

- 1. Last time
 - 2. Intro to file systems
 - 3. Files
 - 4. Implementing files
 - 5. Directories
 - 6. Performance
- } Next time
-

Intro to file systems

vi foo.txt

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

foo.c

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} $\xrightarrow{\text{FS}}$ 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:

OS/360

A. Contiguous allocation

user pre-specifies length

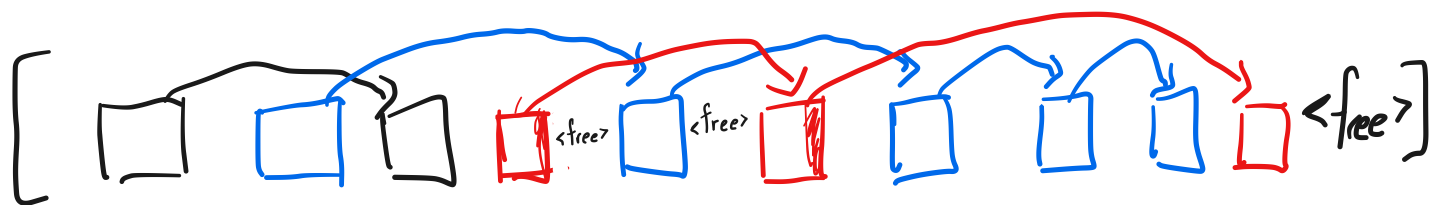
[<free> a1 a2 a3 <5 free> b1 b2 <free>]

- + Simple
- + Low space overhead

- + Random, Sequential
- fragmentation
- length of every file is bounded

B. Linked files

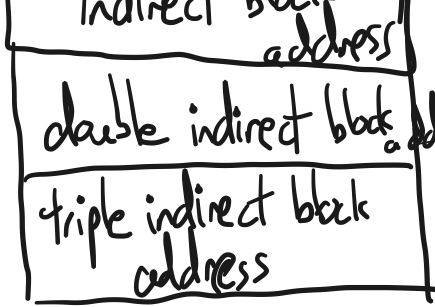
metadata is pointer (disk address) to file's first block



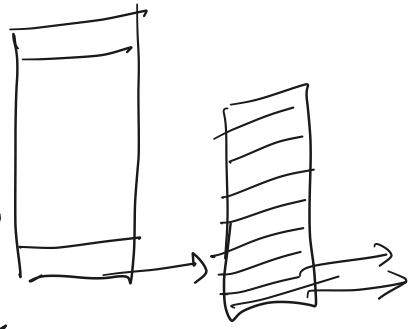
- + no (external) fragmentation
- + sequential access easy/good
- + file length not bounded up front
- random access disaster
- alignment of data is messed up

C. Indexed files

attempt 1



block in the given file



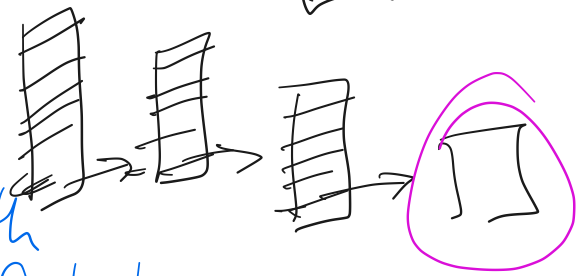
+ small # of accesses for larger files

+ huge max. file length

+ Small files are fast to access

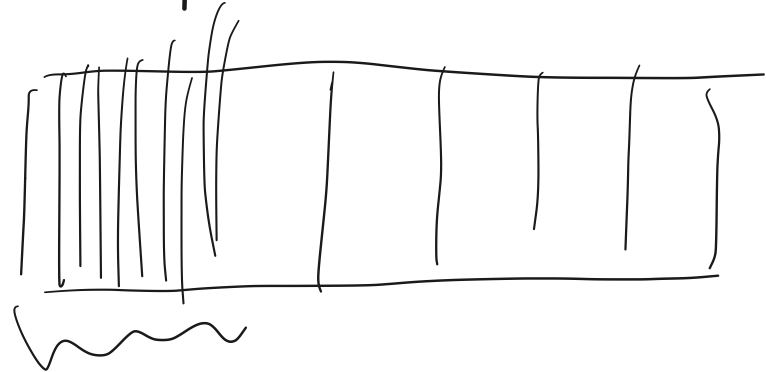
- worst cases pretty bad for space & disk accesses

- data stream throughout the disk



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

vocab: "inumber"



stat (&sb);

slots for
inodes
