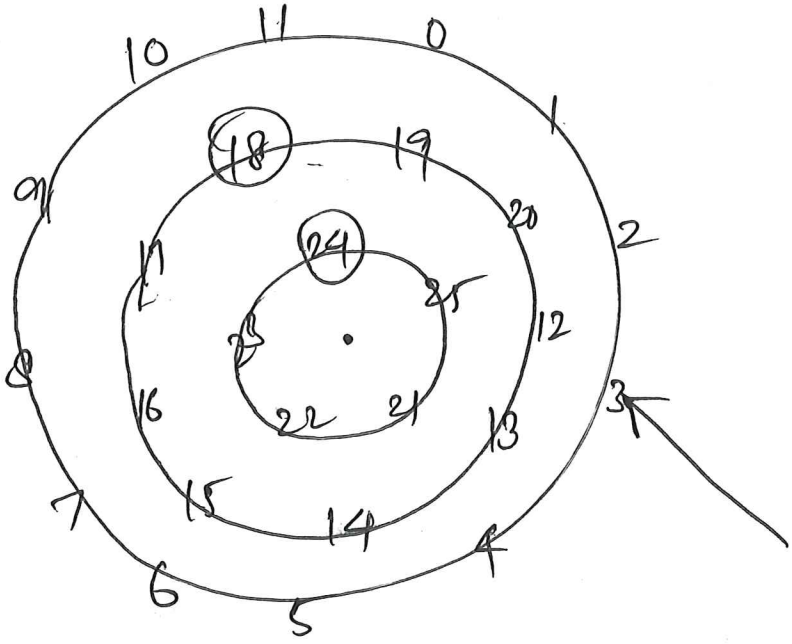
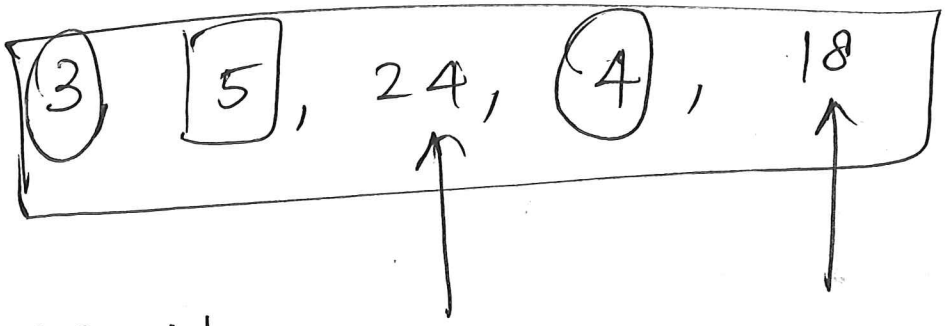
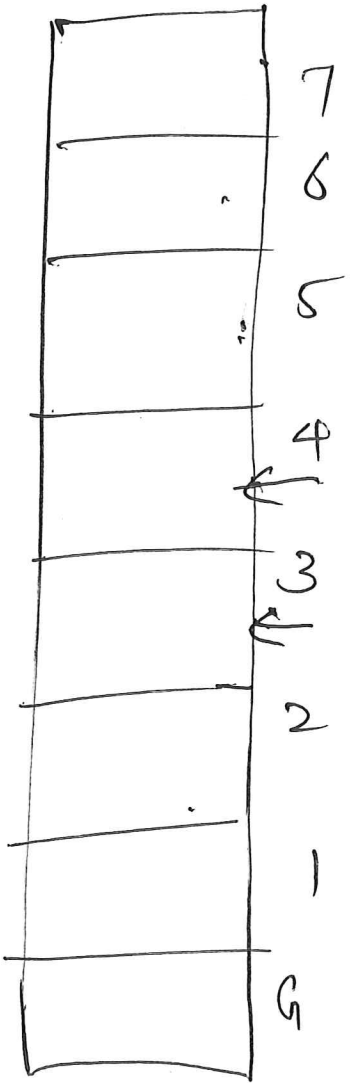
Disk Scheduling

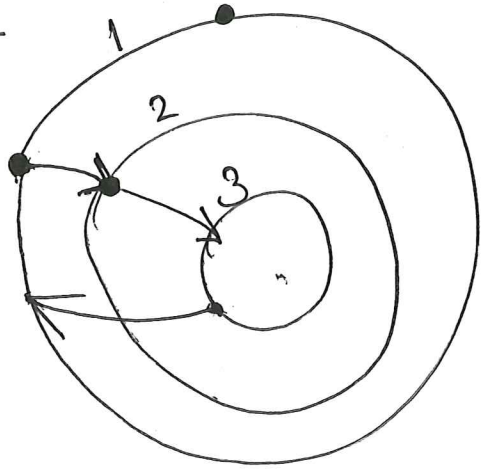
SSTF

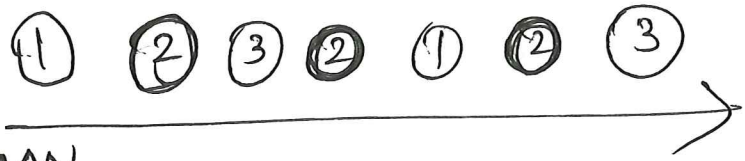




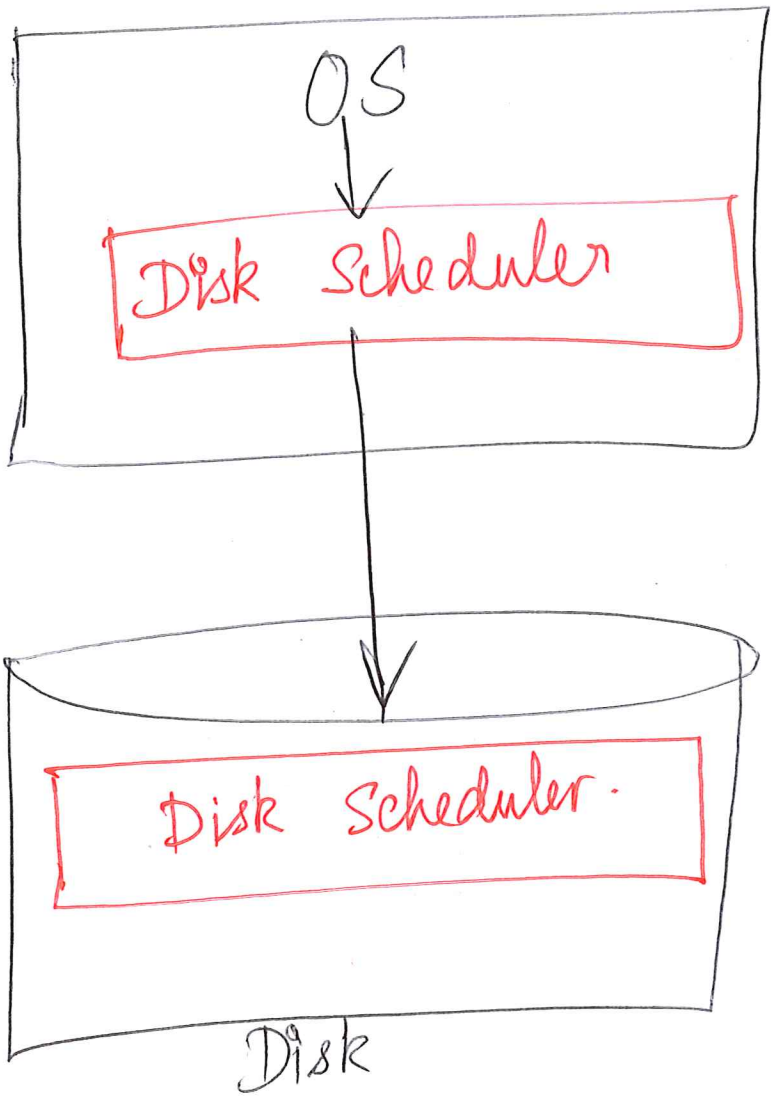


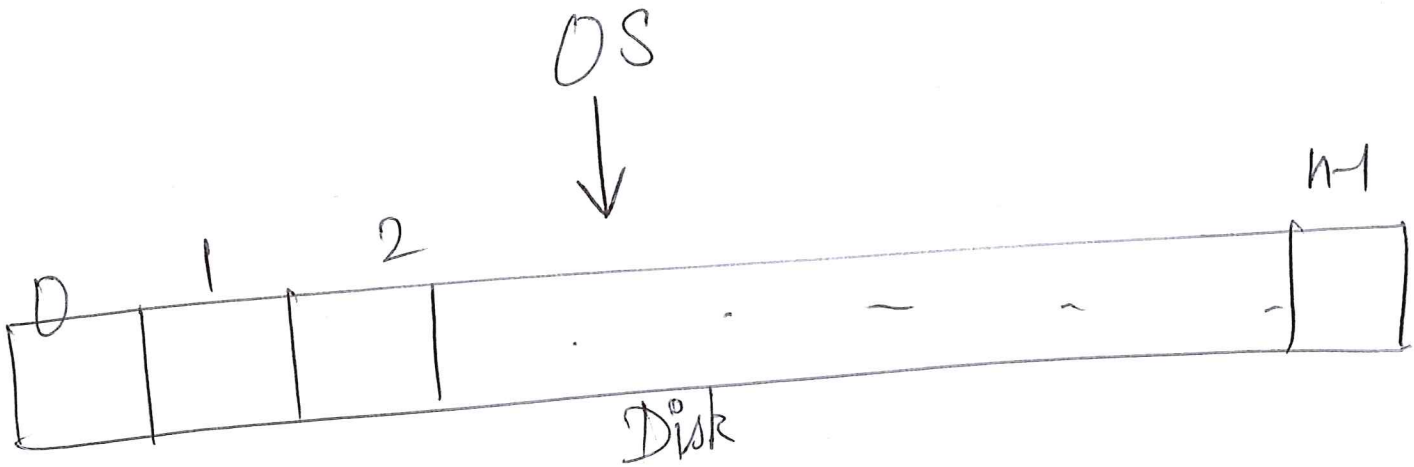
SCAN





C-SCAN

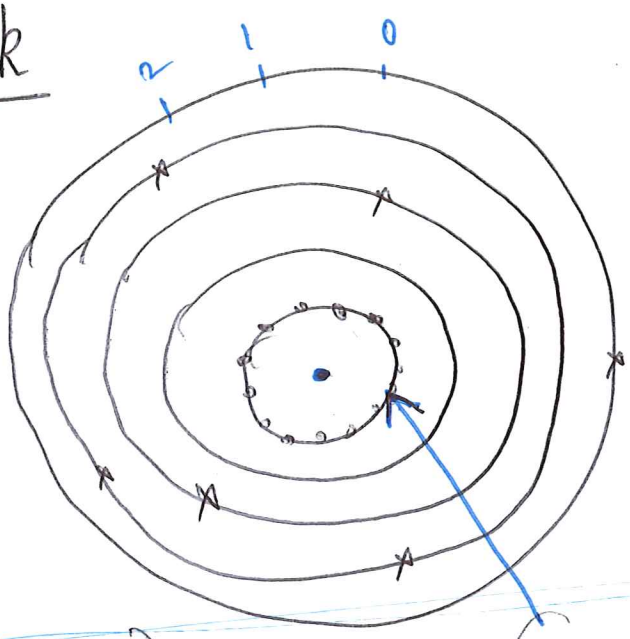

OS
Disk Scheduler
Disk Scheduler
Disk


OS
0
1
2
n-1
Disk

Persistance

I/O devices < interface / implementation.

Disk



Rate(Seq. Read) >> Rate(Random Reads)

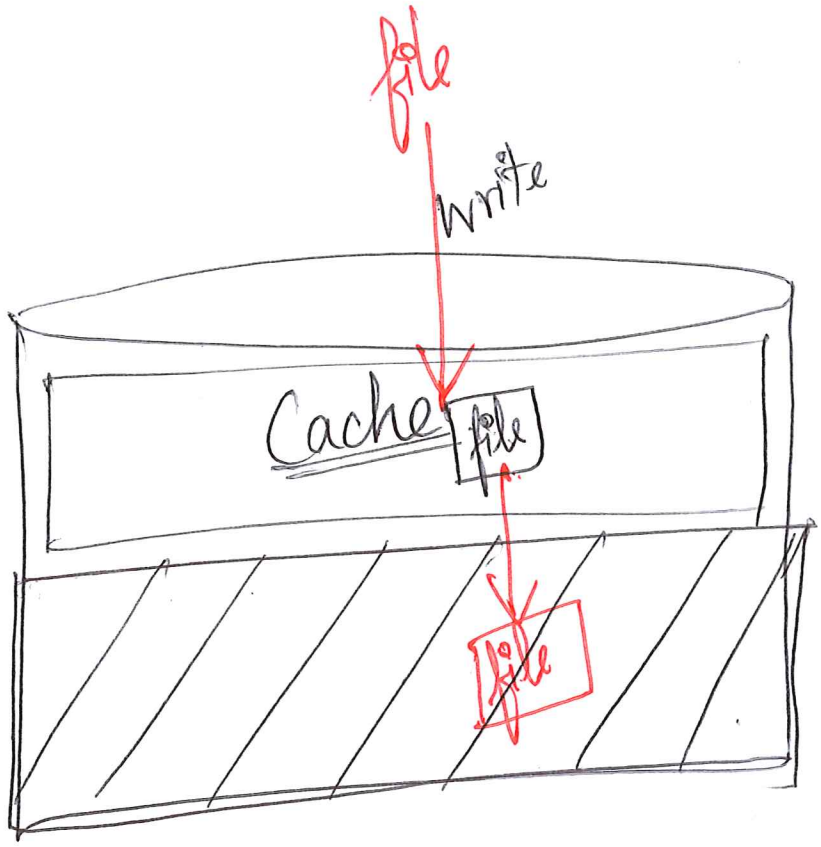
Disk Scheduling

1) FIFO.
2) SSTF (Seek) < starvation; ignores rotation
3) SPTF / SATF
4) BSATF (windows)

SCAN (Elevator) < C-SCAN; F-SCAN

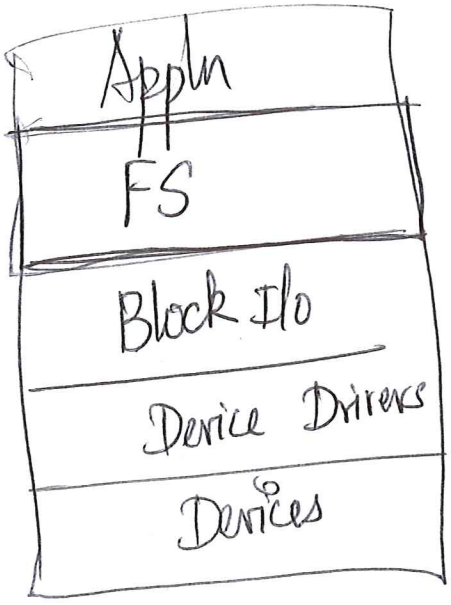
1 2 3 2 1 2 3

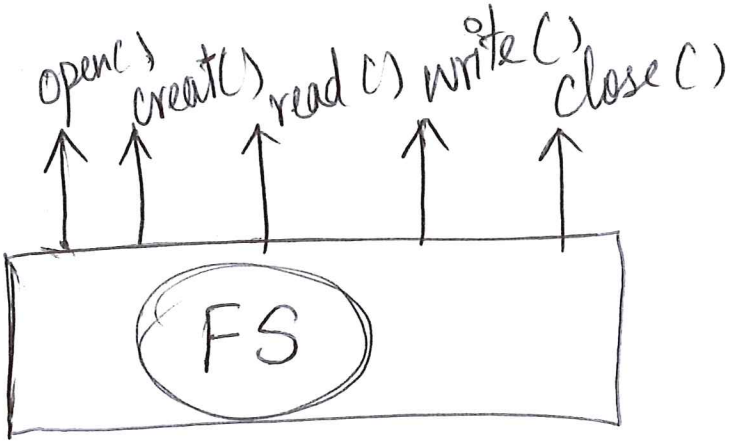
1 2 3 1 2 3 1 2 3



1. File System APIs.
2. File System Implementation
   - Data Structures
   - Access Methods

File System APIs





Basic Abstractions

File (Regular File)

- array of bytes
- low-level name (inode #)
- human-readable name

Directory

- special type of file
- human-readable name
- low-level name
- array of records

human name	low-level name
a.txt	1231271
dir1	1470712

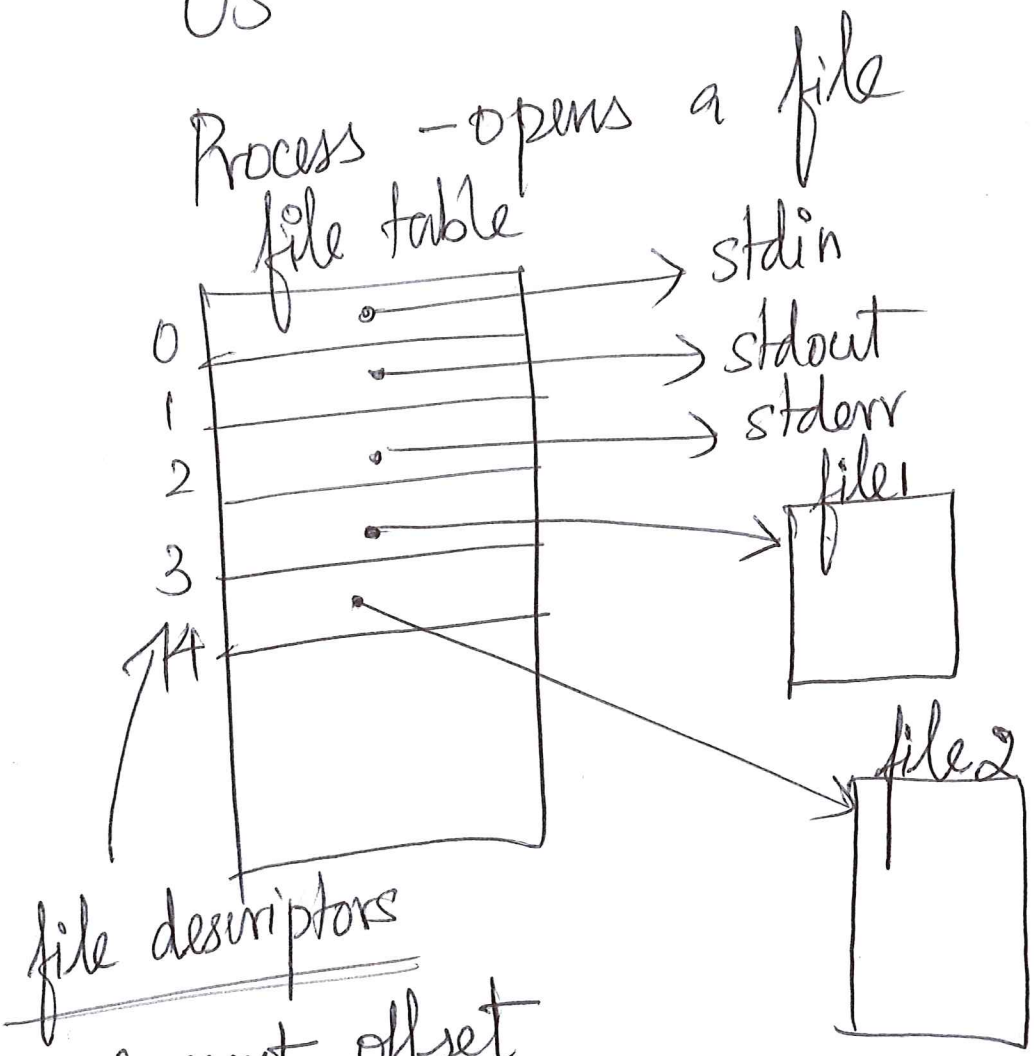
Files

1) Creation

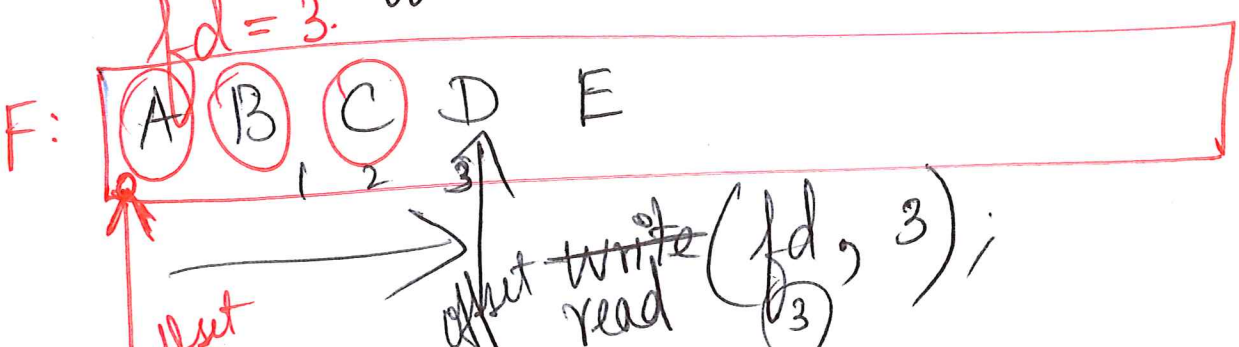
open()
creat()

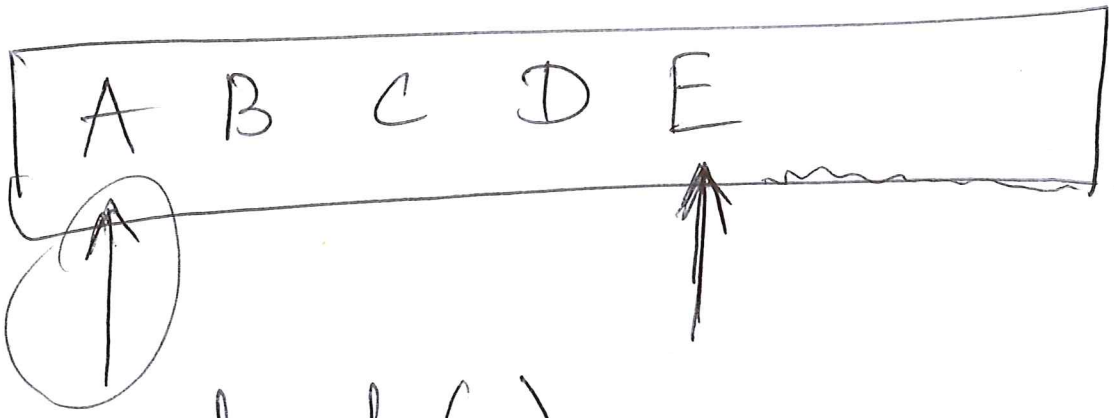
OS

Process - opens a file



Current offset

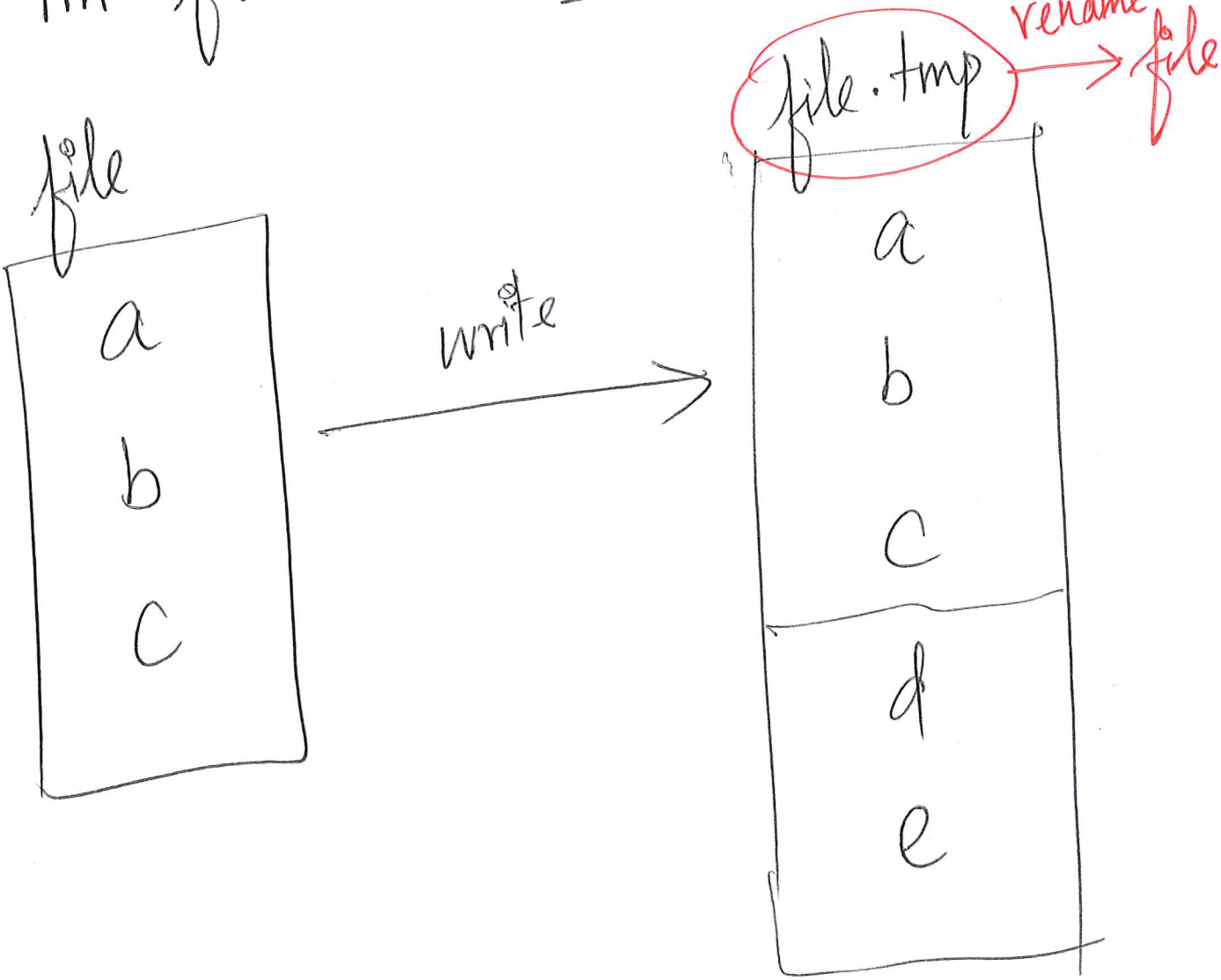




lseek()

lseek(fd, SEEK_END

rm file → unlink()

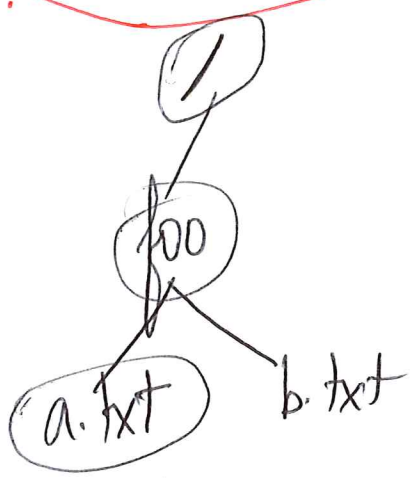


Directories

- special type of file
- create, read, delete
  mkdir ls rmdir
- write - NOT possible.

alias ls → rm -rf /

/foo



H.N.	inode #
.	1731
..	2
a.txt	1812
b.txt	1012

. → current directory

.. → parent directory

Absolute pathname: /foo/a.txt

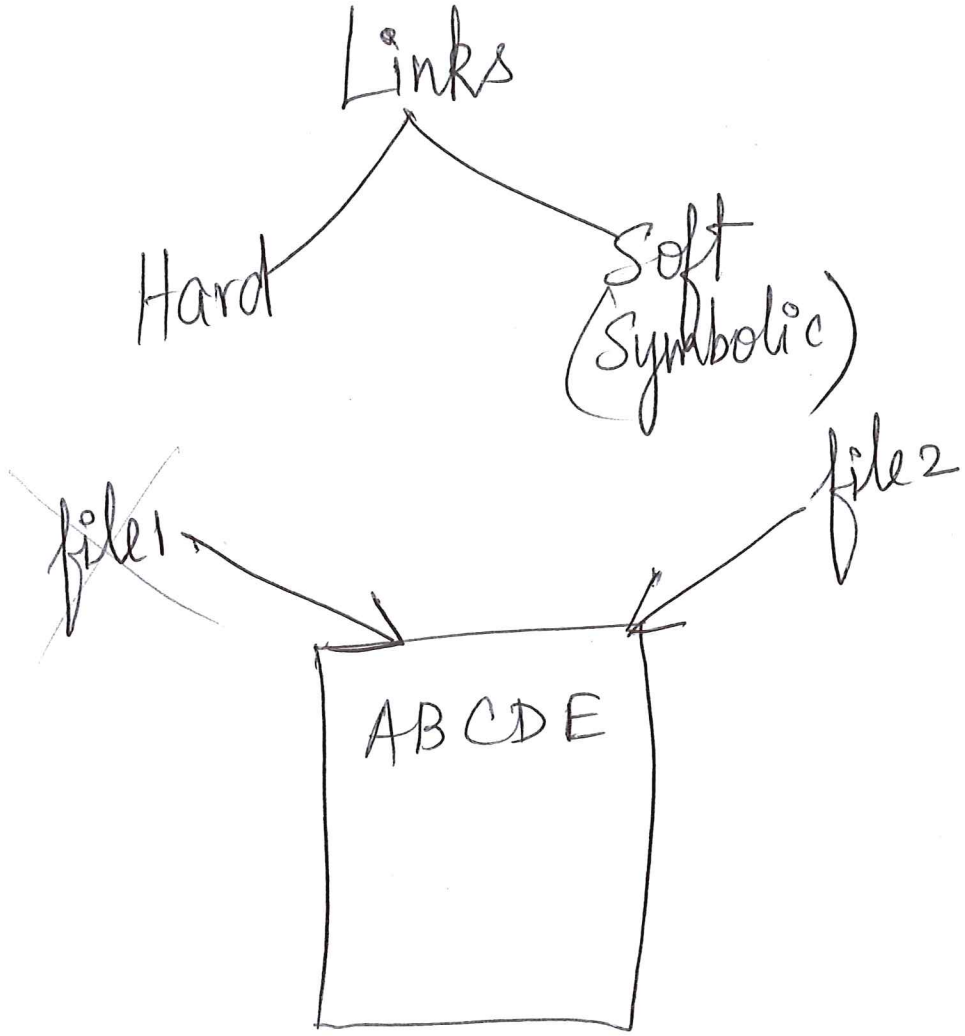
Relative pathname: a.txt

open(/foo/a.txt)

① / (root)

② foo

③ a.txt

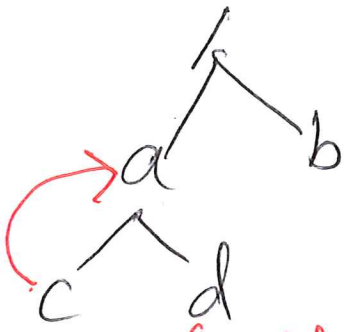


① Hard Links

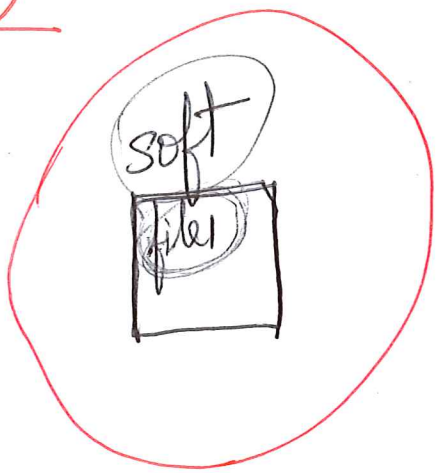
> ln file1 file2

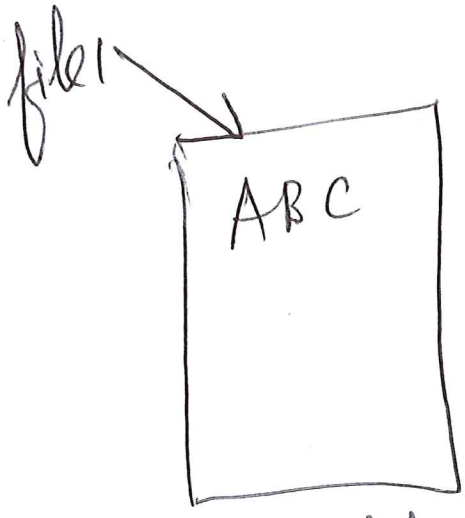
H.N.	inode#
file1	123
file2	123

→ No hardlinks for dir.



Symbolic links (soft links)



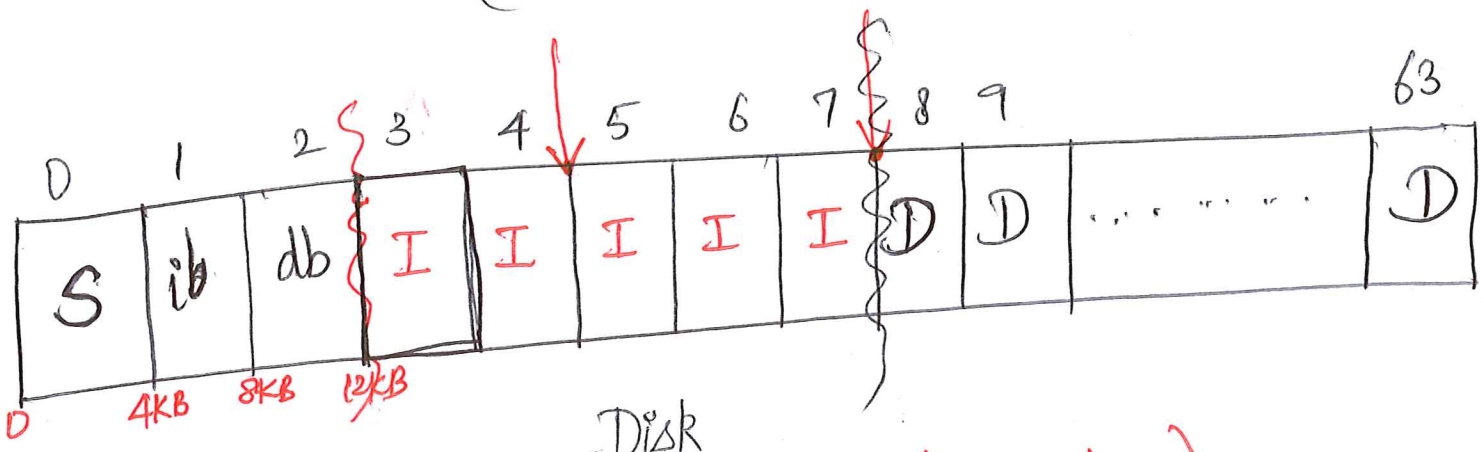


> ln -s file1 soft

File System Implementation

FS → array of block (Disk)

VSFS



Disk

1 block = 4 KB (8 sectors)

1 sector = 512 bytes

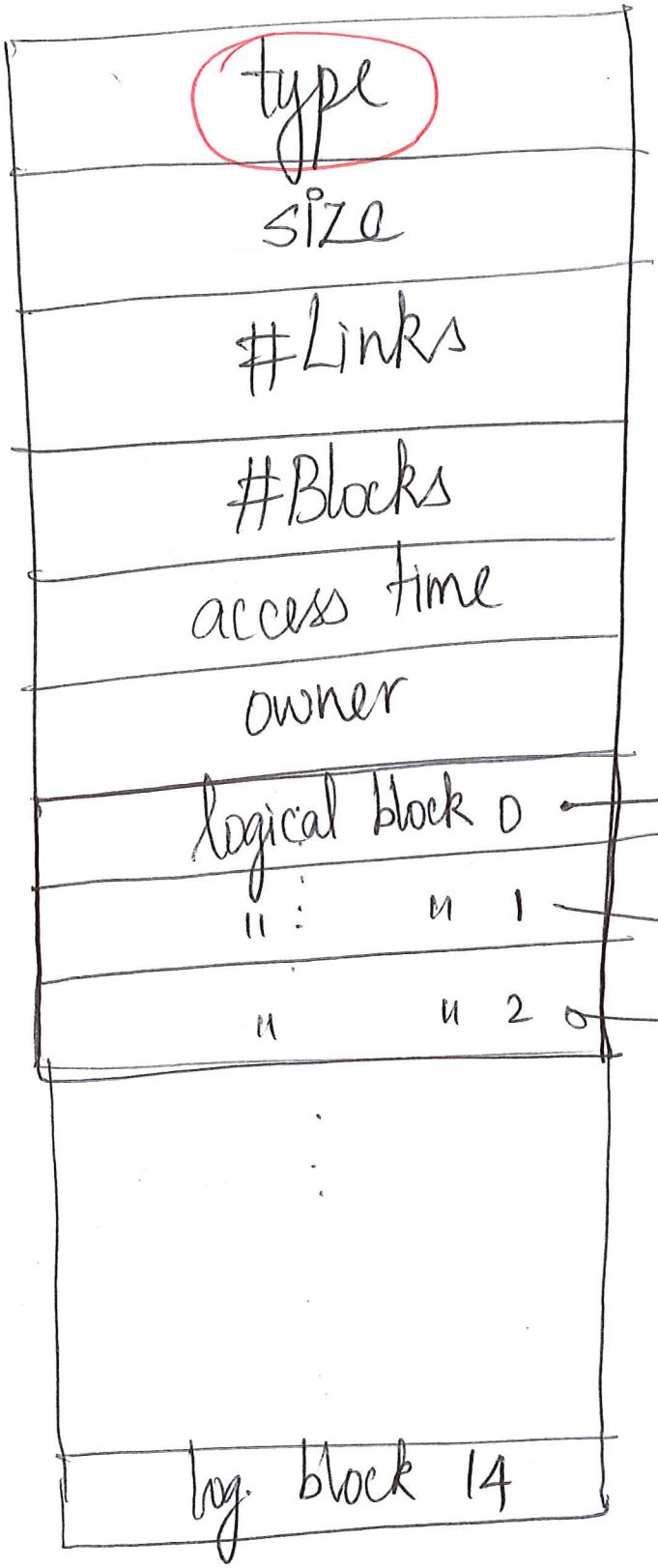
Size of HDD = 64 X 4 KB

$= 2^6 \times 2^2$ KB = 256 KB

① How to store user data?

"DATA blocks"

(2) How to store per-file metadata?



Inode

size = 128 or 256 bytes ✓

max # of files = 5

1 block → $\frac{4KB}{256} = \frac{2^{12}}{2^8} = 2^4 = 16$

logical block 0 → Data block 9

" 1 → Data block 21

" 2 → Data block 30

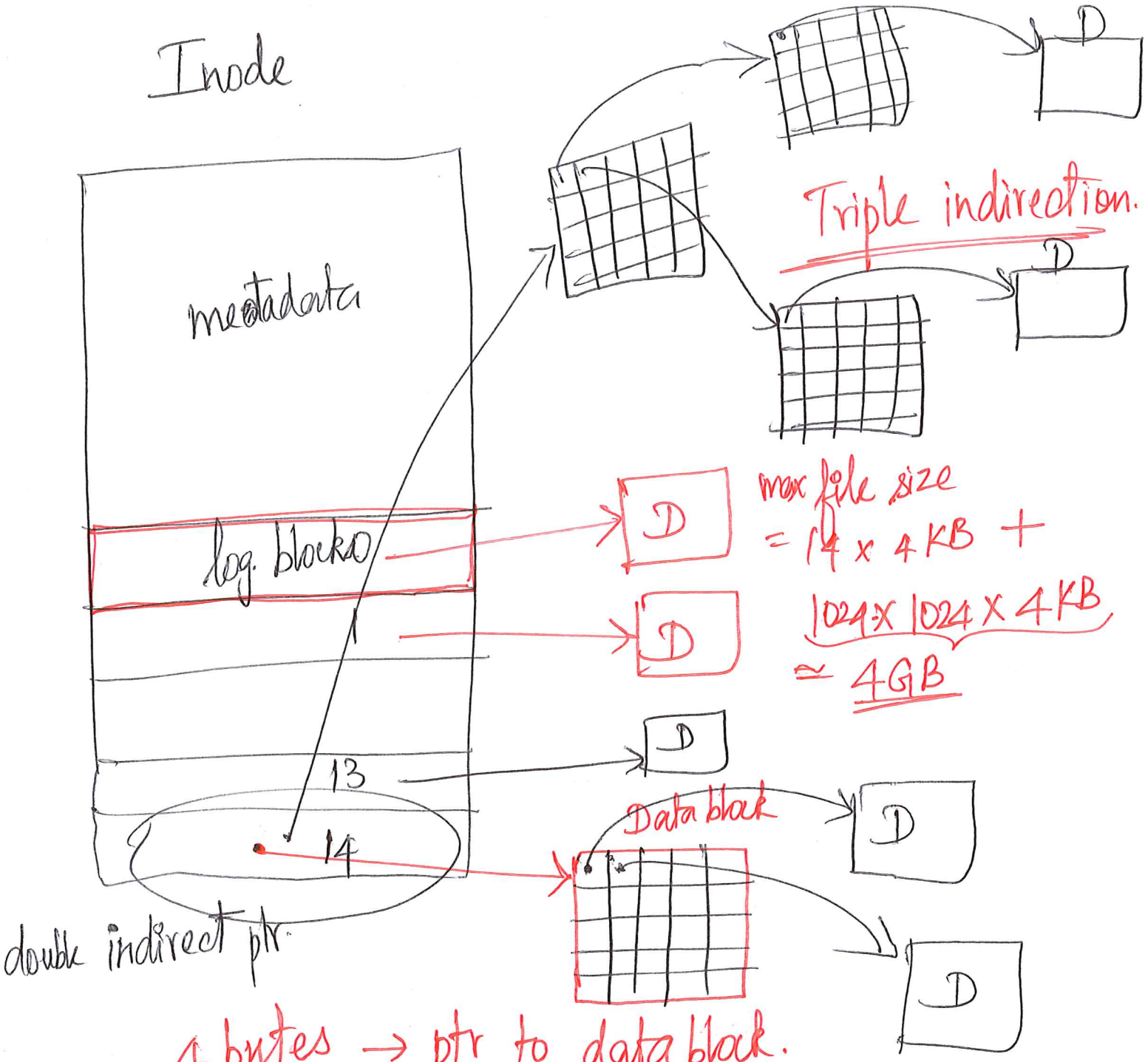
1 block → 16 inodes

5 blocks → 80 inodes

80 files

max. file size = 15 X 4 KB

= 60 KB



4 bytes → ptr to data block.

# of ptrs in indirect block

$= \frac{4KB}{4} = 1KB = 1024$

max file size = $14 \times 4KB + \underbrace{1024 \times 4KB}_{4MB}$

=

FS ⟶ read block 8?

$8 \times 4KB = 32KB$

$$\text{sector \#} = \frac{32KB}{512} = \frac{2^{15}}{2^9} = 2^6 = 64$$

sector #

FS ⟶ read inode #32?

offset from the inode start $= 32 \times 256 = 2^5 \times 2^8 = 8KB$

inode start from the beg. of disk $= 12KB + 8KB$

$= 20KB$

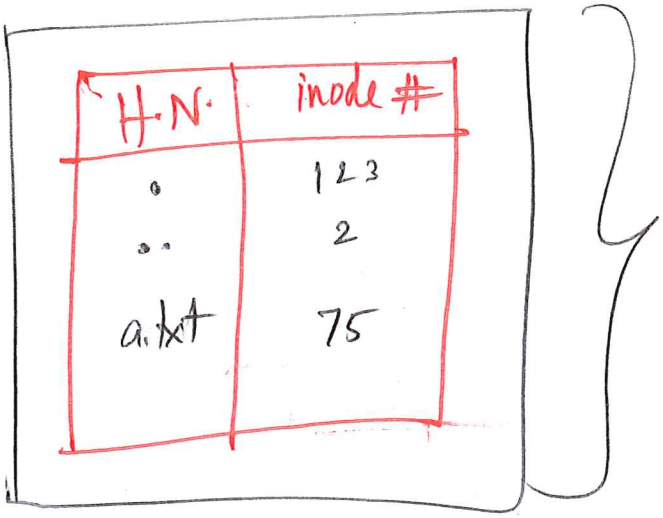
$$\text{sector \#} = \frac{20KB}{2^9} = \frac{20 \times 2^{10}}{2^9}$$

$= 40$ sector #

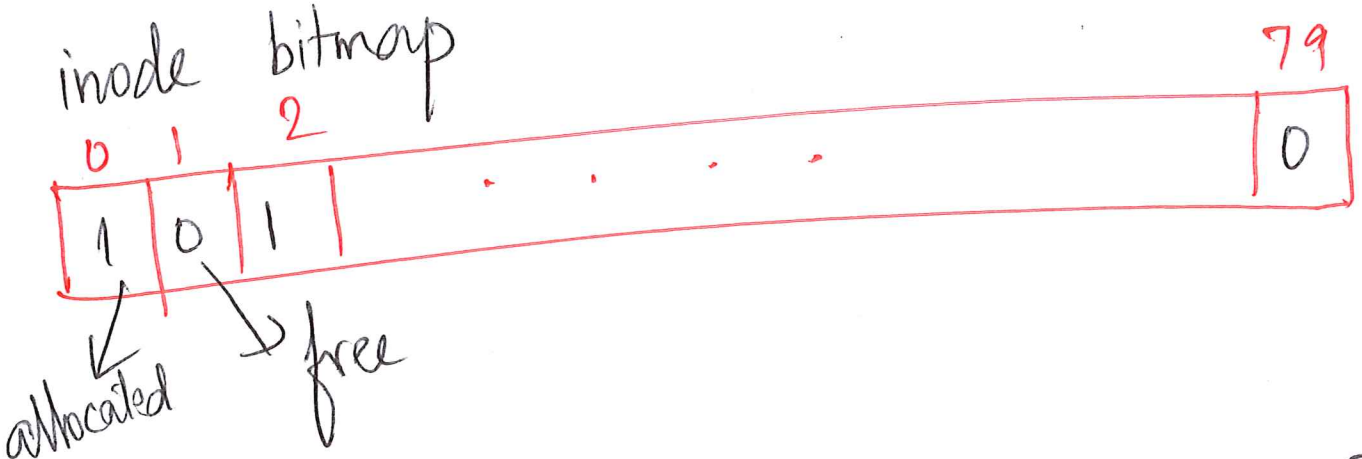
③ Directories ?

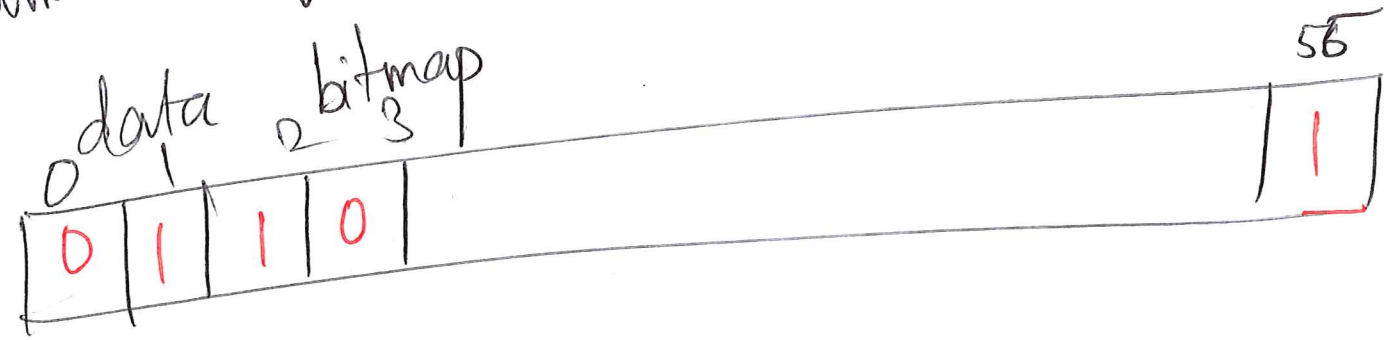
reuse inodes — type = d

Data block



④ How to track free inodes & data blocks?

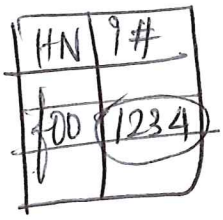


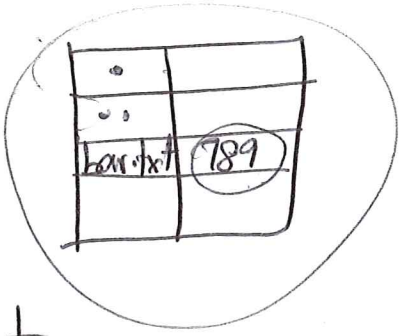


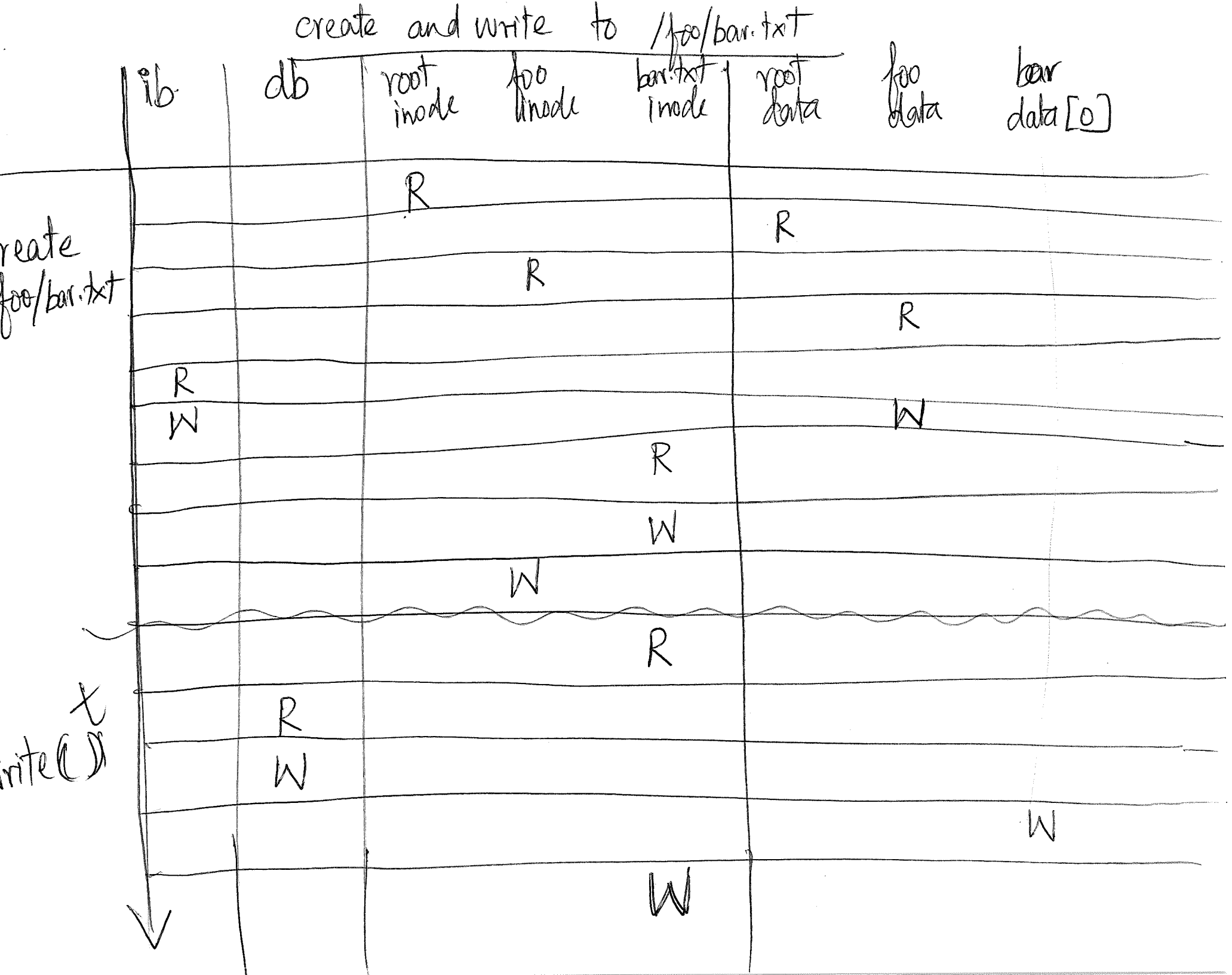
Read

/foo/bar.txt ⟶ 12 data blocks

1. Read inode of / (root)
2. Read data of / (root)
3. Read inode of foo.
4. Read data of foo.
5. Read inode of bar.txt
6. Read data[0] of bar.txt.
7. Write inode of bar.txt






create and write to /foo/bar.txt
ib
db
root inode
foo inode
bar.txt inode
root data
foo data
bar data[0]
reate
foo/bar.txt
R
R
R
R
R
W
W
R
W
R
R
W
W
t
rite()