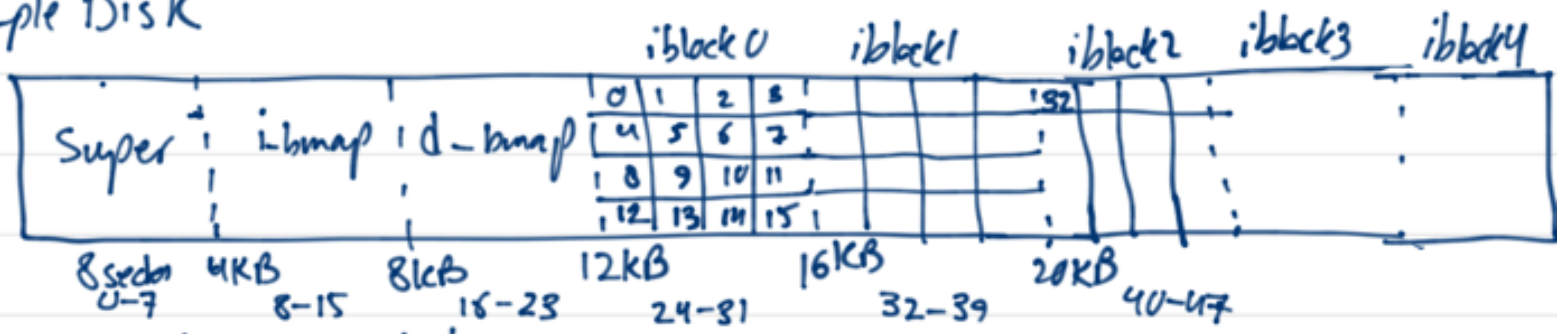
How does it look on the disk?

Disks are secter addressable (not byte addressable)

a set of sectors → Block

Simple Disk




each inode = 256 byte

each block = 4KB ⇒ 5×4kB = 20KB inode data ⇒ 80 inodes

- we want to access inode #32 ⇒ 32×256 = 8KB

8 KB + 12kB = 20KB. Considering secters of size 512

Disk read ⇒ $\frac{20 \times 1024}{512}$ = secter 40 $5 \times 8 = 40$ secters

Read secter 40

Remzi What is a directory? what is an inode?
How are files accessed on a FS?
Directory Organization : entry name, inode num (pointer to the blocks of file.)

where is it? usually treated as a special type of file!

⇒ directory has an inode with the type set as "directory"

How to implement it?

1) Linear List: - simple to program
- not efficient

- directories are searched quiet often (linear search: costly) → keeping a sorted version is helpful, but maintaining a sorted list is not cheap.
- Some systems cache directories to speed up the process. (in RAM rather than disk)
- Balanced tree is a better option.

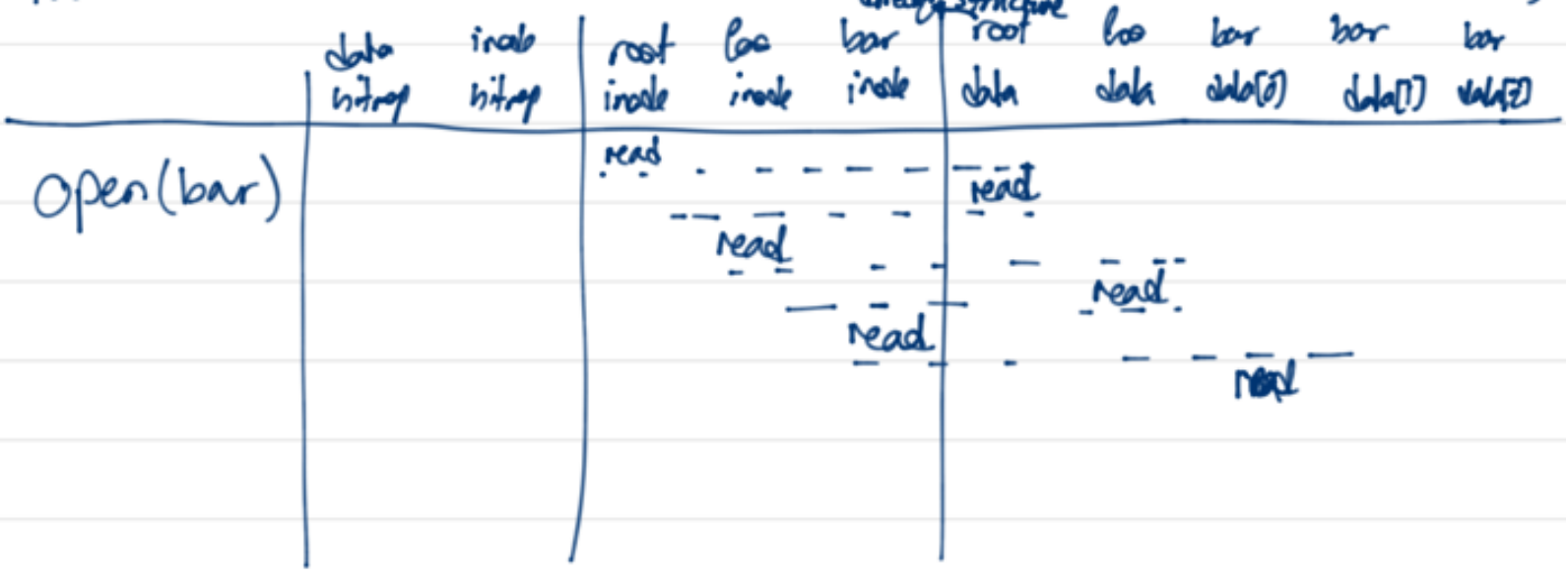
2) Hash Table: - lower directory search time.
- Collisions
- fixed size

Example: assuming that the file system has been mounted & the superblock is already in memory (everything else, inodes, directories on disk)

- we want to open a file /foo/bar read it & then close it

```
open("/foo/bar", O_RDONLY)
```

each directory has an inode itself, usually the root directory has a well-known inode # (2 in Linux) ⟹ after accessing inode 2 we have to load the directory of foo to read its inode val then read bar file inode from data within foo (data block)




⟹ Linear vs Hashed ⟹ faster access

# Important challenge: Allocation Methods:

1) Contiguous allocation: → Sequential, → direct.

Compact ←

Drawbacks: - waste of space (external fragmentation)

Extent-based (segmentation) - waste of space (internal " ) ? user has to estimate size of the file or for a dynamic size file ⇒ overestimate allocate disk blocks in extents. ⇒ waste.

Assuming sequential access to file blocks and to avoid disk head movements

dirEntry entry:


2) Linked allocation (only sequential)

Unless (FAT) → cached

overhead: address of next block

sequential ✓

direct access X

→ store indexes in a different DS:

File Alloc. Table


- how to add a block to a file?

- how does sequential access is affected? - more disk head movements
  - unless cached