Today:
- Links
- Take a step back: how to break down complexity?
- How do we interact with FS as an application?

Links:
- Hard link: Link to a specific `inode` (= directory entry)
- Soft link: Link to a logical path

Ex: `/hello/world/h.txt` @ 20
create `/link.txt`

inode 30
- type: link
  num: `501011`

Block 50
- `/hello/world/hi.txt`
Layering: Building layers that make it one degree easier/simpler for layers above

- Application: "expect "hi" to /hi.txt"
  ➤ Permissions
  ➤ Pathnames: /hi/hello.txt
  ➤ Filename: map names → inode #
  ➤ File: "get me the second block of file #"
  ➤ Inodes: what do I find pieces of this file?
  ➤ Blocks

> Hardware (sectors)
Benefits:
- Split up complexity
- Secure layers easily

**read (...)**
check permissions ( )
interact with sectors ( )

Key: need to prevent programs from doing anything they want w/ hardware, but still need hardware!

Hey!
Can you read the file for me?
ring &; unprivileged

System call: asking the kernel for something

ring 0 = privileged
Things we need to establish:
- What file? Maybe file name
- How much?
- Starting where?
- Where to put the data

Propose 1:

read (char * file name, size_t num_to_read)
size_t start offset, void * dest_buffer);
- Client has to remember some state
- Complexity in edge cases / error handling

\[
\text{int session ID = open (filename, I want to read)};
\]
\[
\text{read (session ID, dest_buffer, num_to_read)};
\]
\[
\text{close (session ID)};
\]
\[
\text{write (session ID, src_buffer, num_to_write)};
\]
I want to use `hello.txt`.

OK, use session 3.

I want to read 29B from session 3.

Here you go.