## Realizzazione del *file system* – 1

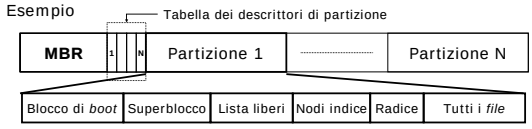
- I *file system* (FS) sono memorizzati su disco
  - I dischi possono essere **partizionati**
  - Ogni partizione può contenere un FS distinto
- Il settore 0 del disco contiene le informazioni di inizializzazione del sistema
  - ***Master Boot Record***
  - L'inizializzazione è eseguita dal BIOS
  - L'**MBR** (ampio 512 B = 1 **settore**) specifica le partizioni presenti e identifica quella attiva
  - Il primo blocco di informazione di ogni partizione contiene le sue specifiche informazioni di inizializzazione (***boot block***)



## Realizzazione del *file system* – 2

- L'unità informativa su disco è il **settore**
- I dischi vengono però letti e scritti a **blocchi** (*cluster* per Microsoft!) per velocizzare le operazioni
  - 1 blocco = N settori (N ≥ 1)
  - Rischio consapevole di frammentazione interna
- La struttura interna di partizione è specifica del FS

Esempio






## Realizzazione dei *file* – 1

- A **livello fisico** un *file* è un insieme di blocchi di disco
  - Occorre decidere quali blocchi assegnare a quale *file* e come tenerne traccia
- 3 strategie di allocazione di blocchi a *file*
  - Allocazione contigua
  - Allocazione a lista concatenata (*linked list*)
  - Allocazione a lista indicizzata

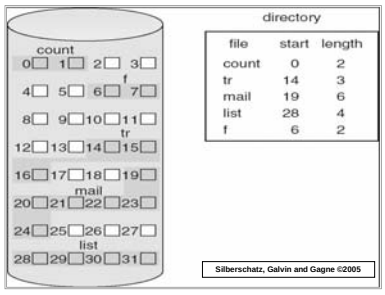


## Realizzazione dei *file* – 2

- **Allocazione contigua**
  - Cerca di memorizzare i *file* su blocchi **consecutivi**
  - Ogni *file* è descritto dall'indirizzo del suo primo blocco e dal numero di blocchi utilizzati
  - Consente sia accesso sequenziale che diretto
  - Un *file* può essere letto e scritto con un solo accesso al disco
    - Ideale per CD-ROM e DVD
  - Ogni modifica di *file* comporta il rischio di frammentazione esterna
    - Ricompattazione periodica molto costosa
    - L'alternativa richiede l'utilizzo dei gruppi di blocchi liberi
      - Mantenere la lista dei blocchi liberi e la loro dimensione
        - Possibile ma oneroso
      - Conoscere in anticipo la dimensione massima dei nuovi *file*
        - Stima difficile e rischiosa



## Allocazione contigua

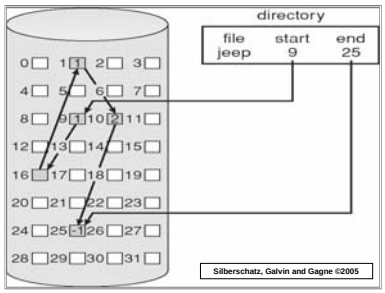





## Realizzazione dei *file* – 3

- **Allocazione a lista concatenata**
  - *File* come lista concatenata di blocchi
  - *File* identificato dal puntatore al suo primo blocco
    - Per alcuni S/O anche dal puntatore all'ultimo blocco del *file*
  - Ciascun blocco di *file* deve contenere il puntatore al blocco successivo (o un marcatore di fine lista)
    - Questo sottrae spazio ai dati
  - L'accesso sequenziale resta semplice da realizzare
    - Ma per *file* grandi può richiedere molte operazioni su disco
  - Accesso diretto molto più complesso e oneroso
    - Raggiungere qualunque posizione logica costa molte operazioni su disco
  - Un solo blocco guasto corrompe l'intero *file*

## Allocazione a lista concatenata






## Realizzazione dei *file* – 4

- **Allocazione a lista indicizzata**
  - I puntatori ai blocchi sono memorizzati in strutture dedicate
    - Ciascun blocco contiene **solo dati**
  - Il *file* è descritto dall'insieme dei puntatori ai suoi dati
  - 2 strategie di organizzazione
    - Forma tabulare (**FAT**, *File Allocation Table*)
    - Forma indicizzata (nodo indice, *i-node*)
  - Non causa frammentazione esterna
  - Consente accesso sequenziale e diretto
  - La dimensione massima di ogni nuovo *file* non deve essere nota a priori
    - Il *file* può crescere a piacere

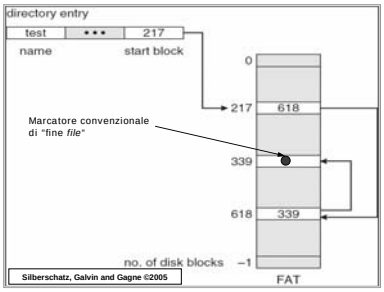


## Allocazione a lista indicizzata – 1

- ***File Allocation Table***
  - La scelta progettuale di *MS-DOS*
    - Base del FS storico di *MS Windows*
- **FAT** = tabella ordinata di puntatori
  - Un puntatore $\forall$ blocco (*cluster*) del disco
  - La tabella cresce con l'ampiezza della partizione
- La porzione di FAT relativa ai *file* in uso **deve** sempre risiedere interamente in RAM
- Consente accesso diretto ai dati
- Un *file* è una catena di indici



## Struttura **FAT**






## Allocazione a lista indicizzata – 2

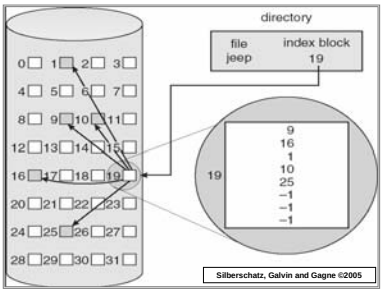
- **Nodi indice** (UNIX $\rightarrow$ GNU/Linux)
  - Una struttura indice (*i-node*) $\forall$ *file* con gli attributi del *file* e i puntatori ai suoi blocchi
    - L'*i-node* è contenuto in un blocco dedicato
  - In RAM una tabella di *i-node* per i soli *file* in uso
    - La dimensione massima di tabella dipende dal massimo numeri di *file* apribili simultaneamente
      - Non più dalla capacità della partizione
  - Un *i-node* contiene un numero limitato di puntatori a blocchi
    - Quale soluzione per *file* composti da un numero maggiore di blocchi?



## Allocazione a lista indicizzata – 3

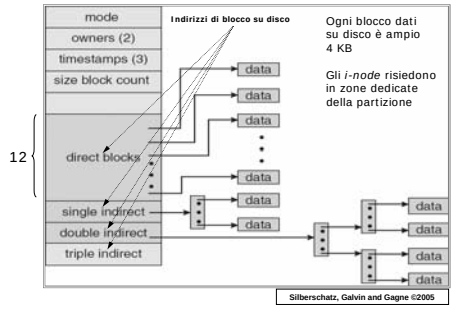
- **Nodi indice** (UNIX $\rightarrow$ GNU/Linux)
  - *File* di piccola dimensione
    - Gli indirizzi dei blocchi dei dati sono ampiamente contenuti in un singolo *i-node*
      - Tipicamente con un po' di frammentazione interna
  - *File* di media dimensione
    - Un campo dell'*i-node* punta a un nuovo blocco *i-node*
  - *File* di grandi dimensioni
    - Un campo dell'*i-node* principale punta a un livello di blocchi (*i-node*) intermedi che a loro volta puntano ai blocchi dei dati
  - Per *file* di dimensioni ancora maggior basta aggiungere un ulteriore livello di indirezione

## Struttura a nodi indice






## FS in UNIX v7



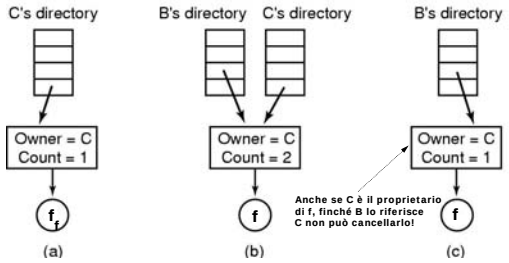



## Realizzazione dei *file* – 5

- **Gestione dei *file* condivisi**
  - Come preservarne la consistenza senza costi eccessivi
    - **Non** porre blocchi di dati nella *directory* di residenza del *file*
    - ∀ *file* condiviso porre nella *directory* remota un ***symbolic link*** verso il *file* originale
      - Esiste così **1 solo** nodo indice (*i-node*) del *file* originale
      - L'accesso condiviso avviene tramite cammino sul FS
    - Altrimenti si può porre nella *directory* remota il puntatore diretto (***hard link***) al nodo indice (*i-node*) del *file* originale
      - Più possessori di descrittori dello stesso *file* condiviso
      - Un solo proprietario effettivo del *file* condiviso
      - Il *file* condiviso **non può** più essere distrutto fin quando esistano suoi descrittori remoti anche se il suo proprietario avesse inteso cancellarlo



## Gestione della condivisione



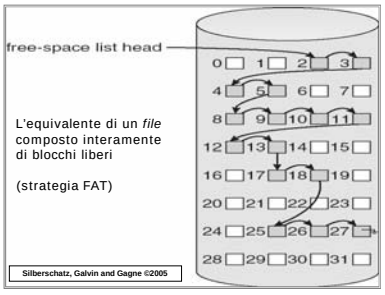



## Realizzazione dei *file* – 6

- **Gestione dei blocchi liberi**
  - **Vettore di *bit*** (*bitmap*) dove ogni *bit* indica lo stato del corrispondente blocco
    - 0 = libero
    - 1 = occupato
  - **Lista concatenata** di blocchi sfruttando i campi puntatore al successivo
    - Gli indici di blocco liberi sono nella FAT e rappresentano un *file* fittizio
  - Oppure riempiendo blocchi dedicati con gli indici corrispondenti
    - Un blocco da 1 KB può contenere fino a 256 indici da 4 B ciascuno



## Lista concatenata dei blocchi liberi



## Realizzazione delle *directory* – 1

- Per ogni suo *file* la *directory* specifica
  - Nome
  - Collocazione
  - Attributi
- *File* e *directory* risiedono in aree logiche **distinte**
- Conviene minimizzare la complessità gestionale della struttura interna di *directory*
  - Meglio una struttura a lunghezza **fissa**
    - Per quanto il suo contenuto sia di ampiezza variabile!
  - [`Nome + attributi`] oppure
  - [`Nome + puntatore a nodo indice con attributi`]
    - Frammentazione interna trascurabile per nomi di *file* fino a 8 caratteri + 3 di estensione
    - Problema più serio per nomi lunghi



## Realizzazione delle *directory* – 2

- La ricerca di un *file* correla il nome (stringa ASCII) alle informazioni necessarie all'accesso
  - Nome e *directory* di appartenenza del *file* sono determinati dal percorso indicato dalla richiesta
- La ricerca **lineare** in *directory* è di realizzazione facile ma di esecuzione onerosa
- La ricerca mediante tabelle *hash* è più complessa ma più veloce
  $F$(nome) = posizione in tabella → puntatore al *file*
- Si può anche creare in RAM una *cache software* di supporto alla ricerca

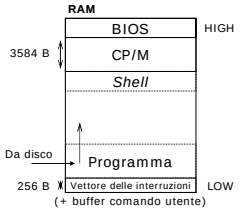


## Cenni storici

- CP/M (1973-1981)
- MS-DOS & Windows 95 (1981 → 1997)
- Windows 98 (1998-1999)
- UNIX v7 (1979)



## **CP/M** (*Control Program for Microcomputers*)




- BIOS minimo
  - Massima portabilità
- Sistema multiprogrammato
  - Ogni utente vede solo i propri *file*
- *Directory* singola con dati a struttura fissa
  - In RAM solo quando serve
- *Bitmap* per blocchi liberi memorizzata in RAM
  - Distrutta a fine esecuzione
- Nome *file* limitato a 8 + 3 caratteri
  - Dimensione inizialmente limitata a 16 blocchi da 1 KB
    - Puntati direttamente dalla *directory*



## **MS-DOS** (*Microsoft Disk Operating System*)

- **Non** multiprogrammato
  - Ogni utente vede **tutto** il FS
- FS **gerarchico** senza limite di profondità e **senza condivisione**
  - Fino a 4 partizioni per disco (C: D: E: F: )
- *Directory* a lunghezza variabile con *entry* di 32 B
  - Nomi di *file* a 8+3 caratteri (normalizzati a maiuscolo)
- Allocazione *file* a lista (FAT)
  - **FAT-X** per **X** = numero di *bit* per indirizzo di blocco ($12 \le$ **X** $< 32$)
  - Blocchi di dimensione multipla di 512 B (1 settore)
  - **FAT-16** *: File* e partizione limitati a 2 GB
    - $2^{16}$ = 64K (puntatori a) blocchi di 32 KB ciascuno = 2 GB
  - **FAT-32 :** blocchi da 4 ÷ 32 KB e indirizzi da 28 *bit* (!)
    - Perché 2 TB è il limite **intrinseco** di capacità per partizione
      - $2^{32}$ settori (*cluster*) da 512 B = $2^2 \times 2^{30} \times 2^9$ B = $2^{41}$ B = 2 TB
      - $2^{28}$ blocchi da **8** KB = $2^8 \times 2^{20} \times 2^3 \times 2^{10}$ B = $2^{41}$ B = 2 TB



## Il FS in MS-DOS 7.0

Non più di FAT-16!

Struttura di *directory entry* (32 B)

| | | | |
|---|---|---|---|
| 1. Nome *file* | 8 B | 5. Ora modifica | 2 B |
| 2. Estensione *file* | 3 B | 6. Data modifica | 2 B |
| 3. Attributi | 1 B | 7. Indice I blocco | 2 B |
| 4. Riservati | 10 B | 8. Dimensione | 4 B |

*(unsigned)*
5 *bit* × ore [0-23]
6 *bit* × minuti [0-59]
5 *bit* × ~2 secondi [0-29]

*(unsigned)*
7 *bit* × anno+1980 [-2107]
4 *bit* × mese [1-12]
5 *bit* × giorno [1-31]

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 8 | 3 | 1 | 10 riservati per sviluppi futuri | 2 | 2 | 2 | 4 |

Usato per Windows 98 (FAT-32, orario accurato, nomi *file* lunghi e *case sensitive*)

# Il FS in MS-DOS 7.1 – 1




# Il FS in MS-DOS 7.1 – 2

- L'utilizzo dello spazio riservato (10 B) ha permesso importanti miglioramenti nella espressività della *directory*
  - Nomi lunghi
    - *Case sensitive* e con estensioni variabili
  - Indice di blocco su 4 B
  - Attributi più significativi
- Al costo di maggior complessità nella sua gestione *software*

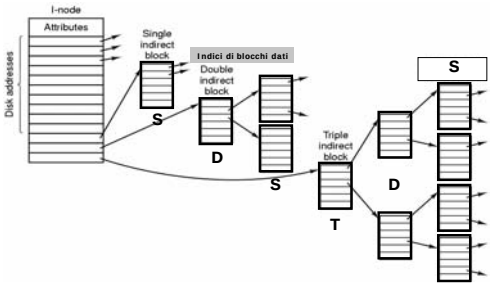


# Il FS di **UNIX** v7 – 1

- Concepito e realizzato tra il 1969 e il 1979 da Ken Thompson e Dennis Ritchie
  - Struttura ad albero con radice e condivisione di *file*
    - Grafo **aciclico**
  - Nomi di *file* fino a 14 caratteri ASCII (escluso /)
  - *Directory* contiene nome *file* (14 B) e puntatore (2 B) al suo *i-node* descrittore
    - Max $2^{16}$ *i-node* distinti
      - Max 64 K *file* per FS
  - L'*i-node* (64 B) contiene gli attributi del *file*
    - Incluso il contatore di *directory* che puntano al *file* tramite un *link* di tipo *hard*
      - Se contatore = 0, il nodo e i blocchi del *file* diventano liberi

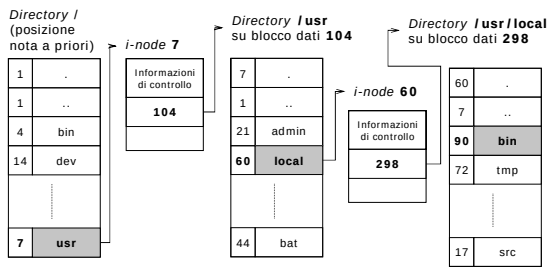


# Il FS di **UNIX** v7 – 2






# Il FS di **UNIX** v7 – 3




Esecuzione parziale del comando "cd /usr/local/bin/"



# *File system* su CD-ROM

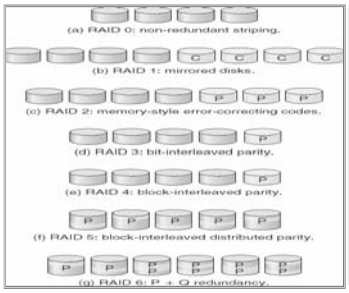
- **ISO 9660**
  - Supporta fino a $2^{16}-1$ dischi partizionabili
  - Dimensione di blocco 2 ÷ 8 KB
  - *Directory* a struttura variabile internamente ordinate alfabeticamente
  - FS limitato a 8 livelli di annidamento
    - E nomi di *file* "corti"
- **Rock Ridge**
  - Estensione di ISO 9660 definita dal mondo UNIX per meglio rappresentare le caratteristiche del proprio FS
    - Iniziativa resa obsoleta dall'avvento di Joliet
- **Joliet**
  - Estensione definita da Microsoft per lo stesso motivo
    - Nomi "lunghi" e annidamento profondo

# Integrità del *File System* – 1

- **Gestione dei blocchi danneggiati**
  - Via *hardware*
    - Creando e mantenendo in un settore del disco un elenco di blocchi danneggiati e dei loro sostituti
  - Via *software*
    - Ricorrendo a un falso *file* che occupi tutti i blocchi danneggiati
- **Salvataggio del FS**
  - Su nastro
    - Tempi lunghi, anche per incrementi
  - Su disco
    - Con partizione di *back-up*
    - Oppure mediante architettura RAID
      - *Redundant Array of Inexpensive Disks*
      - Oggi la I vale come "*Independent*"



# Livelli RAID




***Striping***: i dati vengono sezionati (per *bit* o *byte*) e ciascuna sezione viene scritta in parallelo su un disco

Al crescere del "livello" RAID cresce la sicurezza dei dati

**C**: alcuni dischi sono destinati a contenere **copia** dei dati di dischi "gemelli"

**P**: alcuni dischi (o **parti**) sono destinati a contenere codici di controllo di integrità dei dati

Silberschatz, Galvin and Gagne ©2005



# Integrità del *File System* – 2

- **Consistenza del FS**
  - Un *file* viene aperto, modificato e poi salvato
  - Se il sistema cade tra la modifica e il salvataggio il contenuto del *file* su disco diventa inconsistente
- **Consistenza dei blocchi**
  - 2 liste di blocchi con un contatore ∀ blocco
    - Lista dei blocchi in uso dei *file*
    - Lista dei blocchi liberi
  - **Consistenza**
    - Ciascun blocco appartiene a una e una sola lista
  - **Perdita**
    - Un blocco non appartiene ad alcuna lista
  - **Duplicazione**
    - Il contatore del blocco è >1 in una delle due liste



# Prestazioni del *File System*

- Una porzione di RAM viene usata come *cache software* di alcune migliaia di blocchi
  - Per ridurre la frequenza di accesso ai dischi
  - L'accesso ai blocchi localizzati in "*cache*" avviene tramite ricerca *hash*
  - La gestione richiede specifica politica di rimpiazzo blocchi
- Occorre però garantire la consistenza dei dati su disco
  - **MS-DOS**
    - I blocchi modificati vengono copiati **immediatamente** su disco
      - *Write through*
    - Alto costo ma consistenza sicura (specie con dischi rimovibili)
  - **UNIX → GNU/Linux**
    - Un processo periodico (`sync`) effettua l'aggiornamento dei blocchi su disco
    - Basso costo e basso rischio con dischi fissi affidabili