Today:

- Links
- Take a step back: how to break down complexity?

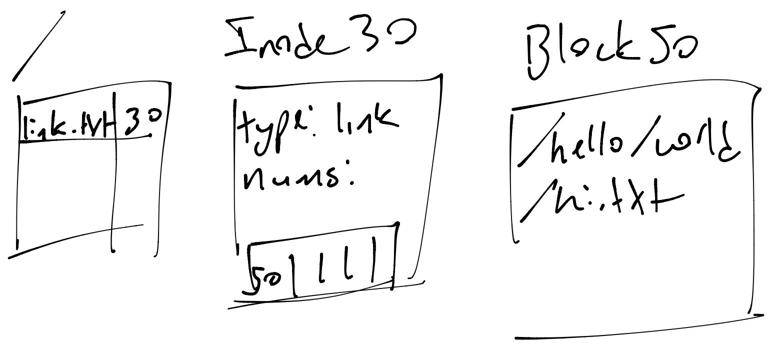
- How do we interact with fs as an application?

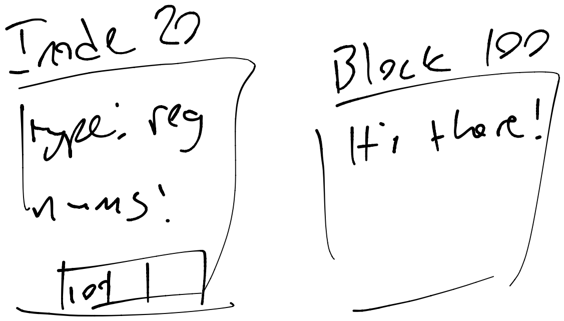
Links:

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

Ex: /hello/world/hi.txt @ 20

create /link.txt

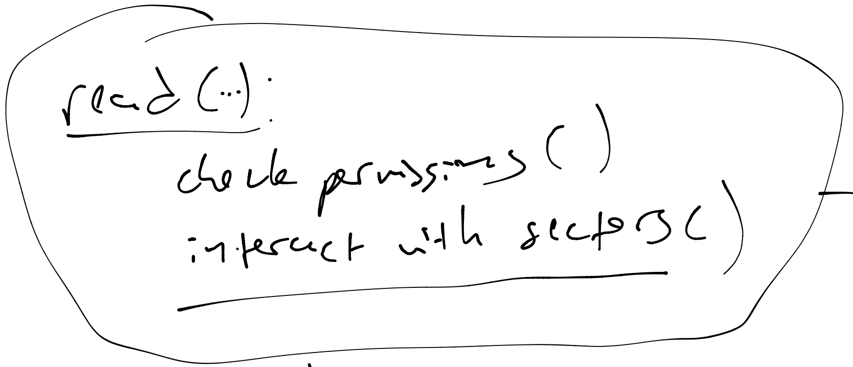




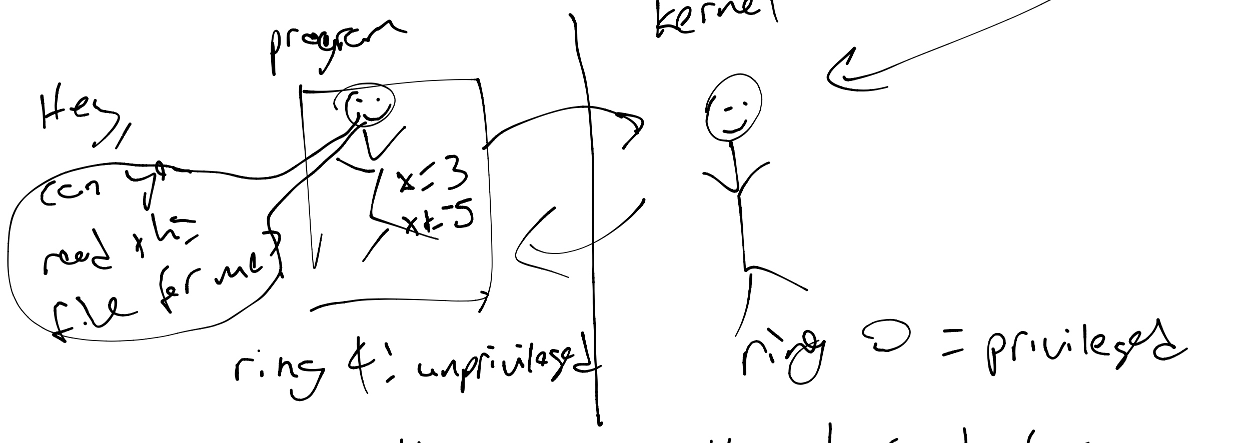
Layering: Building layers that make it one degree easier/simpler for layers above

- Application: "append "hi" to /hi.txt"
  - Permissions
- Pathnames: /hi/hello.txt
- Filename: map names → inode #s
- File: "get me the second block of file #3"
- Inodes: where do I find pieces of this file?
- Blocks
- → [Hardware (sectors)

Benefits:
- Split up complexity
- Swap layers easily



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



System call: asking the kernel for something

Things we need to establish:

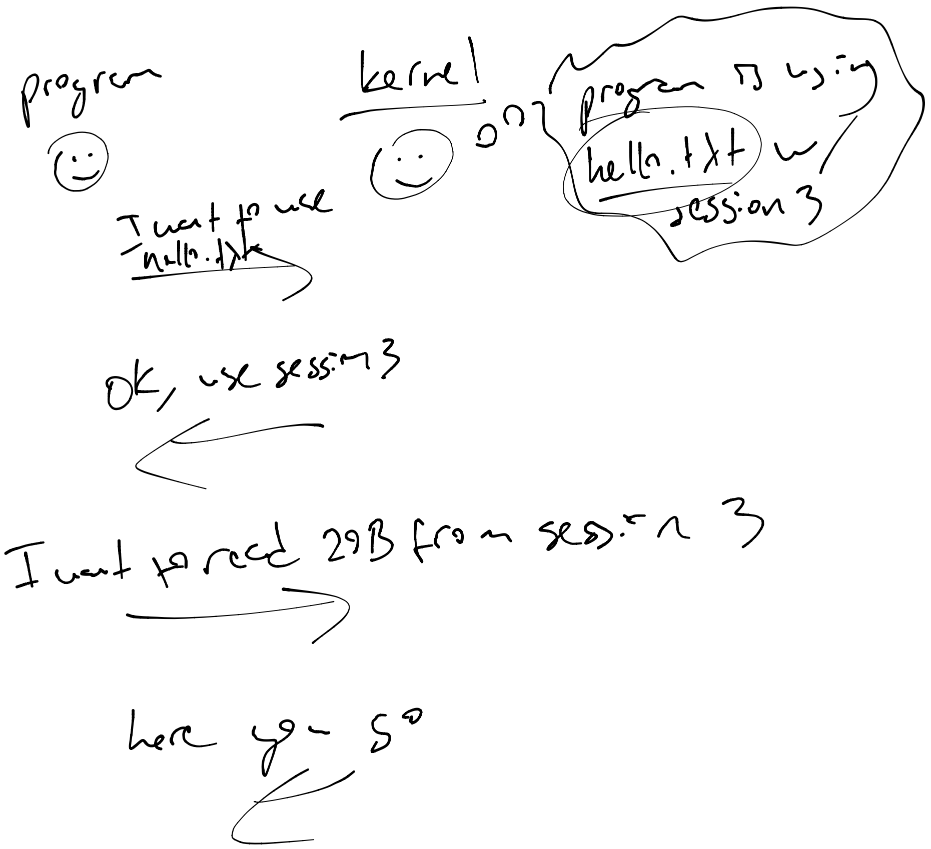
- What file? maybe file name
- How much?
- Starting where?
- Where to put the data

Proposal:

```
read(char * filename, size_t num_to_read,
        size_t start offset, void * dest_buf);
```

- Client has to remember some state
- complexity in edge cases / error handling

```
int sessionID = open(filename, I want to read);
read(session ID, dest_buf, num_to_read);
close(sessionID);
write(session ID, src_buf, num_to_write);
```


program
kernel
ooo
program is using hello.txt w/ session 3
I want to use hello.txt
OK, use session 3
I want to read 29B from session 3
here you go