File Management

Problem: Want persistent data, but
- RAM is volatile.
- RAM is a scarce resource.

Possible Solutions:
- Non-volatile RAM (flash memory, carbon nanotubes)
- External storage (disk)

- Physical device
- Operating System File Abstractions
- Programming Language Support for Files
- File Formats
- Other abstractions
Driver abstraction

Disk blocks accessed by track/sector

(track #, sector #) → disk block/disk page

Driver Buffer Memory → O.S.
Driver abstraction
Disk blocks accessed by track/sector

Too low level of an abstraction!

- sharing — agree on which apps use which space
- protection
- users can’t be expected to remember where the data is

Higher level abstraction for external storage management

- Operating System provides a File System variation among O.S.
- Language can provide higher level abstraction on top of file system, providing a richer set of tools
Disk blocks

File System uses disk blocks as a primitive for building an on-disk data structure

- Latency is slower than RAM
- Fixed "object" size — one block
- Pack data in — lots of locality
- Minimize # of reads to find what you want

Each block in Unix File system

```
inode
```

```
data block
```
Way too slow!!
File System Abstraction

- User View (Files + Folders + String names)
  - Interfaces
    - User Interface
  - API
    - Name a file
    - Open a file
    - Read/write
    - Close
    - Name a directory
    - List its contents
    - Edit a file

O.S. Basic Features

At this point, discussed (on the chalkboard):

- Streams
- Orthogonality of source/destination, stream management, data format
- Example of wrapping streams with various filter types