- [x] 1. Last time
- [x] 2. Intro to file systems
- [x] 3. Files
- [x] 4. Implementing files
- [ ] 5. Directories
- [ ] 6. Performance

} next time

Intro to file systems

What does a FS do?

- provide persistence
- create a way to name data on the disk

FS: can be implemented in lots of places

- We focus on the disk, generalize later

Note: disk is the 1st thing we've seen that is **both** modifiable **and** persistent.

Files

What is a file?

From user's view: a named, contiguous run of bytes

From FS's view: collection of disk blocks

Job of a FS:

map {file, offset in file} $\xrightarrow{FS}$ disk address

operations:

create(file), delete(file), read(), write()

Goal: operations have as few disk accesses as possible and minimal space overhead

4. Implementing files

A. Contiguous

B. Linked files

C. Indexed files

Assume for now that a given file's metadata is known to the system.

Access pattern to support:

- Sequential
- Random access

Ideal is good sequential + good random access performance

Candidate designs:

A. Contiguous allocation

user pre-specifies length

[<free> a1 a2 a3 <5 free> b1 b2 <free>]

+

+

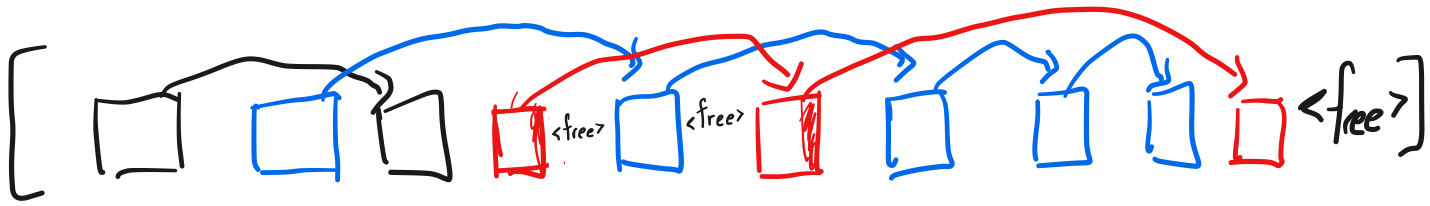
+

-

-

B. Linked files

metadata is pointer (disk address) to file's first block



+

+

+

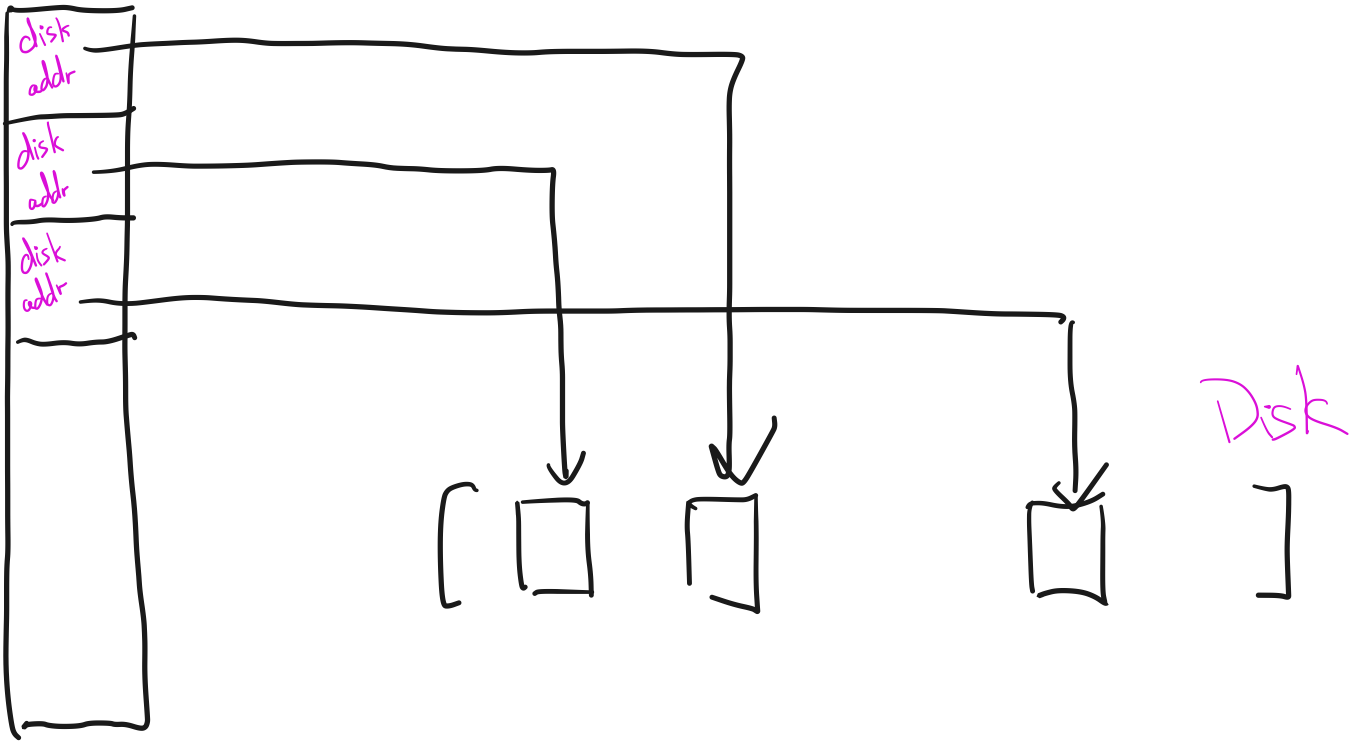
-

-

C. Indexed files

attempt 1

metadata:



+ Sequential good
+ R.A. good
- Very large

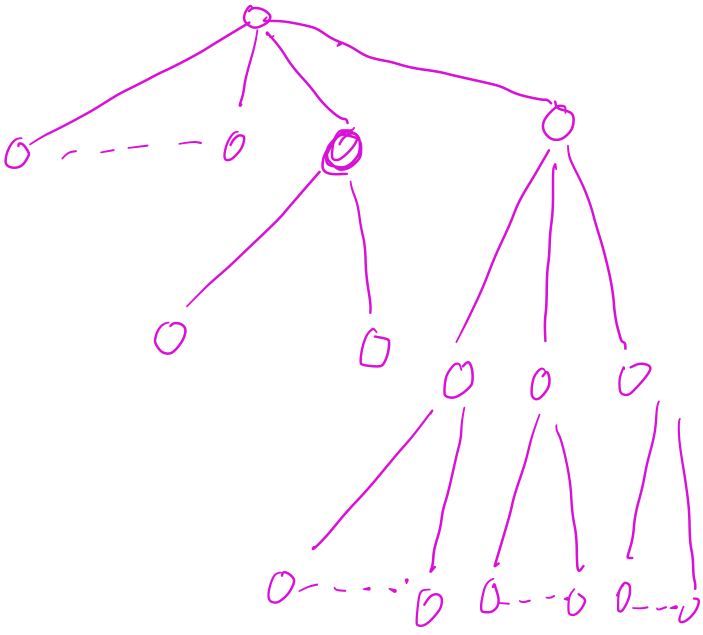
attempt 2

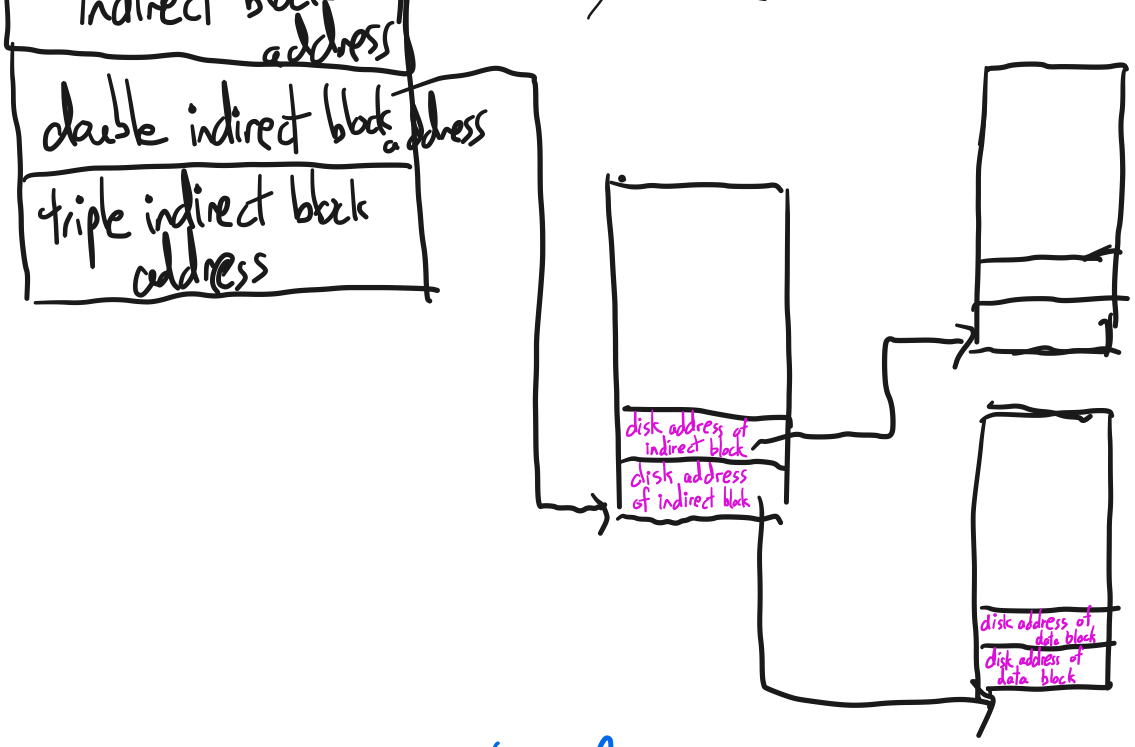
+

-

attempt 3

Metadata: inode

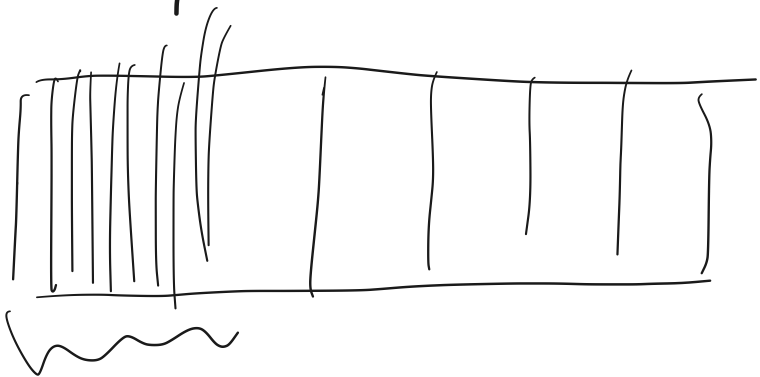




+ Simple, easy to build
+ Fast access for small files
+ Enormous max file length
- Worst case # disk accesses can be bad
- Space ovhd in worst case can be bad
- Data, metadata wind up all over the disk

inodes: stored in a fixed-size array, known location

vocab: "inumber"



stat (&sb);

slots for inodes

---

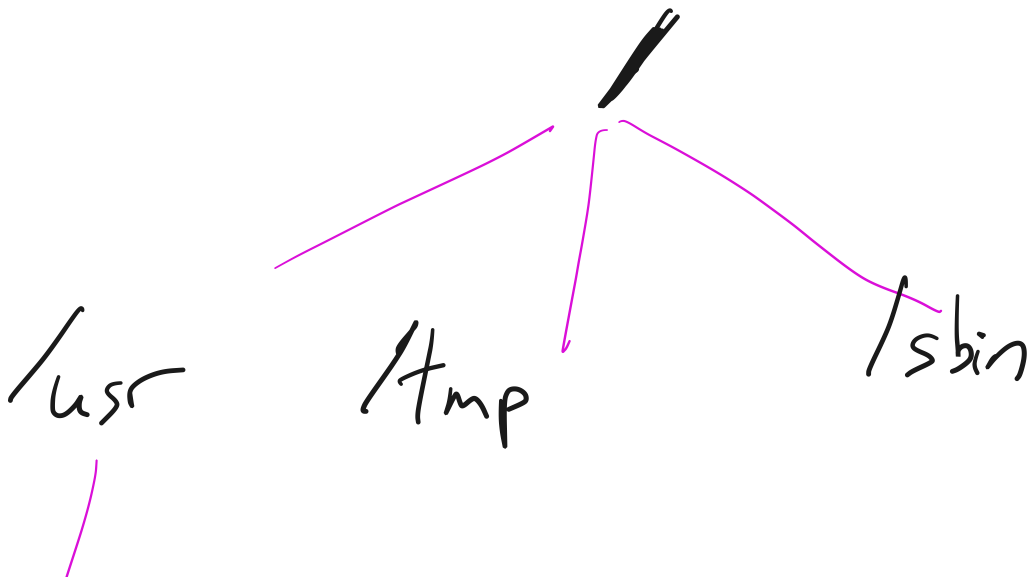
5. Directories (next time)

/bin/

/sbin

/usr

/tmp




/usr/mw
/lab4
kernel.c ...