



---

# File System Design

David E. Culler  
CS162 – Operating Systems and Systems  
Programming  
Lecture 23  
October 22, 2014

Reading: A&D 13.1-3a  
HW 4 out  
Proj 2 out

# Big hairy thought-provoking question to help review recent topics

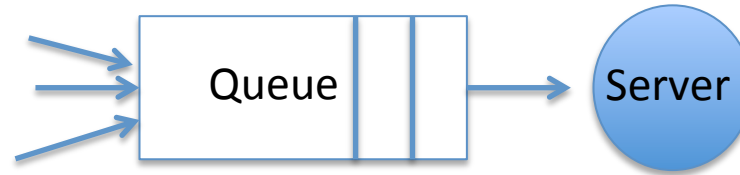
---



- The One vs The All
- Is improving *response time* (the One) in alignment or in opposition to improving *throughput* (the All) ?
- E.g., C-SCAN ?
- Delay servicing queue?



# Performance: multiple outstanding requests



- Suppose each read takes 10 ms to service.
- If a process works for 100 ms after each read, what is the utilization of the disk?
  - $U = 10 \text{ ms} / 110\text{ms} = 9\%$
- What if there are two such processes?
  - $U = (10 \text{ ms} + 10 \text{ ms}) / 110\text{ms} = 18\%$
- What if each of those processes has two such threads?



# How else do we hide I/O latency?

---

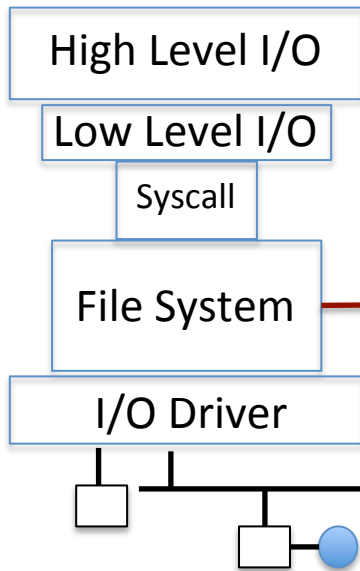
- **Blocking Interface: “Wait”**
  - When request data (*e.g.*, `read()` system call), put process to sleep until data is ready
  - When write data (*e.g.*, `write()` system call), put process to sleep until device is ready for data
- **Non-blocking Interface: “Don’t Wait”**
  - Returns quickly from read or write request with count of bytes successfully transferred to kernel
  - Read may return nothing, write may write nothing
- **Asynchronous Interface: “Tell Me Later”**
  - When requesting data, take pointer to user’s buffer, return immediately; later kernel fills buffer and notifies user
  - When sending data, take pointer to user’s buffer, return immediately; later kernel takes data and notifies user



# I/O & Storage Layers

## Operations, Entities and Interface

### Application / Service



*streams*

*handles*

*registers*

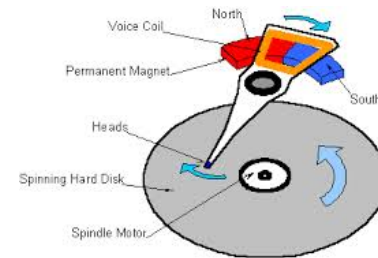
`file_open, file_read, ...` on **struct file \*** & `void *`

*descriptors*

***we are here ...***

*Commands and Data Transfers*

*Disks, Flash, Controllers, DMA*



# So you are going to design a file system ...

---



- What factors are critical to the design choices?



# Recall: C Low level I/O

- Operations on File Descriptors – as OS object representing the state of a file
  - User has a “handle” on the descriptor

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>

int open (const char *filename, int flags [, mode_t mode])
int creat (const char *filename, mode_t mode)
int close (int filedes)
```

Bit vector of:

- Access modes (Rd, Wr, ...)
- Open Flags (Create, ...)
- Operating modes (Appends, ...)

Bit vector of Permission Bits:

- User|Group|Other X R|W|X

[http://www.gnu.org/software/libc/manual/html\\_node/Opening-and-Closing-Files.html](http://www.gnu.org/software/libc/manual/html_node/Opening-and-Closing-Files.html)



# Recall: C Low Level Operations

---

```
ssize_t read (int filedes, void *buffer, size_t maxsize)
```

- returns bytes read, 0 => EOF, -1 => error

```
ssize_t write (int filedes, const void *buffer, size_t size)
```

- returns bytes written

```
off_t lseek (int filedes, off_t offset, int whence)
```

```
int fsync (int filedes) – wait for i/o to finish
```

```
void sync (void) – wait for ALL to finish
```

- When write returns, data is on its way to disk and can be read, but it may not actually be permanent!



# So you are going to design a file system ...

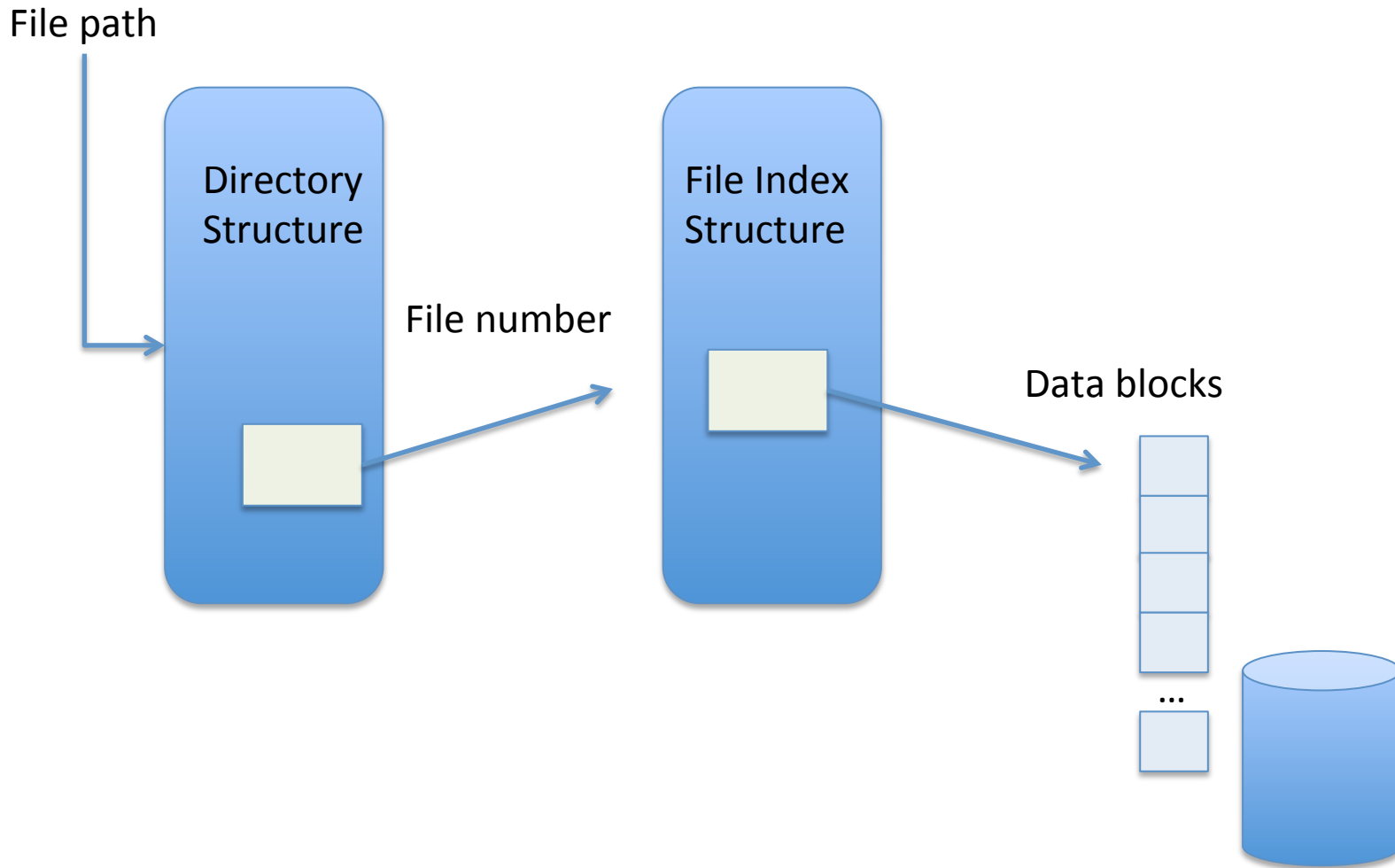
---



- What factors are critical to the design choices?
- Durable data store => it's all on disk
- Disks Performance !!!
  - Maximize sequential access, minimize seeks
- Open before Read/Write
  - Can perform protection checks and look up where the actual file resource are, in advance
- Size is determined as they are used !!!
  - Can write (or read zeros) to expand the file
  - Start small and grow, need to make room
- Organized into directories
  - What data structure (on disk) for that?
- Need to allocate / free blocks
  - Such that access remains efficient



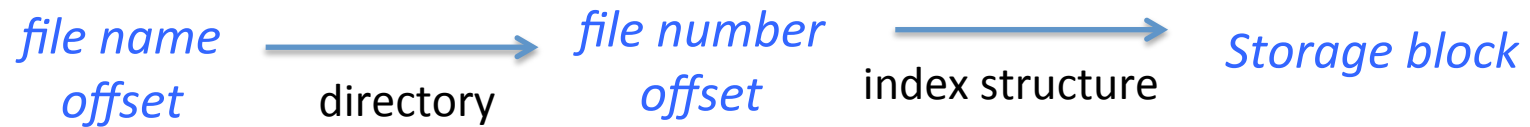
# Components of a File System





# Components of a file system

---



- Open performs *name resolution*
  - Translates pathname into a “file number”
    - Used as an “index” to locate the blocks
  - Creates a file descriptor in PCB within kernel
  - Returns a “handle” (another int) to user process
- Read, Write, Seek, and Sync operate on handle
  - Mapped to descriptor and to blocks

# Directories



FAVORITES	Name	Applications	Date Modified	Size	Kind
Home	culler				
All My Files					
AirDrop					
Applications					
Desktop					
Documents					
Downloads					
DEVICES					
David's M...					
Remote Disc					
TAGS					
Red					
Orange					
Yellow					
Green					
Blue					
Purple					
Gray					
All Tags...					
	bse		Yesterday, 6:21 PM	--	Folder
	Classes		Oct 13, 2014, 10:19 PM	--	Folder
	AIT2008		Oct 13, 2014, 10:11 PM	--	Folder
	CS-Scholars		Oct 13, 2014, 10:11 PM	--	Folder
	cs61cl-f08		Oct 13, 2014, 10:17 PM	--	Folder
	cs61cl-f09		Oct 13, 2014, 10:19 PM	--	Folder
	cs162		Today, 8:36 AM	--	Folder
	AndersonDahlin		Oct 13, 2014, 10:11 PM	--	Folder
	fa14		Today, 8:36 AM	--	Folder
	162prereqcheckSept8.xlsx		Sep 10, 2014, 3:20 PM	36 KB	Micros...kbook
	coursecomparison.xlsx		Aug 6, 2014, 7:50 AM	31 KB	Micros...kbook
	CS 162 apps.xlsx		Jun 29, 2014, 6:35 AM	53 KB	Micros...kbook
	cs162git		Sep 23, 2014, 11:33 AM	--	Folder
	devel		Oct 15, 2014, 11:40 AM	--	Folder
	exams		Oct 13, 2014, 10:12 PM	--	Folder
	gitprojects		Oct 8, 2014, 4:52 PM	--	Folder
	group0		Today, 8:35 AM	--	Folder
	pintos		Today, 8:35 AM	--	Folder
	src		Today, 8:35 AM	--	Folder
	gradesheet.xls		Sep 19, 2014, 4:48 PM	68 KB	Micros...kbook
	GSI Section Coverage.xlsx		Aug 22, 2014, 1:29 PM	11 KB	Micros...kbook
	Lectures		Today, 8:22 AM	--	Folder
	pintos-notes.txt		Sep 14, 2014, 2:10 PM	1 KB	Plain Text
	pintos.pdf		Jul 21, 2014, 10:17 AM	549 KB	PDF Document
	roster-9-13.xls		Sep 13, 2014, 5:12 PM	83 KB	Micros...kbook
	roster-9-19.xls		Sep 19, 2014, 4:39 PM	84 KB	Micros...kbook
	staff.xlsx		Aug 6, 2014, 7:14 AM	34 KB	Micros...kbook
	student		Oct 13, 2014, 10:12 PM	--	Folder
	studentsExcelFile-10-20		Yesterday, 9:53 AM	84 KB	Micros...kbook
	syllabus-fa14.xlsx		Sep 12, 2014, 10:00 AM	38 KB	Micros...kbook
	tmp		Oct 13, 2014, 10:12 PM	--	Folder
	pintos		Aug 8, 2014, 6:06 AM	--	Folder
	sp14		May 14, 2014, 9:02 PM	--	Folder
	cs194		Oct 13, 2014, 10:16 PM	--	Folder
	cs262b		Aug 7, 2013, 7:55 AM	--	Folder



# Directory

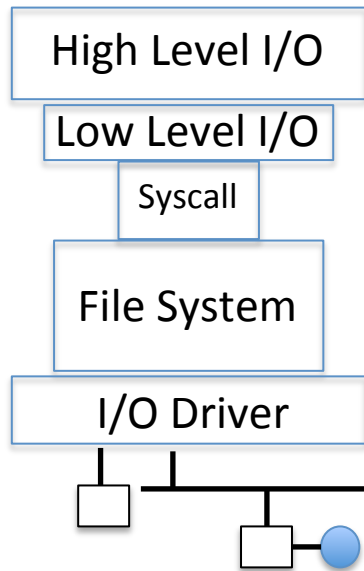
---

- Basically a hierarchical structure
- Each directory entry is a collection of
  - Files
  - Directories
    - A link to another entries
- Each has a name and attributes
  - Files have data
- Links (hard links) make it a DAG, not just a tree
  - Softlinks (aliases) are another name for an entry



# I/O & Storage Layers

## Application / Service



*streams*

*handles*

*registers*

*descriptors*

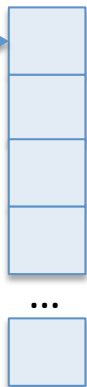
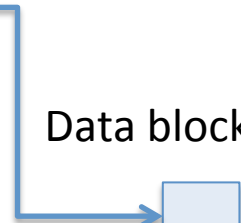
*Commands and Data Transfers*

*Disks, Flash, Controllers, DMA*

#4 - handle



Data blocks

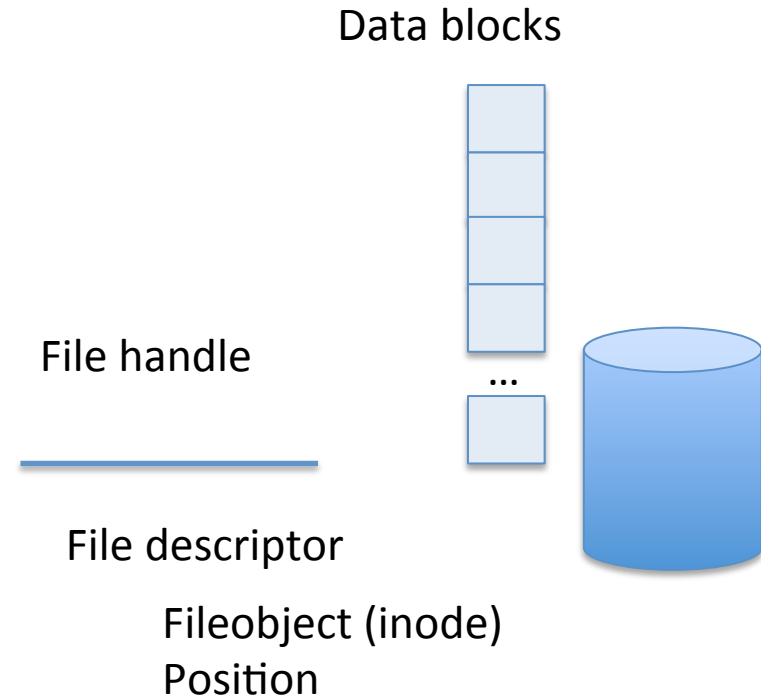


Directory Structure



# File

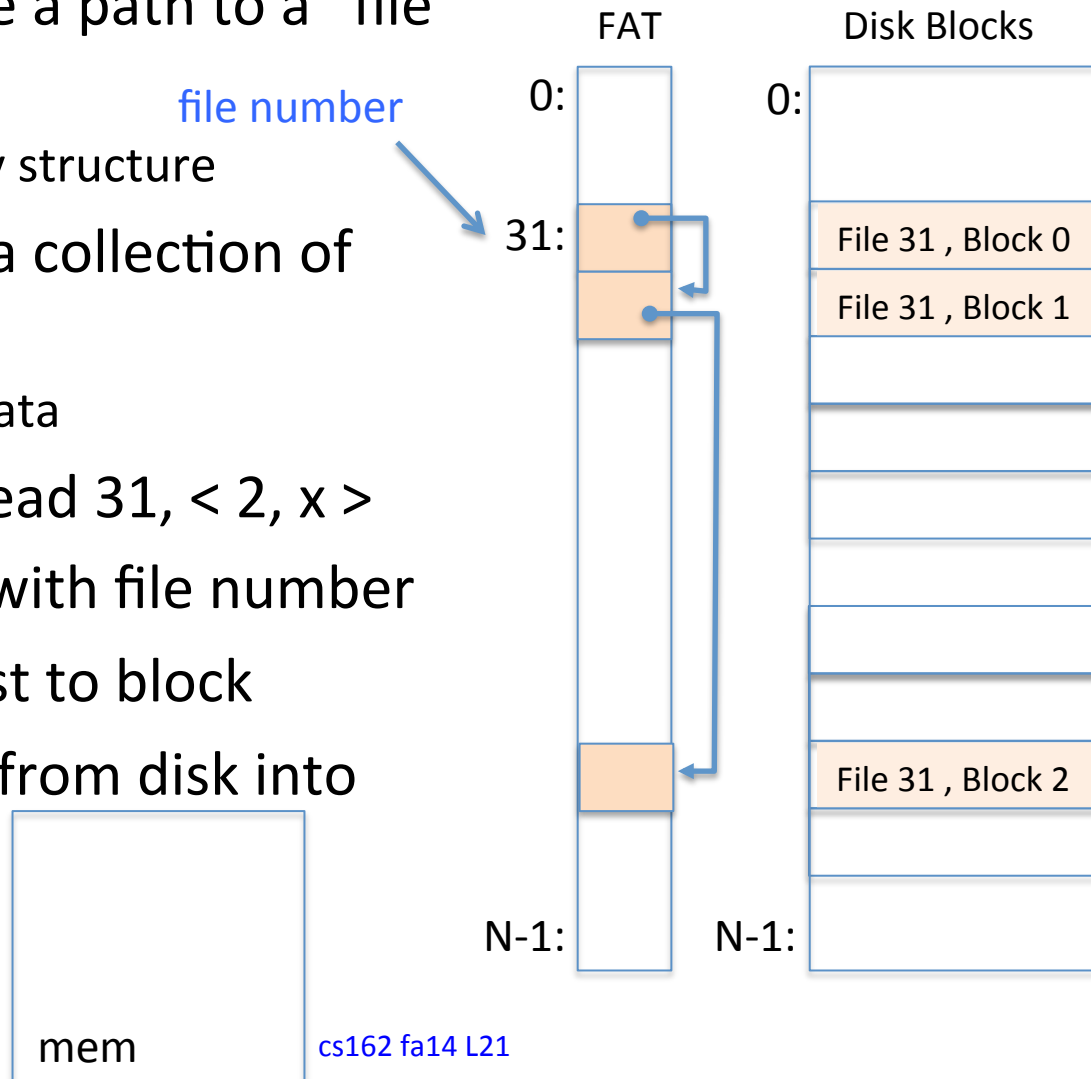
- Named permanent storage
- Contains
  - Data
    - Blocks on disk somewhere
  - Metadata (Attributes)
    - Owner, size, last opened, ...
    - Access rights
      - R, W, X
      - Owner, Group, Other (in Unix systems)
      - Access control list in Windows system





# FAT (File Allocation Table)

- Assume (for now) we have a way to translate a path to a “file number”
  - i.e., a directory structure
- Disk Storage is a collection of Blocks
  - Just hold file data
- Example: `file_read 31, < 2, x >`
- Index into FAT with file number
- Follow linked list to block
- Read the block from disk into mem

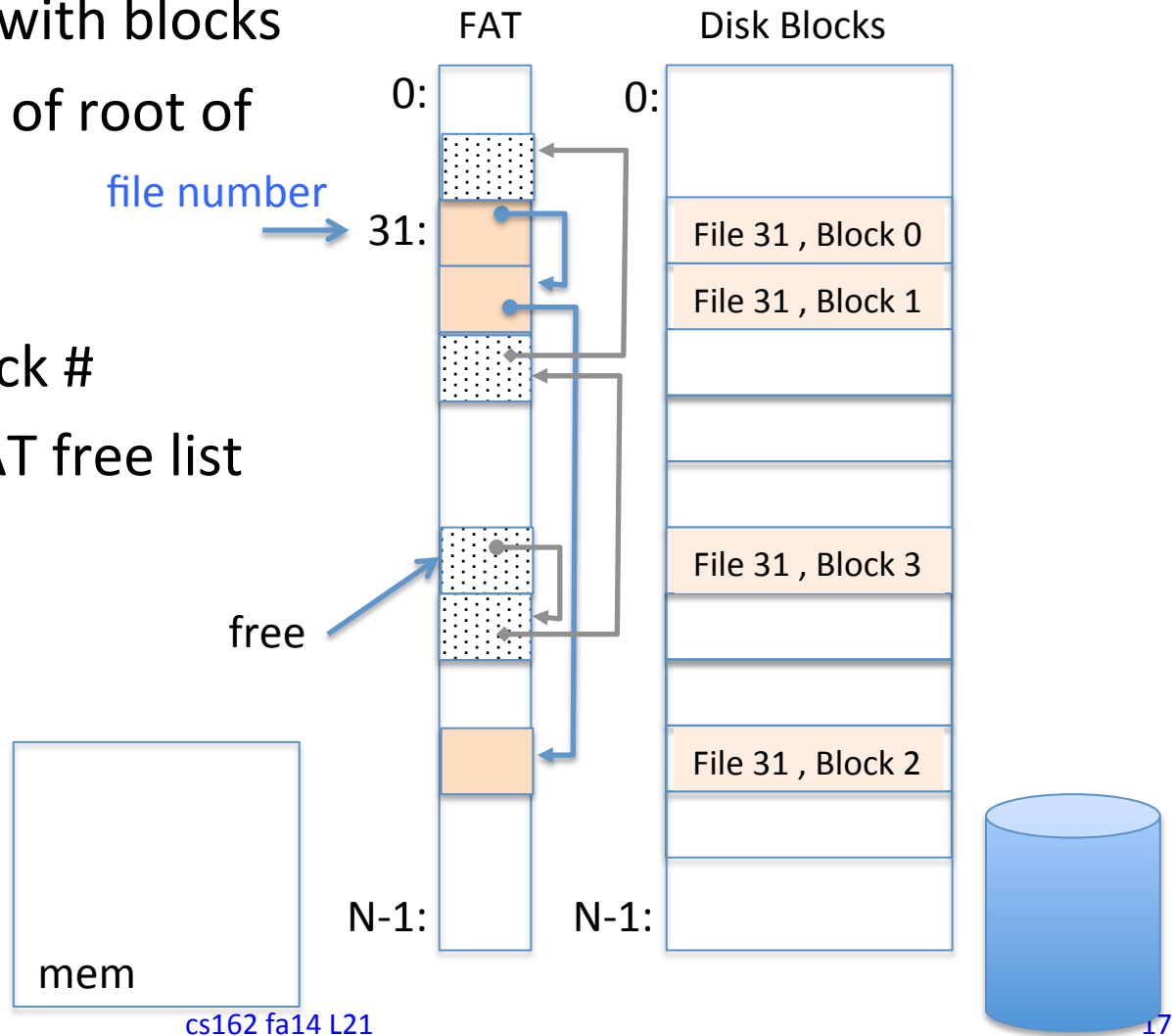






# FAT (File Allocation Table)

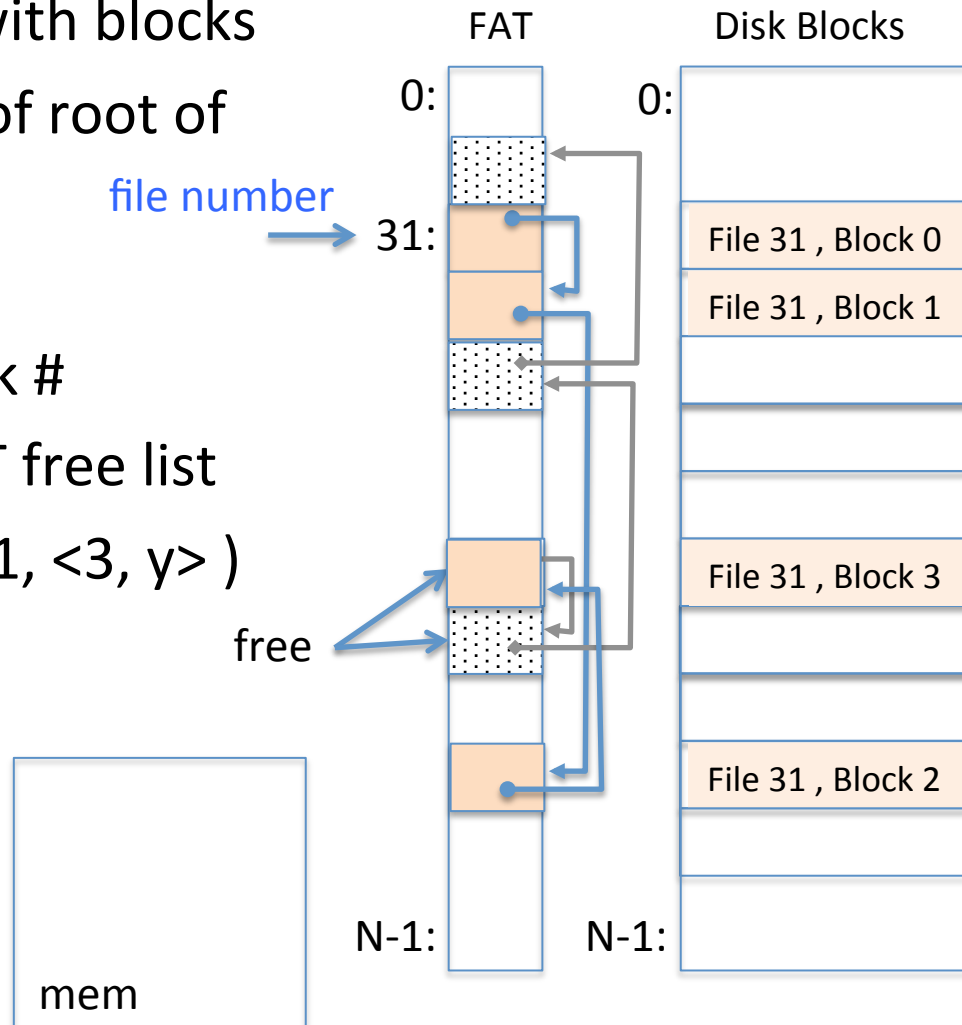
- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- File offset ( $o = B:x$ )
- Follow list to get block #
- Unused blocks  $\Leftrightarrow$  FAT free list





# FAT (File Allocation Table)

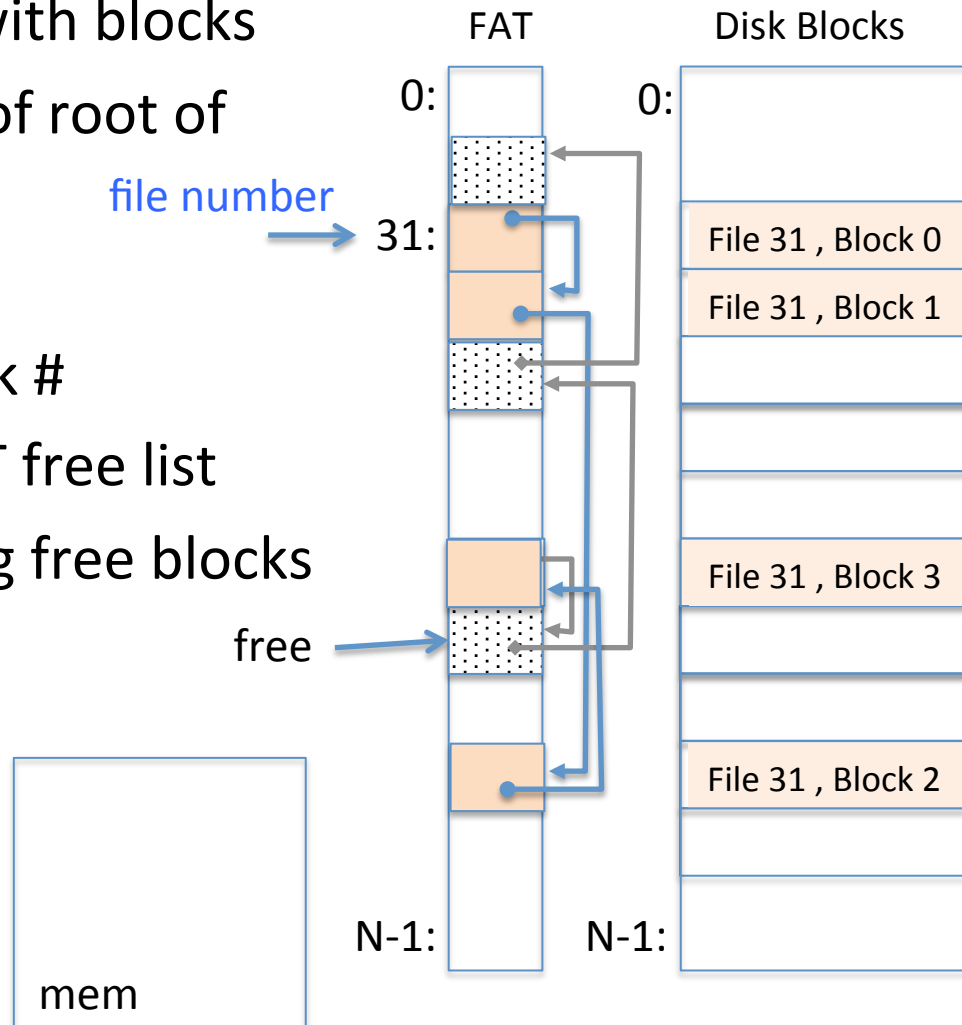
- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- File offset ( $o = B:x$ )
- Follow list to get block #
- Unused blocks  $\Leftrightarrow$  FAT free list
- Example: `file_write(51, <3, y>)`





# FAT (File Allocation Table)

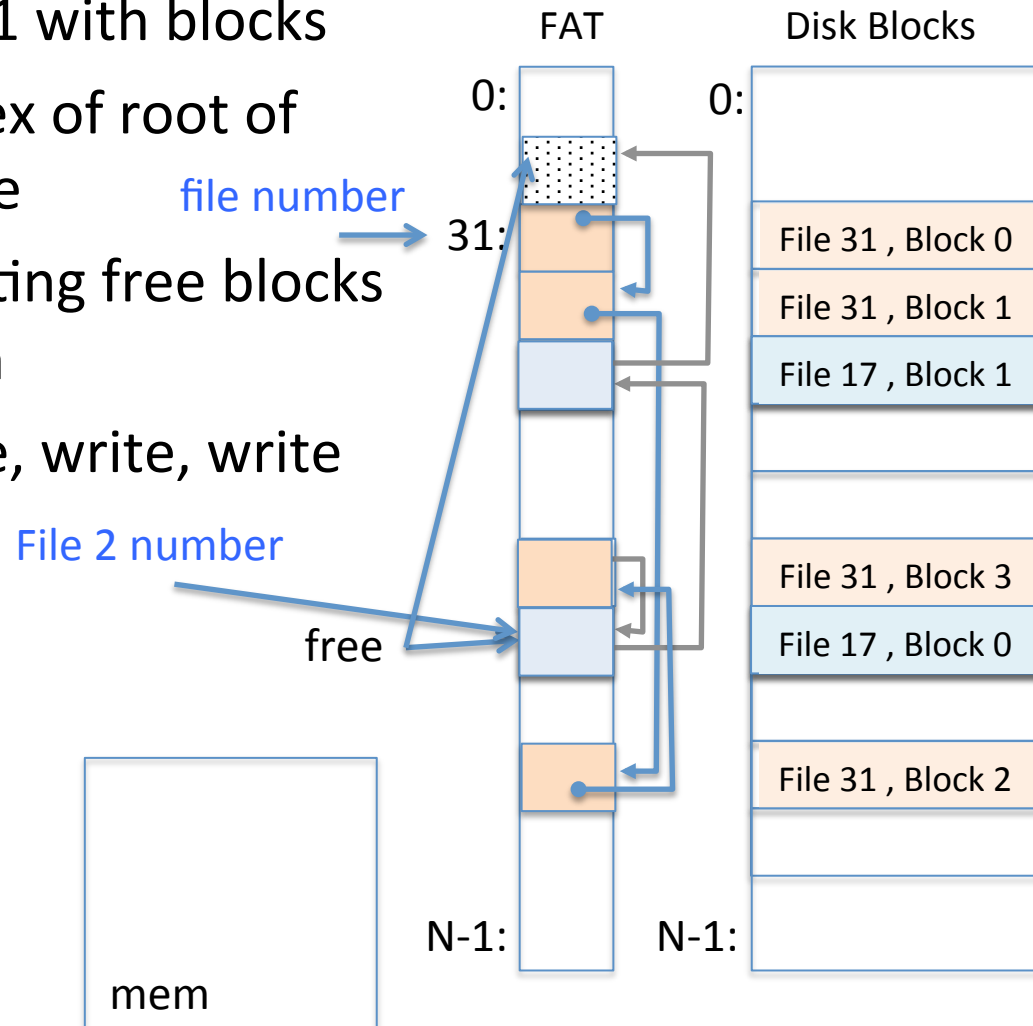
- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- File offset ( $o = B:x$ )
- Follow list to get block #
- Unused blocks  $\Leftrightarrow$  FAT free list
- Grow file by allocating free blocks and linking them in



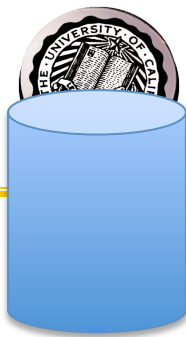


# FAT (File Allocation Table)

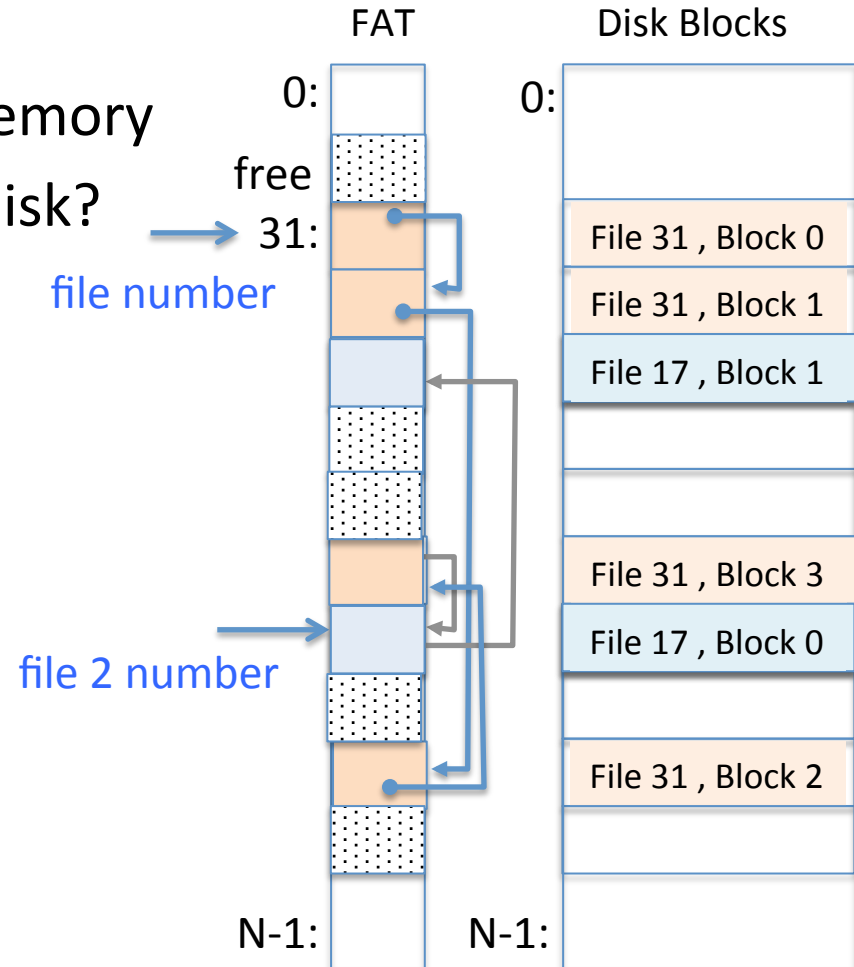
- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- Grow file by allocating free blocks and linking them in
- Example Create file, write, write

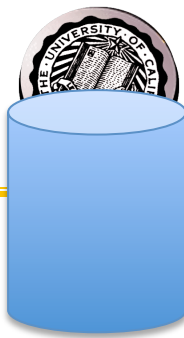


# FAT Assessment



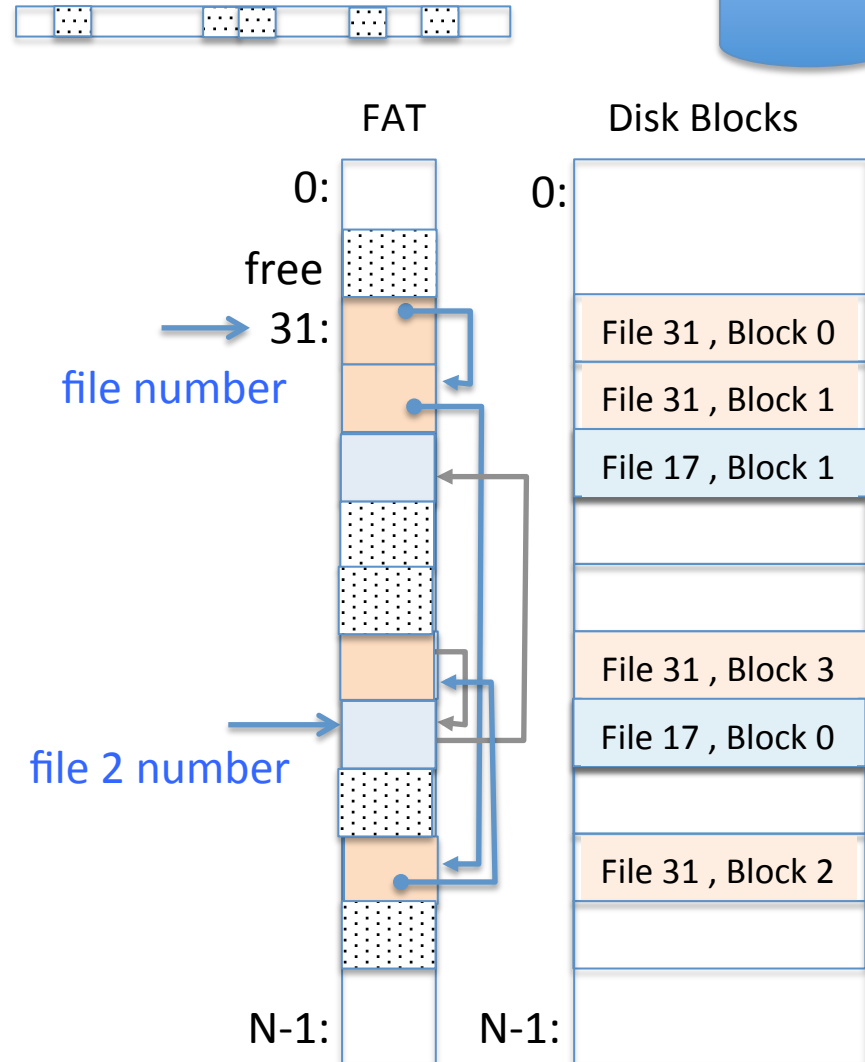
- Used in DOS, Windows, thumb drives, ...
- Where is FAT stored ?
- On Disk, restore on boot, copy in memory
- What happens when you format a disk?
- Zero the blocks, link up the FAT free file number
- Simple





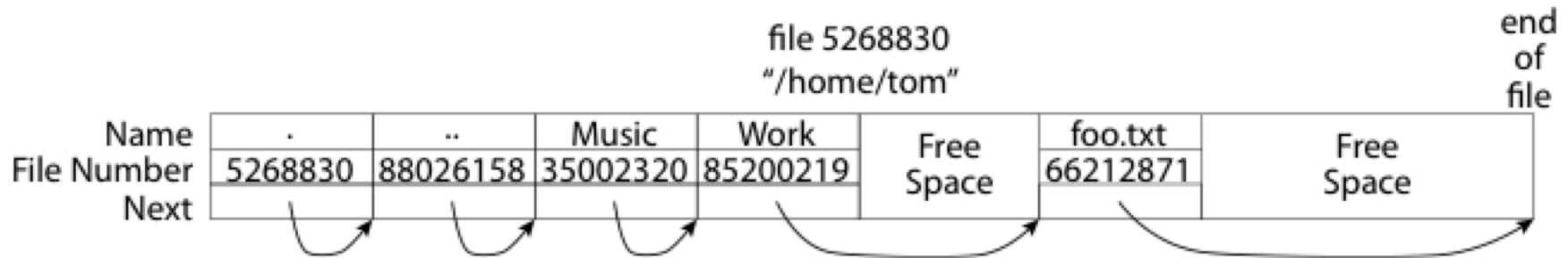
# FAT Assessment

- Time to find block (large files) ??
- Free list usually just a bit vector
- Next fit algorithm
- Block layout for file ???
- Sequential Access ???
- Random Access ???
- Fragmentation ???
- Small files ???
- Big files ???





# What about the Directory?

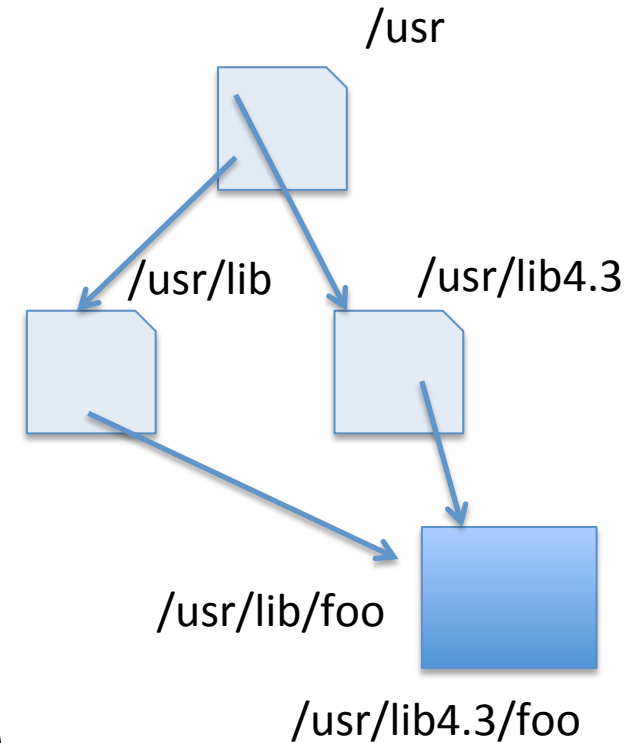


- Essentially a file containing `<file_name: file_number>` mappings
- Free space for new entries
- In FAT: attributes kept in directory (!!!)
- Each directory a linked list of entries
- Where do you find root directory ( `"/"` )



# Bit more on directories

- Stored in files, can be read, but don't
  - System calls to access directories
  - Open / Creat traverse the structure
  - mkdir /rmdir add/remove entries
  - Link / Unlink
    - Link existing file to a directory
      - Not in FAT !
    - Forms a DAG



- **libc support**

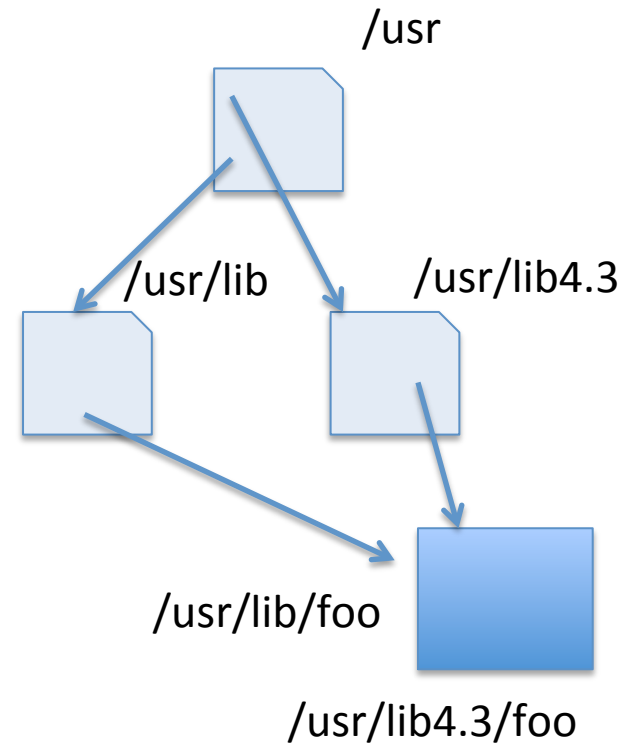
- DIR \* opendir (const char \*dirname)
- struct dirent \* readdir (DIR \*dirstream)
- int readdir\_r (DIR \*dirstream, struct dirent \*entry, struct dirent \*\*result)





# When can a file be deleted ?

- Maintain reference count of links to the file.
- Delete after the last reference is gone.





# Big FAT security holes

---

- FAT has no access rights
- FAT has no header in the file blocks
- Just gives an index into the FAT
  - (file number = block number)



# Characteristics of Files

A Five-Year Study of File-System Metadata

NITIN AGRAWAL  
University of Wisconsin, Madison  
and  
WILLIAM J. BOLOSKY, JOHN R. DOUCEUR, and JACOB R. LORCH  
Microsoft Research

- Most files are small
- Most of the space is occupied by the rare big ones

A Five-Year Study of File-System Metadata • 9:9

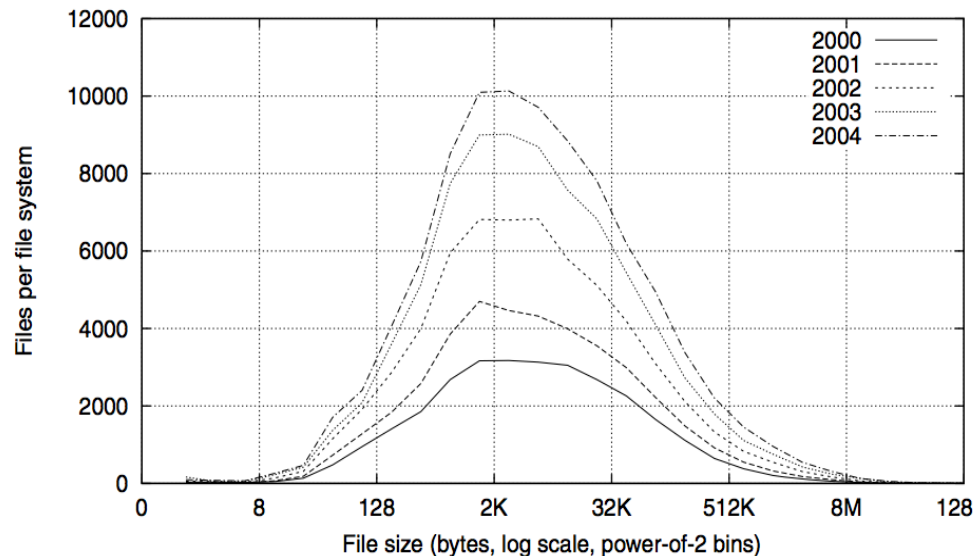


Fig. 2. Histograms of files by size.

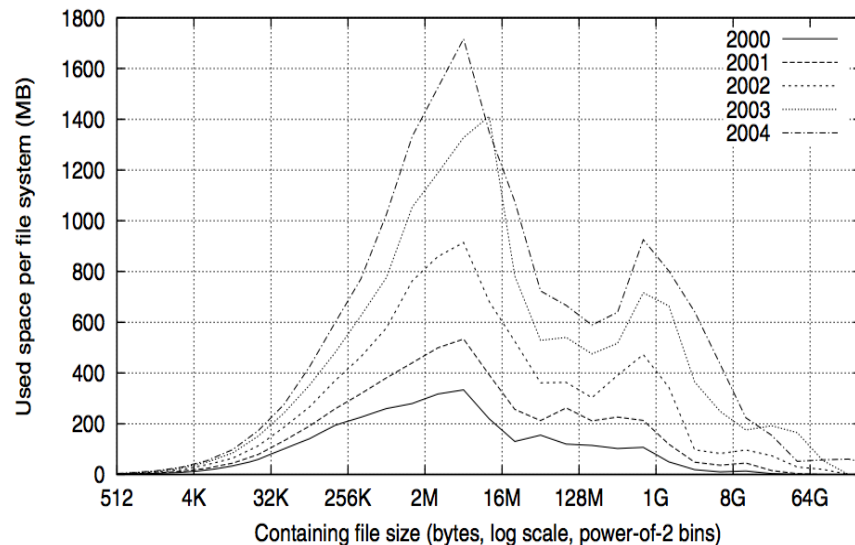


Fig. 4. Histograms of bytes by containing file size.



# So what about a “real” file system

- Meet the inode

