^{29} \text{ bits}}{8 \text{ bits}} \text{ byte}$$

$$= 64 \text{ MB}$$

- reserve space (lie to the user about free space)
- symbolic links
- atomic "rename" (`$ mv abc.txt def.txt`)
- cylinder groups (for clustering)



[superblock | bookkeeping | inodes | bitmap | data blocks  
(512 bytes each)]

attempt to: put inodes and their data blocks in the same cyl. group

attempt to: put inodes of files in the same dir in the same cyl. group.

new directory: place in cyl. group w/ higher than avg. # of free inodes

as a file grows, after it crosses 40KB, spill to next cyl. group, and do likewise for every 1MB thereafter.

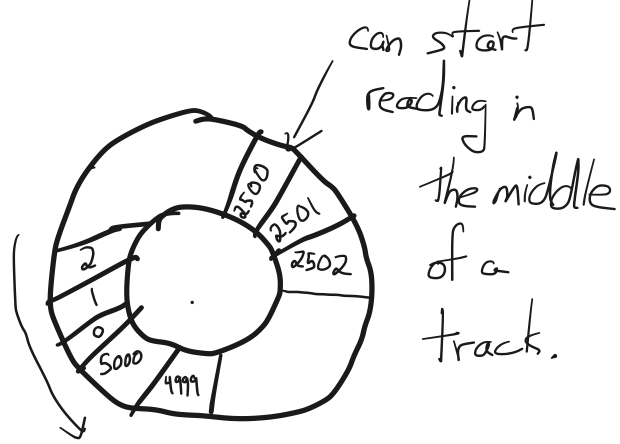
total perf:

20-40% of disk bandwidth for large files

10-20x improvement on the predecessor

Other things they did:

- buffer cache
- read entire track
- write in big chunks
- read ahead in big chunks (64KB)





# Directories

Approach 1: Single dir for whole system

map : <name, inumber>

mike-todo.txt, 64  
chris-todo.txt, 121

⋮

Approach 2: Single dir for each user



mkdir



Approach 3: Hierarchical name space

Directory maps from names to files OR  
other directories

File system then forms a tree or graph.

Large name spaces tend to be hierarchical

Ex: IP addresses, domain names

Cloud computing infrastructure has changed that! Google docs, for example.

www.cs.nyu.edu.

DNS



edu.

www.cs.nyu.edu.

/usr/mike/cs202/lab3/4.cpp

/usr/jol\_