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 ⇒ waste.
allocate disk blocks in extents.

File.txt	start	length
	0	2
⋮		

2) Linked allocation (only sequential)

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

Unless (FAT) → cached

overhead: address of next block

sequential ✓

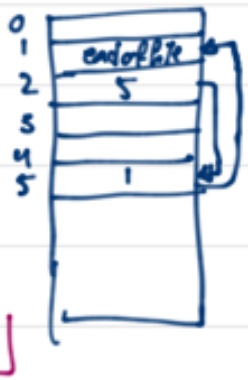
direct access X

→ store indexes in a different DS:
File Alloc. Table

dirEntry entry:

$$\frac{400GB}{4KB} = \frac{100 \times 2^{30}}{2^{10}} = 100 \times 2^{20} > 1{,}000{,}000$$

OK 100MB



- how to add a block to a file?
- how does sequential access is affected? - more disk head movements
  - unless cached

Linked Allocation

	start	end
file.txt	2	8

disadv:

1) { Sequential ✓
   Random? read the file sequentially to get to the desired position.

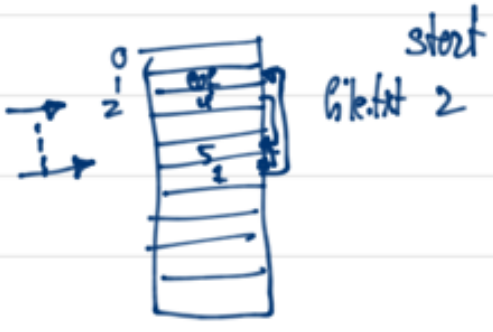
2) pointers occupy space in each block
   ↳ cluster of blocks (every 4 blocks) ⇒ internal frag.

3) Reliability: what if a pointer is corrupted?
   ↳ doubly linked list.
   ↳ or keep relative block number, filename in each block ⇒ recover.

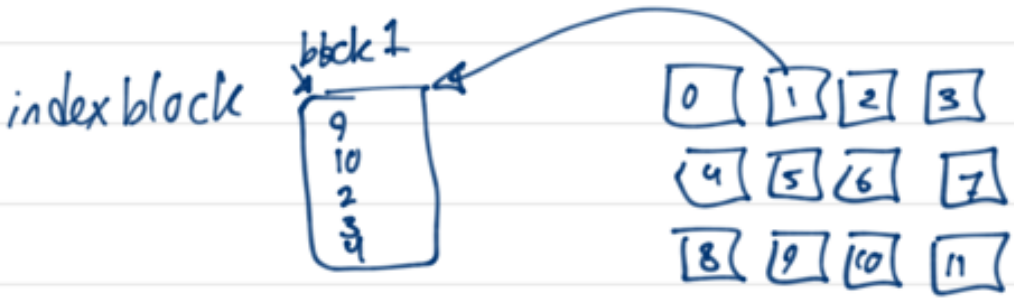
⇓

How can we improve this?

File Allocation Table (FAT): Keep the links (LL) in a separate Data Structure. Remember disk head movements are costly ⇒ we have to keep this table cached.



Putting FAT & Linked Allocation together ⟹ Indexed Allocation



Similar to paging: Logical block addr (i-th) ⟶ physical block

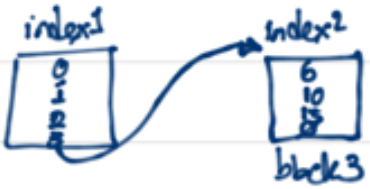
- Direct access ✓
- wasted space (small files) wastes a whole block for a couple of indexes!!!
- How big should an index block be?

Small — not have enough pointers for a large file

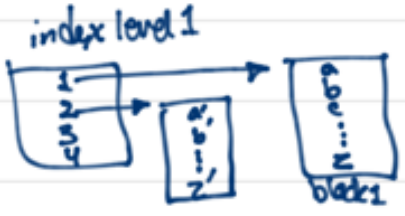
Large — wastes a lot of space for small files

How to deal with this?

1) Linked indexes: index block can be linked to another one



2) Multilevel index



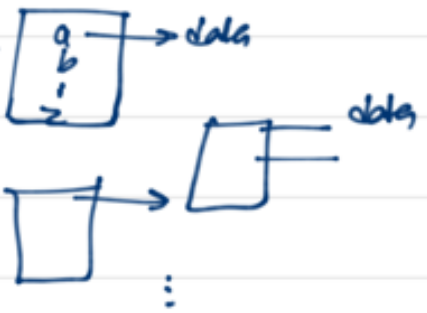
4KB blocks ⟹ 1024 4byte pointers

2 level ⟹ $1024 \times 1024 = 2^{20}$

$2^{20} \times 4KB \Rightarrow 4GB$ file

3) Combined Scheme : Linux inode

12 direct blocks
1 single indirect
1 double indirect
1 triple



Free Space Management:

Free-space list: Why we didn't study this in RAM?
RAM is fast.
disk is slow. } but similar ideas apply

1) Bit vector: allocated → 0 2,3,4,5,8,9,10: free
free → 1 0011110011 1

- Simple & efficient to find the first free block or n consecutive free blocks (only & operation)

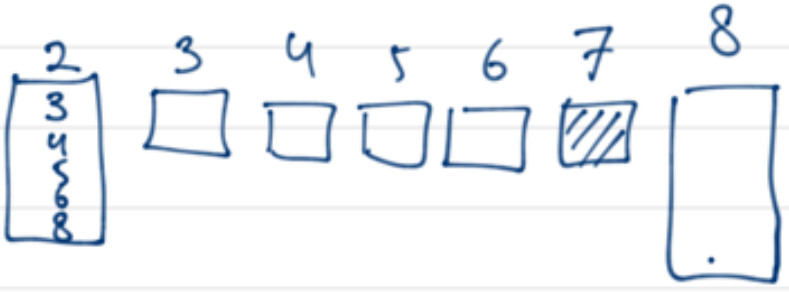
- Cons: Could be very large. 1.3 GB disk 512 block
332KB

1 TB disk 4KB blocks
⟹ 32 MB for bitmap. (Considerable)

2) Linked list: Link free blocks together.
and cache the listore.
Usually we won't traverse free blocks.

FAT: free block accounting

3) Grouping:



4) Counting: usually several contiguous blocks may be allocated or freed → keep address of first block & (n) of free contiguous blocks that follow it.