Sistemi Operativi
File System (parte 2)

Docente: Claudio E. Palazzi

cpalazzi@math.unipd.it

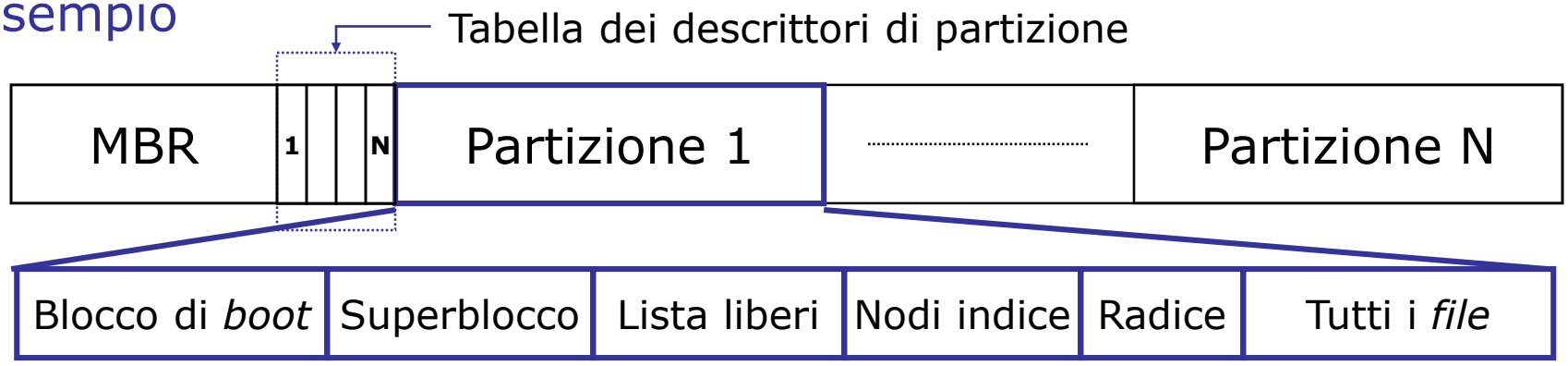
Realizzazione del *file system* – 1

- 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** contiene (in 512 B) una descrizione delle partizioni che 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!)
  - 1 blocco = N settori (N ≥ 1)
  - Rischio consapevole di frammentazione interna
- La struttura interna di partizione è specifica del FS

Esempio



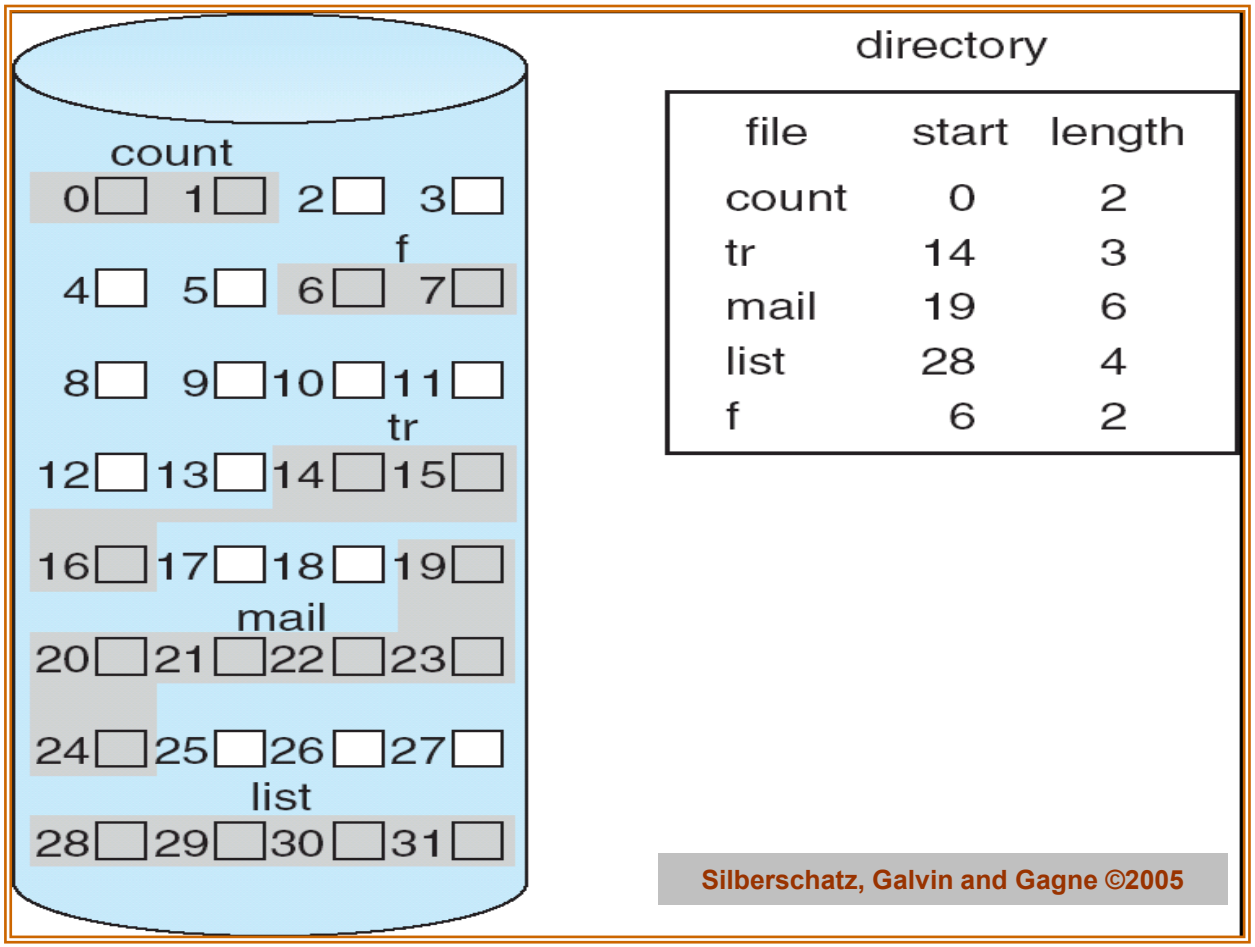
Realizzazione dei *file* – 1

- 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* per farli stare in un blocco libero
        - » Stima difficile e rischiosa

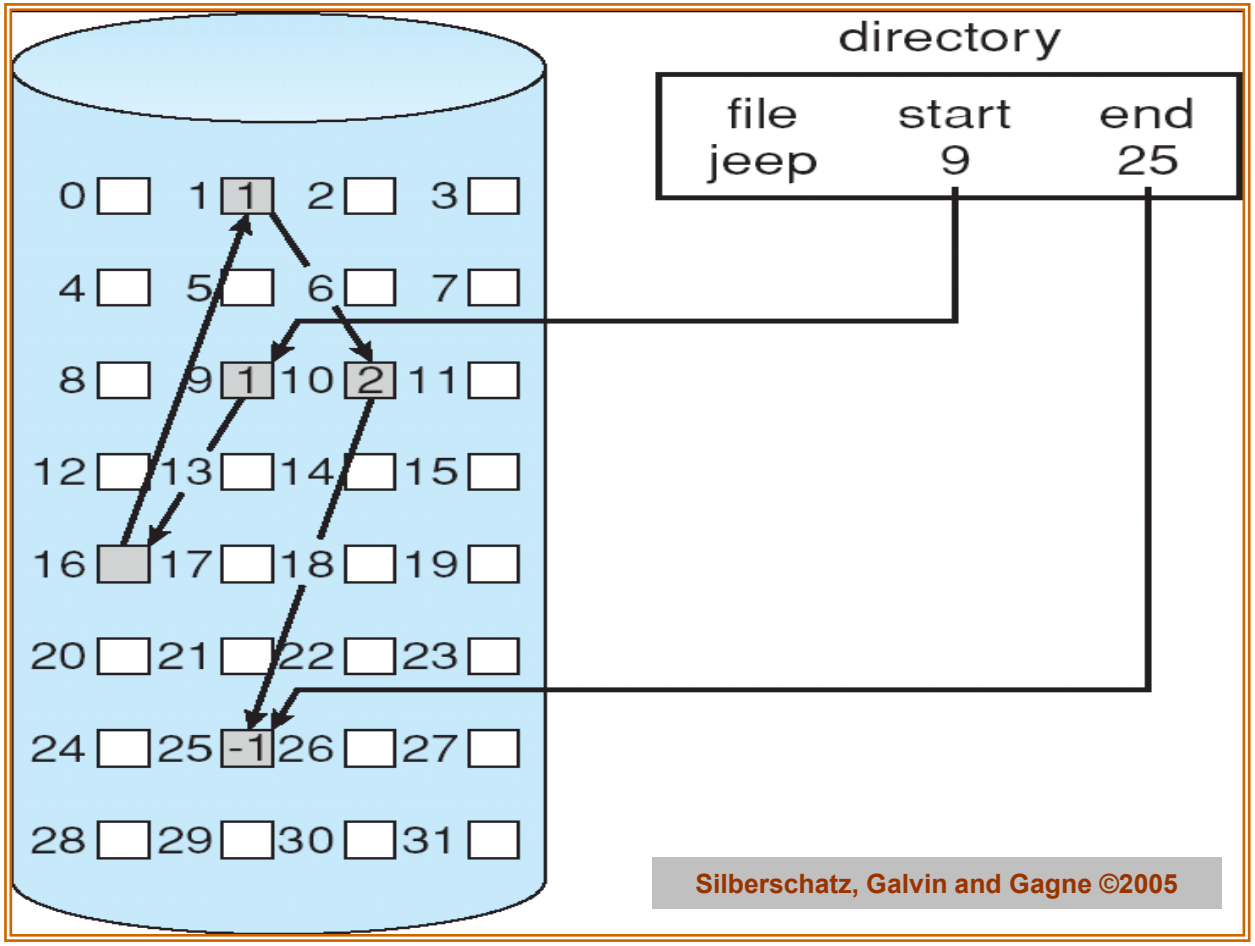
Allocazione contigua



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 ma può richiedere molte operazioni su disco
    - Accesso diretto molto più complesso e oneroso (lento)
  - Un solo blocco guasto corrompe l'intero *file*

Allocazione a lista concatenata



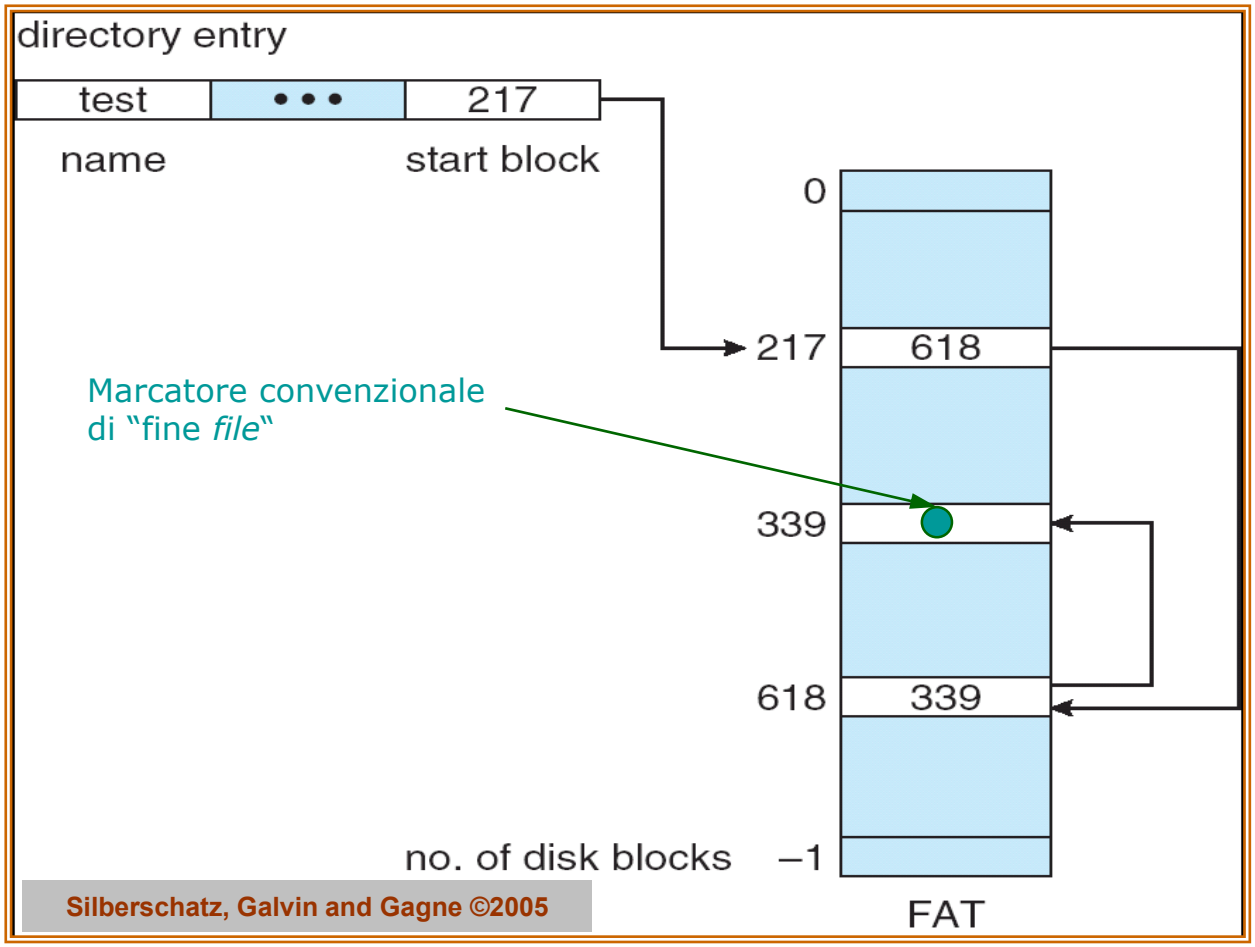
Realizzazione dei *file* – 4

- **Allocazione a lista indicizzata**
  - Si pongono i puntatori ai blocchi in strutture apposite
    - Ciascun blocco contiene **solo dati**
  - Il *file* è descritto dall'insieme dei suoi puntatori
  - 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
  - Non richiede di conoscere preventivamente la dimensione massima di ogni nuovo *file*

Allocazione a lista indicizzata – 1

- ***File Allocation Table***
  - La scelta progettuale di *MS-DOS*
    - Base di *MS Windows*
- **FAT** = tabella ordinata di puntatori
  - Un puntatore ∀ 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 seguendo sequenzialmente i collegamenti ma senza accessi a disco
  - Es. disco da 200 GB, blocchi da 1 KB, serve FAT di 200 M righe ciascuna di 3-4 Bytes: 6-800 MB di memoria impiegati!
- Un *file* è una catena di indici

Struttura **FAT**



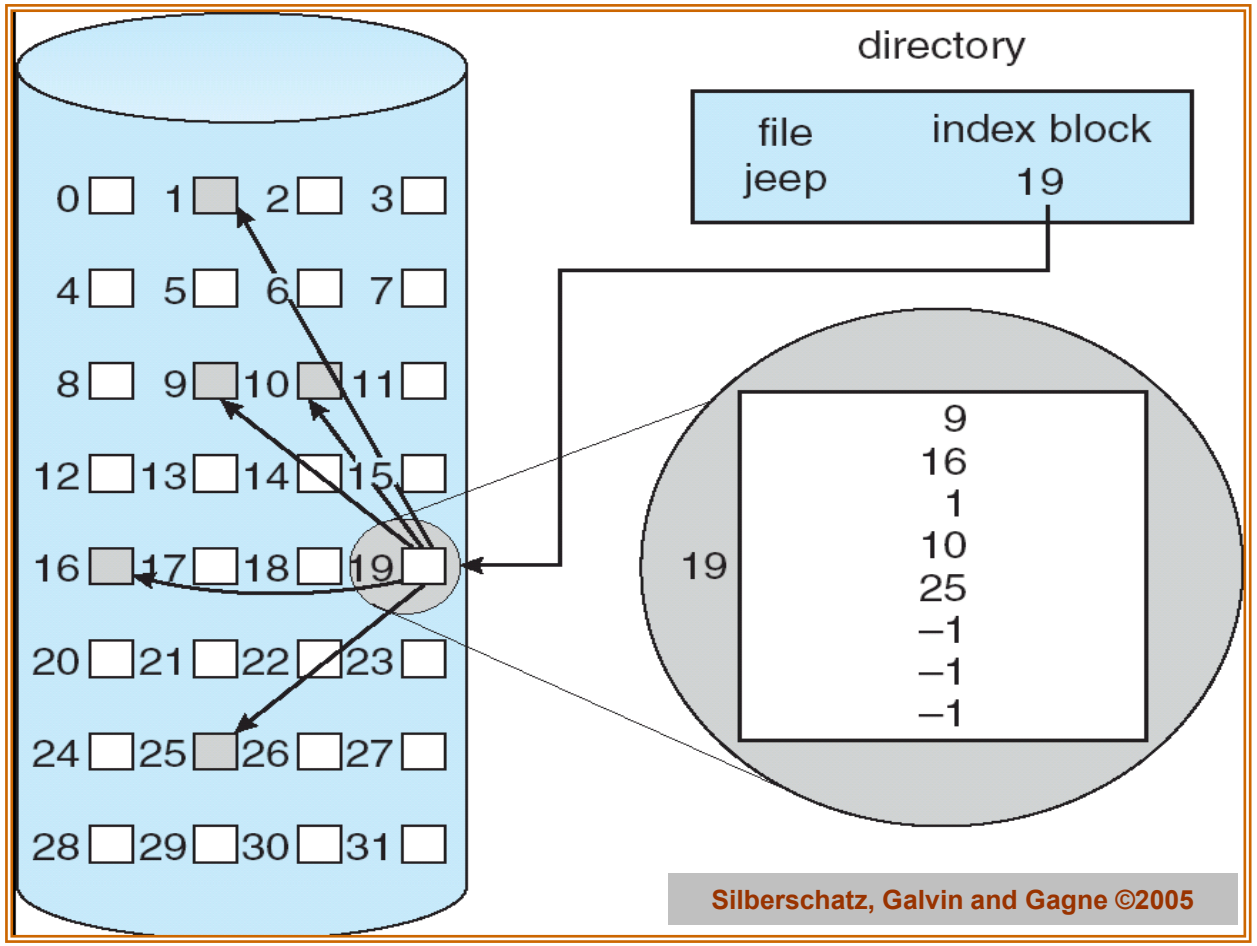
Allocazione a lista indicizzata – 2

- **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?

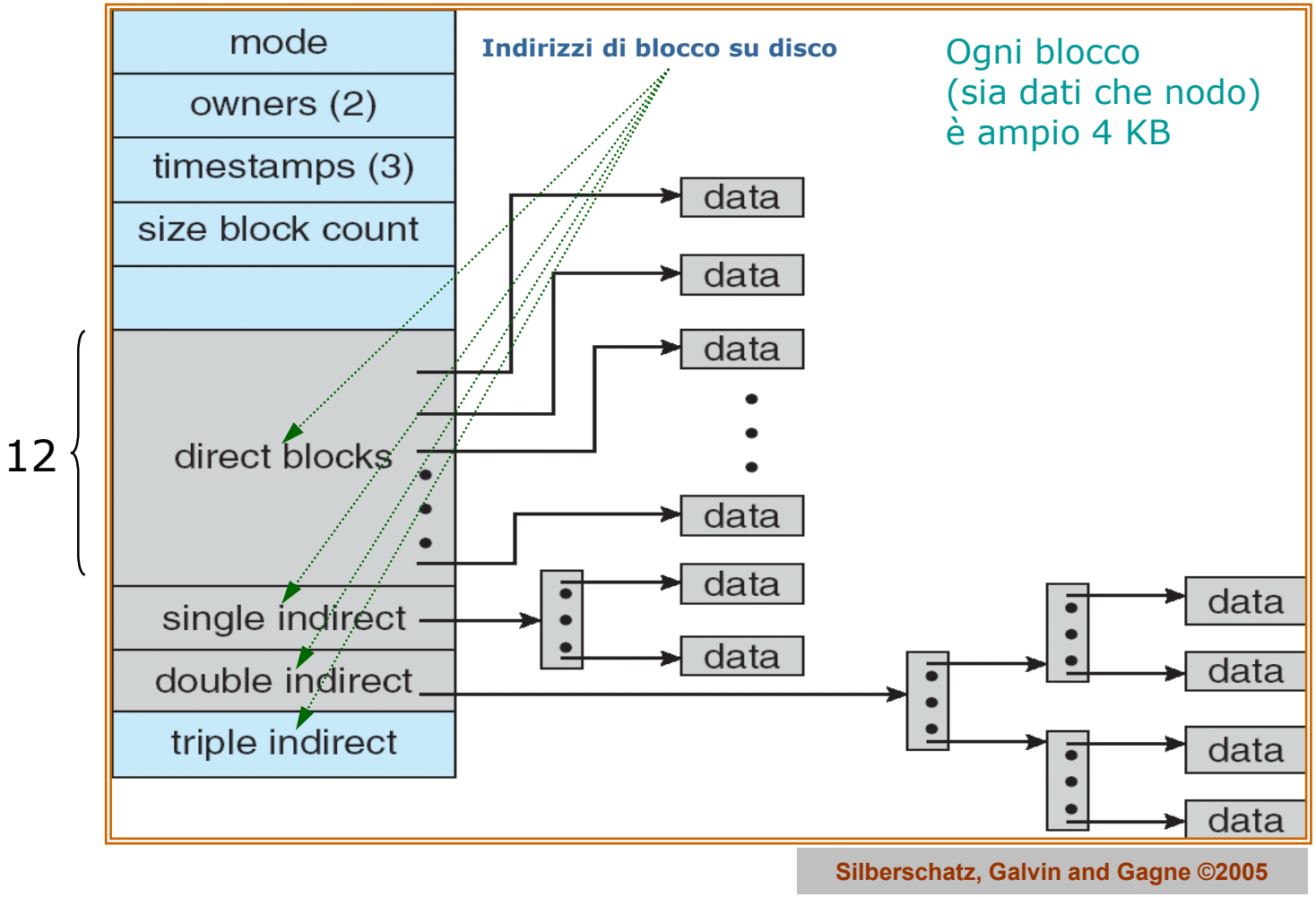
Allocazione a lista indicizzata – 3

- **Nodi indice** (UNIX → 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



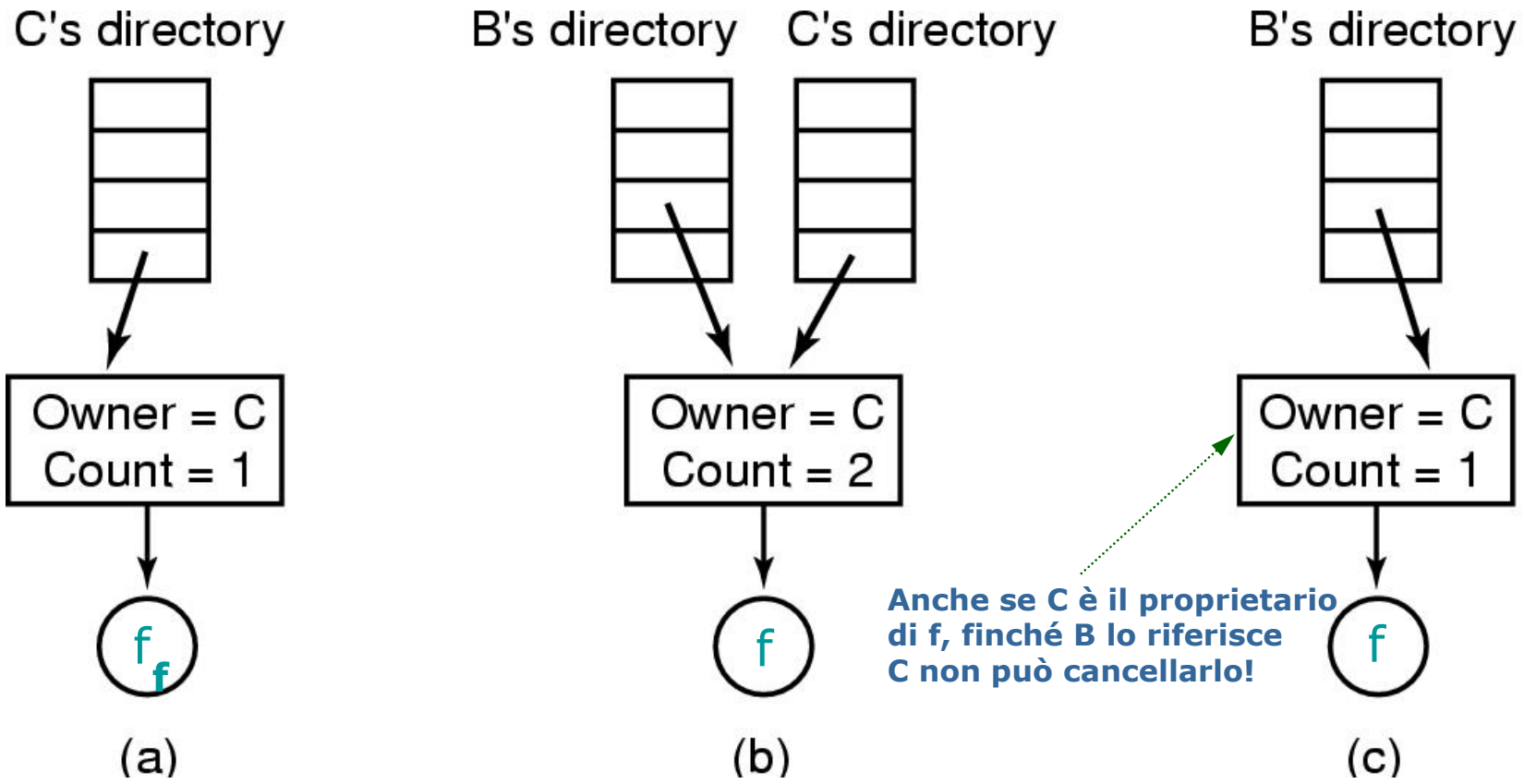
FS in UNIX v7


mode
owners (2)
timestamps (3)
size block count
direct blocks
single indirect
double indirect
triple indirect
12
Indirizzi di blocco su disco
Ogni blocco
(sia dati che nodo)
è ampio 4 KB
data
data
data
data
data
data
data
data
data
data
Silberschatz, Galvin and Gagne ©2005

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 descrittore** (*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 descrittore (*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

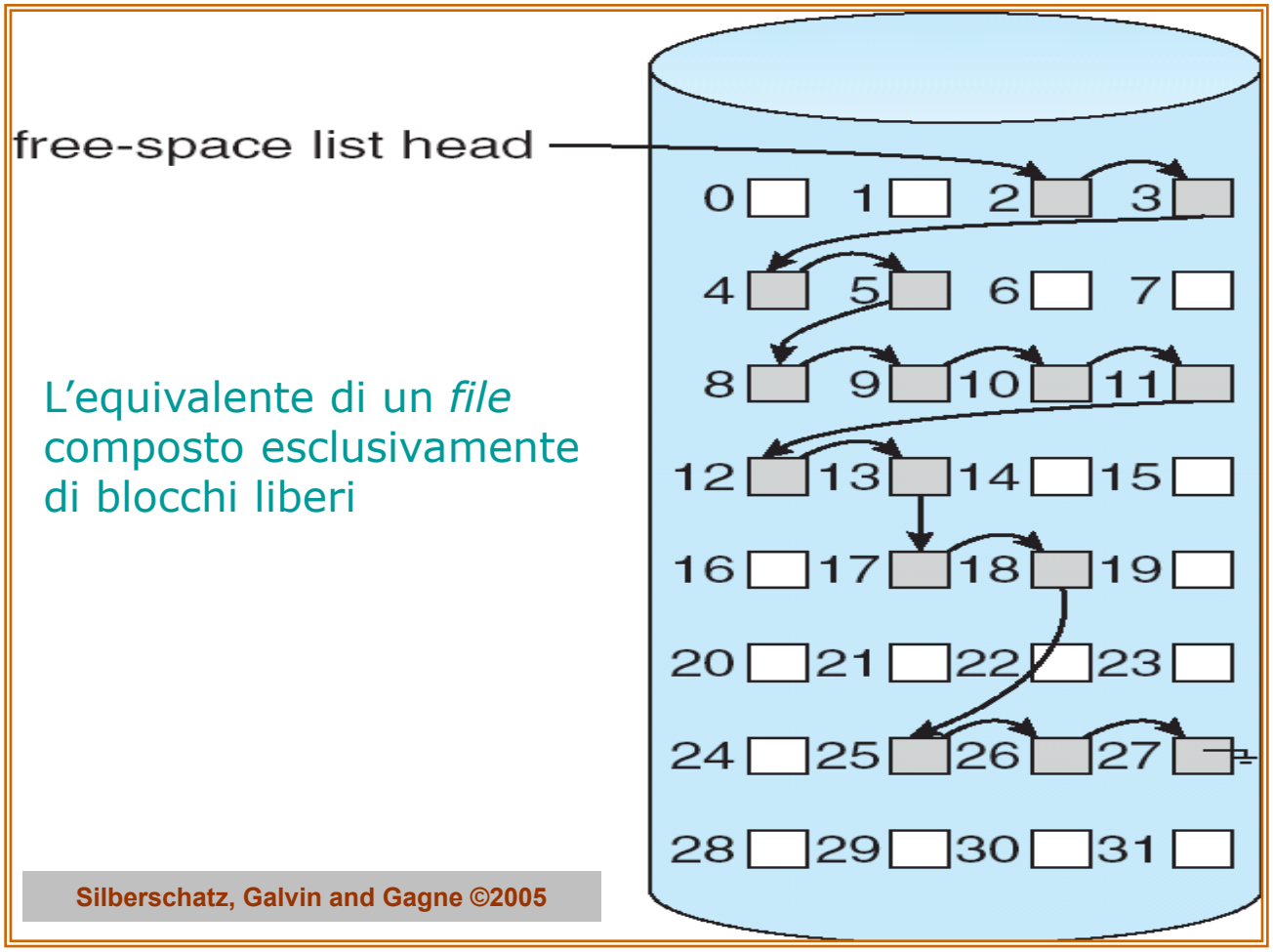
Gestione della condivisione



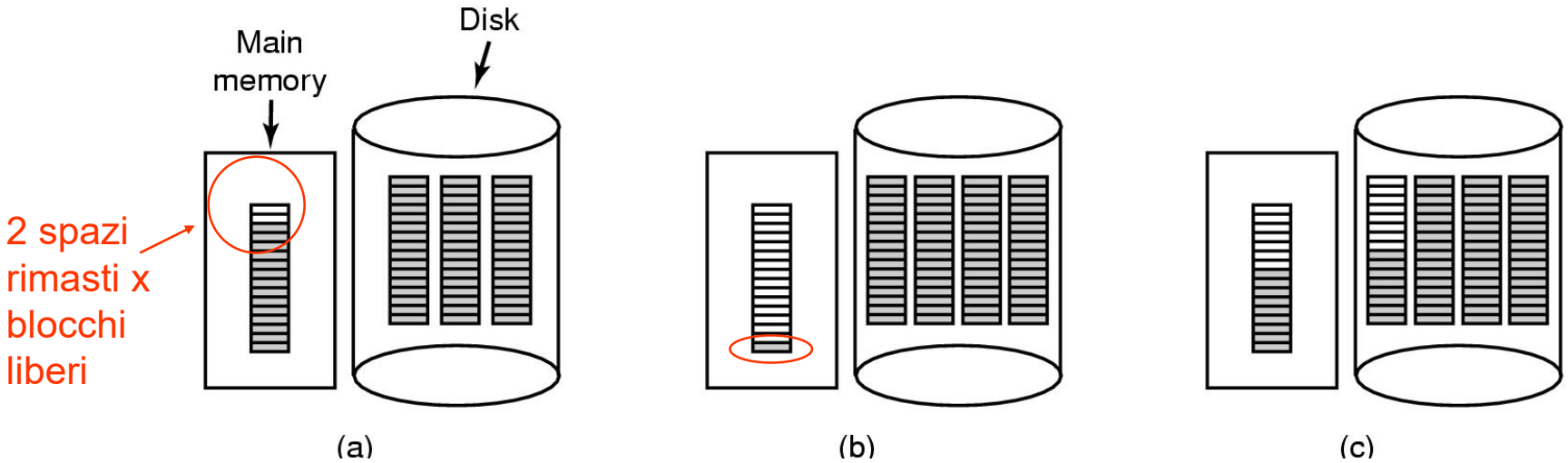
Realizzazione dei *file* – 6

- **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
    - Questa è la scelta nell'architettura FAT

Lista concatenata dei blocchi liberi



Gestione Spazio su Disco



(a) In RAM: Blocco di puntatori a blocchi liberi su disco
- Altri su disco (non c'è bisogno di averli sempre tutti in RAM)

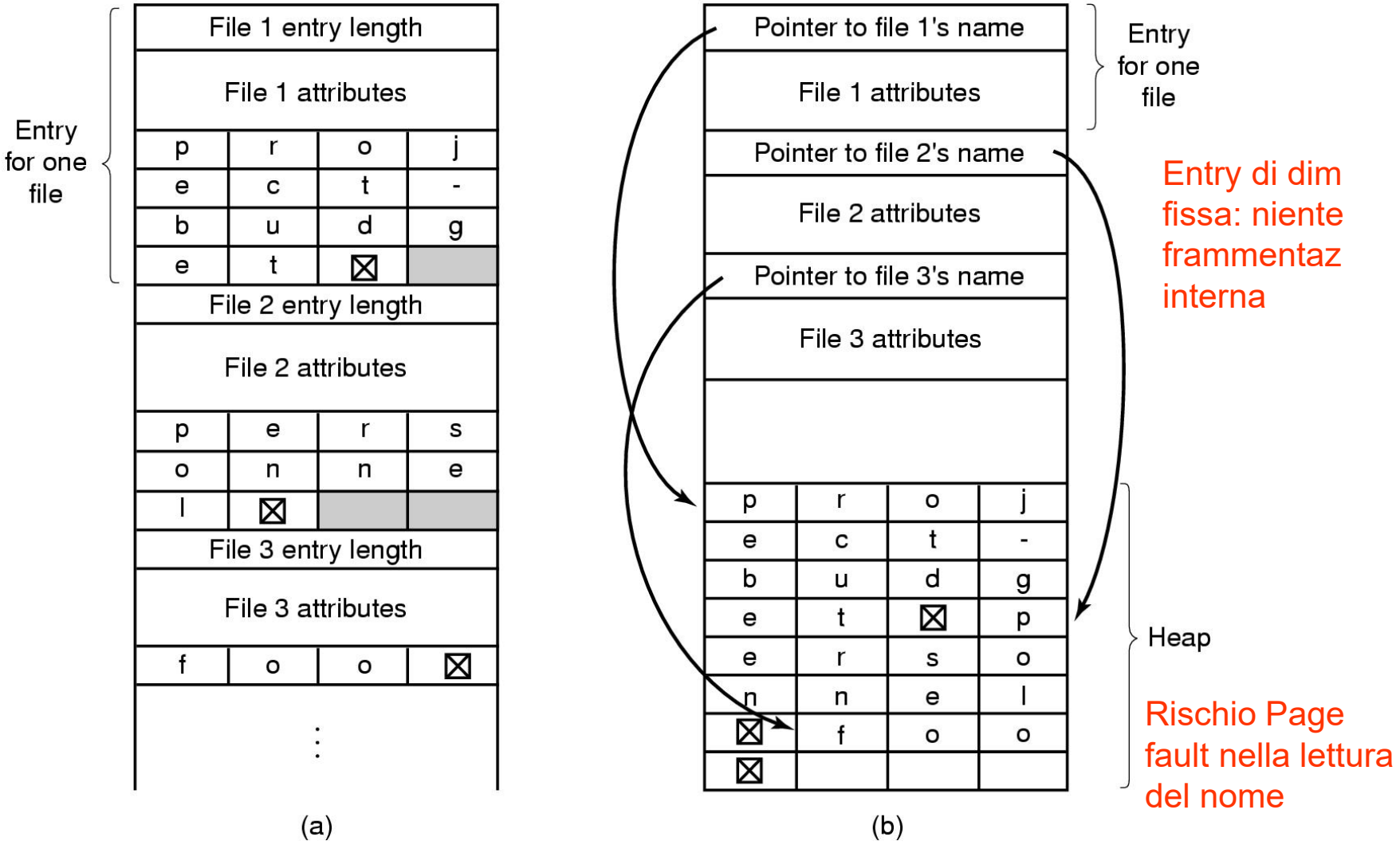
(b) Dopo aver cancellato un file di 3 blocchi
- ma se poi riscrivo 3 blocchi? (devo ricaricare blocco di prima…)

(c) Strategia alternativa per la gestione dei blocchi
- Una volta riempito il blocco, si dividono i puntatori in due parti: metà in blocco in RAM e metà in blocco su disco
- Riempito quello in RAM, si fa scambio

Realizzazione delle *directory* – 1

- La *directory* fornisce informazioni su
  - Nome
  - Collocazione
  - Attributi
- Di *file* appartenenti a quel particolare catalogo
- *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 intrinsecamente 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
    - Il problema diventa però più serio per nomi lunghi

Realizzazione delle *directory* – 2



Entry di dim fissa: niente frammentaz interna

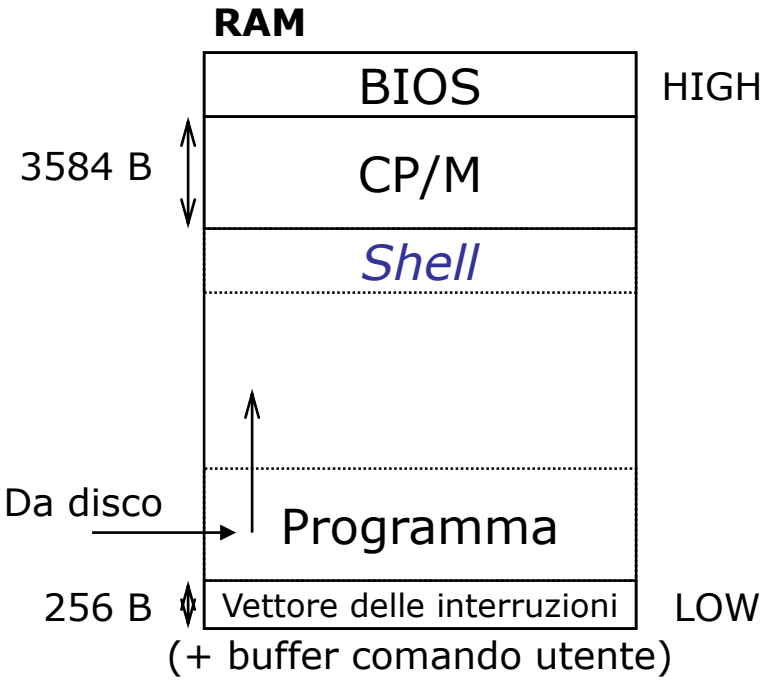
Rischio Page fault nella lettura del nome

Prospettiva storica

- 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
  - 17 I/O calls (massima portabilità)
- Sistema multiprogrammato
  - Ogni utente vede solo i propri *file*
- *Directory* singola con dati a struttura fissa (32 B entry)
  - In RAM solo quando serve
- *Bitmap* in RAM per blocchi di disco liberi
  - Distrutta a fine esecuzione
- Nome *file* limitato a 8 + 3 caratteri
  - Dimensione inizialmente limitata a 16 blocchi da 1 KB
    - Puntati da *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 ($12 \leq$ **X** <32)
  - Blocchi/Cluster di dimensione multipla di 512 B;
    - Max partition size è 2^{12} x 512B = 2MB
    - Estendendo blocchi fino a 4KB si arriva a 16 MB max
  - **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 Win95
      - 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

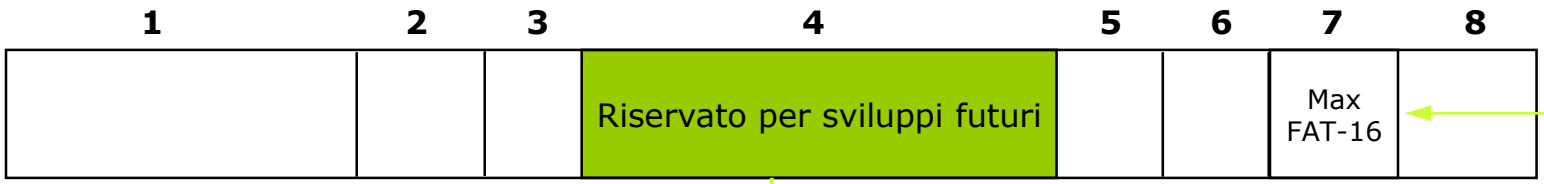
Il FS in MS-DOS 7.0 – 1

Struttura di *directory entry* (32 B)

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

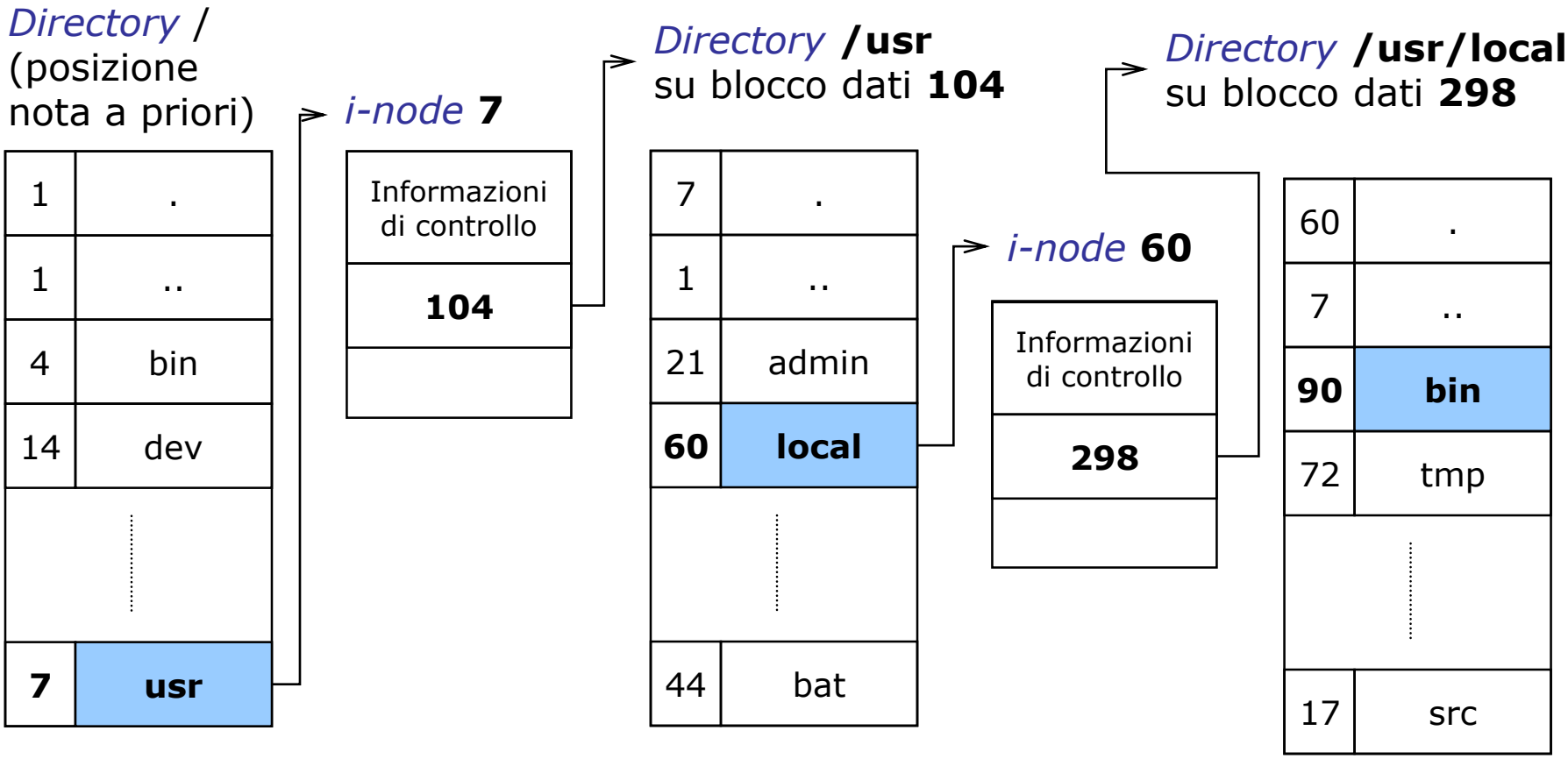


Usato per Windows 98 (FAT-32, orario accurato, nomi *file* lunghi e *case sensitive*)

UNIX v7

- 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 / e NUL)
  - *Directory* contiene nome *file* e puntatore (su 2 B) al suo *i-node* descrittore
    - Max 64 K *file* per FS (2^{16} *i-node* distinti)
  - 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 in UNIX



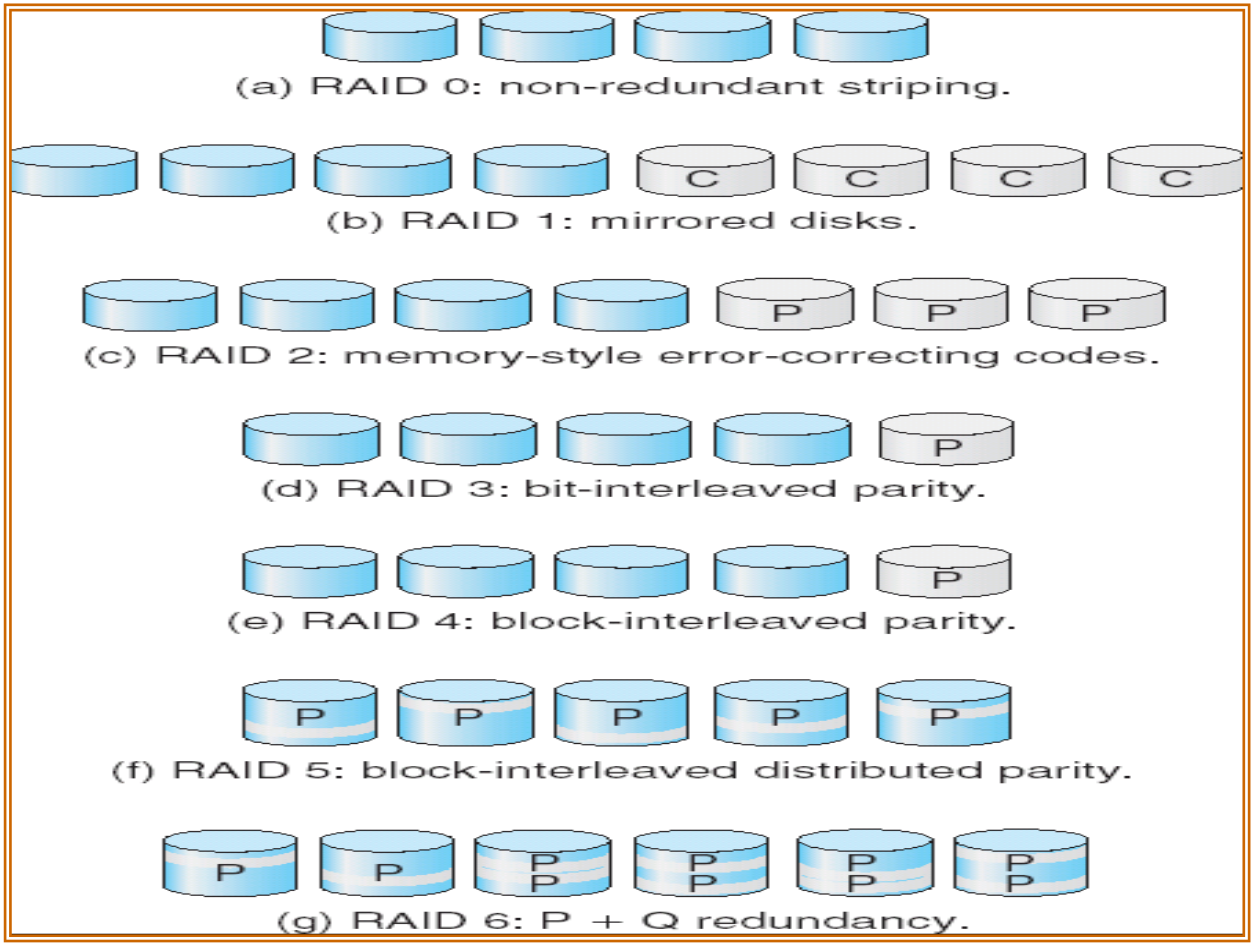
Esecuzione parziale del comando "cd /usr/local/bin/"

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

Redundant Array of Inexpensive Disks



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

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 memoria principale viene usata come *cache* 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