File Systems

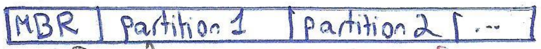
*Old OS: before bytes, files were a sequence of records

*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

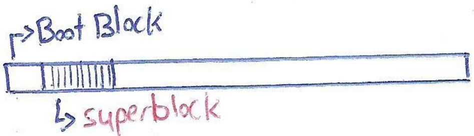
• Directories: files that organize a hirearchy of other files

*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:


Master Boot Record
- boots the PC
- ends with a partition table

only 1 can be active at a time

Partitions


- #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

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

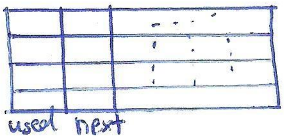
disadvs
- file grows => may have to move it.
- internal fragmentation inside a single block
- external fragmentation between files

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

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

disadvs - terible 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.



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

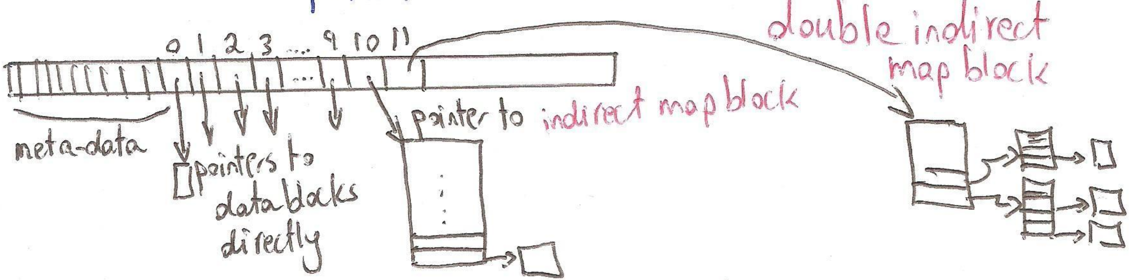
disadvs - entire table must be in memory at all times.

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)


0 1 2 3 ... 9 10 11
double indirect map block
pointer to indirect map block
meta-data
pointers to data blocks directly

1 KB block & 4B pointer (entry)

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

$0 \to 9$: 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

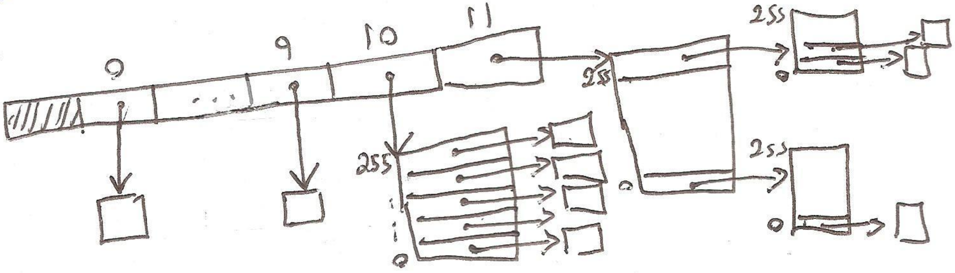
=> if we go up to 12 we can store $10kB + 256KB + 256^2 KB + 256^3 kB \approx 16 GB$ max file size

• Effective size of file = (# map blocks x block size) + (# data blocks x block size) + size(inode)

Note: . map block: blocks that point to other blocks

. data block: blocks that hold data.

ex)



data blocks: $\overset{0-9}{10} + \overset{10}{256} + \overset{11}{256^2}$

map blocks: $10 + (256+1) + (256^2+256+1)$