File System Design

David E. Culler

CS162 – Operating Systems and Systems Programming

Lecture 23

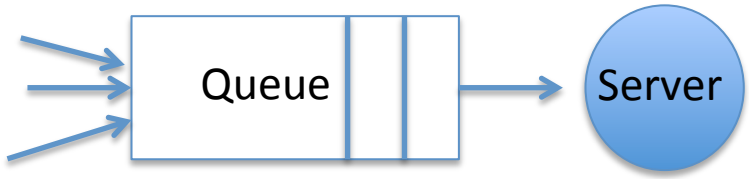
October 22, 2014

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 ms / 110ms = 9%
- What it there are two such processes?
  - U = (10 ms + 10 ms) / 110ms = 18%
- What if each of those processes have two such threads?

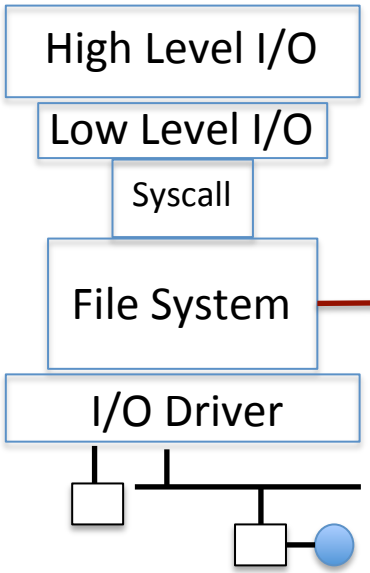
How else do we hide I/O latency?

- 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

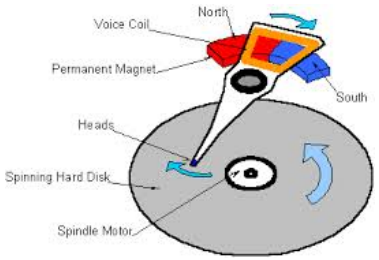
file_open, file_read, … on **struct file *** & void *

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

Recall: C Low Level Operations

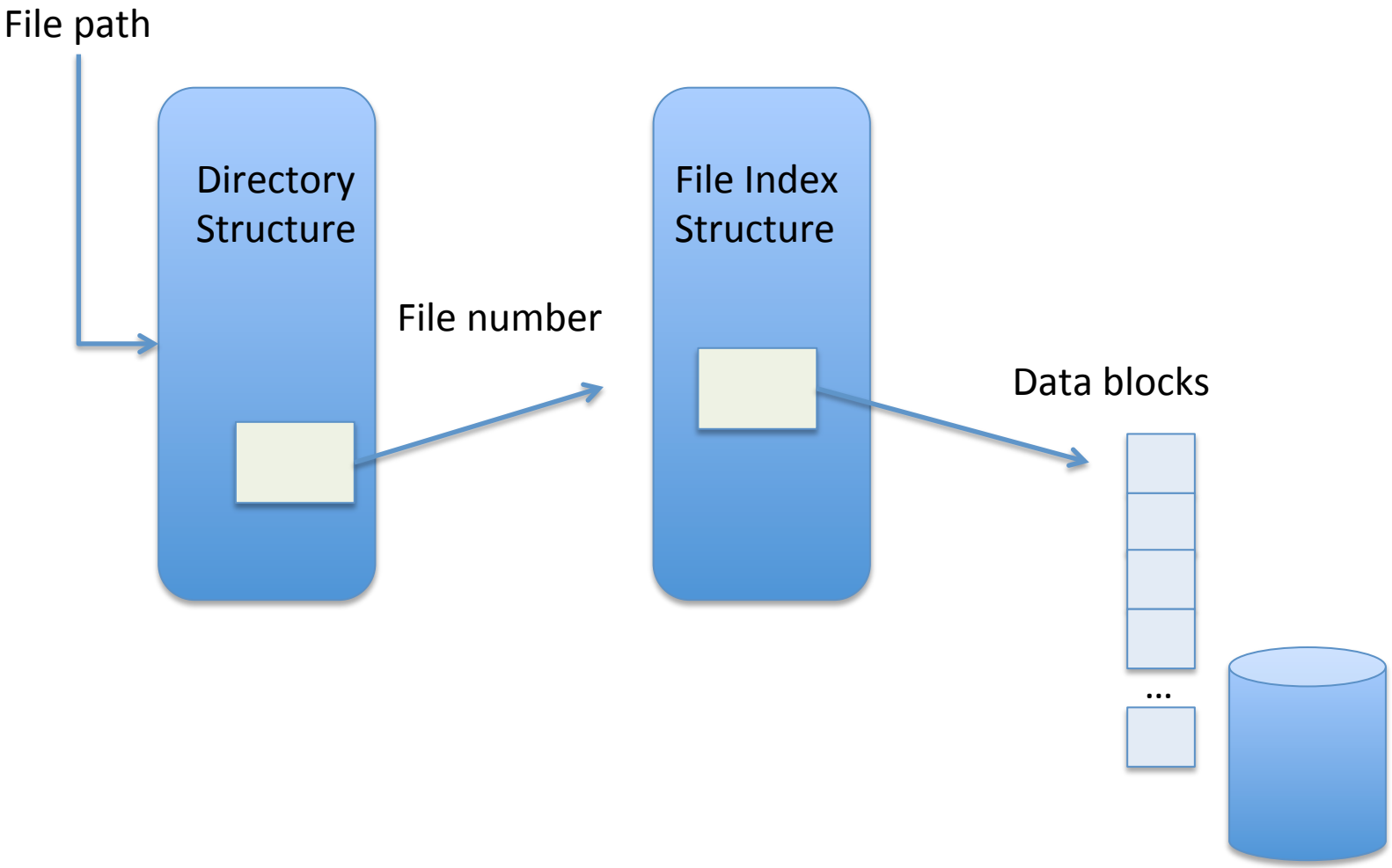
```
ssize_t read (int filedes, void *buffer, size_t maxsize)
 - returns bytes read, 0 => EOF, -1 => error
ssize_t write (int filedes, const void *buffer, size_t size)
 - returns bytes written

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

int fsync (int fildes) – 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!

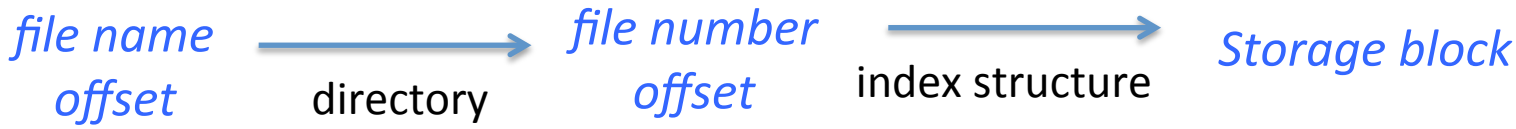
So you are going to design a file system …

- 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

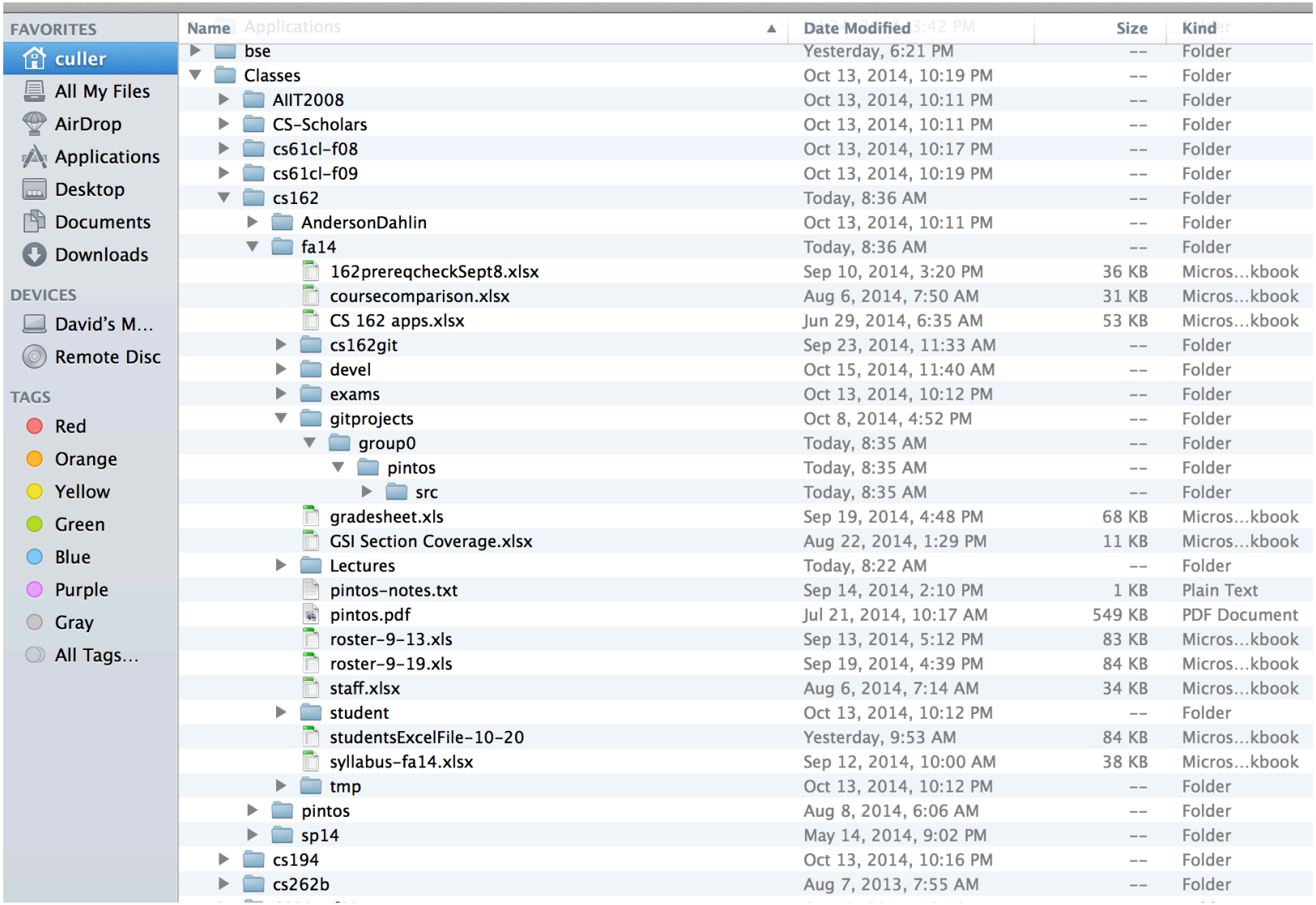


Components of a file system



- 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

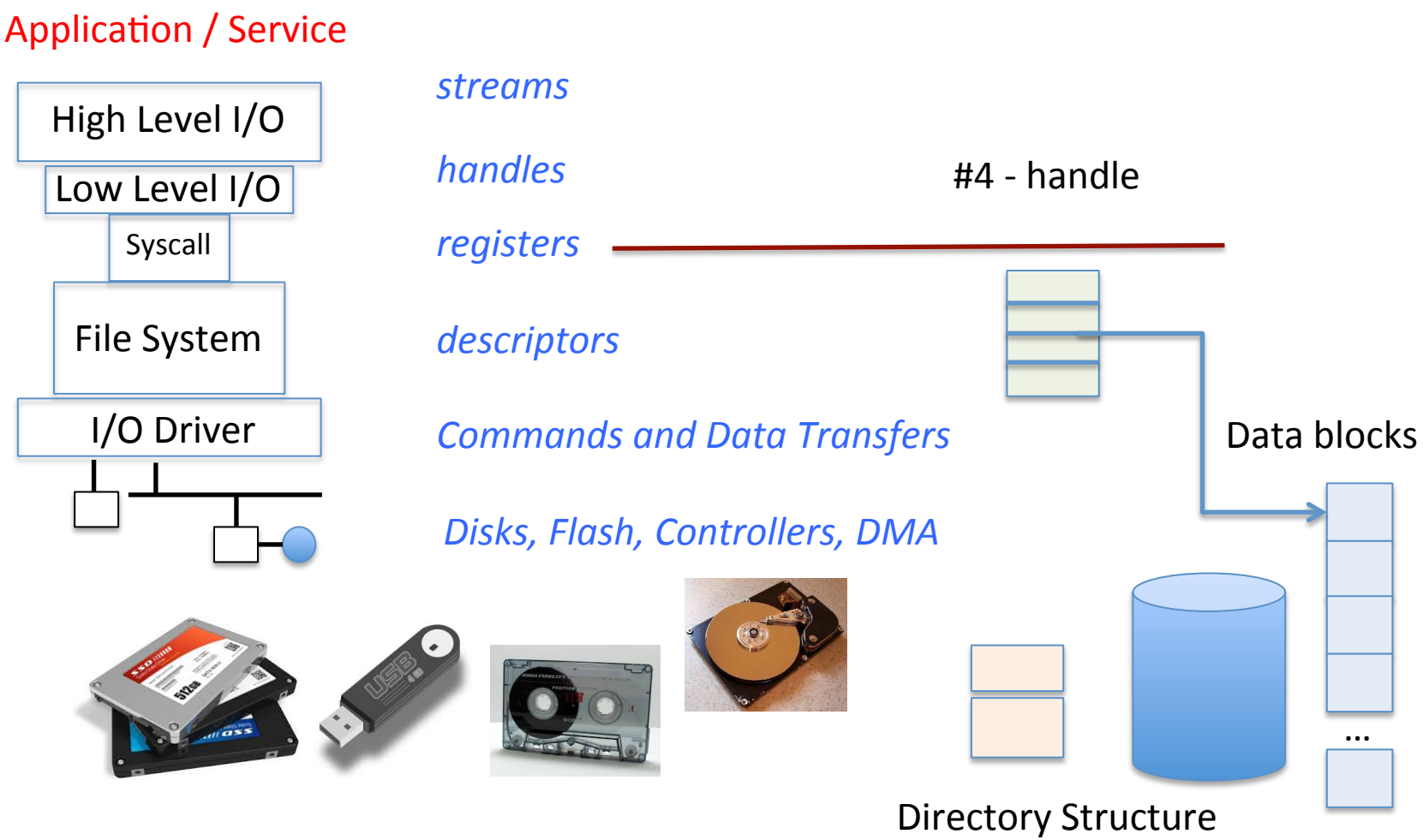
Directories



Directory

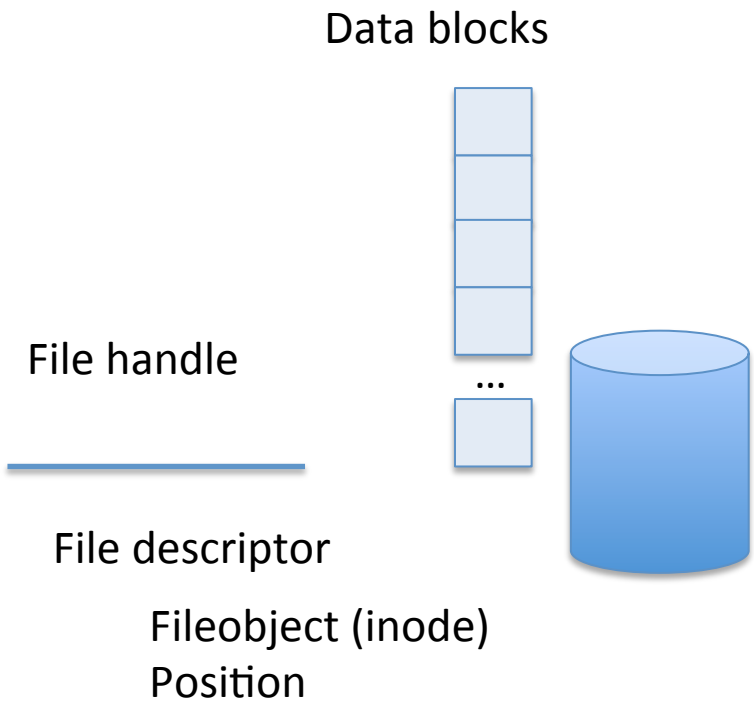
- Basically a hierarchical structure
- Each directory entry is a collection of
  - Files
  - Directories
    - A link to another entries
- Each has a name and attributes
  - Files have data
- Links (hard links) make it a DAG, not just a tree
  - Softlinks (aliases) are another name for an entry

I/O & Storage Layers



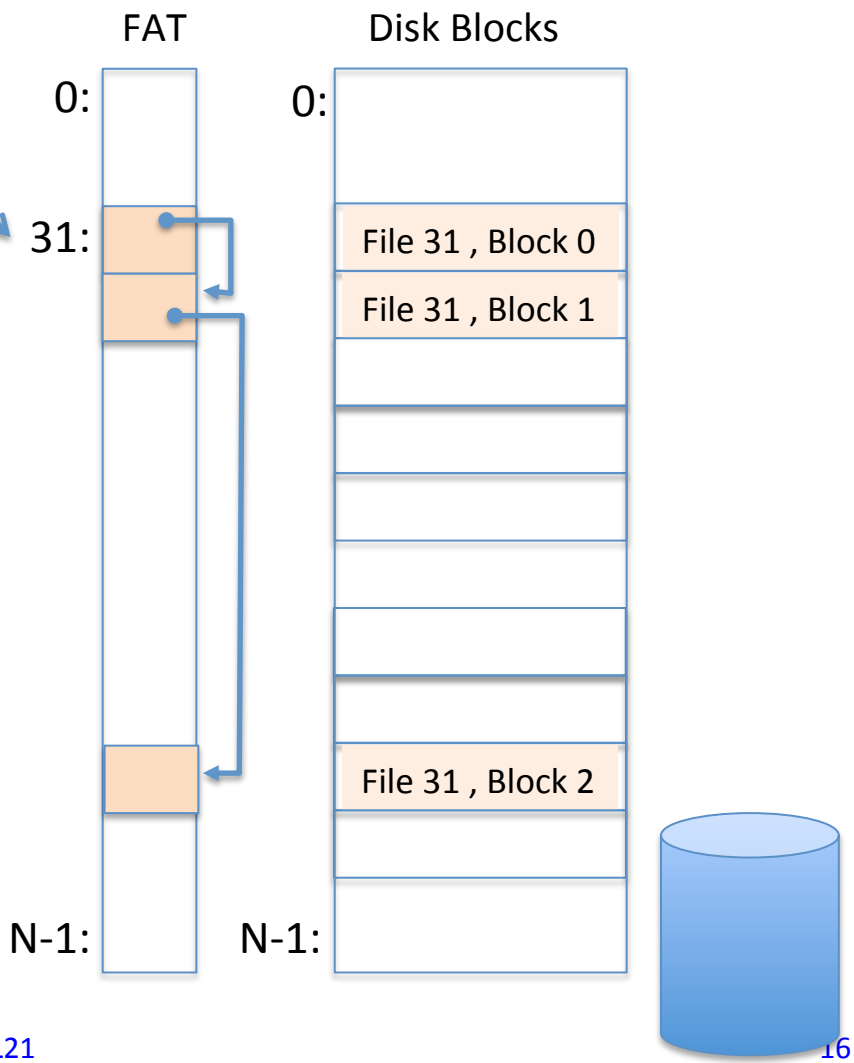
File

- 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



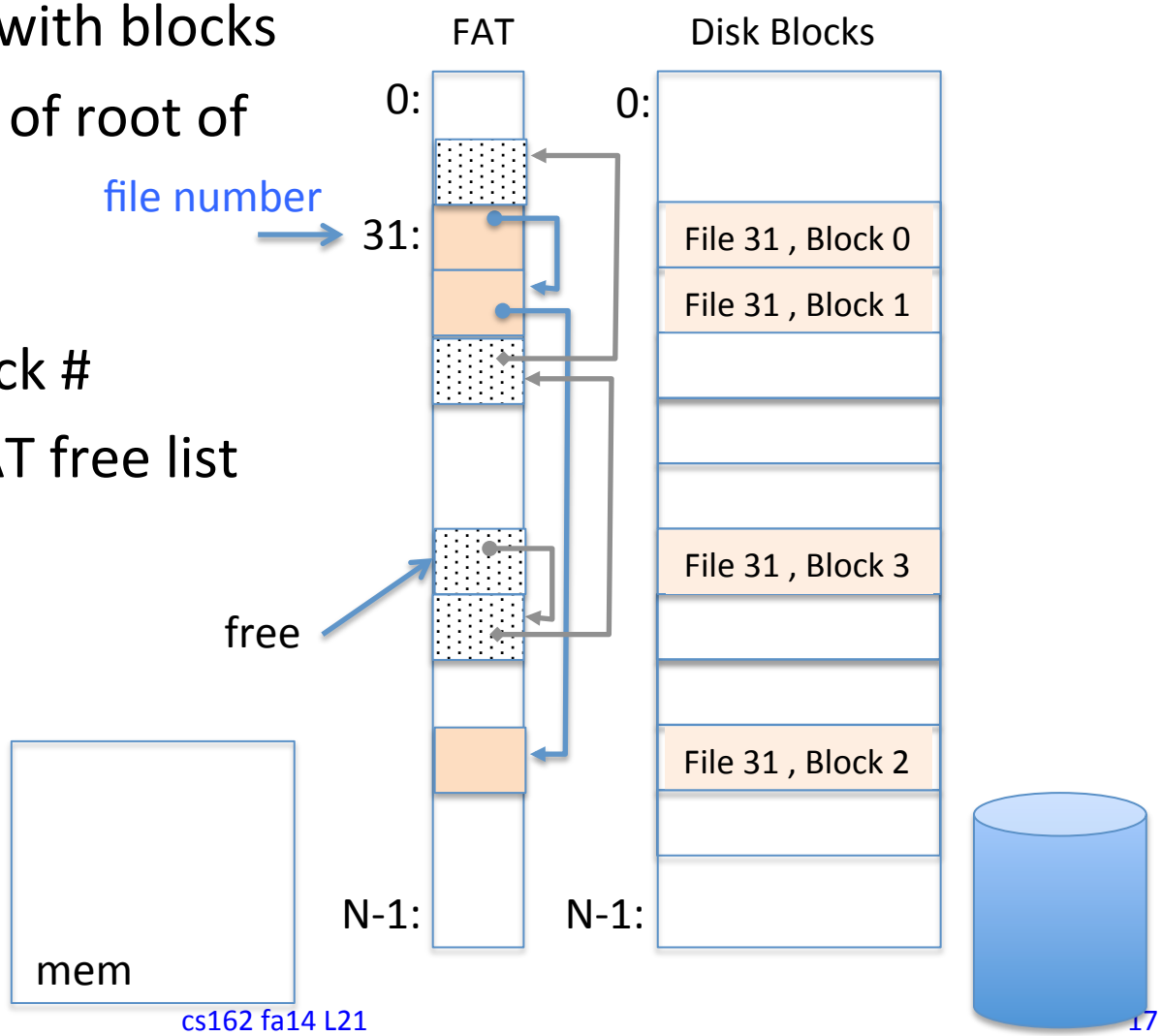
FAT (File Allocation Table)

- 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



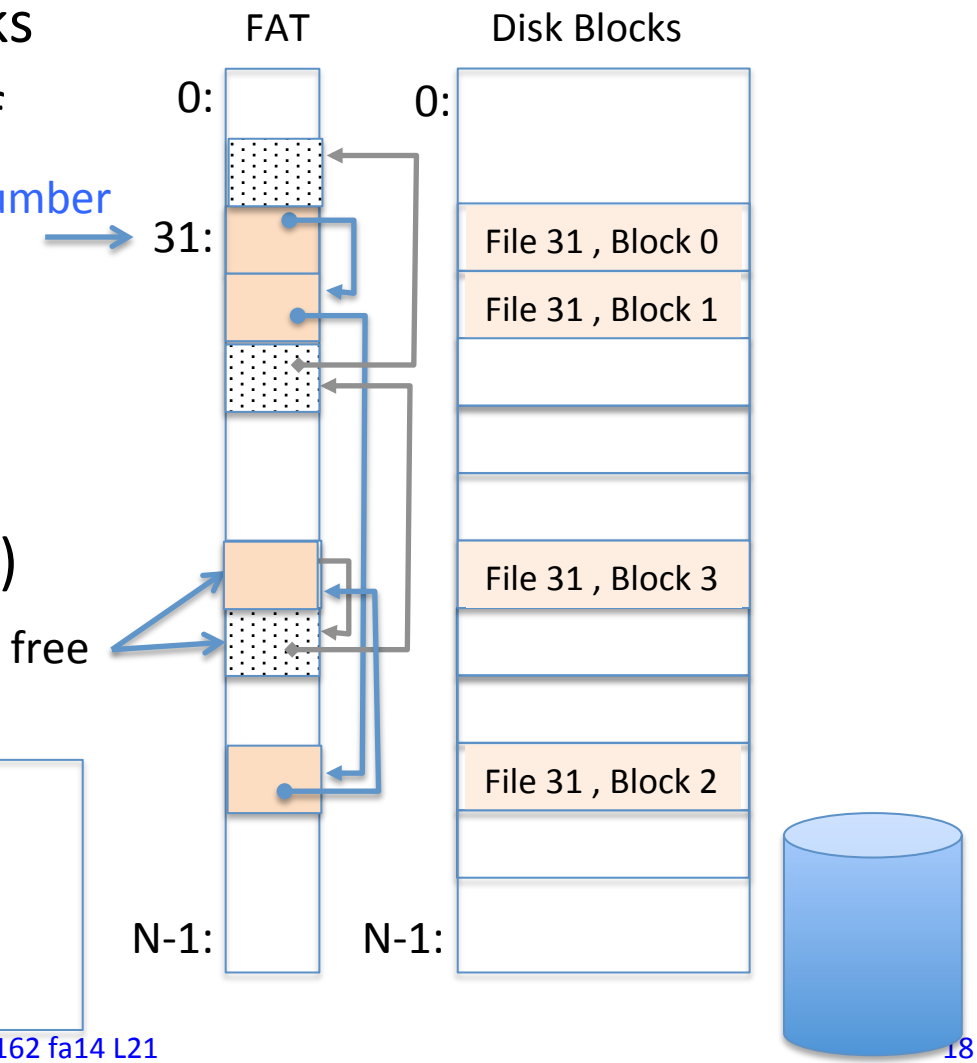
FAT (File Allocation Table)

- 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 ⇔ FAT free list



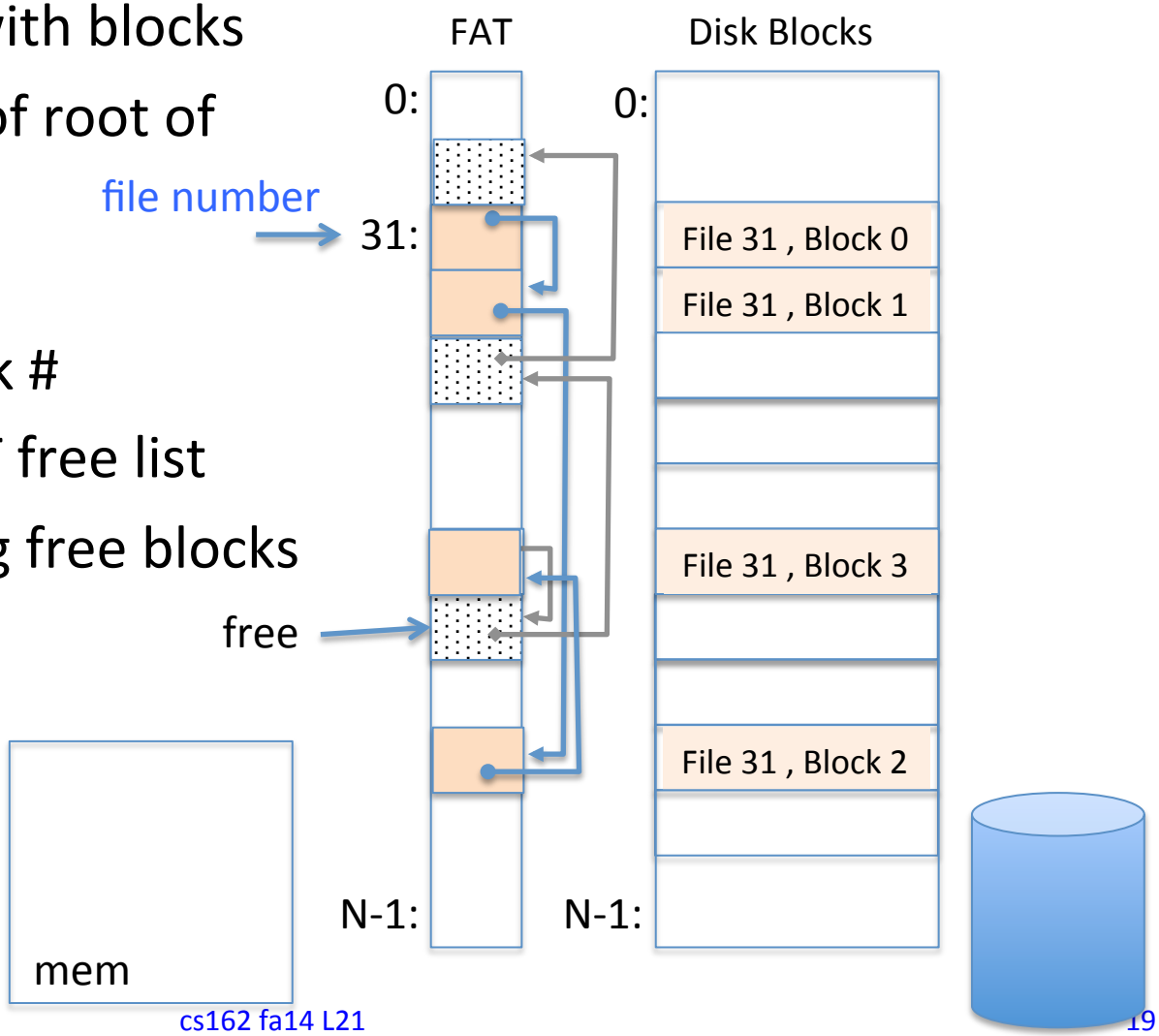
FAT (File Allocation Table)

- 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 ⇔ FAT free list
- Example: file_write(51, <3, y>)



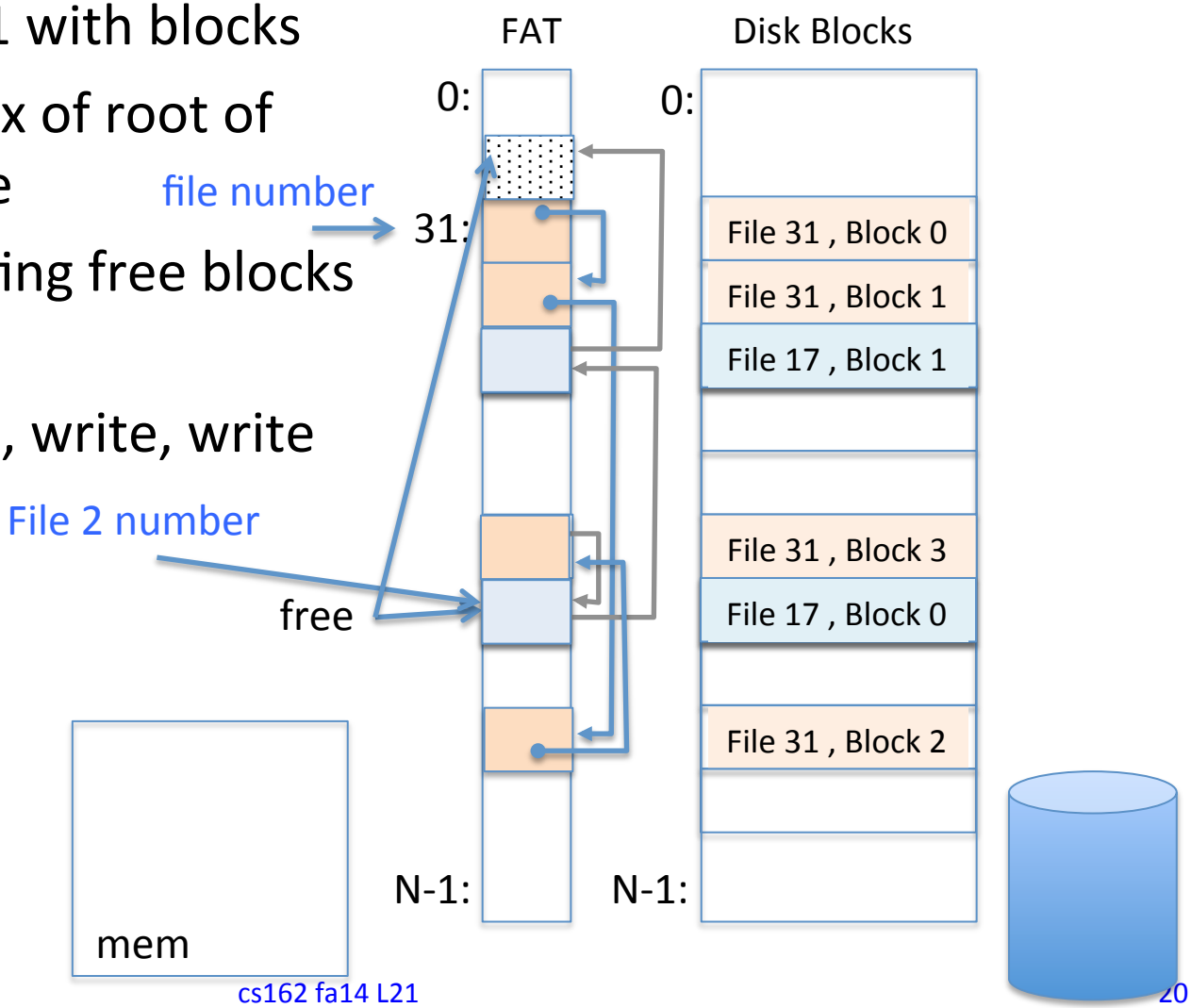
FAT (File Allocation Table)

- 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 ⇔ FAT free list
- Grow file by allocating free blocks and linking them in

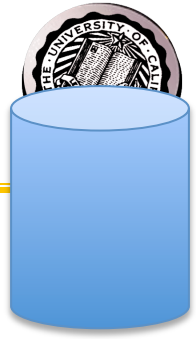


FAT (File Allocation Table)

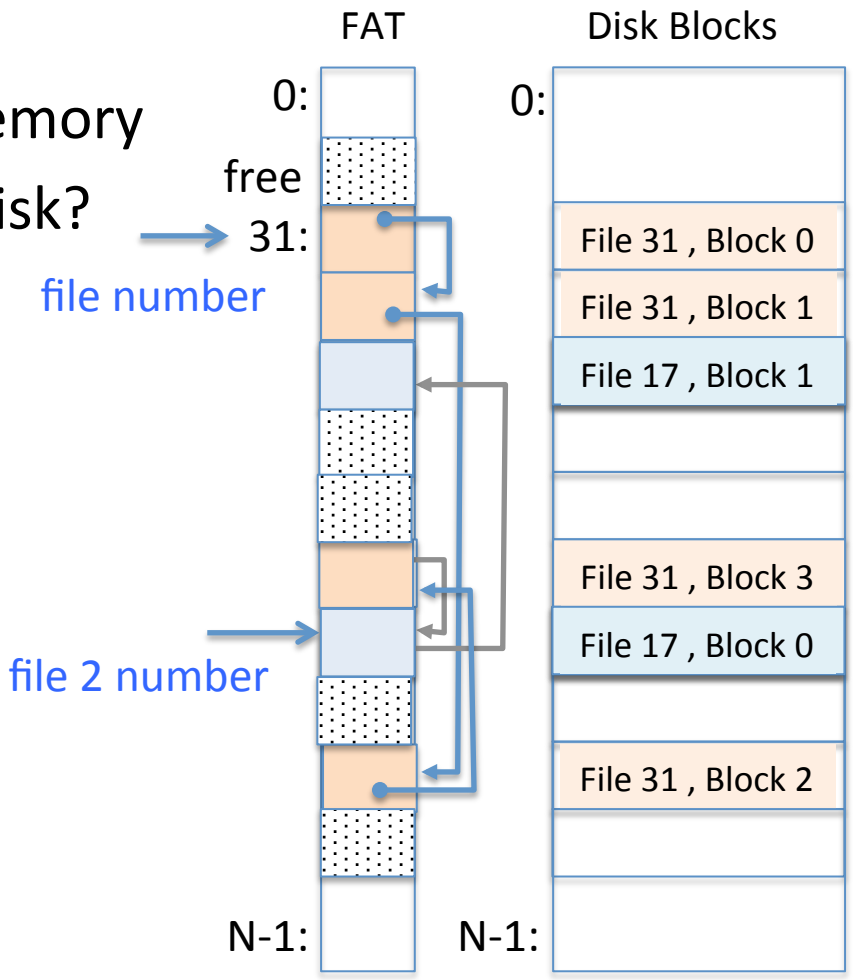
- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- Grow file by allocating free blocks and linking them in
- Example Create file, write, write



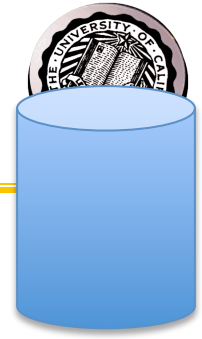
FAT Assessment



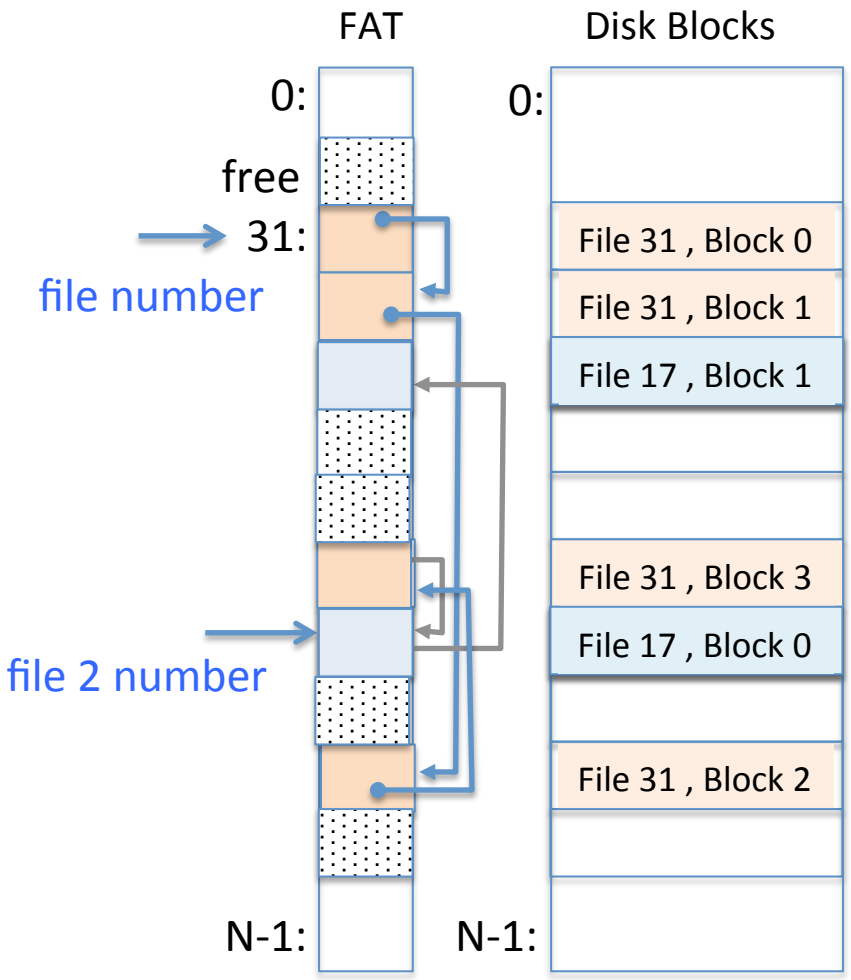
- Used in DOS, Windows, thumb drives, …
- Where is FAT stored ?
- On Disk, restore on boot, copy in memory
- What happens when you format a disk?
- Zero the blocks, link up the FAT free
- Simple



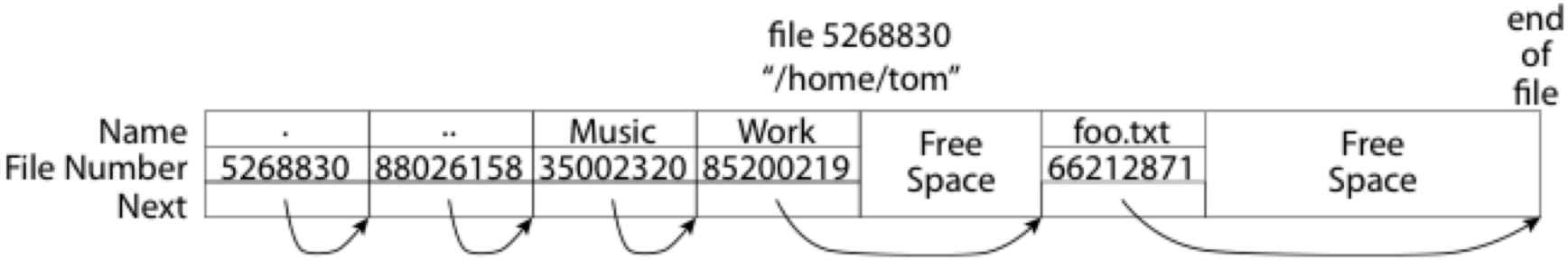
FAT Assessment



- 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 ???



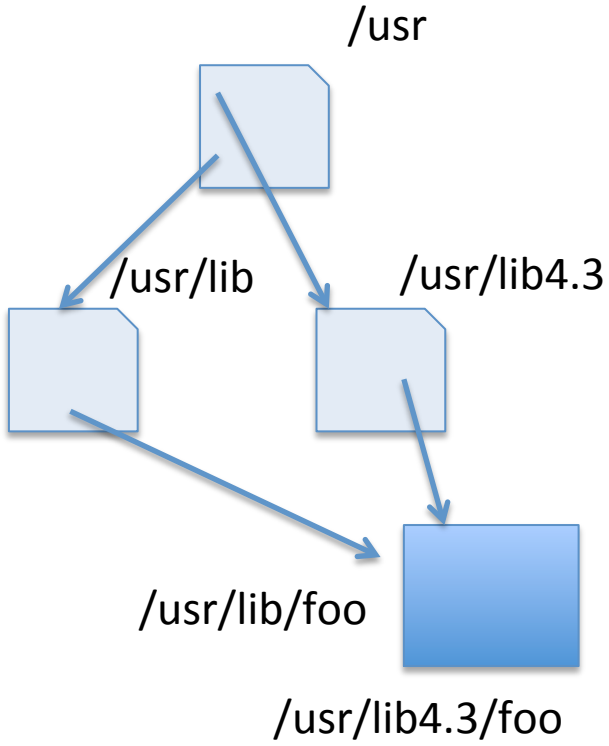
What about the Directory?



- 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 ("/")

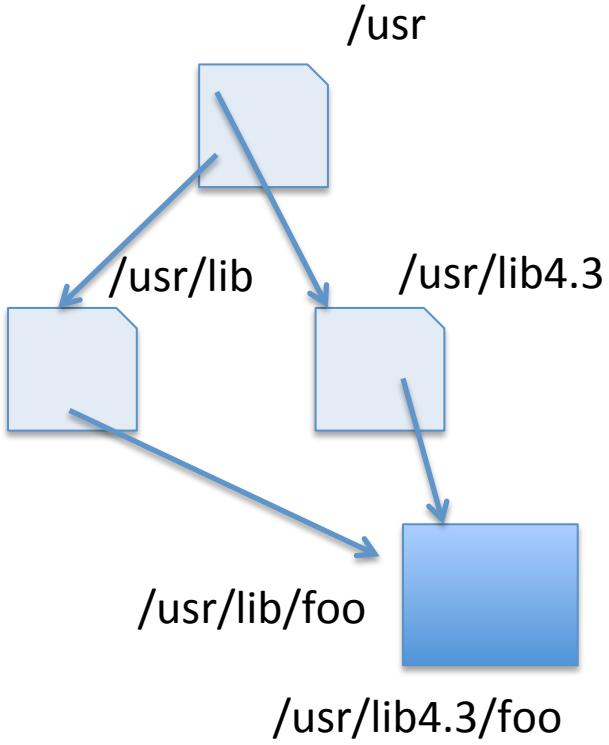
Bit more on directories

- 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 and 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

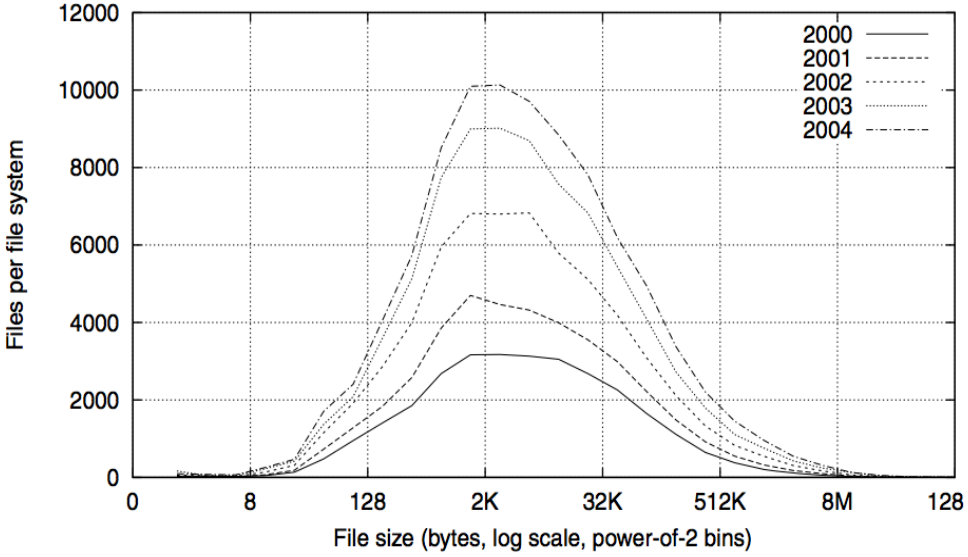


Fig. 2. Histograms of files by size.

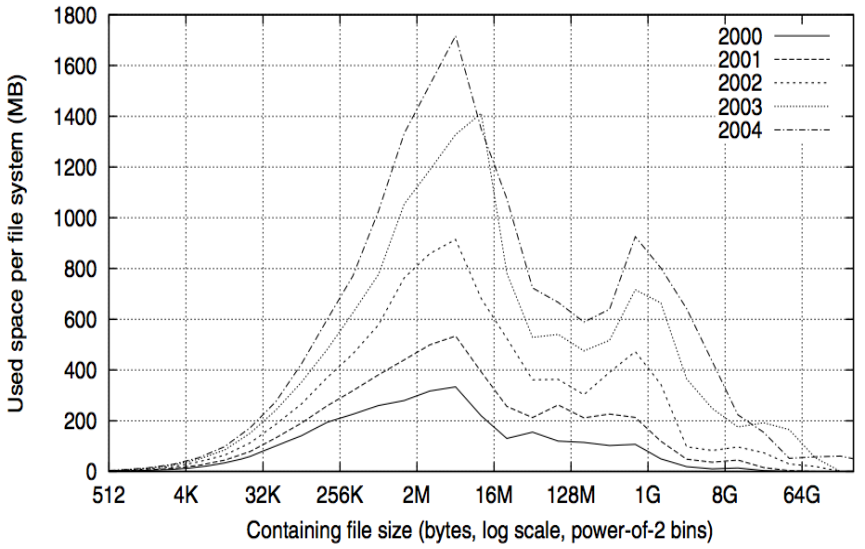


Fig. 4. Histograms of bytes by containing file size.

So what about a "real" file system

- Meet the inode