17

Files and Directories

Introduction to
Systems Software

wilkie

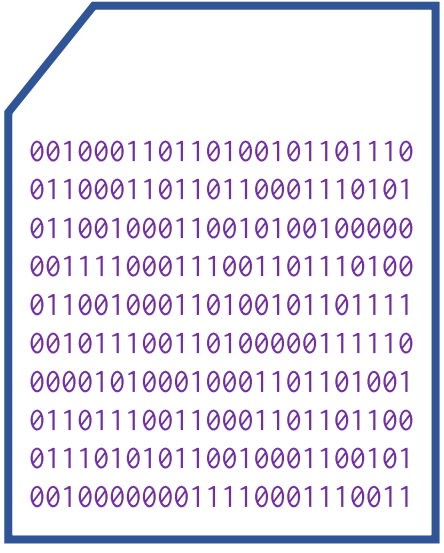
Spring 2019/2020

The Nature of Data

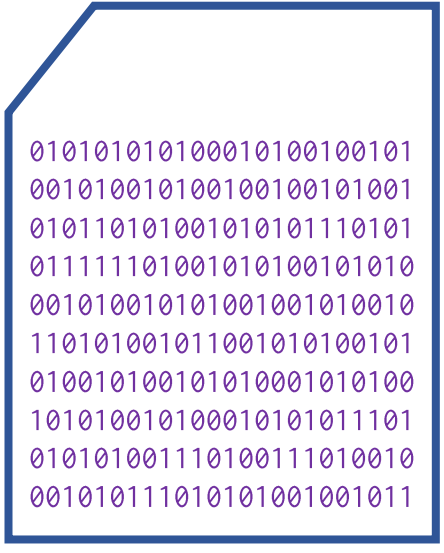
The Lord of the Files

Files and Data

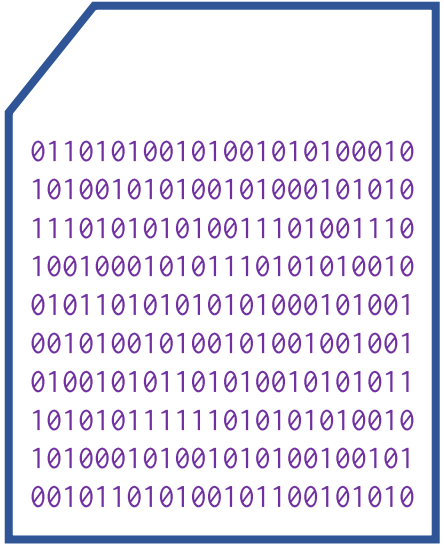
- What is a file? What is “data”? What is a binary file vs. a text file?



files.c



files.o



cat.gif

Formats

- **What is a file "format"? Why are there so many different image types??**
  - All files are flat binary blobs of information. How can we tell them apart?
- **Well, remember ELF? Our executable format?**
  - And the MAGIC NUMBER inside the box?
- **This is one way we differentiate different files.**
  - Archivists and librarians keep track of different file formats when they digitize, store, and retrieve data. They maintain the PRONOM database of formats.
- **Ok. How do we read files?**

C Programming: Manipulating Files with stdio

C (`gcc -o files files.c`)

```c
#include <stdio.h>   // for fopen, fread, etc
#include <string.h>

void main(void) {
  const char* path = "myfile.txt";
  const char* data = "hello!";
  char buffer[100] = {0};

  FILE* f = fopen(path, "w+");
  fwrite(data, strlen(data), 1, f);
  fclose(f);

  FILE* f2 = fopen(path, "r");
  fseek(f2, 0, SEEK_END);
  size_t filesize = ftell(f2);
  fseek(f2, 0, SEEK_SET);
  fread(buffer, filesize, 1, f2);
  buffer[filesize] = '\0';
  fclose(f2);

  printf("The file says: %s\n", buffer);
}
```

Open for writing

Open read-only

Seek 0 from end

Seek to byte 0

Politely close the file

- Here is a simple C program that creates a file called "`myfile.txt`" and writes a string to it, then opens it again to print it out.

- `fopen`: Opens a file with the given path. The string that follows is the access mode. "`w+`" opens for writing; overwrites file. "`r`" opens read-only.
- `fread`: Reads to the provided buffer the given number of bytes. (The "`1`" is the number of elements to read if reading an array.)
- `fwrite`: Writes from the provided buffer the given number of bytes. (Similar to `fread`)
- `fseek`: Moves the current file position.
- `ftell`: Returns the current file position.

C Programming: Manipulating Files with syscalls

C (`gcc -o files_unix files_unix.c`)

```c
#include <stdio.h>   // for printf
#include <string.h>  // for strlen
#include <fcntl.h>   // for open syscall
#include <unistd.h>  // for read/write syscalls

void main(void) {
  const char* path = "myfile.txt";
  const char* data = "hello!";
  char buffer[100] = {0};

  int fd = open(path, O_RDWR | O_CREAT | O_TRUNC);
  write(fd, data, strlen(data));
  close(fd);

  int fd2 = open(path, O_RDONLY);
  size_t filesize = lseek(fd2, 0, SEEK_END);
  lseek(fd2, 0, SEEK_SET);
  read(fd2, buffer, filesize);
  buffer[filesize] = '\0';
  close(fd2);

  printf("The file says: %s\n", buffer);
}
```

Open for writing

Open read-only

Seek to end

Seek to byte 0

Politely close the file

- Here is a simple C program that creates a file called "`myfile.txt`" and writes a string to it, then opens it again to print it out.

- `open`: Opens a file with the given path. The value that follows is the access mode.
  - `O_RDWR` opens for writing.
  - `O_CREAT` creates the file, if needed.
  - `O_TRUNC` removes the data in the file.
  - `O_RDONLY` opens read-only.
- `read`: Reads to the provided buffer the given number of bytes.
- `write`: Writes from the provided buffer the given number of bytes. (Similar to `read`)
- `lseek`: Moves the current file position. Returns the new position.

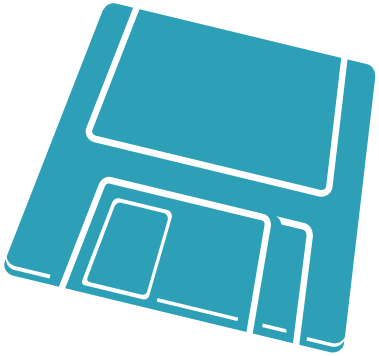
Everything is a “file”

- UNIX makes lots of things “files” in a non-traditional sense.
- Sockets, named pipes, all kinds of exotic things. Directories are files in the traditional sense (stored on disk properly)
- You can use the `read`, `write`, and `close` system calls with any of these diverse set of data streams.

The Internet

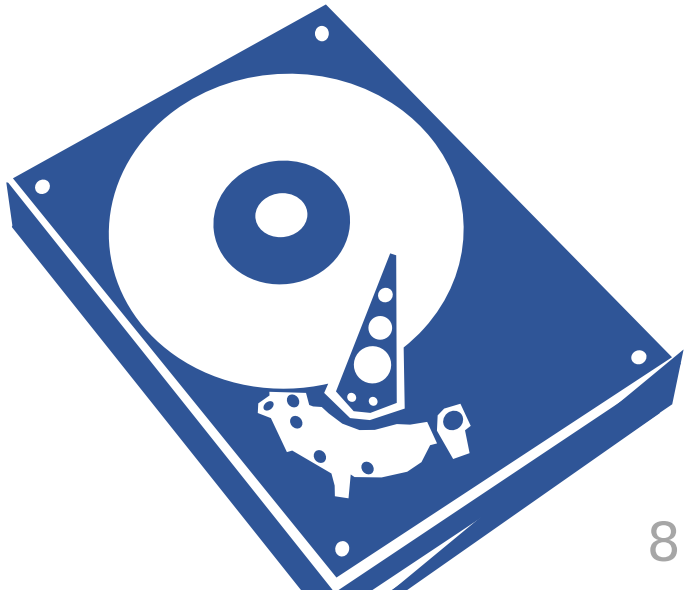
How data is stored…

- Data in RAM is generally **volatile memory**.
  - It disappears after you shut off your computer.



- So, you want some kind of **persistent memory**.
  - Storing data on disk involves creating a physical representation of that binary data.

- Fun fact: there are non-volatile (persistent) main memories in development. (NVRAM)
  - They are really neat! (and slow!)
  - But wow they really complicate things!!
    - Consider the implications.



- Let's dig in…

Disks

You throw them and dogs chase them. Wait.. no… don’t do that.

Floppy Disks

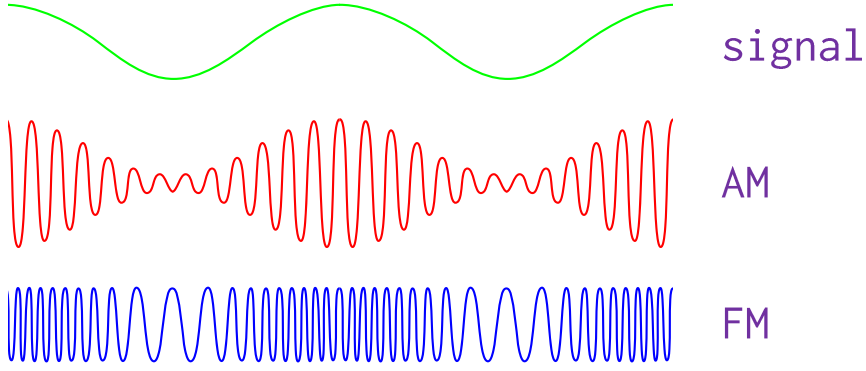
- Data is stored in analog on magnetic material.
  - I love them.
    - Buy me a random box for my birthday, please.
      - When is my birthday? It is everyday.

- Termed “floppy” due to the soft, flexible nature of the magnetic material.
- Wait. Magnets?

- How do you store data… with magnets?
  - If you’re thinking “they have two poles… so they are binary natured,” then you are on to something.

Representing continuous data…

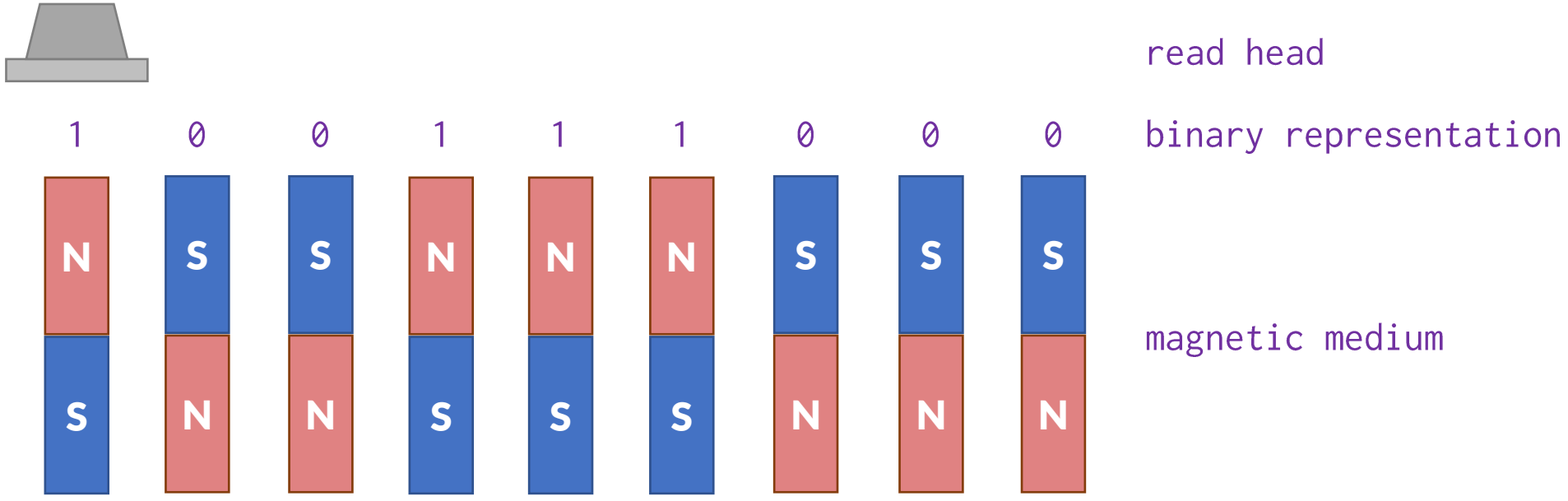
- If you have some (continuous) data, represented by a waveform…
- How to transmit/store that wave?
- Amplitude modulation…
  - Send pulses of data sampling the wave.
  - Data encoded in the amplitude of pulse.
- Frequency modulation…
  - Data encoded in variation of frequency of pulse. (Yes, like FM radio)
  - Disks actually store data using a form of this encoding!



Different waveform streams

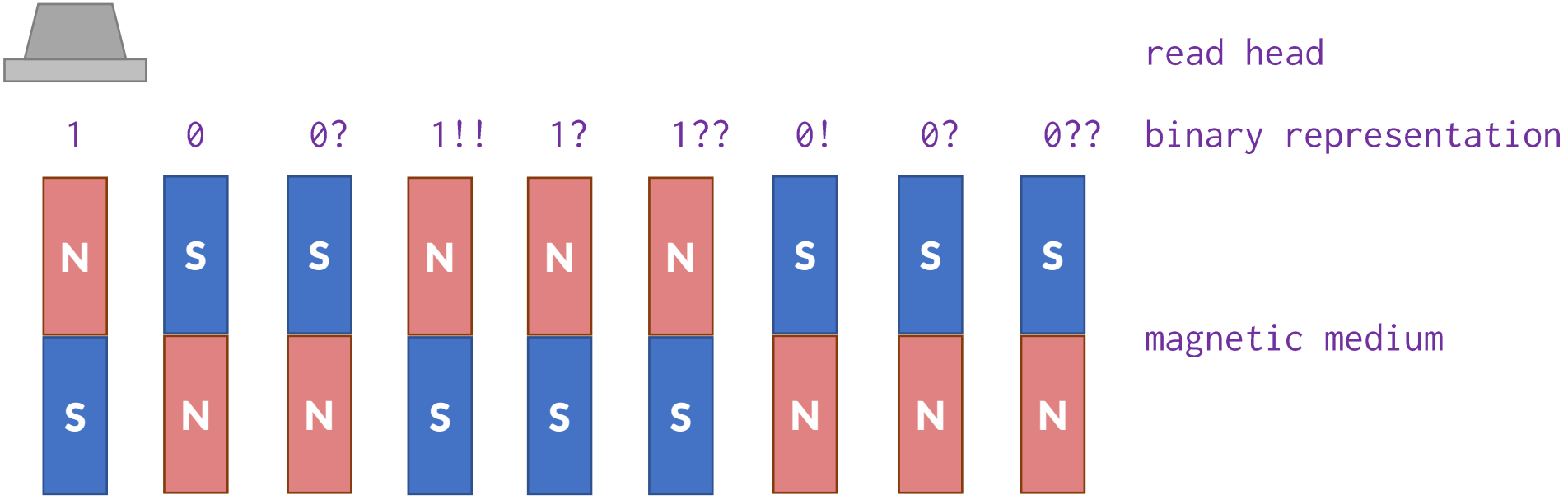
Data and magnets… how do they work

- We can use magnets to represent `0` and `1` (discrete binary)
  - The drive's read head contains a sensor that detects the "magnetic flux"
  - It can sense a change in magnetism over time.
    - This shows the ideal world, without any modulation:



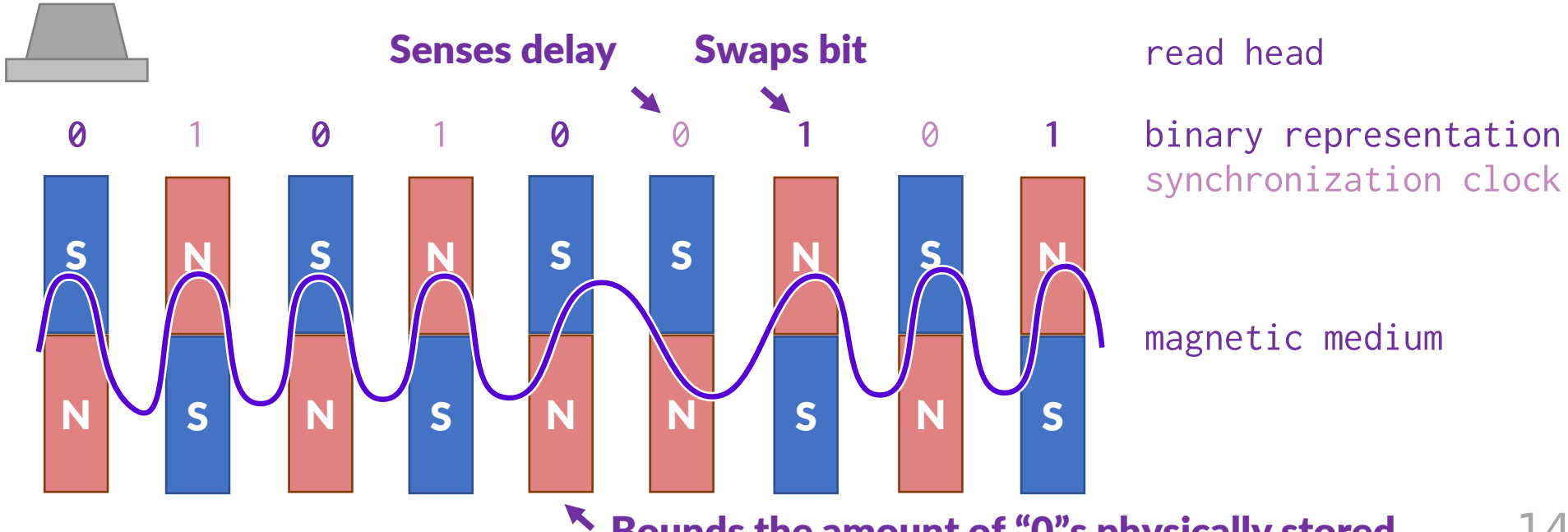
The peril of nature…

- The magnetic drives read the change in magnetism.
  - It is difficult to tell the difference between two consecutive magnets…
  - This is also because co-aligned magnets HATE being next to each other.
    - Opposites attract, n'at. (They repel and affect each other's signal)



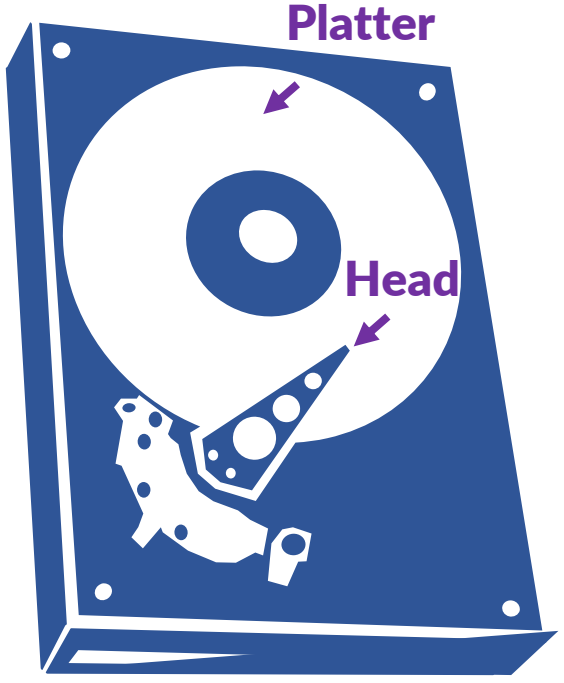
The peril of nature…

- So, we can give up almost half of our data to add synchronization.
  - When we read, every other "sense" affects the next read.
  - If we read a "0" and then sense a change, the next bit stays the same.
    - If we sense a delay (long frequency), it is a "1" and we continue. (**Modified-FM Encoding**)



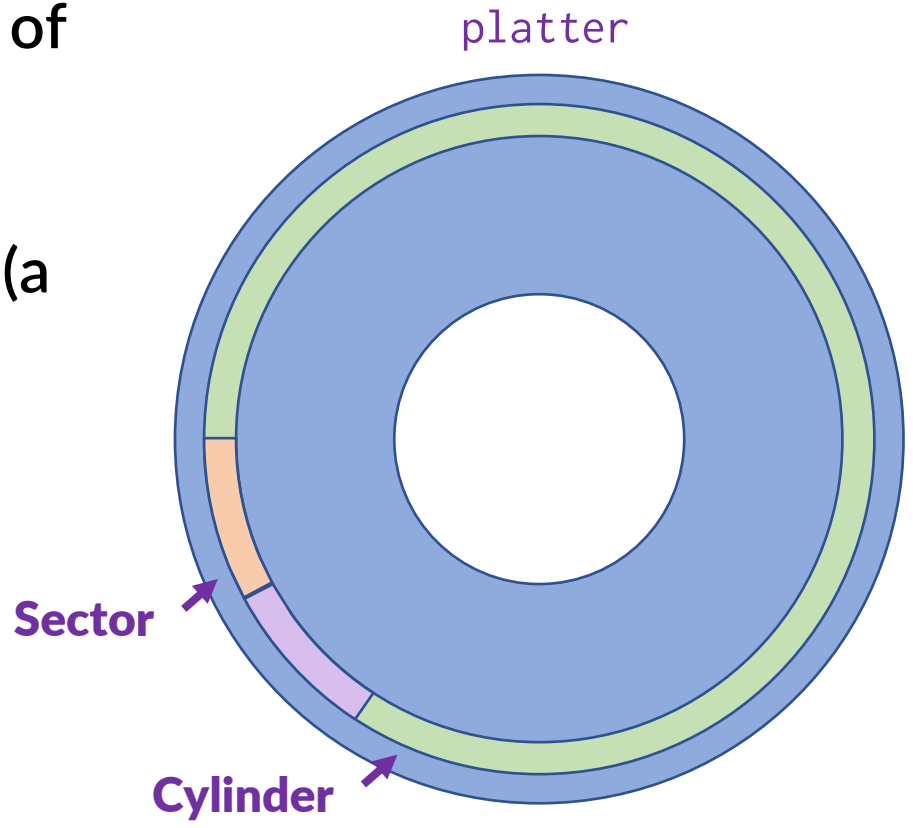
Disk Drives

- Also known as a “hard disk” due to the inflexible nature of its magnetic material.
- Data is also stored digitally using a physical medium, such as, again, magnets.
  - Uses a similar yet stronger encoding scheme.
- Mechanical parts.
  - Can read random access, but it is slower than reading data sequentially (in physical order).
- Bits are hard… let’s start abstracting…



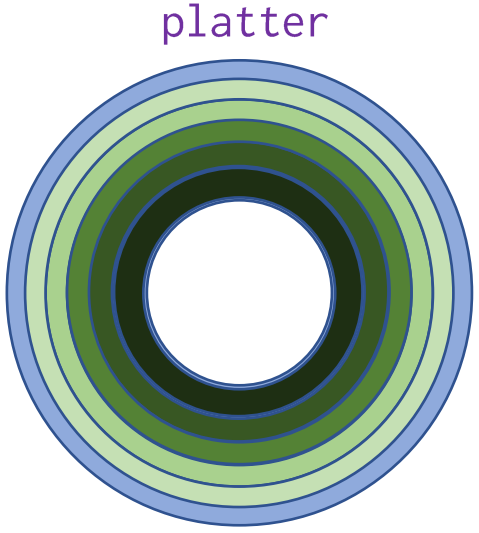
The platter matters:

- Magnetic disk is represented by a set of stacked platters with magnetic bits.
- A **cylinder** is a subdivision of platters (a **track** is such a subdivision on a single platter.)
- A **sector** is a subdivision of a cylinder/track.
  - You typically read information from a disk in units of sectors.
  - Files are, generally, a set of sectors.

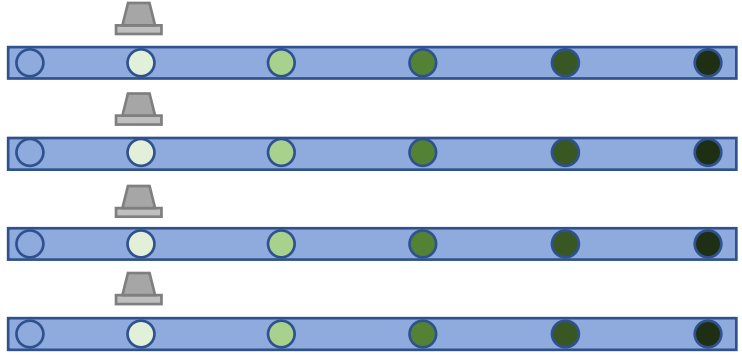


Making heads turn (actually, they don't turn at all)

- Magnetic disk is represented by a set of stacked platters with magnetic bits.
  - There may be several platters.
  - Each read by at least one **head**.
  - **Access time** is how long it takes to read a sector.



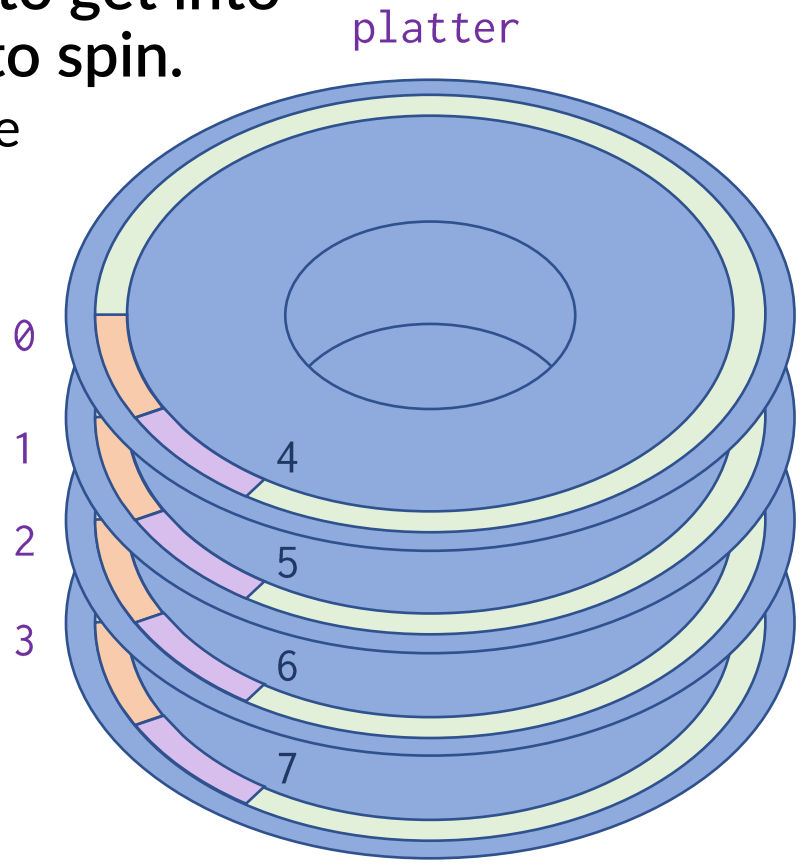
- As a head moves, it goes to a different cylinder.
  - As the platter spins, the head reads a different sector.



- You can potentially read multiple sectors in parallel.
  - So how should we layout data on disk to take advantage of this?

Making best use of sequential access

- **Seek time** is the time it takes for the head to get into position. **Latency**: the time for the platter to spin.
  - Data is located at a two-dimensional coordinate on a spinning surface.
    - so the math is not trivial.

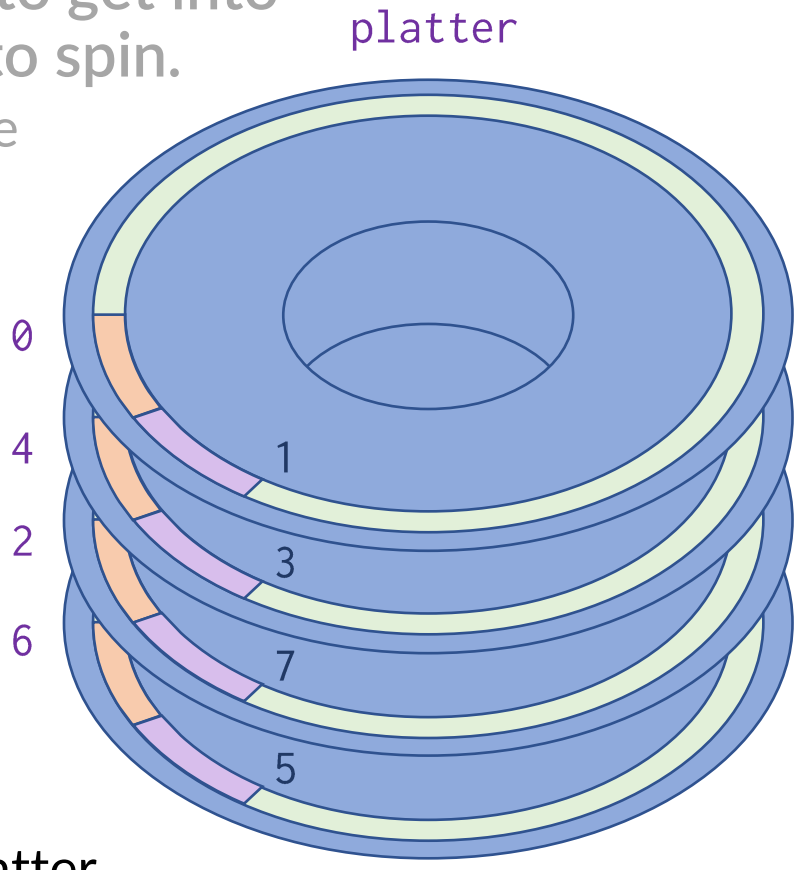


- Seek time is relative to the current position of the head.
  - The closer the next bit of data you need…
    - The sooner it will get there.

- So… to reduce the seek time to nil…
  - We position adjacent data in the same cylinder and respective sector.
  - Next set goes into subsequent sector. Heads don't move; the platters spin.

Ain't no platter like a hard disk platter 'cause a hard disk platter don't stop

- **Seek time** is the time it takes for the head to get into position. **Latency**: the time for the platter to spin.
  - Data is located at a two-dimensional coordinate on a spinning surface.
    - so the math is not trivial.

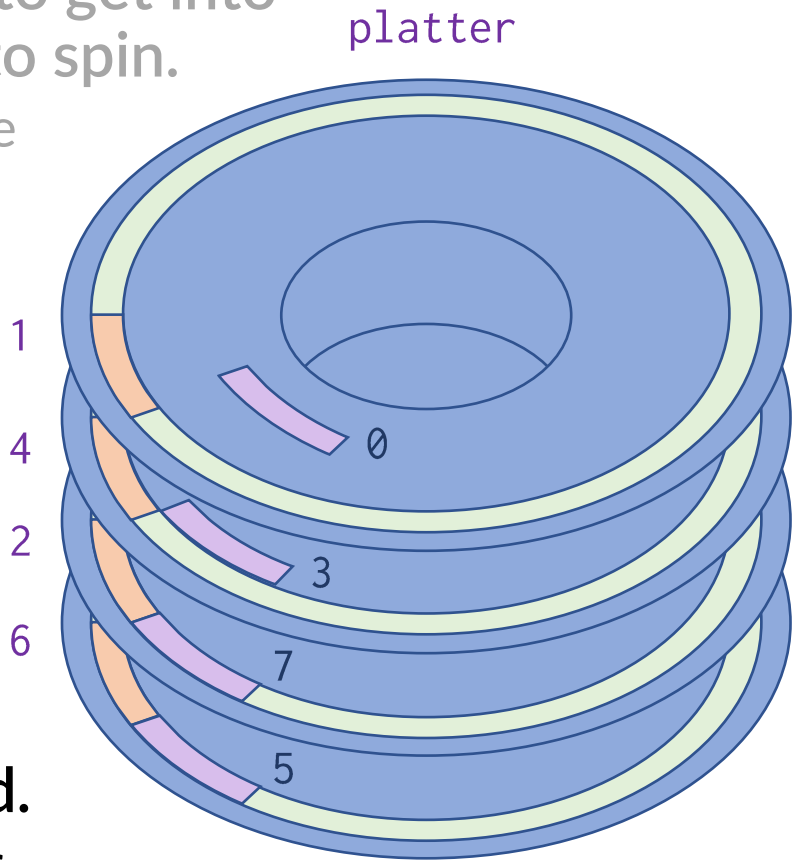


- Seek time is relative to the current position of the head.
  - The closer the next bit of data you need…
    - The sooner it will get there.

- **Here, the head does not have to move at all and blocks 0 and 1 are read easily.**
  - Yet, to read block 2, we have to wait for the platter to completely spin back around!! Seek time is zero, but maximum latency loss.

Ain't no platter like a hard disk platter 'cause a hard disk platter don't stop

- **Seek time** is the time it takes for the head to get into position. **Latency**: the time for the platter to spin.
  - Data is located at a two-dimensional coordinate on a spinning surface.
    - so the math is not trivial.
- Seek time is relative to the current position of the head.
  - The closer the next bit of data you need…
    - The sooner it will get there.
- **Yikes! Blocks are in different cylinders and subsequent blocks are behind the head.**
  - Worst case! Latency and seek time really suffer.
  - Need to keep data in order! How do we organize data on disk?



File Systems

Yet another abstraction… moving toward applications.

File Systems

- There are many ways of representing files on the disks themselves.

- As you know, you are familiar with:
  - Files having names!
  - Directories/folders for organization
  - Perhaps special files such as symbolic-links/shortcuts

- A **file system** entails describing how we represent:
  - File data (of course)
  - The location of the file (a file **path**)
  - Meta data about the file (what kind of file?)
  - Access control (who can access the file)

File Metadata

- There is a long list of possible metadata associated with files:
  - The file size.
  - The file name.
  - When it was last accessed.
  - Who created it and when.

- And access control:
  - Who can read it.
  - Who can write it.
  - Who can run it.

Linux/UNIX `stat()` metadata:

```
/* Metadata returned by the stat and fstat functions */
struct stat {
    dev_t         st_dev;      /* Device */
    ino_t         st_ino;      /* inode */
    mode_t        st_mode;     /* Protection and file type */
    nlink_t       st_nlink;    /* Number of hard links */
    uid_t         st_uid;      /* User ID of owner */
    gid_t         st_gid;      /* Group ID of owner */
    dev_t         st_rdev;     /* Device type (if inode device) */
    off_t         st_size;     /* Total size, in bytes */
    unsigned long st_blksize;  /* Blocksize for filesystem I/O */
    unsigned long st_blocks;   /* Number of blocks allocated */
    time_t        st_atime;    /* Time of last access */
    time_t        st_mtime;    /* Time of last modification */
    time_t        st_ctime;    /* Time of last change */
};
```

Operating Systems and Files

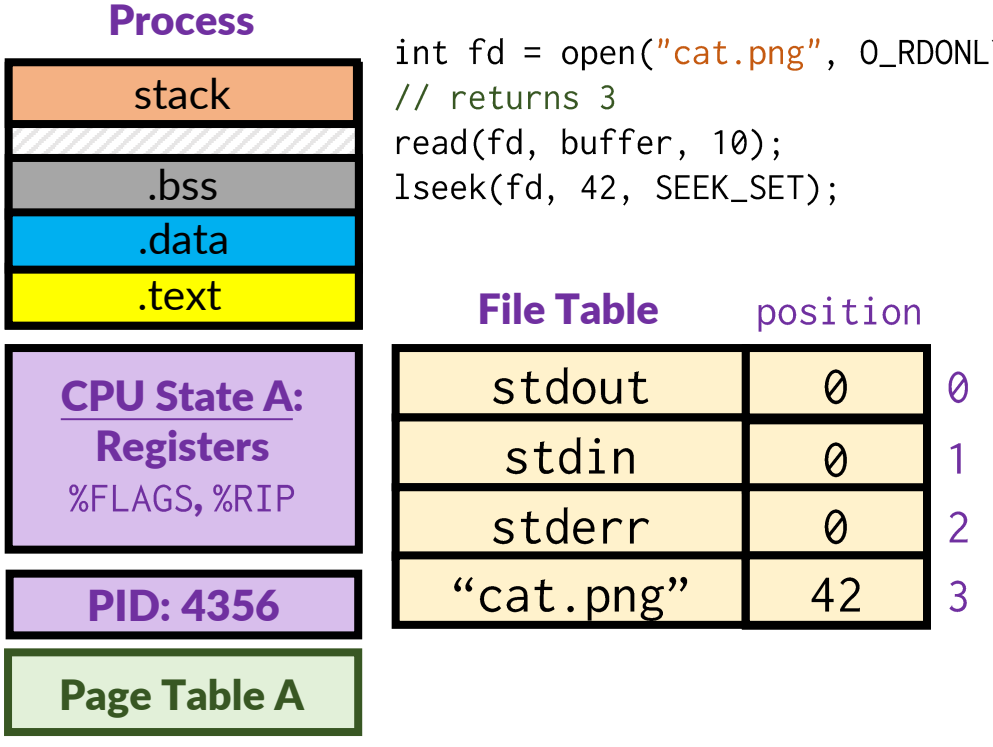
- The `open` function returns a **file descriptor**, an integer that identifies the open file in the process.
  - Every process can have open files, but none are shared across processes.

- On Linux/UNIX, some file descriptors are established automatically for every process by the shell:
  - `stdout` – the output file (can be a file on disk! Recall terminal redirection.)
  - `stderr` – a file for error output.
  - `stdin` – the input file (could be a file on disk… or user input in the terminal.)

- The OS maintains a table of open files per process. When it sees a syscall such as `read` or `write`, it uses that table to determine the file.

Processes and Files

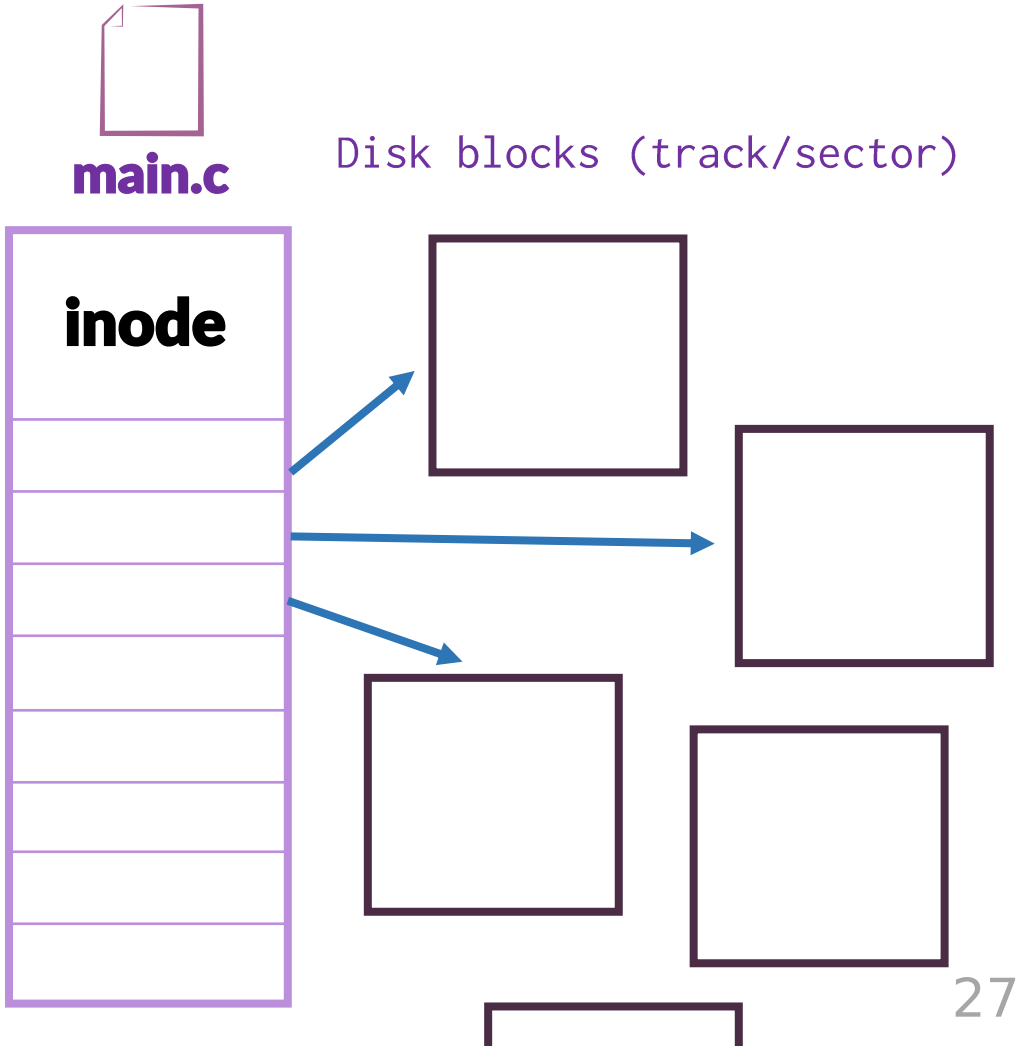
- The OS maintains a table of open files per process. When it sees a syscall such as `read` or `write`, it uses that table to determine the file.



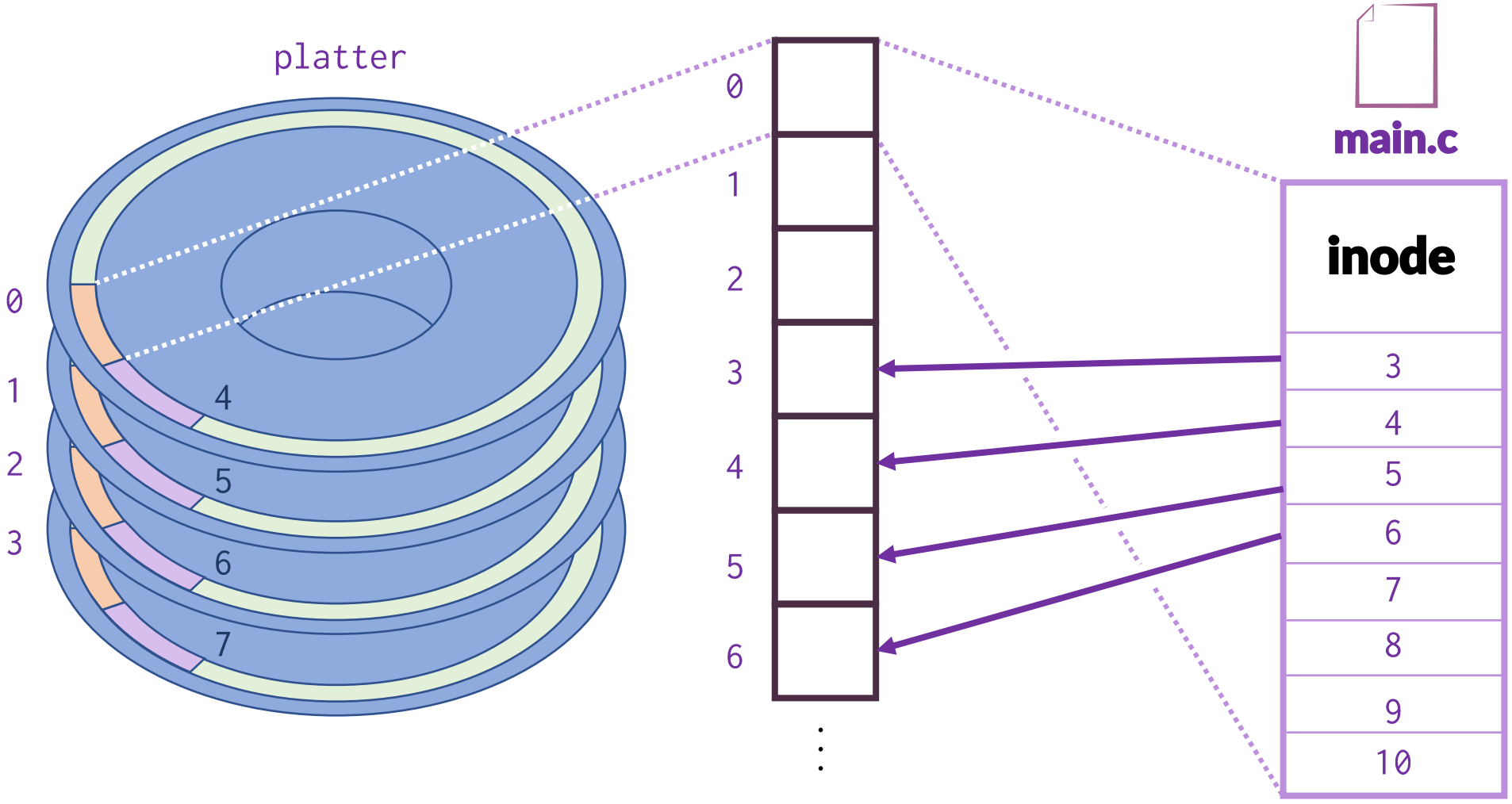
- The table contains a set of open files indexed by the file descriptor.
- Several files are generally opened for you by the shell.
- Each open file maintains its own current position.
  - `fseek`/`ftell` manipulate it.

I nodes, you nodes, we all nodes for inodes

- Files are a set of disk blocks.
  - Hopefully laid out in a nice order!
- How do we organize these?
  - Similar to virtual memory!
- We use a disk block that holds addresses to other blocks.
  - It is a simple table. The blocks that make up the file are in the order reflected by the table.
- An **index node** is this main block.
  - Often seen shortened to “inode”

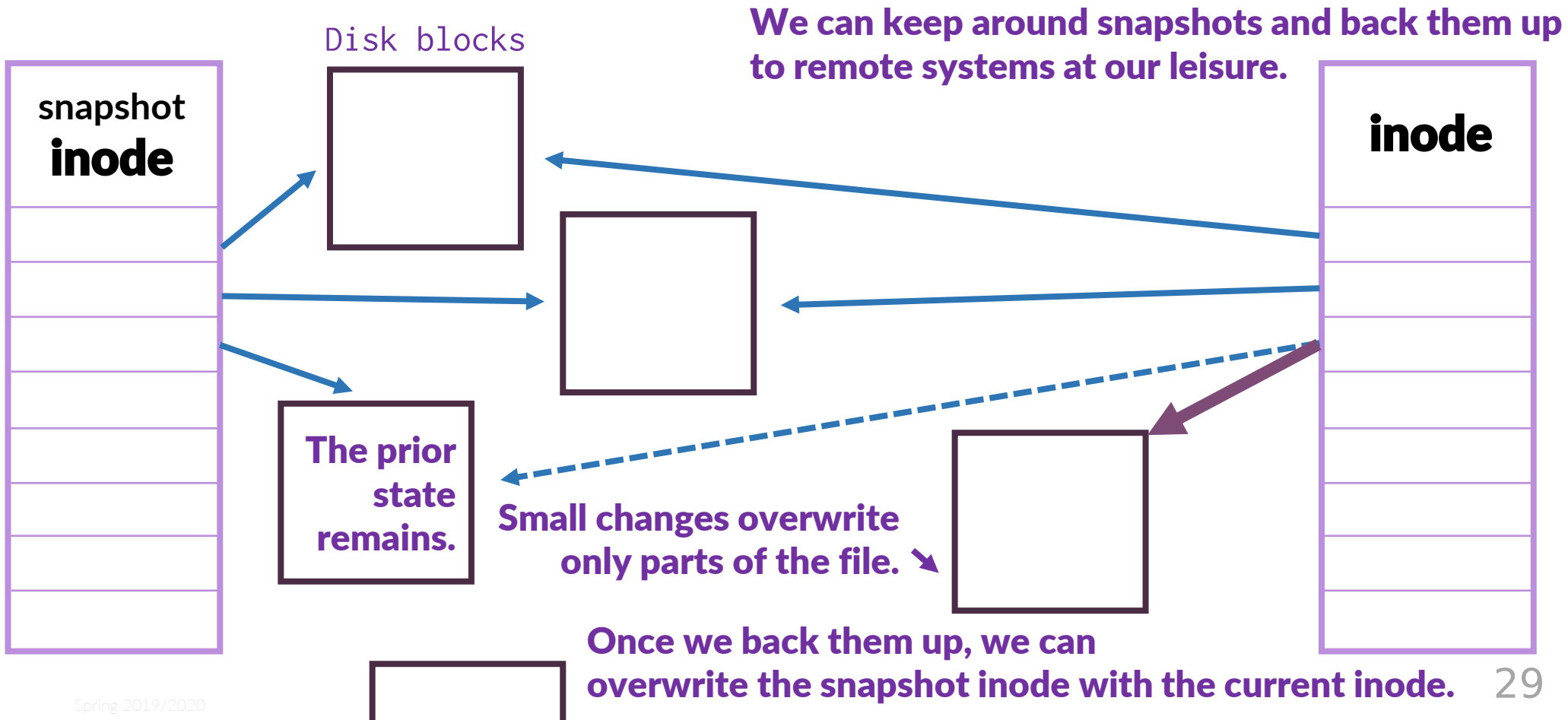


File systems are about organizing the disk



Cheap Versioning: WAFL

Here is WAFL performing "snapshot" backups of files:



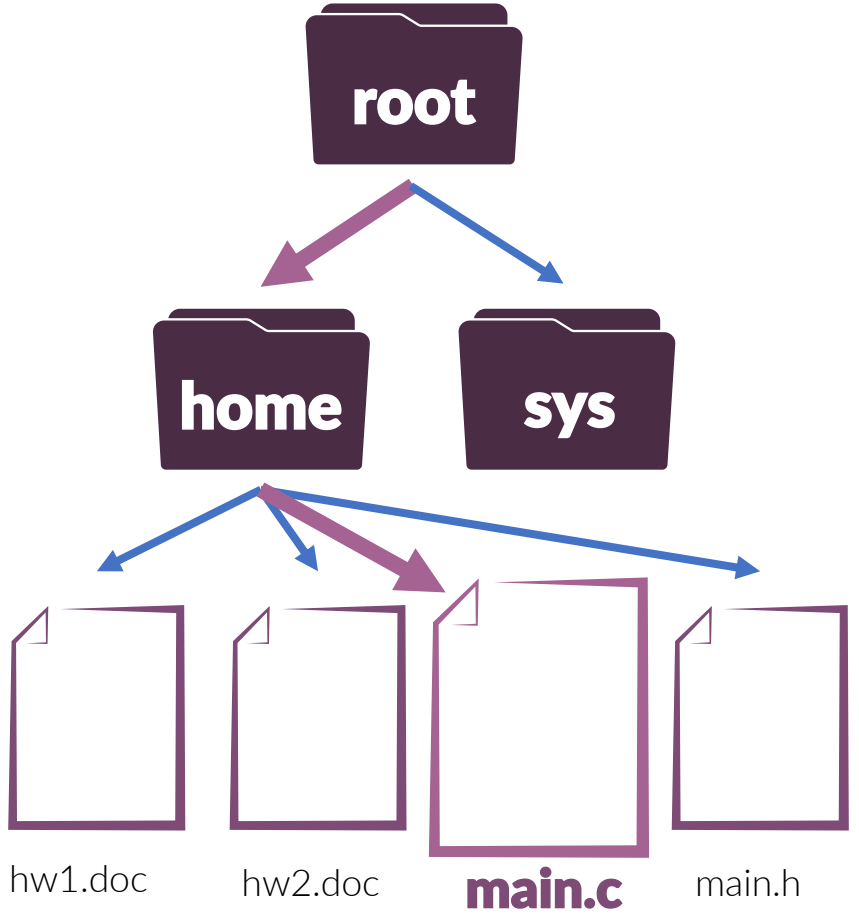
Hierarchies

- **Directories maintain strict hierarchical structure for files in the system.**
  - For instance, your home folder is often something like `/home/wilkie`.

- **An absolute path is a fully-qualified name for a file that indicates exactly where in the hierarchy it is located.**
  - Often organized by a human being in some logical way:
    - `/music/rock/queen/news-of-the-world/we-will-rock-you.mp3`
  - There are many special paths. OS data structures go in:
    - `/proc`
  - Devices go in:
    - `/dev`
  - Use the Linux `which` command to find out where your system binaries go!

Directories (Folders)

- A **directory** is a file that contains a set of named links to other files.
  - The earliest file systems did not even have directories… just a bunch of files.
- Groups a set of files together under a single name.
  - Strictly hierarchical…
  - Generally, a file can only be within one directory.
  - Although, a directory can also be within a directory… creating a cascade.

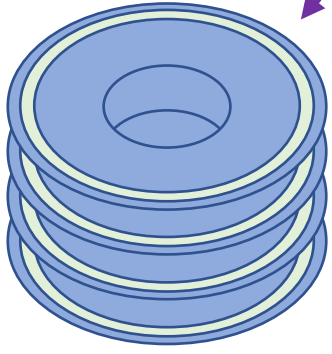


Implementing Directories

- **Directories can simply be text files, if you want!**
  - Every line contains a name and then a block address on disk for the inode.
- **Obviously, there are a variety of ways to do this.**
  - Do you keep a sorted order to make searching directories easier?
  - Can a directory refer to a file that is part of a different disk?
    - A file from a completely different machine??
- **If a directory is the only thing linking to a file, and removes that link, what happens?**
  - How do you access a file if you cannot look up where it is?
  - Deleting a file is really as simple as removing it from the directory.
  - (And marking its blocks on disk free)

A directory

Name	Block Address	Type
..	a4b3	dir
malloclab-handout	b224	dir
cachelab-handout	1266	dir
main.c	566d	file
main.h	453b	file
Makefile	22ab	file
main.o	65df	file

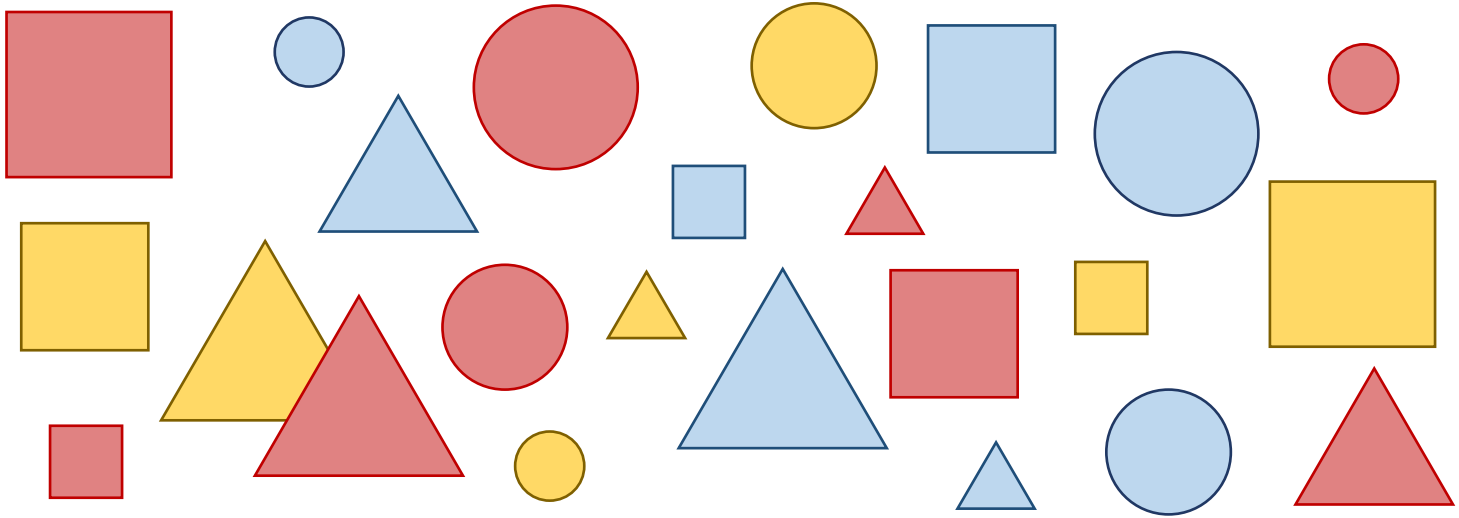


- A directory is a simple table, implemented as a file, that maps names to inodes.
  - Each file system (NTFS, FAT, HFS, EXT) will implement it slightly differently.
- It may also contain metadata about the file for each entry.
  - Creation date, author, file size.
- How does a directory know its parent?
  - Special entry! “. .” points to parent.
  - “cd . .” is quite literal.

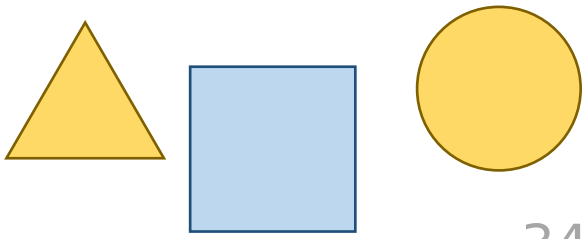
Heretic of the Day: Alternatives to Hierarchies

- Margo Seltzer: Hierarchical File-Systems are Dead (HotOS '09)
  - In this paper, she re-orients file systems around human beings and our own needs.

- She asks the simple task: "Group these."

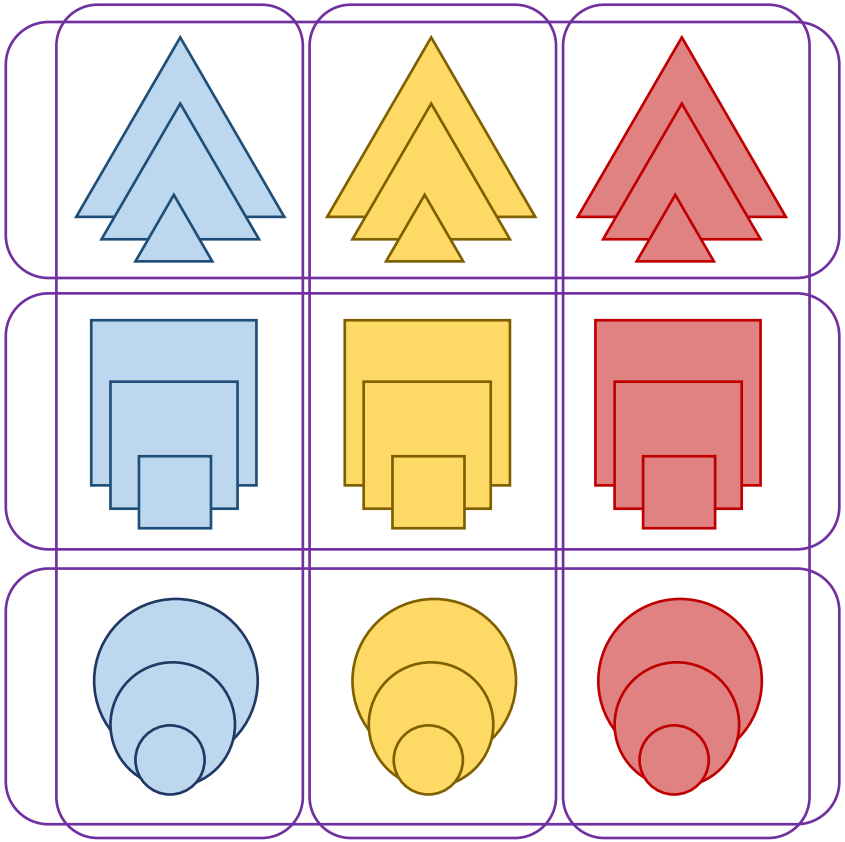


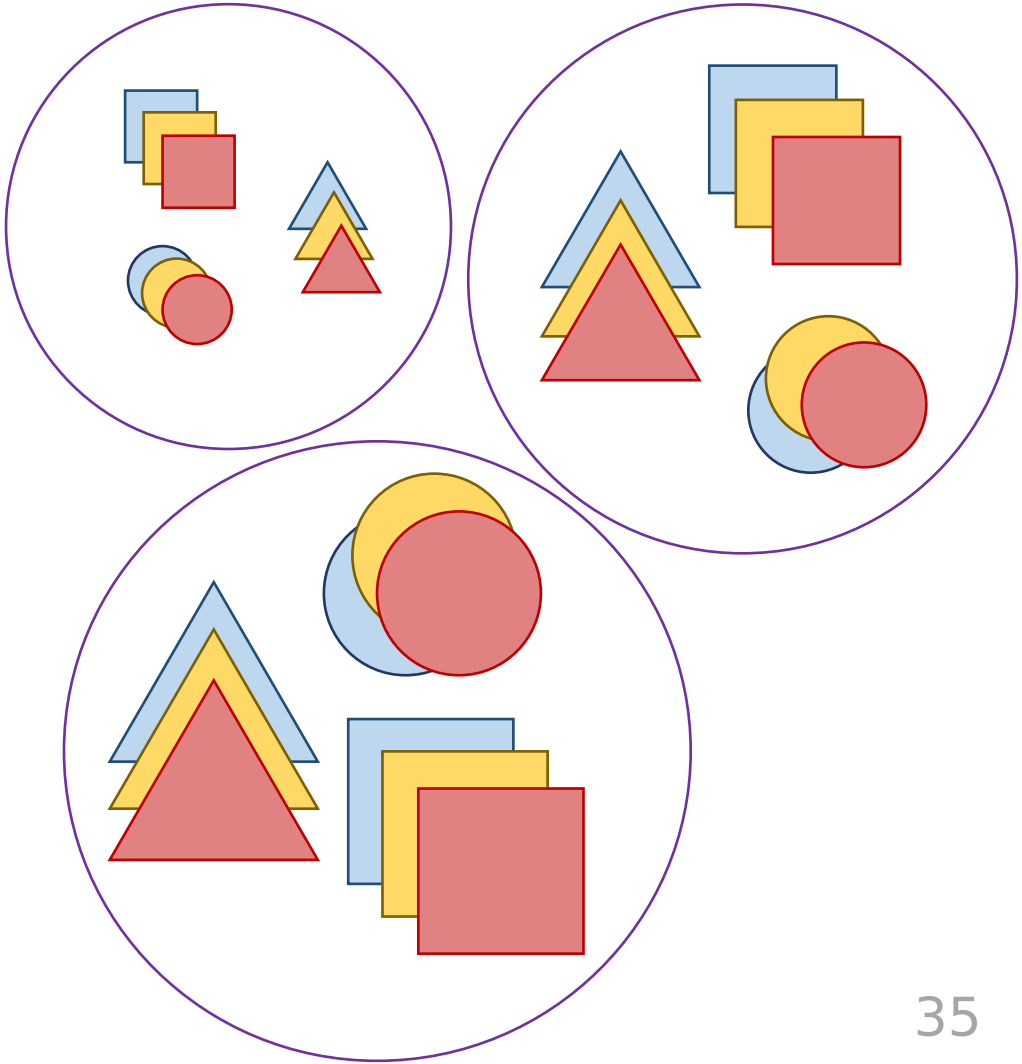
Professor Margo Seltzer
University of British Columbia



Aww… Human nature at work…

- How did you group them?

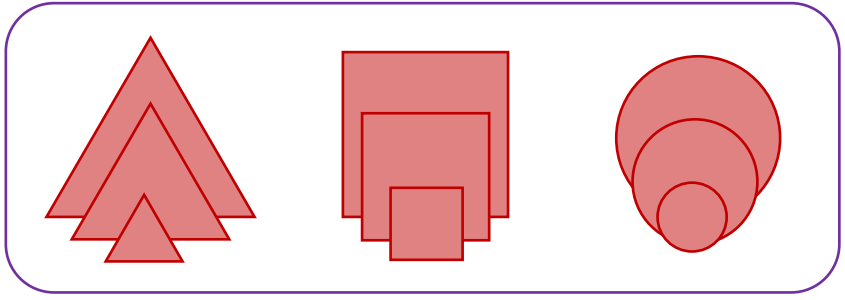




Great Expectations

- How can we expect anybody to use hierarchies?
  - It does not seem to be how we actually think.

- Organize by description instead.
  - No more placing files in directories.
  - Tag files based on what they are.
  - Search for files based on tag / keyword.



- "I want to see all images that are red."
  - "I want all images of squares…," etc.
  - "I want to list all music that is 135bpm."

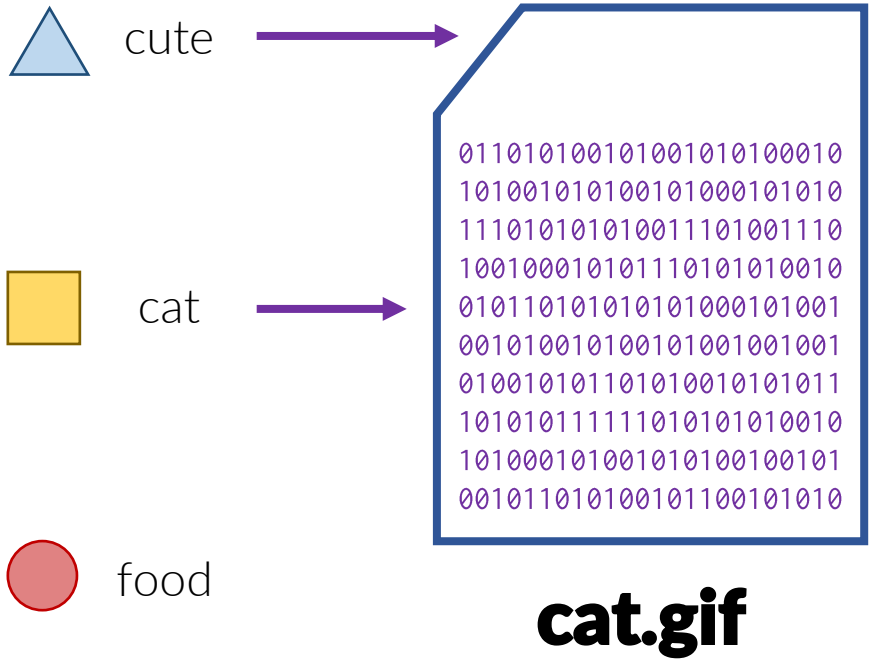
- Draws inspiration from the web: we often search by keywords.

Files with tags…

- We can attach tags to data files.

- Then, when I'm feeling down and out, I can ask my computer: "Hey, show me all the pictures that are cute."
  - No longer looking into random directories to find what I need.

Image by Dimitri Houtteman



cat.gif

POSIX: Contradicting the definition of “path”

- One piece of trouble with being heretical is that everybody asks you “how will you implement this and get people to switch????”

- You don’t want to completely change C functions… so… let’s make use of traditional ideas to implement tags.
  - `ls /cat/cute` can still list all the files with both tags!
  - `fopen("/cat/cute/tabby.png", "rb")` can open a particular file.
  - `fopen("/cut/cats/tabby.png", "rb")` is, naturally but oddly, the same file.

- And you can also fit traditional POSIX paths to tags. Files in “`/usr/home/wilkie`” can simply be files tagged with “`/usr/home/wilkie`”
  - We don’t do this. Why… don’t we just do this? … … Well, change is hard.

C File I/O Summary

- C Standard IO
  - `#include <stdio.h>`
  - `int fread(void* buffer, size_t num, size_t count, FILE* fd);`

    Reads to the given buffer the given number of bytes from the file indicated by the file descriptor. Returns the number of bytes read or 0 if the file reached its end. Negative on error.
  - `int fwrite(void* buffer, size_t num, size_t count, FILE* fd);`

    Similarly writes to the given buffer to the file indicated by the file descriptor.
  - `int fseek(FILE* fd, size_t offset, int SEEK_SET or SEEK_END or SEEK_CUR);`

    Modifies and returns the file position for the given file descriptor to the given offset from the reference point.
  - `int fclose(FILE* fd);`

    Closes the file for this process. No subsequent action can be taken on this file. Returns negative on error.
  - `int fopen(const char* path, const char* flags); // "r", "rb", "wb+", etc`

    Opens the given file at the provided path with access depending on the provided flags. If the flags consist of a string with “r” in it, it will be read-only. If it contains a “w” it will be writable. If it contains a “b” it will not be interpreted as a text file, but as “binary” instead. If it is “w+”, it will completely overwrite the file. If it has an “a”, it will automatically write to the end of the existing file.

C File I/O Summary

- **UNIX System Calls**
  - `#include <unistd.h>`
  - `int read(int fd, void* buffer, size_t num);`

    Reads to the given buffer the given number of bytes from the file indicated by the file descriptor. Returns the number of bytes read or 0 if the file reached its end. Negative on error.
  - `int write(int fd, void* buffer, size_t num);`

    Similarly writes to the given buffer to the file indicated by the file descriptor.
  - `int lseek(int fd, size_t offset, int SEEK_SET or SEEK_END or SEEK_CUR);`

    Modifies and returns the file position for the given file descriptor to the given offset from the reference point.
  - `int close(int fd);`

    Closes the file for this process. No subsequent action can be taken on this file. Returns negative on error.

  - `#include <fcntl.h>`
  - `int open(const char* path, int flags); // O_CREAT, O_RDONLY, O_RDWR, O_TRUNC`

    Opens the given file at the provided path with access depending on the provided flags. If the O_CREAT flag is given, the file is created if it does not exist. If the O_TRUNC is passed and the file is writable, it will remove all the data in the file after it opens. If O_RDONLY is specified, no writes can occur.

Summary

- **Files are just binary blobs of information.**
  - Disambiguating that data requires a specification and consistency.
- **Disks are physical and rely on nature, which is chaotic.**
  - We have strategies for encoding digital data on analog (magnetic) media.
  - Physical addressing requires care in where blocks of data are stored.
- **File systems are an opinionated space related to how humans organize data on disk (and share/discover that data)**
  - Files are generally organized in trees, much like our virtual memory!
  - Hierarchical file systems still dominate: directory structures.
  - Other file systems are possible: relational searches and tags.

Distributed Filesystems and Storage

- Now… what happens when we have file systems that span machines?
  - The power of networks and storage combined!
- What are some unique issues to files that span multiple systems?
- How can we create better methods of transmitting data between machines?
  - What if we break down the “client-server” model.
  - What if files and data need not be in one particular place?
  - How do we find information, then?
- Stay tuned!