File Systems

* Old OS: before bytes, files were a sequence of records
* Files: • regular files (ASCII readable, Binary executable) • special files (Char/byte I/O, Block hard drive)
* File access: • sequential (must go in order, magnetic tape) • random (can access any part)
• Directories: files that organize a hirearchy of other files
* UNIX sees the disk as a sequence of physical blocks of fixed size. each disk can be divided into partitions each with its own file system.

Disk: [MBR | partition 1 | partition 2 | ...]
Master Boot Record
- boots the PC
- ends with a partition table

only 1 can be active at a time

Partitions: [Boot Block | superblock]
- # of blocks in FS
- FS size
- index of 1st free block
- list of free blocks
- size of inode table
- list of free inodes
- index of first free inode

Disk Allocation Layouts:

1- Contiguous: Store each file as a contiguous set of physical data blocks

adv: - quick & easy to find block holding my data
- almost no seek required for sequential data.
- excelled file reading performance

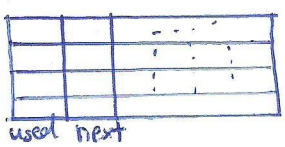
disadv: - file grows => may have to move it.
- internal fragmentation inside a single block
- external fragmentation between files

2- Linked List: the first word of each block contains the address of the next block

adv: - no more variable sized file allocation
- no external fragmentation.

disadv - terrible performance in some cases
- long seek time
- ammount of data per block is no longer a power of 2

3- Linked List with File Allocation Table (FAT): one entry per physical block
The table is loaded into memory on fs boot.



adv: - size of block is full (of data)
- random access much easier since entire chain is in the memory.

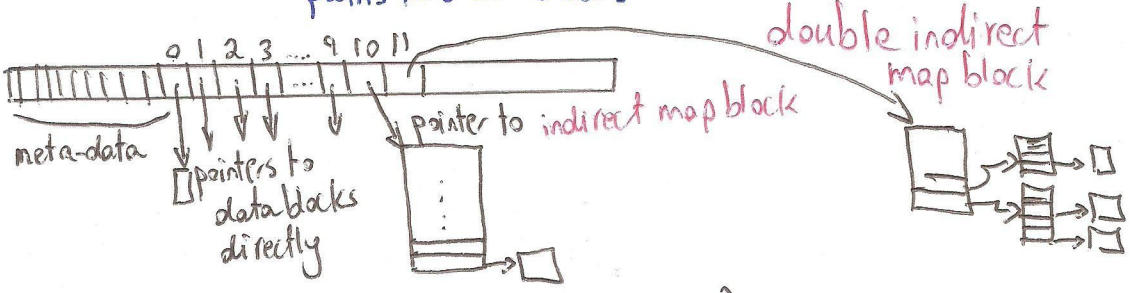
disadv - entire table must be in memory at all times.

4 - inodes: (UNIX) a data structure associated with the file.

open file => store in inode and release when closed.

The inode contains – management (metadata)
– paths to data blocks

- ex)



1 KB block & 4B pointer (entry)

=> 1KB/4 = 256 entries in map block

0 → 9 : 1kB each => 10 kB

if file > 10 kB => use 10

10 : 256 entries => 256 kB

if file > 266 kB => use 11

11 : $(256)^2$ kB

12 : $(256)^3$ kB

=> if we go up to 12 we can store

10 kB + 256 kB + 256^2 kB + 256^3 kB $\approx$ 16 GB

max file size

- Effective size of file = # map blocks + # data blocks + size(inode)