FILESYSTEM

↳ modulo dell' S.O. per estrarre meccanismi memorizzazione di massa

FILE → insieme (collezione) di info omogenee memorizzate in modo indip. dal tipo di dispositivo fisico

↳ sono associati attributi: nome, tipo, locaz, dim, protez., ora e data, proprietà

→ sono possibili più azioni: 1→ creaz, allocaz. descrittore su filesystem;
2→ scrittura, aggiunta di dati a file esistente;
3→ lettura, preleva dati;
4→ riposiz., sposta puntatori lett e scritt;
5→ cancellazione, deallocando spazio;

→ esiste una **tabella dei file in uso**: → OPEN, aggiungo file a tabi
↳ CLOSE, lo rimuovi;

PROTEZIONE DATI → 1. da danni fisici → SOL: backup e mirroring
2. da accessi impropri → SOL: def. **politica di accesso** → accesso controllato

liste di accesso ← in base a chi vuole accedere e a cosa vuole fare

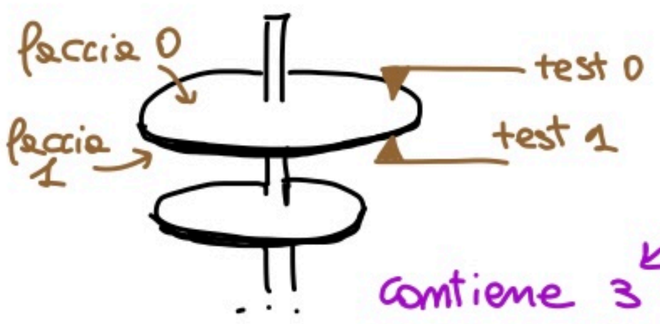
SOL: creo classi: owner, group, others ← PROB: possono essere molto pesanti

Cntr access list → 3 gruppi per own, group, other → a loro volta ridivisi in base a R, W, X (read, write, execute)

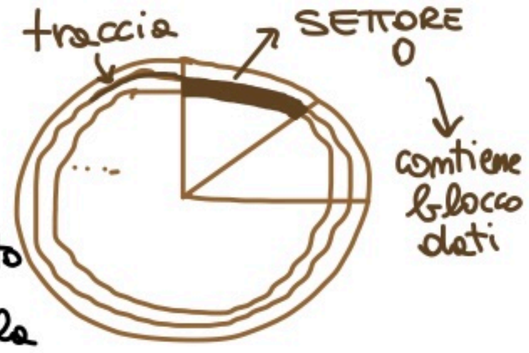
STRUTTURA FISICA → file org. in seq. di blocchi di **lunghezza fissa** (e dipende da disp)

NON PUÒ ESSERE ELIMINABILE ← **PROB FRAMM INTERNA**: rischio di "buttare via" delle parti di memoria siccome i blocchi hanno capienza maggiore grandezza del file (alloc. mem. fissa)

↳ **HARD DISK**: dischi impilati coassiali, a 2 facce con testina R/W a testa;



→ ogni faccia: – tracce concentriche
– divisa in settori
– 1 blocco dati



↳ all'interno del **settore** vi è uno strato magnetico ricoperto che a seconda della polarità determina valore logico bit

contiene 3 sez. logiche: • info allineam;
• dati veri e propri;
• codice rilev. errori;

(vede il fsys a blocchi)

STRUTTURA LOGICA → gestisce rappr. valori logici tram filesystem formato da: volume, directory, file; (file → contiene dati; volume → disco logico; directory → indice gruppo file)

directory fondamentale nella strutt. filesys → è in grado di svolgere ogni funz. su file (tram. descrittore)

→ **DIRECTORY**, per gestirla senza rallentare sistema divido in 2: **root** (gestione affidata a amministratore), la principale contiene directory per ogni elem;
home, contiene file di un utente;

↳ per accedere ai file o uso nome direttamente o pathname di accesso
↳ per file condivisi si usa **metodo di riferimento**, con puntatore a descrittore;

→ il **filesystem** si occupa di visionare nel loro insieme: file, directory, link, attr e politiche accesso operaz, mapping logico su fisico
↳ struttura a livelli con accesso da alto verso basso

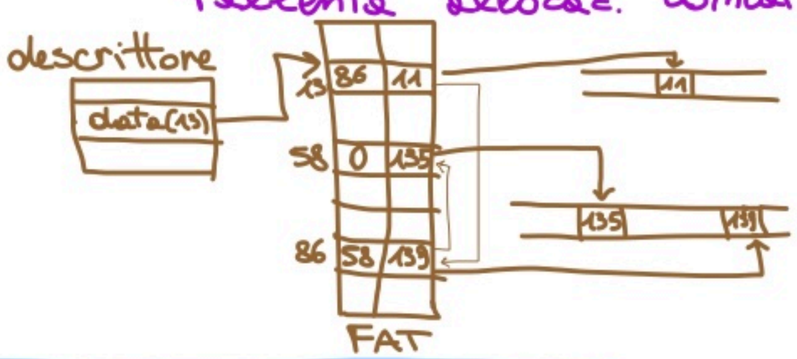
ALLOCAZIONE CONTIGUA → blocchi adiacenti (2 compi: blocco, n°) con dim fissa;
↳ facile e rapido accesso ma 'brucia' facilmente spazio
(b,s) → b base ovvero blocco inizio, s size file
↳ nel tempo crea zone inutiliz. e inaccessibili della memoria;
↳ **FRAMMENTAZIONE ESTERNA** ovvero il blocco contiguo crea buchi inutili
3 politiche risolut:
1 → first-fit;
2 → best-fit;
3 → worst-fit;
↳ **SOL**: o riscrivo disco eliminando spazi vuoti;
memorizzo in blocchi differenti;

DESCRITORE FILE → indica: sez principale, dim della sez, base extent (parte salvata altrove), dim extent;

ALLOCAZIONE CONC → Si crea una lista tram extent: il descrittore conosce 1° e ultimo blocco e ogni blocco punta al next;
nel blocco: 

↳ accesso meno efficiente perché bisogna passa tutti i rif. in pila.
↳ posso raggruppare blocchi in **cluster** (ovvero gruppi);
↳ maggior framm int, riduz spazio, miglior prestaz.

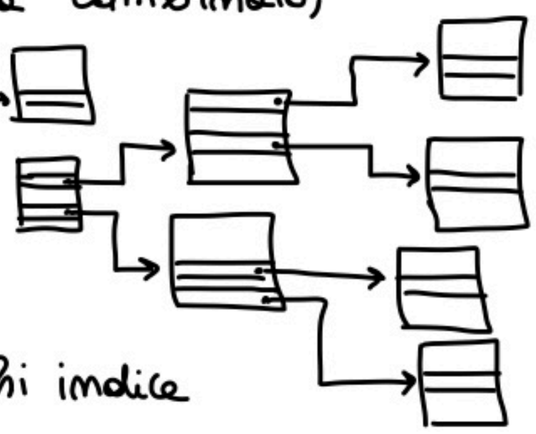
→ per proteggere i dati si usa **FAT**: file allocation table, contiene per ogni blocco l'indice al succ;
in questo modo il descrittore deve conosere solo il 1° blocco;
rallenta allocaz. concat



ALLOCAZIONE INDICIZZATA → utilizza un blocco indice che raggruppa tutti i rif. ai cluster ed è parte stessa del file e del descrittore
↳ elimina framm. esterna e tram accesso casuale ai blocchi è più efficiente
↳ però richiede uno spazio ai rif. maggiore, quindi se un file ha pochi riferimenti è una tecnica ridondante

↳ nel caso di gestione di file di dim variabile:
1 → schema concat;
2 → schema multilivello;
3 → schema combinato;

1 → più blocchi indice concatenati 

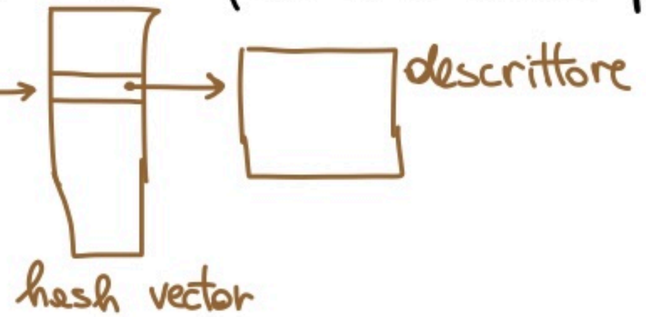
2 → come sopra ma a più livelli (tipo albero) 1°, 2°, 3° 

3 → metà rif a blocchi file e metà a blocchi indice


REALIZZAZIONE DIRECTORY → metodo più efficace utilizza una funzione hash perfetta

↳ se la funz. non è perfetta (no clash e no 2 chiavi per lo stesso indice) → dal vettore dei valori partono liste concatenate che vanno scorse fino a trovare la chiave giusta. (più lento)

foo → index=hash(name) → hash vector → descrittore



(i-node → descrittore per amb UNIX, LINUX a schema indic. combinato)

FORMATTAZIONE → cambio dim del blocco visto da fsys logico (ovviamente non fisico)

→ **hard link** e **soft link** ?

hard link: copia di un altro file con stesso i-node del file target
↳ ln pathorig. pathsimb.
↳ sostituisce direttamente nel blocco fsys del target

soft link: ↳ link simbolico contenente un collegamento a un altro file non contiene dati
↳ ln -s path_orig path_simb
↳ crea i-node rif a un blocco che conterrà la directory
↳ per l'accesso ci saranno 2 livelli di blocchi