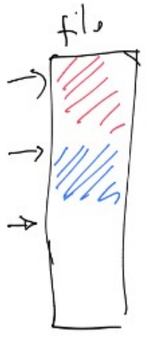
Lecture 17: FS Organization

- File access API
- Naming & directories
- VFS
- Storage
  - Contiguous, LL, FAT, Inodes

- Disk optimized for sequential r/w
- File API optimized for " "

- open (filename) → find file, prepare file for R/W, return file descriptor (fd) (number identifying opened file)

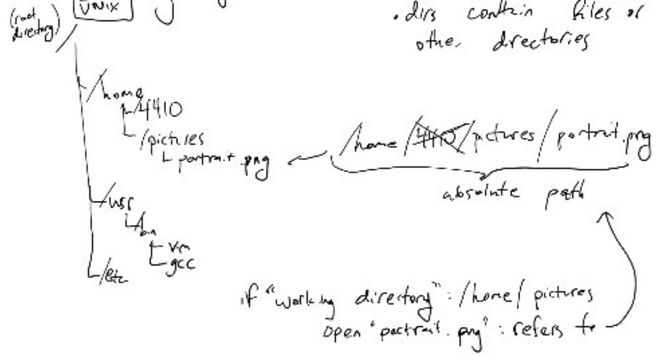
associated with each fd, there is a "current position", starting @ 0



- read (fd, [buf, length]):
  reads data from file into buffer starting from "current position", moves current position forward
- write [similar to read]:
  write to current position, advance curr. position
- seek:
  move "current position" pointer (non-sequential: inefficient)
- close

memory mapped files (mmap (fd, address range))
- file cached in (os) memory
- mapped into the process' address space, read/write from file with loads & stores.
- because working on a cached copy, need to explicitly sync data to disk.

sync(fd): block until data written back to disk.



Windows has multiple directory trees, one per device:

CDRom → A:\Hone

Thumb drive → B:\

Hard drive → C:\Windows
└ ...
C:\Desktop

UNIX: multiple device filesystems are all part of same logical filesystem

Virtual Filesystem (VFS)

/
- /media
  - /thumbdrive
    - <FS for thumbdrive>
- /home
  - /network <FS for communicating over network>
- /proc <PCB filesystem>

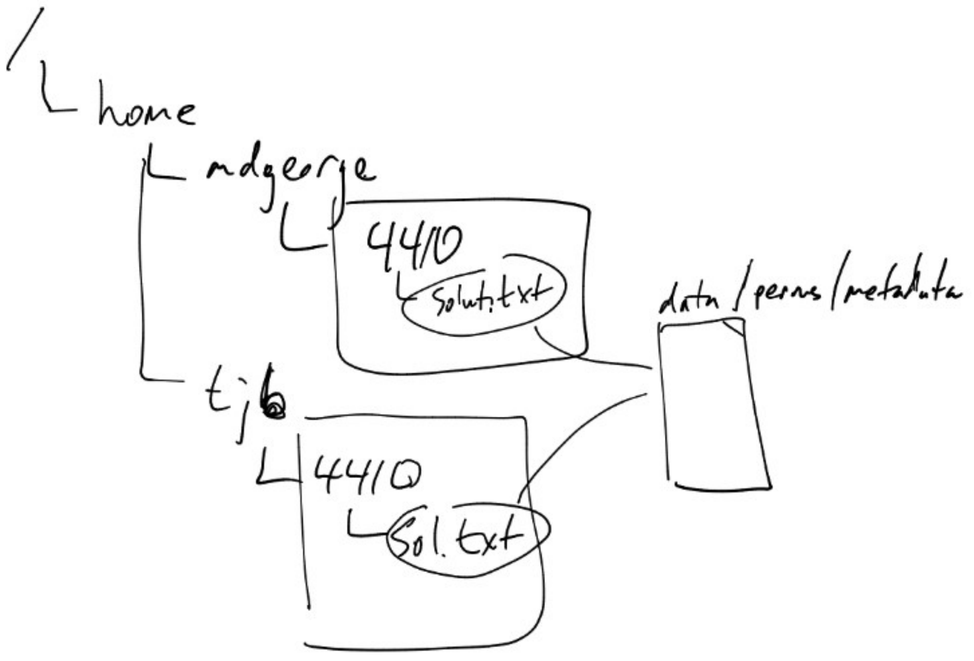
way open, read, write, ... operate depends on path to file

thumbdrive is "mounted" at /media/thumbdrive.

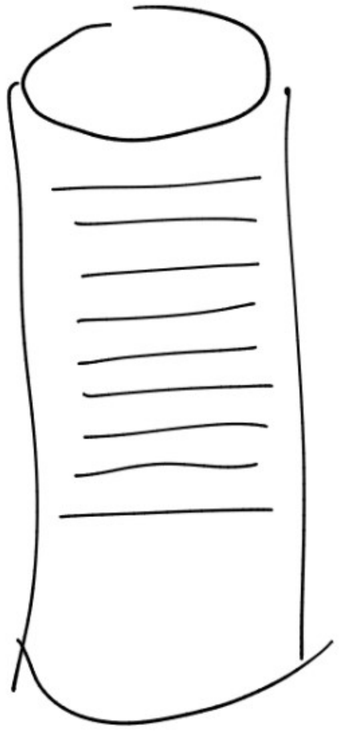
Some filesystems don't support directories at all.
- distributed filesystems. (e.g. Amazon S3)

Filesystem features:

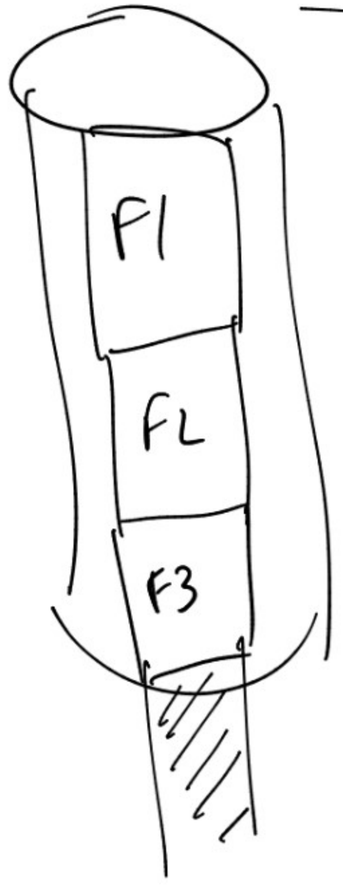
- Metadata stored with files:
  - size
  - ownership/permissions
  - type of file
    - Windows: part of filename (.exe: executable, .txt: text)
    - Unix: stored inside file (executable files "ELF header")
  - name of file
    - usually stored as part of directory
      - supports "hard links"



Storing files

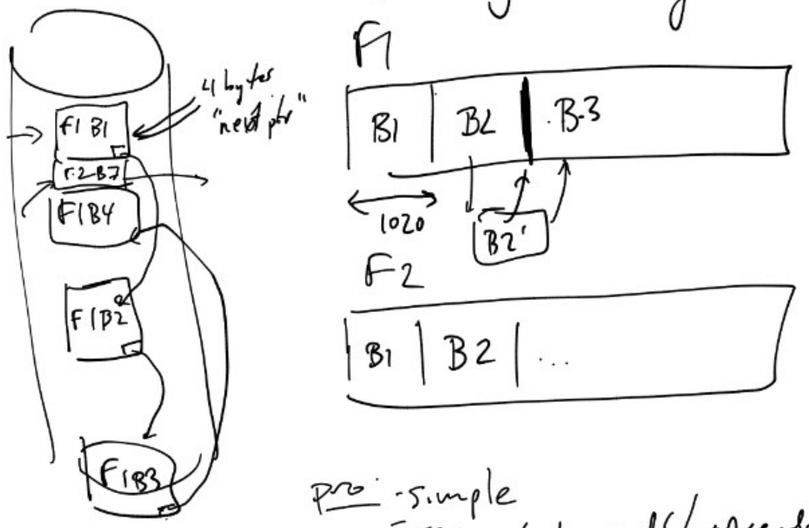


Contiguous alloc:



- internal/external frag.
- need to be able to expand files.

2nd strawman: Linked list of blocks per file
- sequentially read by following pointers.

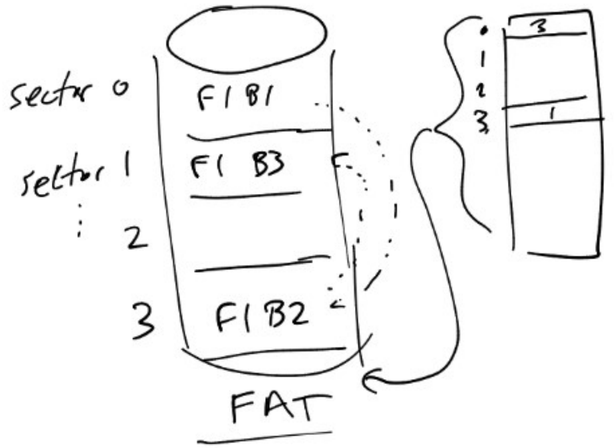


pro: - simple
- sequential reads/appends easy

con: - seeking is expensive
- hard to write into middle (except inserting blocks @ block boundary)
- storage is not divided into nice power-of-two sized chunks.

File allocation table (FAT)

- Like linked list allocation, but "next" pointers all stored on the side.



next pointers FAT (cached in memory, if small enough) } works for small disks, not great for large disks.

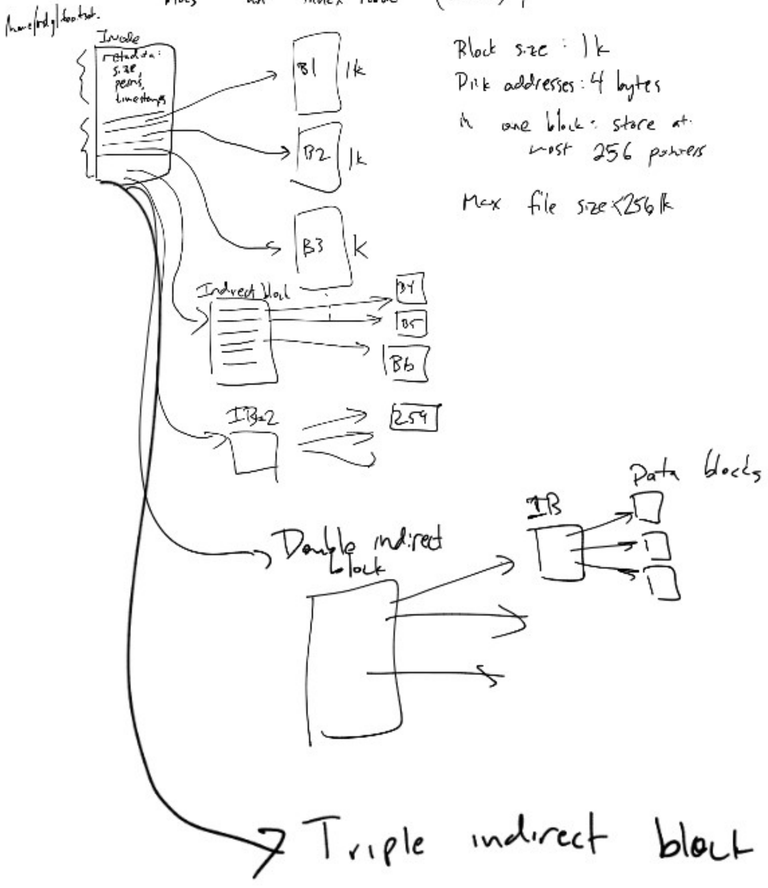
Seeking: O(n) memory accesses ← basically free, but small # of disk accesses

↗ additional metadata: which file starts where?

MS-DOS FS (designed by Microsoft).
→ used in Windows.
→ used everywhere.

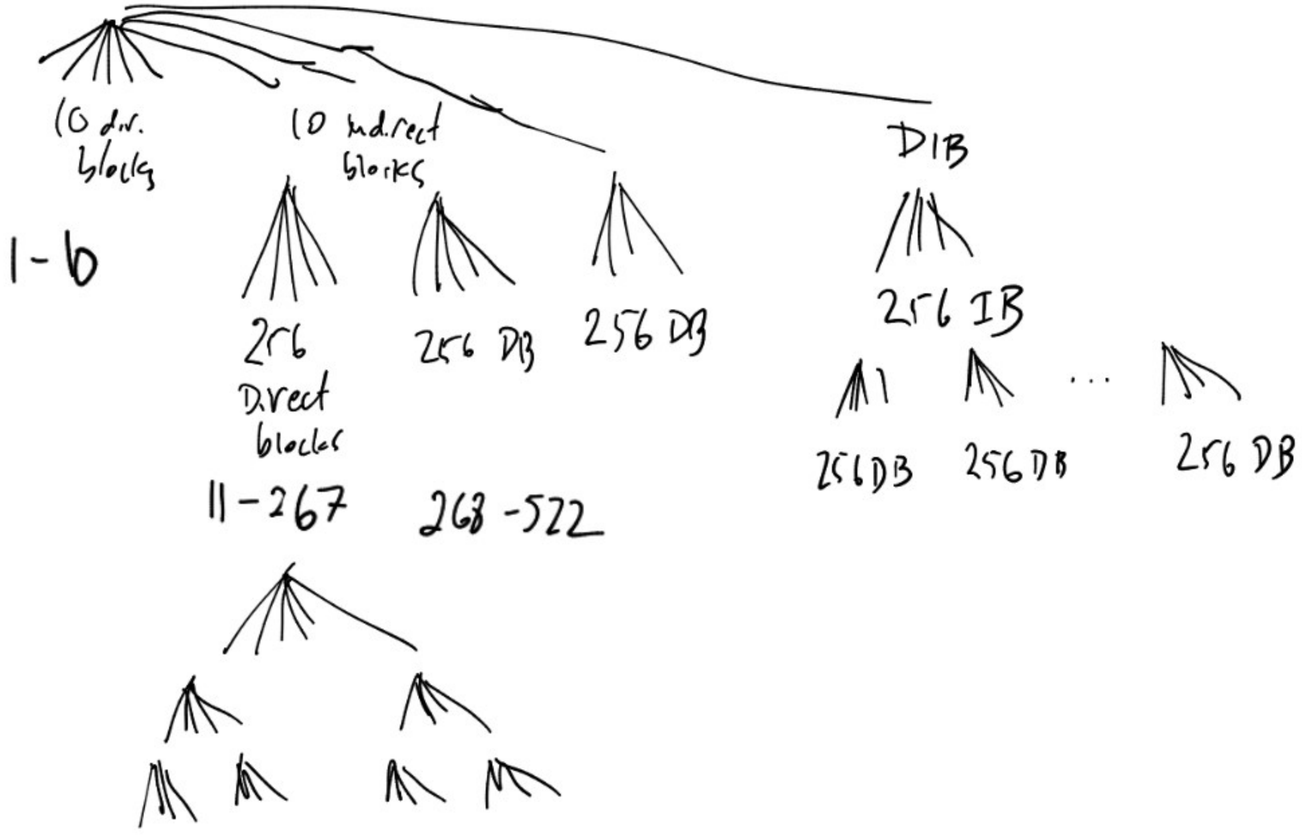
Unix filesystem (UFS)

- metadata stored with file ⇒ each file has an "index node" (inode) per file.



Block size: 1k
Disk addresses: 4 bytes
In one block: store at most 256 pointers

Max file size < 256 k



UFS directories

- Directories are files
- Contents of that file (not metadata) have mapping from filenames of contents to inodes of contents.