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

---

2. Intro to file systems

What does a FS do?

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

FS's 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.

---

3. 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

file's metadata is known

Assume for now that a given files metadata

to the system.

Access patterns to support:

- Sequential
- Random access

Ideal is good sequential + good random access performance.

Candidate designs:

read(fi6, 1496)

A. contiguous allocation

IBM os/360

user pre-specifies length

metadata contains location, size

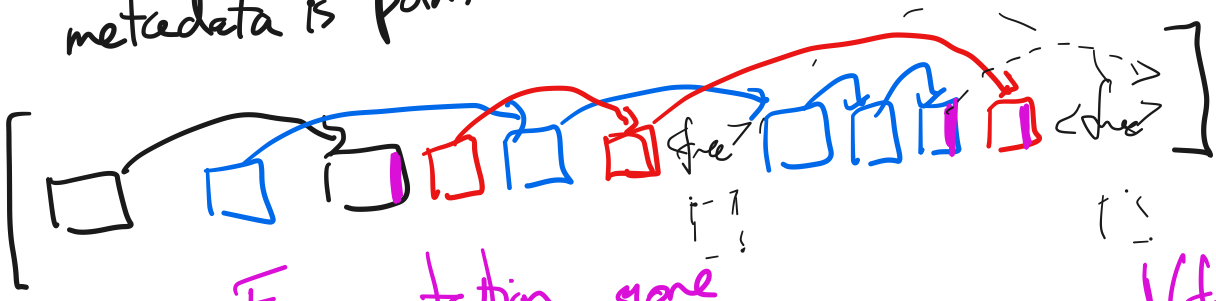
[<free> a1 a2 a3 <free> b1 b2 <free>]

+ Simple: easy, low overhead
+ Fast access for both seq + random
- Fragmentation

B. linked files

(disk address) of file's first block

metadata is pointer (disk addr)



+ Fragmentation gone

+ Seq access pretty good + easy

- Random access is a disaster

- Pointers take up room in blocks; messes up data alignment

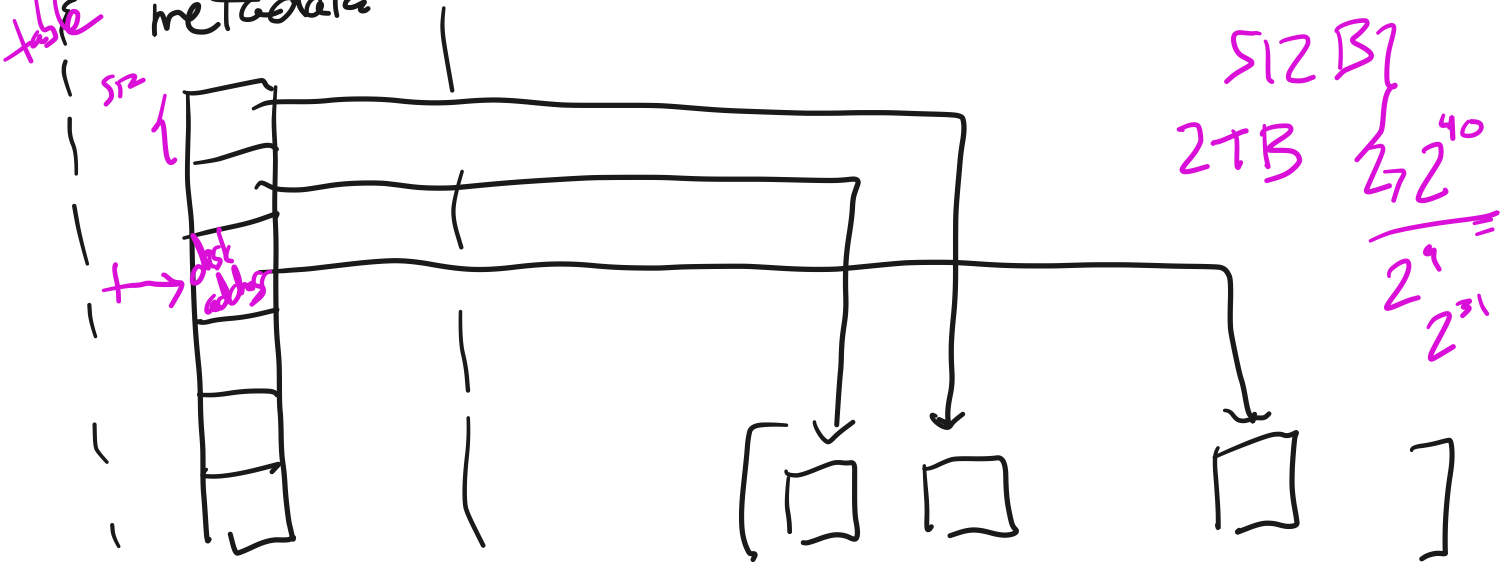
read(fd, 2500) ! 10^6

C. Indexed files

attempt 1

read(fd, 1500)

table metadata:



512B

2TB $\rightarrow 2^{40}$

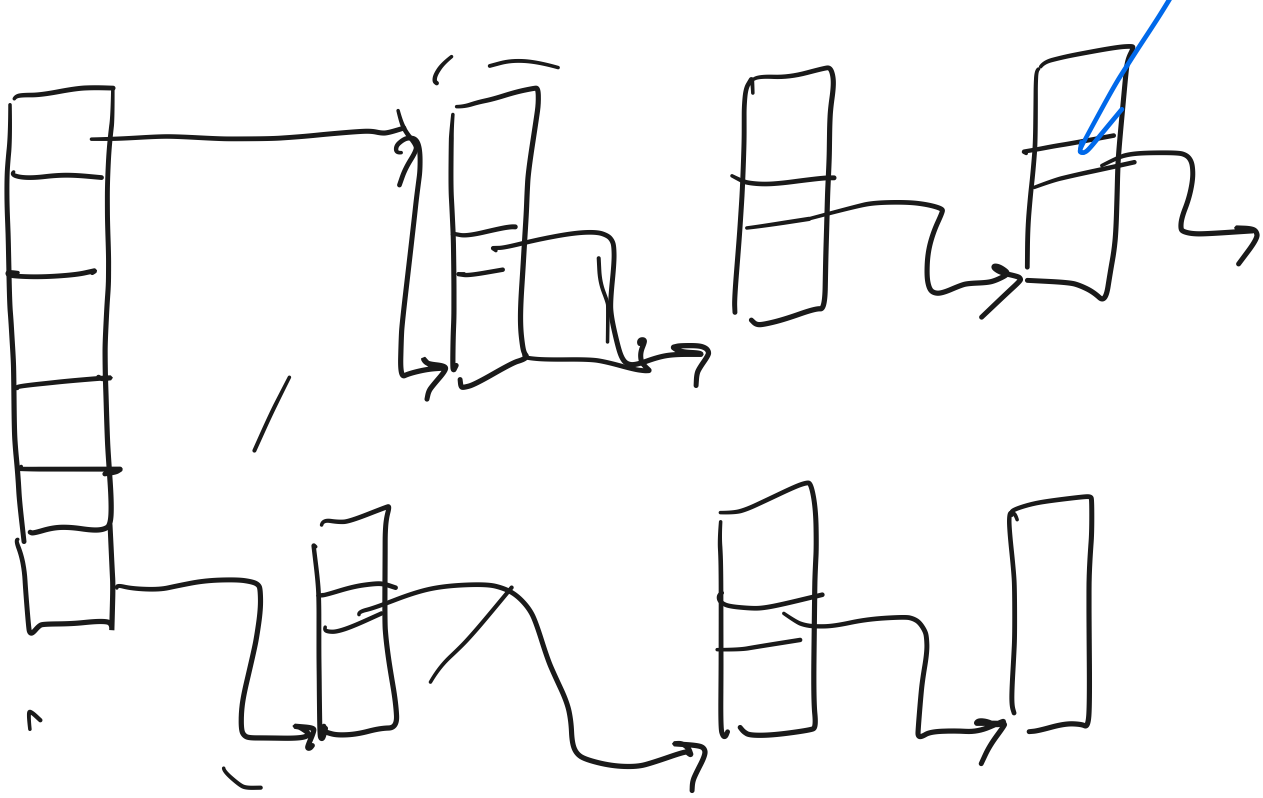
$\frac{2^{40}}{2^9} = 2^{31}$

+ Seq, RA are both easy

- Metadata enormous

attempt 2

disk addr of data



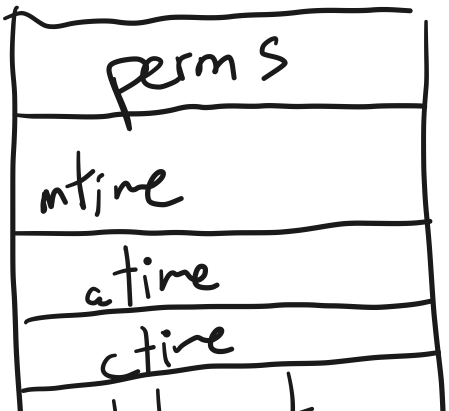
attempt 3 (find)
imbalanced tree

classic Unix file system

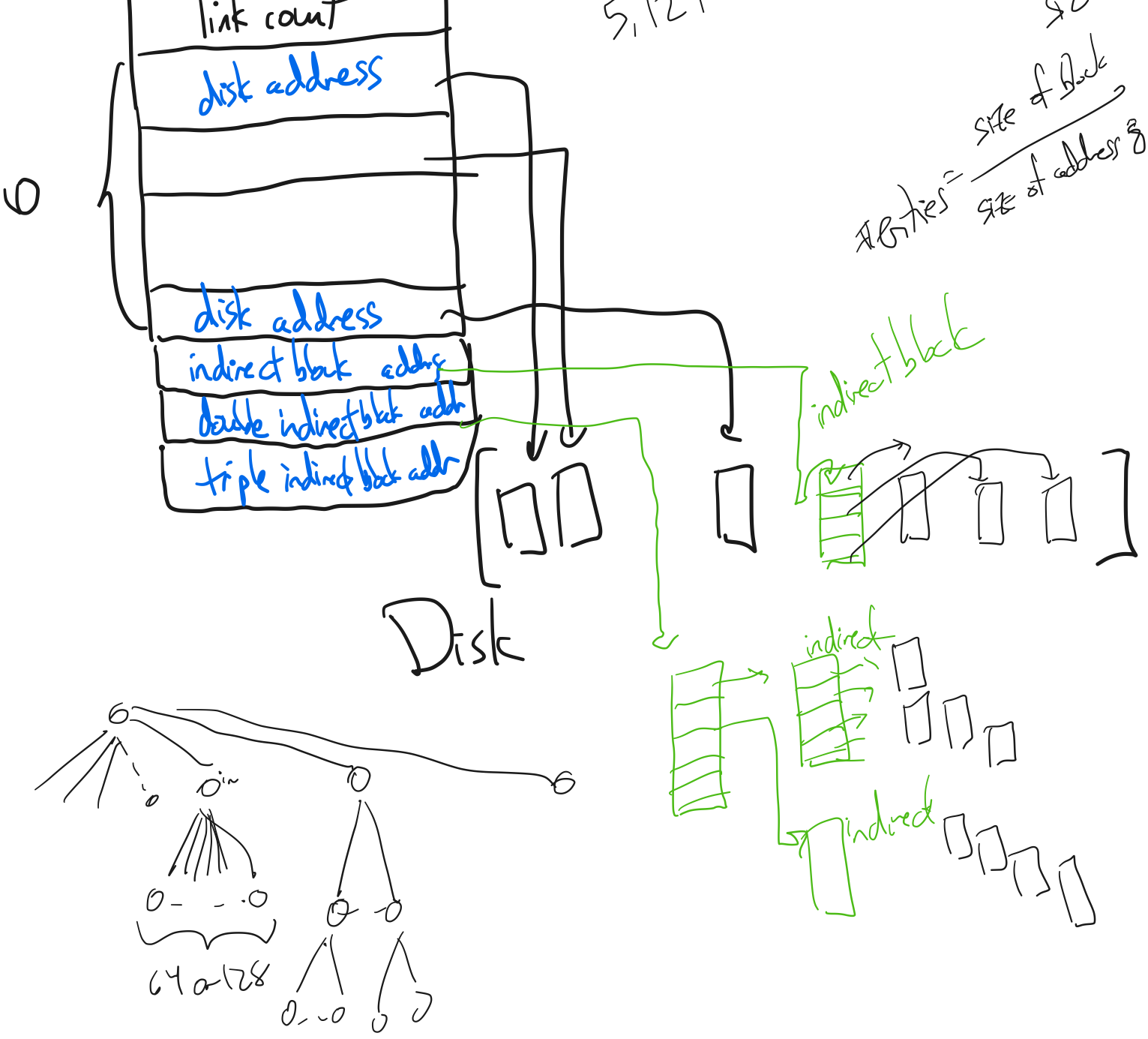
"inode"

Metadata: inode

inode



5,120



+ Simple, easy to build, fast access to small files
+ Max file length can be enormous
- Worst case seeks/accesses not great
- Worst case overhead (e.g., 11-block file) not ideal
- metadata + data strewn throughout the disk

inodes

stored in a fixed-size array, known location

vocab: "inumber"

stat(&_);

---

5. Directories

multics → Unix

Approach 0: Users remember their data location

Short history of directories

Approach 1: Single dir. for whole system

Dir

name1, inum

name2, inum

name3, inum

foo, 576

bar, 110578

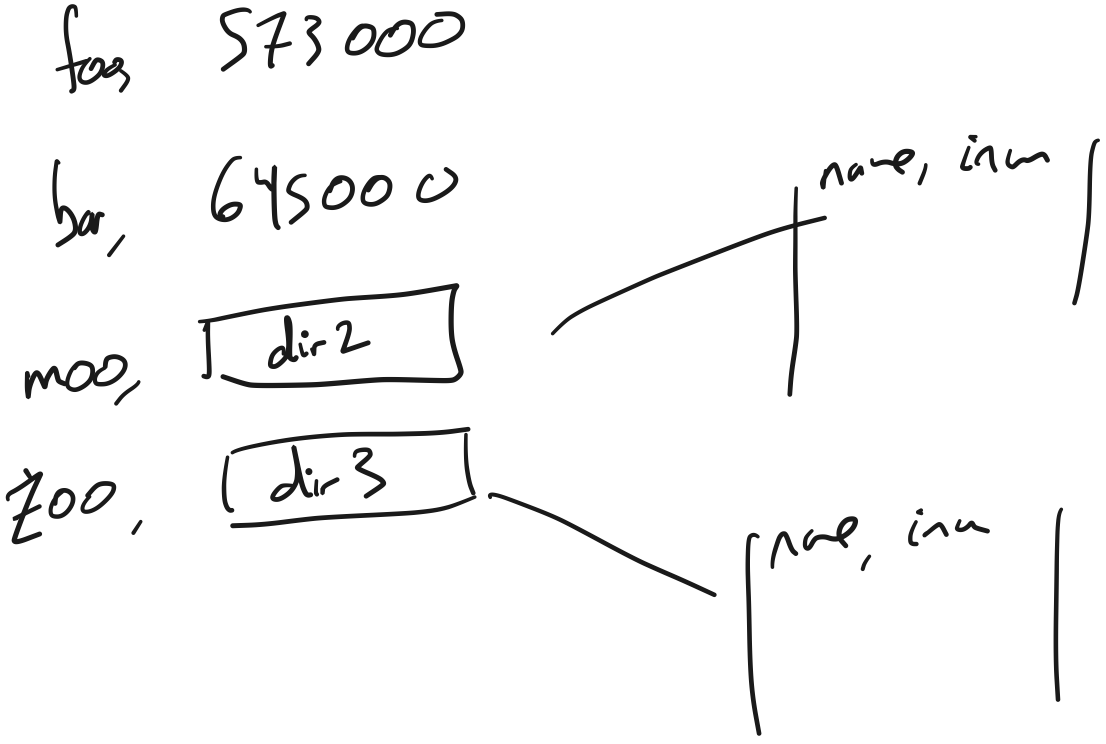
labyrinth, 205208

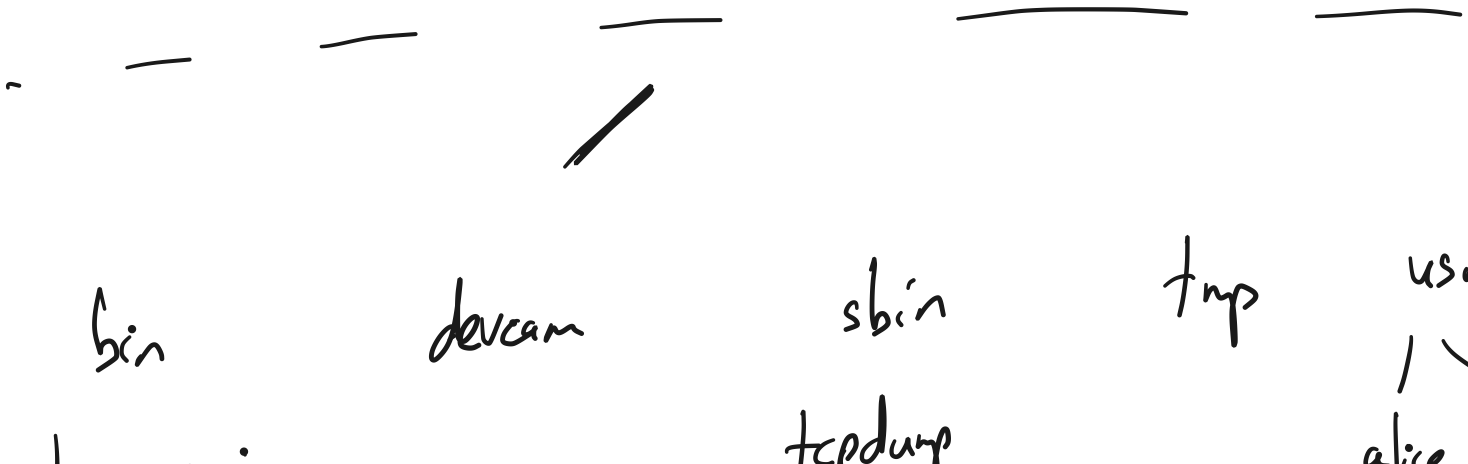
answers.txt

Approach 2: Single dir. for each user

- clumsy
- ls ⟹ 10,000

Approach 3: Hierarchical name space





ls, grep, vim

ping

dir:

<name, inode #>

<bin, 1011>

<sbin, 2048>

could be the inum of a directory

:

inode

bootstrapping:

"/" is inode 2

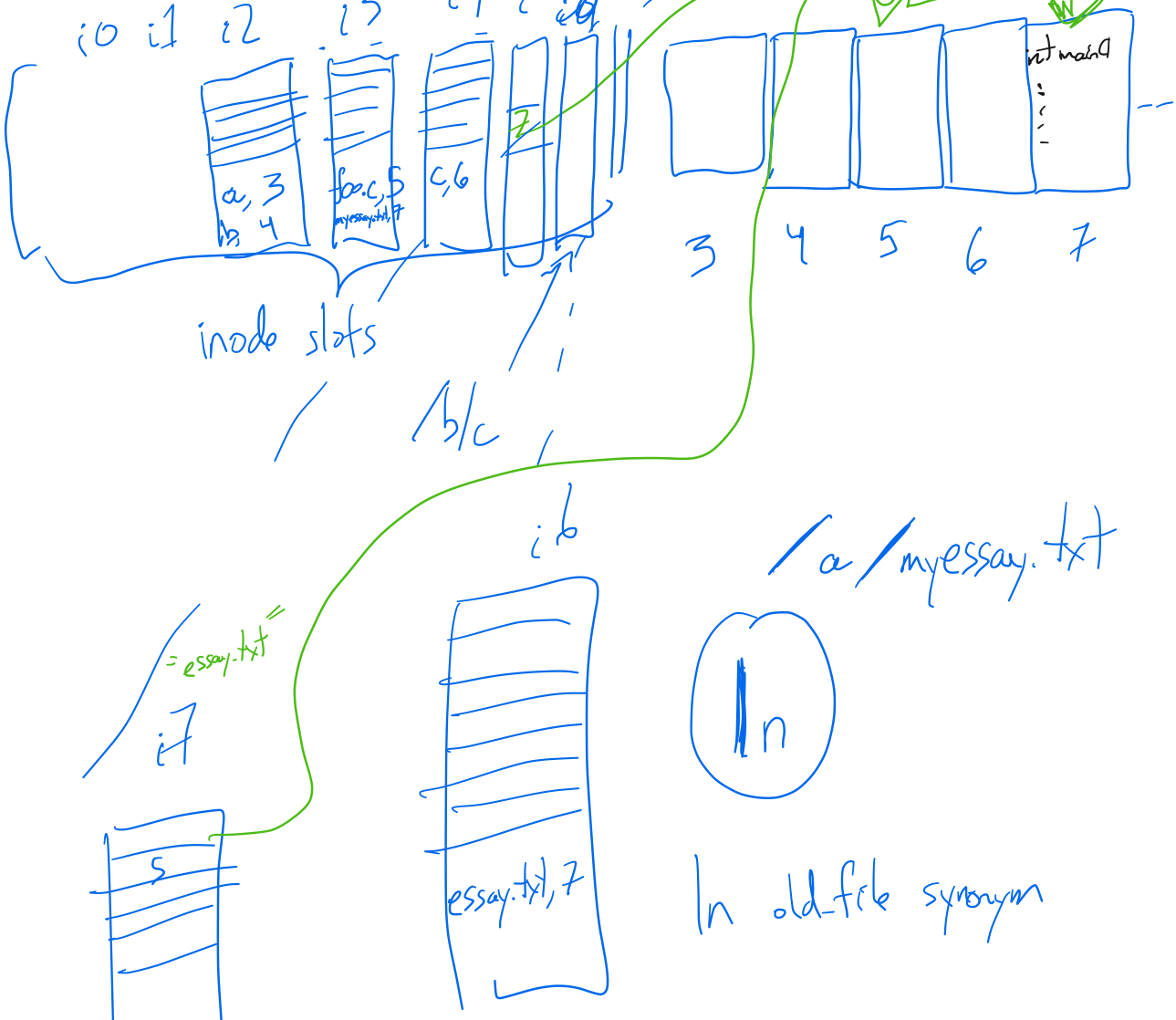
example:

/a/foo.c

/b/c/essay.txt

/a /b /a/foo.c

data blocks



delete:

rm

↓

unlink