Sistemi Operativi
2021/2022

Modulo 8: File System

Renzo Davoli
Alberto Montresor

1. Visione utente

Introduzione

- **I computer possono utilizzare diversi media per registrare in modo permanente le informazioni**
  - esempi: dischi rigidi, floppy, nastri, dischi ottici
  - ognuno di questi media ha caratteristiche fisiche diverse
- **Compito del *file system* è quello di astrarre la complessità di utilizzo dei diversi media proponendo una interfaccia per i sistemi di memorizzazione:**
  - comune
  - efficiente
  - conveniente da usare

Introduzione

- **Dal punto di vista dell'utente, un file system è composto da due elementi:**
  - *file*: unità logica di memorizzazione
  - *directory*: servono per organizzare e fornire informazioni sui file che compongono un file system
- **Il concetto di file**
  - è l'entità atomica di assegnazione/gestione della memoria secondaria
  - è una collezione di informazioni correlate
  - fornisce una vista logica uniforme ad informazioni correlate

Attributi dei file

- **Nome:**
  - stringa di caratteri che permette agli utenti ed al sistema operativo di identificare un particolare file nel file system
  - alcuni sistemi differenziano fra caratteri maiusc./minusc., altri no
- **Tipo:**
  - necessario in alcuni sistemi per identificare il tipo di file
- **Locazione e dimensione**
  - informazioni sul posizionamento del file in memoria secondaria
- **Data e ora:**
  - informazioni relative al tempo di creazione ed ultima modifica del file

Attributi dei file

- **Informazioni sulla proprietà**
  - utenti, gruppi, etc.
  - utilizzato per accounting e autorizzazione
- **Attributi di protezione:**
  - informazioni di accesso per verificare chi è autorizzato a eseguire operazioni sui file
- **Altri attributi**
  - flag (sistema, archivio, hidden, etc.)
  - informazioni di locking
  - etc.

Tipi di file

- **A seconda della struttura interna**
  - senza formato (stringa di byte): file testo,
  - con formato: file di record, file di database, a.out,...
- **A seconda del contenuto**
  - ASCII/binario (visualizzabile o no, 7/8 bit)
  - sorgente, oggetto, ...
  - eseguibile (oggetto attivo)

Tipi di file

- **Alcuni S.O. supportano e riconoscono diversi tipi di file**
  - conoscendo il tipo del file, il s.o. può evitare alcuni errori comuni, quali ad esempio stampare un file eseguibile
- **Esistono tre tecniche principali per identificare il tipo di un file**
  - meccanismo delle estensioni
  - utilizzo di un attributo "tipo" associato al file nella directory
  - magic number

Tipi di file

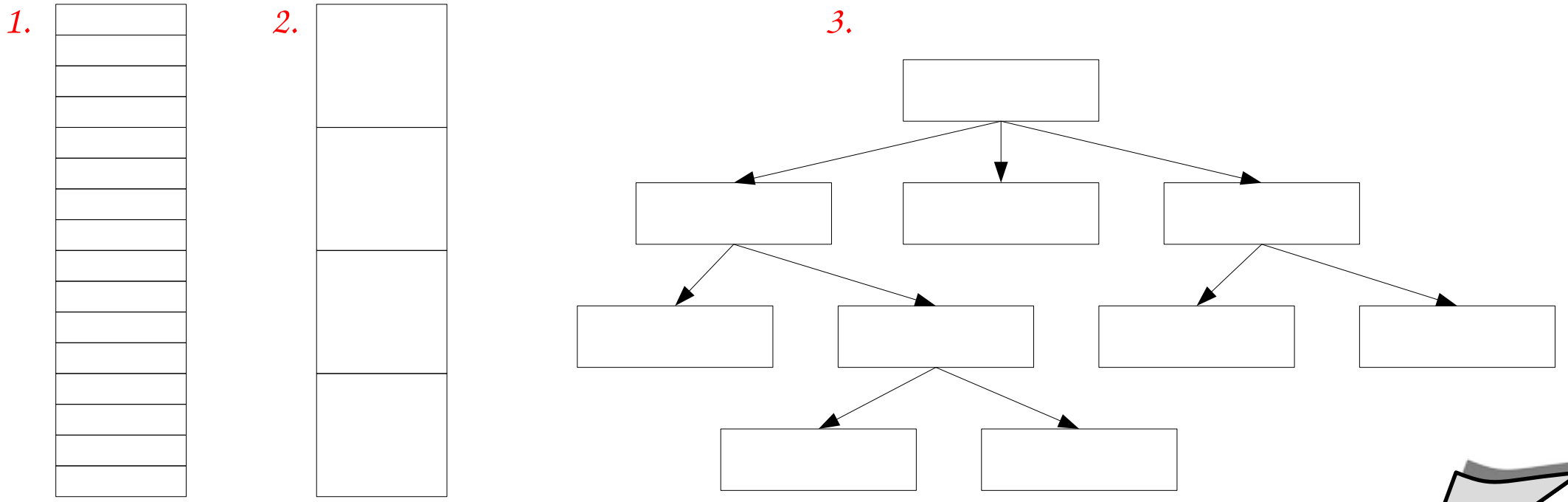
- **MS-DOS, free-DOS:**
  - nome del file 8+3 (nome + estensione)
  - riconoscimento delle estensioni .COM, .EXE, .BAT
- **Windows 9x / NT / 7/8/9/10**
  - nomi/estensioni di lunghezza variabile
  - riconoscimento delle estensioni .COM, .EXE, .BAT
  - associazione estensione / programma
- **Mac OS**
  - programma creatore del file come attributo
- **Unix/Linux**
  - magic number + estensione + euristica (UNIX).

Tipi di file: ulteriori distinzioni

- **In un file system nei sistemi UNIX, soo presenti:**
  - *file regolari*
    - *Sequenze di byte*
  - *directory*
    - file di sistema per mantenere la struttura del file system
  - *file speciali a blocchi*
    - utilizzati per modellare dispositivi di I/O come i dischi (*)
  - *file speciali a caratteri*
    - utilizzati per modellare device di I/O seriali come terminali, stampanti e reti (*)
  - altri file speciali
    - ad es., pipe (*)
  - (*) NON sono file!

Struttura dei file

- **I file possono essere strutturati in molti modi:**
  1. sequenze di byte
  2. sequenze di record logici
  3. file indicizzati (struttura ad albero)



Struttura dei file

- **I sistemi operativi possono attuare diverse scelte nella gestione della struttura dei file:**
  - scelta minimale
    - i file sono considerati semplici stringhe di byte, a parte i file eseguibili il cui formato è dettato dal s.o.
    - e.g., UNIX e MS-DOS
  - parte strutturata/parte a scelta dell'utente
    - e.g. Macintosh (resource fork / data fork)
  - diversi tipi di file predefiniti
    - e.g., VMS, MVS

Supporto alla struttura dei file

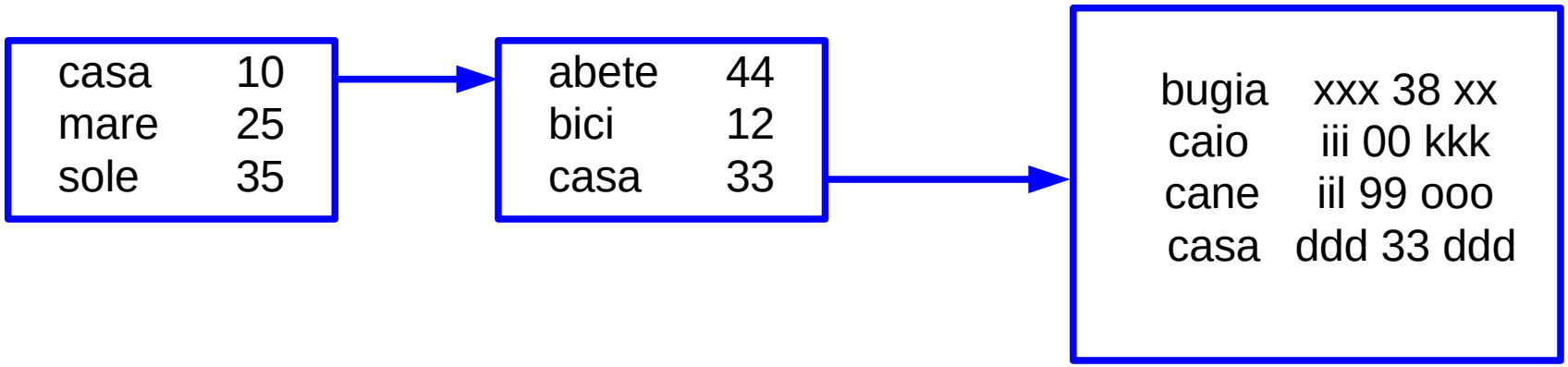
- **E’ un trade-off:**
  - più formati:
    - codice di sistema più ingombrante
    - incompatibilità di programmi (accesso a file di formato differente
    - *MA* gestione efficiente e non duplicata per i formati speciali
  - meno formati
    - codice di sistema più snello

Metodi di accesso

- **Sequenziale**
  - read, write
- **Ad accesso diretto**
  - read *pos*, write *pos* (oppure operazione seek)
- **Indicizzato**
  - read *key*, write *key*
  - Tipico dei database

Metodi di accesso: indice

- **Indice**
  - è una tabella di corrispondenza chiave-posizione
- **Può essere memorizzato:**
  - in memoria: metodo efficiente ma dispendioso
  - su disco
- **Esempio**



Operazioni sui file

- **Operazioni fondamentali sui file**
  - creazione
  - apertura/chiusura
  - lettura/scrittura/append
  - posizionamento
  - cancellazione
  - troncamento
  - lettura/scrittura attributi

Operazioni sui file

- **L'API (interfaccia per la programmazione) relativa alle operazioni su file è basata sulle operazioni open/close**
  - i file devono essere “aperti” prima di effettuare operazioni e “chiusi” al termine.
- **L’astrazione relativa all’apertura/chiusura dei file è utile per**
  - mantenere le strutture dati di accesso al file
  - controllare le modalità di accesso e gestire gli accessi concorrenti
  - definire un descrittore per le operazioni di accesso ai dati

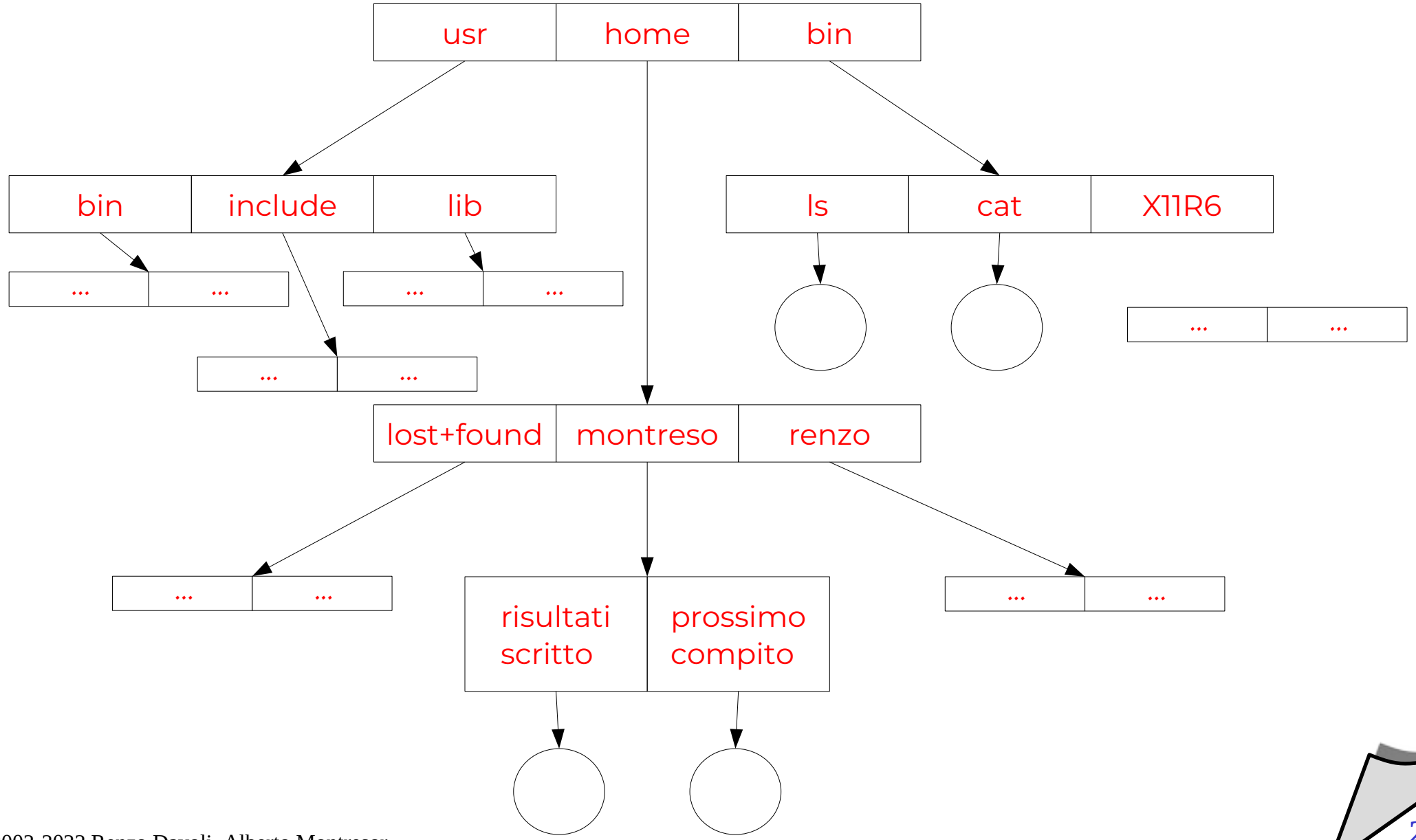
Directory

- **L'organizzazione dei file system**
  - è basata sul concetto di directory, che fornisce un'astrazione per un'insieme di file
  - in molti sistemi, le directory sono file (speciali)
- **Operazioni definite sulle directory**
  - creazione
  - cancellazione
  - apertura di una directory
  - chiusura di una directory
  - lettura di una directory
  - rinominazione
  - link/unlink

Directory

- **Struttura di una directory**
  - a livello singolo
  - a due livelli
  - ad albero
  - a grafo aciclico
  - a grafo

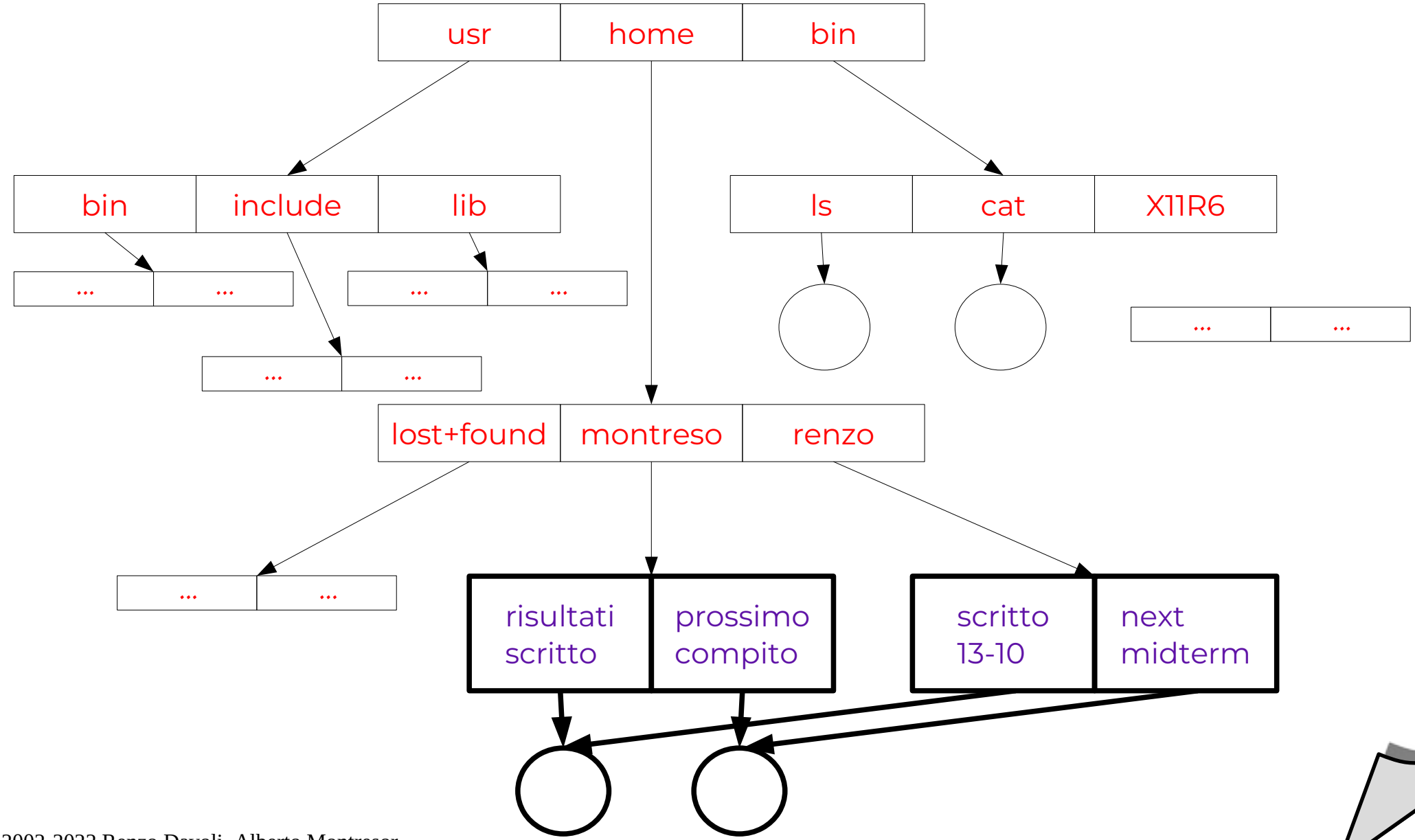
Directory strutturata ad albero



Directory strutturate a grafo aciclico

- **Nella struttura ad albero:**
  - ogni file è contenuto in una directory univoca
- **E' anche possibile considerare grafi diversi dagli alberi**
  - un file può essere contenuto in due o più directory
  - esiste un'unica copia del file suddetto:
    - ogni modifica al file è visibile in entrambe le directory
- **La struttura risultante prende il nome di *grafo diretto aciclico* (*DAG*)**

Directory strutturata a grafo aciclico



Semantica della coerenza

- **In un sistema operativo multitasking, i processi accedono ai file indipendentemente**
- **Come vengono viste le modifiche ai file da parte dei vari processi?**
- **In UNIX**
  - le modifiche al contenuto di un file aperto vengono rese visibili agli altri processi immediatamente.
  - esistono due tipi di condivisione del file:
    - condivisione del puntatore alla posizione corrente nel file (condivisione ottenibile con fork)
    - condivisione con distinti puntatori alla posizione corrente.
- **Non è sempre così. Es. semantica delle sessioni in AFS**

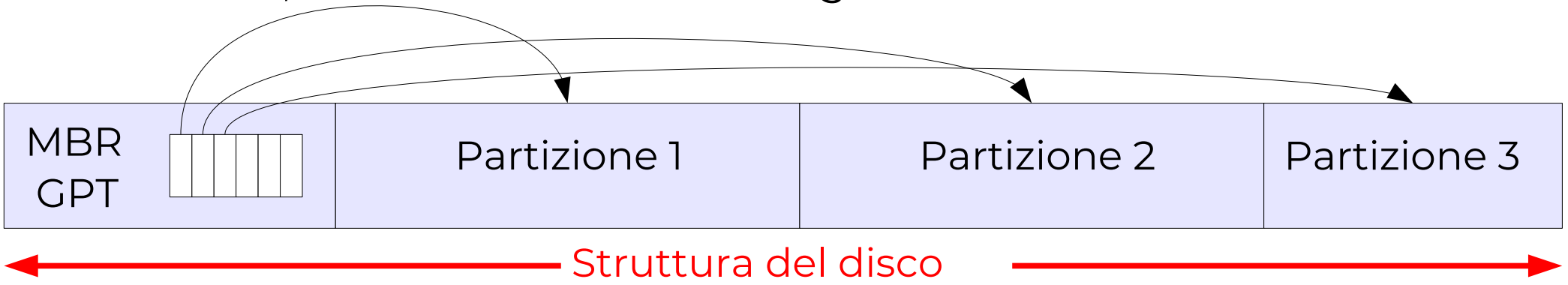
2. Visione implementatore

Implementazione del file system

- **Problemi da tenere in considerazione**
  - organizzazione di un disco
  - allocazione dello spazio in blocchi
  - gestione spazio libero
  - implementazione delle directory
  - tecniche per ottimizzare le prestazioni
  - tecniche per garantire la coerenza

Organizzazione del disco

- **Struttura di un disco**
  - un disco può essere diviso in una o più *partizioni*, porzioni indipendenti del disco che possono ospitare file system distinti
  - il primo settore dei dischi è il cosiddetto *master boot record* (MBR)
    - è utilizzato per fare il boot del sistema
    - contiene la *partition table* (tabella delle partizioni)
    - contiene l'indicazione della partizione attiva
  - al boot, il MBR viene letto ed eseguito



Organizzazione del disco

- **Struttura di una partizione**
  - ogni partizione inizia con un boot block
  - il MBR carica il boot block della partizione attiva e lo esegue
  - il boot block carica il sistema operativo e lo esegue
  - l'organizzazione del resto della partizione dipende dal file system

Organizzazione del disco

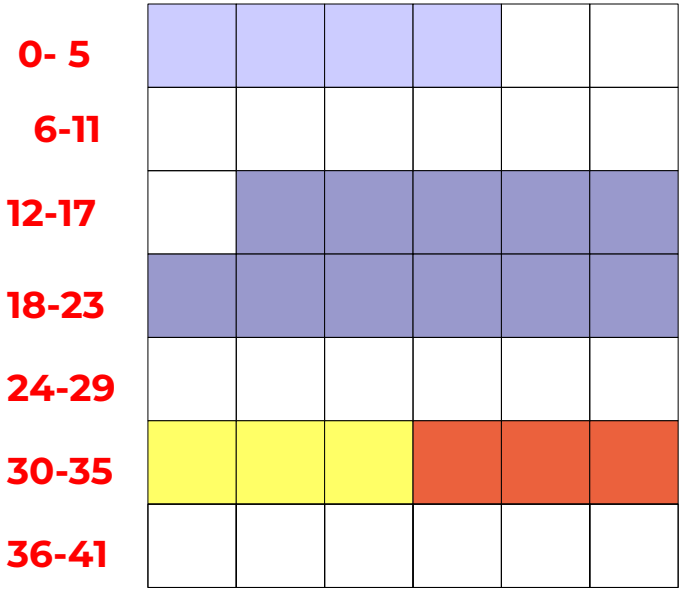
- **In generale**
  - *superblock*
    - contiene informazioni sul tipo di file system e sui parametri fondamentali della sua organizzazione
  - *tabelle per la gestione dello spazio libero*
    - struttura dati contenente informazioni sui blocchi liberi
  - *tabelle per la gestione dello spazio occupato*
    - contiene informazioni sui file presenti nel sistema
    - non presente in tutti i file system
  - *root dir*
    - directory radice (del file system)
  - file e directory

Allocazione

- **Problema**
  - l'hardware e il driver del disco forniscono accesso al disco visto come un insieme di blocchi dati di dimensione fissa.
  - partendo da questa struttura come si implementa l'astrazione di file?
  - in altre parole: come vengono scelti i blocchi dati da utilizzare per un file e come questi blocchi dati vengono collegati assieme a formare una struttura unica
- **Questo è il problema dell'allocazione**

Allocazione contigua

- **Descrizione**
  - i file sono memorizzati in sequenze contigue di blocchi di dischi
- **Vantaggi**
  - non è necessario utilizzare strutture dati per collegare i blocchi
  - l'accesso sequenziale è efficiente
    - blocchi contigui non necessitano operazioni di seek
  - l'accesso diretto è efficiente
    - **block= offset/blocksize;**
    - **pos= offset%blocksize;**



directory

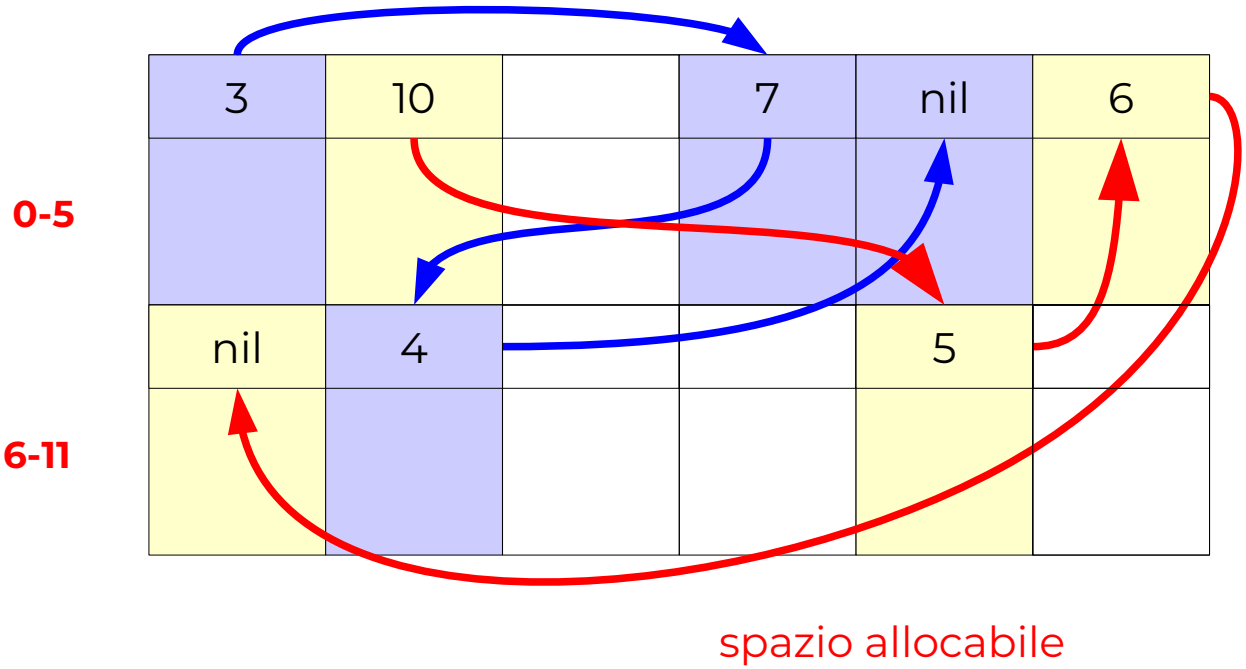
Name	Start	Size
a	0	4
b	13	11
c	30	3
d	33	3

Allocazione contigua

- **Svantaggi**
  - si ripropongono tutte le problematiche dell'allocazione contigua in memoria centrale
    - frammentazione esterna
    - politica di scelta dell'area di blocchi liberi da usare per allocare spazio per un file: first fit, best fit, worst fit....
  - inoltre:
    - i file non possono crescere!
- **Ma esiste almeno un tipo di file system in cui viene utilizzata allocazione contigua....**

Allocazione concatenata

- **Descrizione**
  - ogni file è costituito da un lista concatenata di blocchi.
  - ogni blocco contiene un puntatore al blocco successivo
  - il descrittore del file contiene i puntatori al primo e all'ultimo elemento della lista



directory

Name	Start	End
a	0	4
b	1	6

Allocazione concatenata

- **Vantaggi**
  - risolve il problema della frammentazione esterna
  - l'accesso sequenziale o in "append mode" è efficiente
- **Svantaggi**
  - l'accesso diretto è inefficiente
  - progressivamente l'efficienza globale del file system degrada (i blocchi sono disseminati nel disco, aumenta il n. di seek)
  - la dimensione utile di un blocco non è una potenza di due
  - se il blocco è piccolo (512 byte) l'overhead per i puntatori può essere rilevante

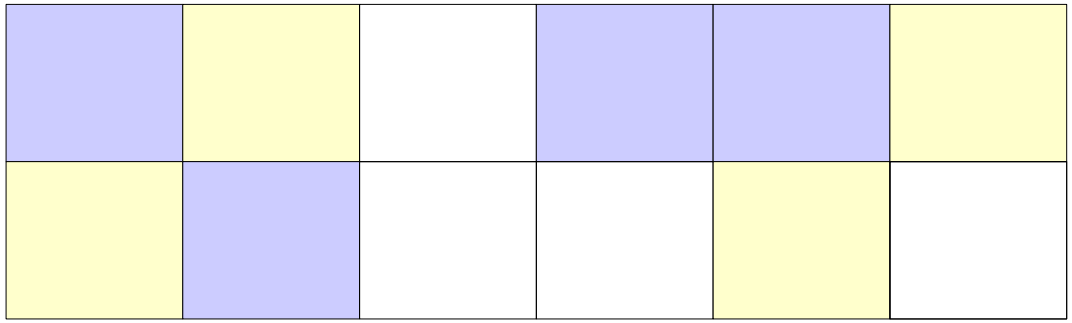
Allocazione concatenata

- **Minimizzare l'overhead dovuto ai puntatori**
  - i blocchi vengono riuniti in cluster (contenenti 4, 8, 16 blocchi) e vengono allocati in modo indivisibile
- **Vantaggi**
  - la percentuale di spazio utilizzato per i puntatori diminuisce
- **Svantaggi**
  - aumenta lo spazio sprecato per la frammentazione interna

Allocazione basata su FAT

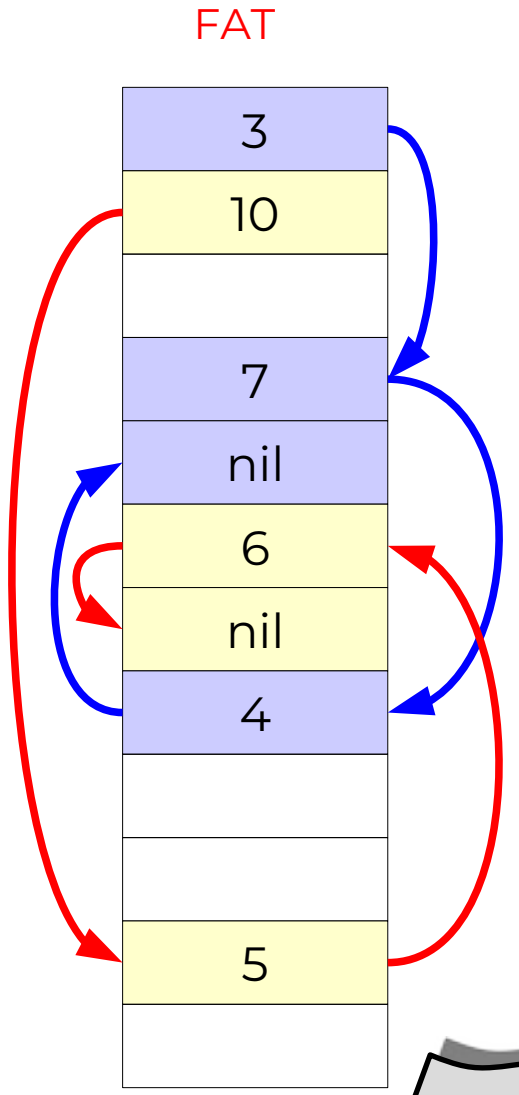
- **Descrizione**
  - invece di utilizzare parte del blocco dati per contenere il puntatore al blocco successivo si crea una tabella unica con un elemento per blocco (o per cluster)

spazio allocabile



directory

Name	Start	End
a	0	4
b	1	6

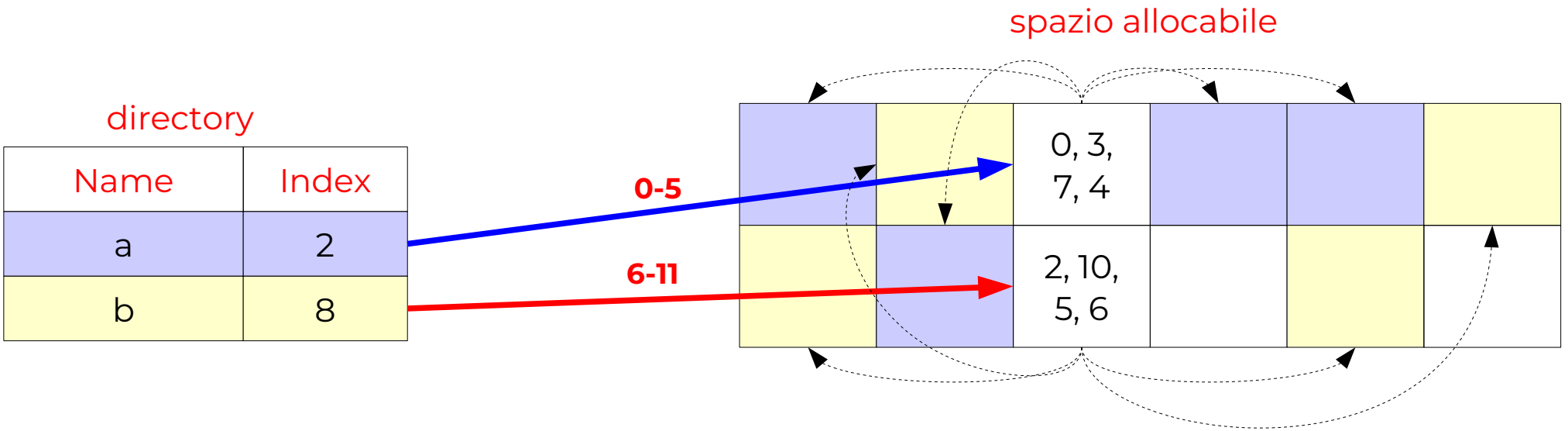


Allocazione basata su FAT

- **Vantaggi**
  - i blocchi dati sono interamente dedicati… ai dati
- **Svantaggi**
  - la scansione richiede anche la lettura della FAT, aumentando così il numero di accessi al disco.
- **Vantaggi**
  - è possibile fare caching in memoria dei blocchi FAT
  - l'accesso diretto diventa così più efficiente, in quanto la lista di puntatori può essere seguita in memoria
- **E' il metodo usato da DOS, macchine fotografiche, chiavette USB (off-the-shelf), …**

Allocazione indicizzata

- **Descrizione**
  - l'elenco dei blocchi che compongono un file viene memorizzato in un blocco (o area) indice
  - per accedere ad un file, si carica in memoria la sua area indice e si utilizzano i puntatori contenuti

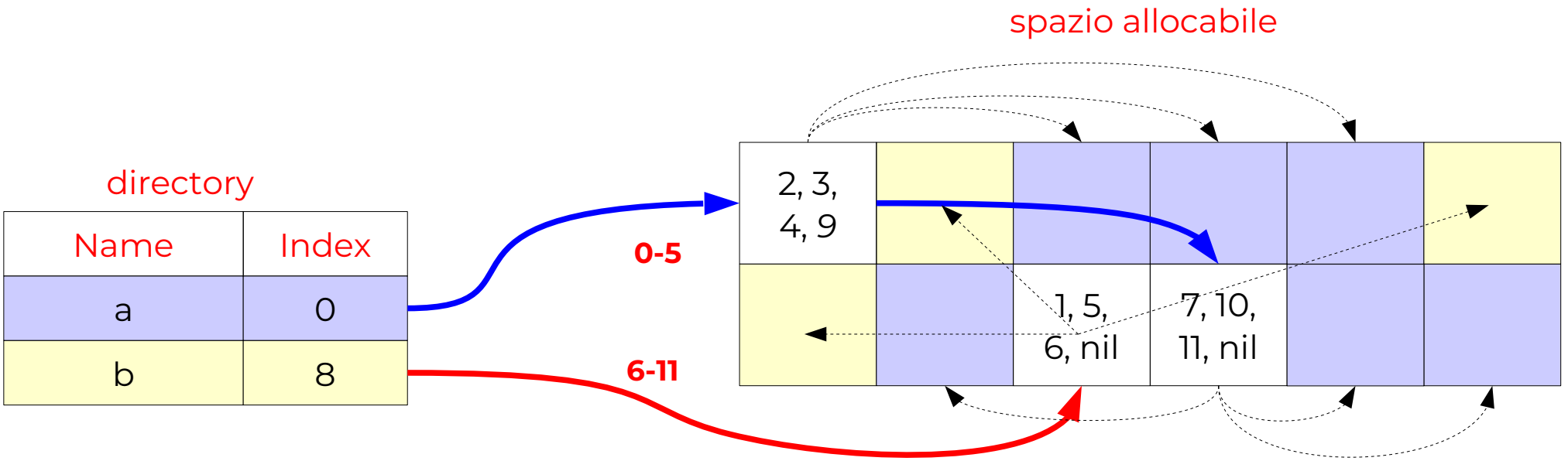


Allocazione indicizzata

- **Vantaggi**
  - risolve (come l'allocazione concatenata) il problema della frammentazione esterna.
  - è efficiente per l'accesso diretto
  - il blocco indice deve essere caricato in memoria solo quando il file è aperto
- **Svantaggi**
  - la dimensione del blocco indice determina l'ampiezza massima del file
  - utilizzare blocchi indici troppo grandi comporta un notevole spreco di spazio
- **Come risolvere il trade-off?**

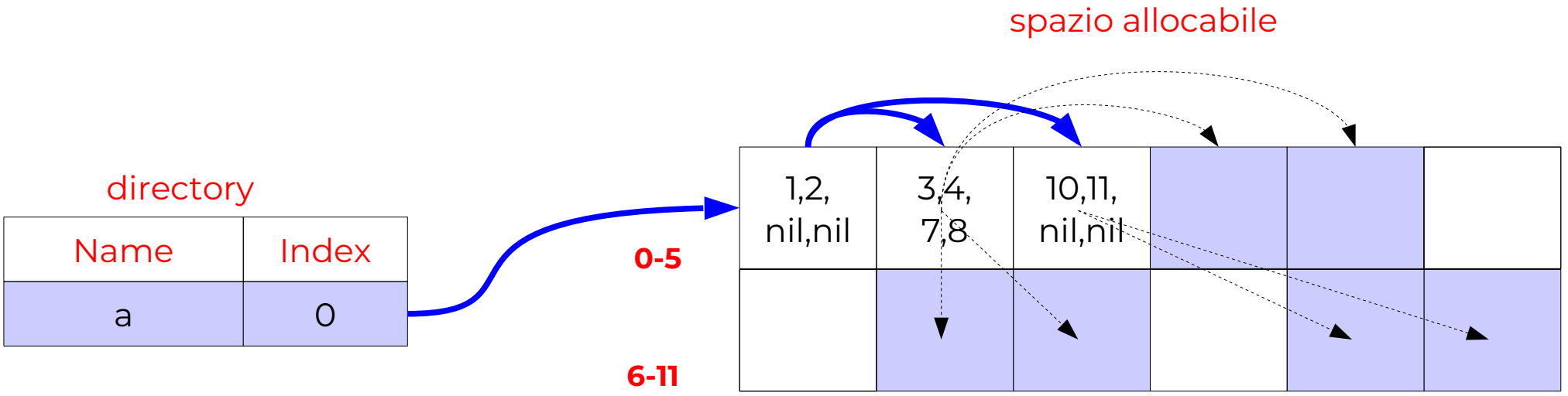
Allocazione indicizzata - Possibili soluzioni

- **Concatenazione di blocchi indice**
  - l'ultimo elemento del blocco indice non punta al blocco dati ma al blocco indice successivo
  - si ripropone il problema per l'accesso diretto a file di grandi dimensioni



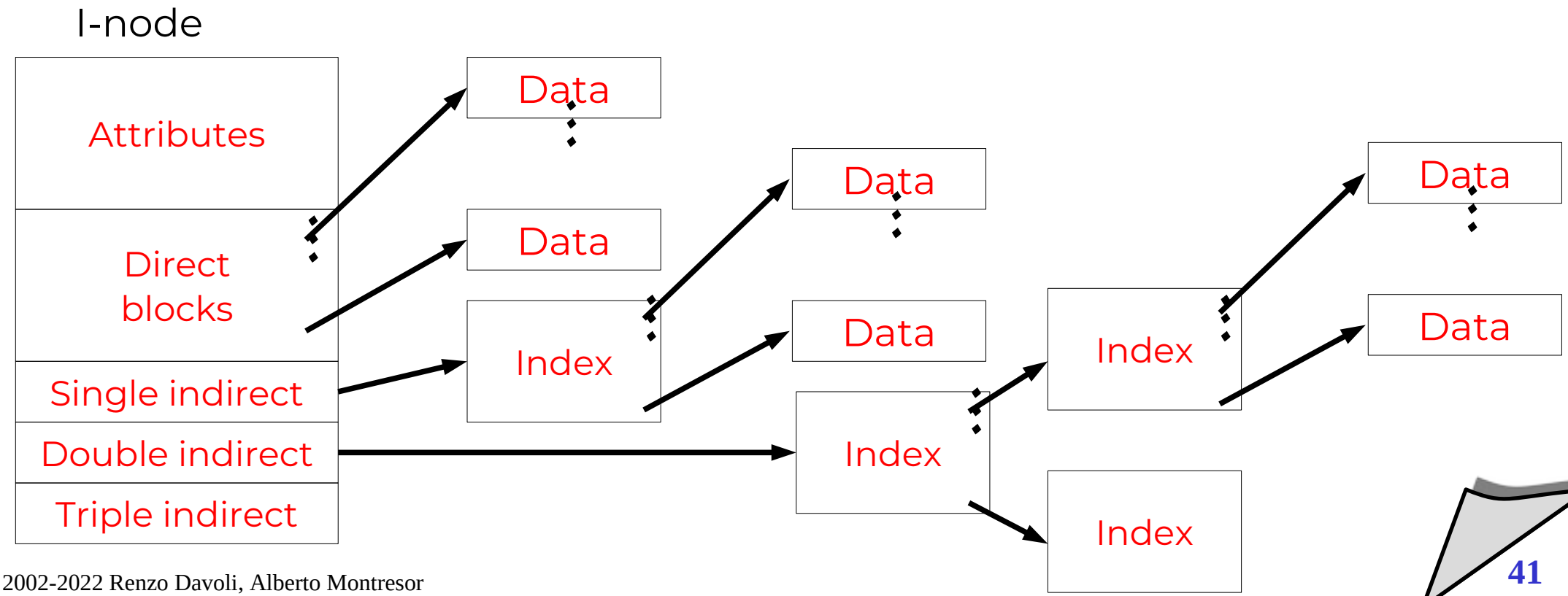
Allocazione indicizzata - Possibili soluzioni

- **Indice multilivello**
  - si utilizza un blocco indice dei blocchi indice
  - degradano le prestazioni, in quanto richiede un maggior numero di accessi



Allocazione indicizzata e UNIX

- **In UNIX**
  - ogni file è associato ad un i-node (index node)
  - un i-node è una struttura dati contenente gli attributi del file, e un indice di blocchi diretti e indiretti, secondo uno schema misto

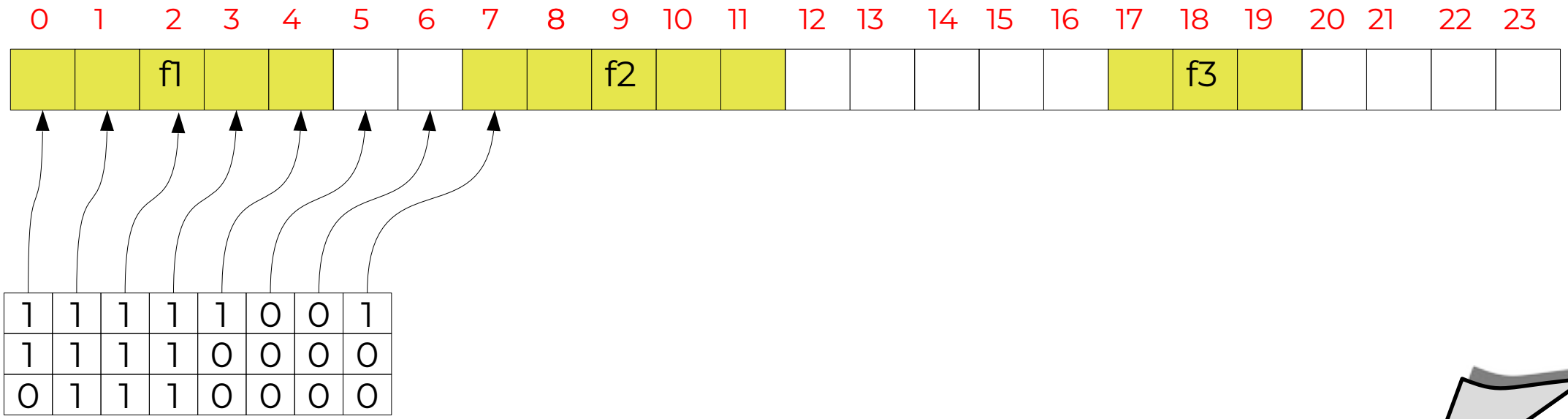


Allocazione e performance

- **Lo schema UNIX**
  - garantisce buone performance nel caso di accesso sequenziale
  - file brevi sono acceduti più velocemente e occupano meno memoria
- **Ulteriori miglioramenti**
  - pre-caricamento (per esempio nell'allocazione concatenata fornisce buone prestazioni per l'accesso sequenziale).
  - combinazione dell'allocazione contigua e indicizzata
    - contigua per piccoli file ove possibile
    - indicizzata per grandi file
    - Indicizzata di chunk (sottosequenze) contigue (ext4)

Gestione spazio libero - Mappa di bit

- **Descrizione**
  - ad ogni blocco corrisponde un bit in una bitmap
  - i blocchi liberi sono associati ad un bit di valore 0, i blocchi occupati sono associati ad un bit di valore 1

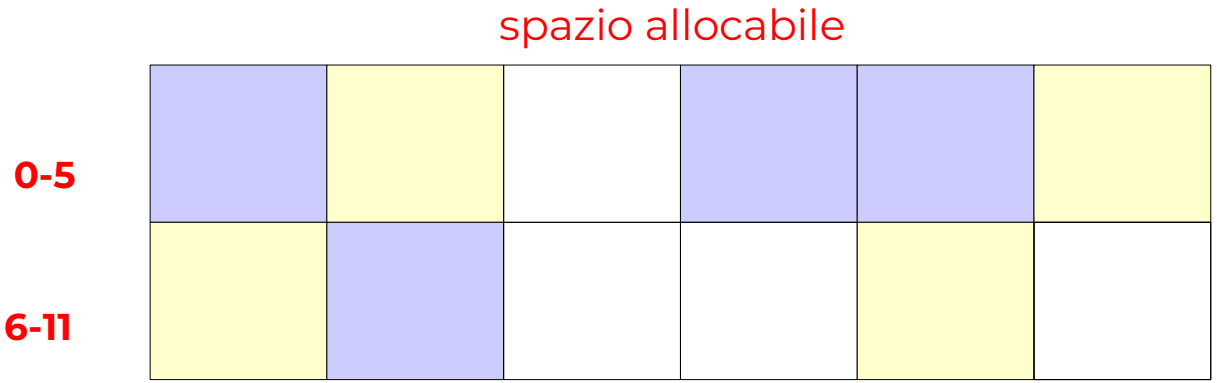


Gestione spazio libero - Mappa di bit

- **Vantaggi**
  - semplice, è possibile selezionare aree contigue
- **Svantaggi**
  - la memorizzazione del vettore può richiedere molto spazio
- **Nota:**
  - Intel 80386 e succ. Motorola 68020 e succ. hanno istruzioni per trovare l'offset del primo bit acceso/spento in una word

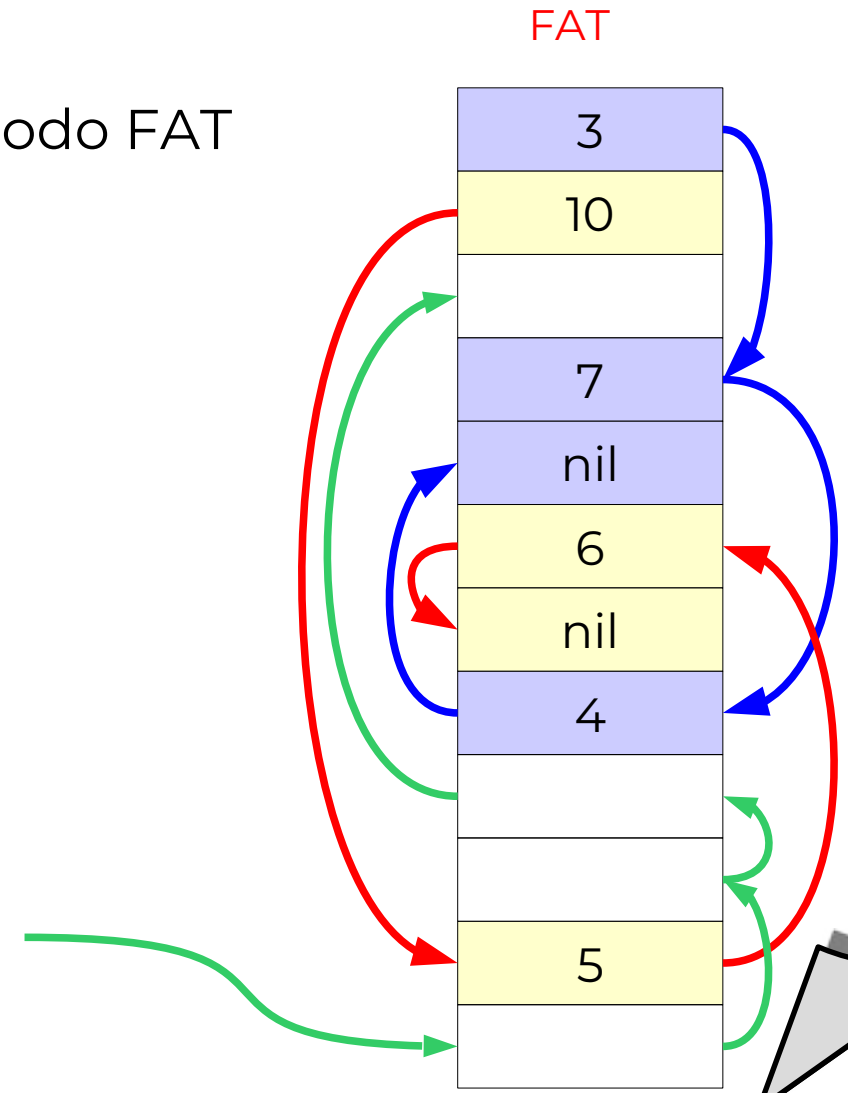
Gestione spazio libero - Lista concatenata

- **Descrizione**
  - i blocchi liberi vengono mantenuti in una lista concatenata
  - si integra perfettamente con il metodo FAT per l'allocazione delle aree libere



directory

Name	Start	End
a	0	4
b	1	6

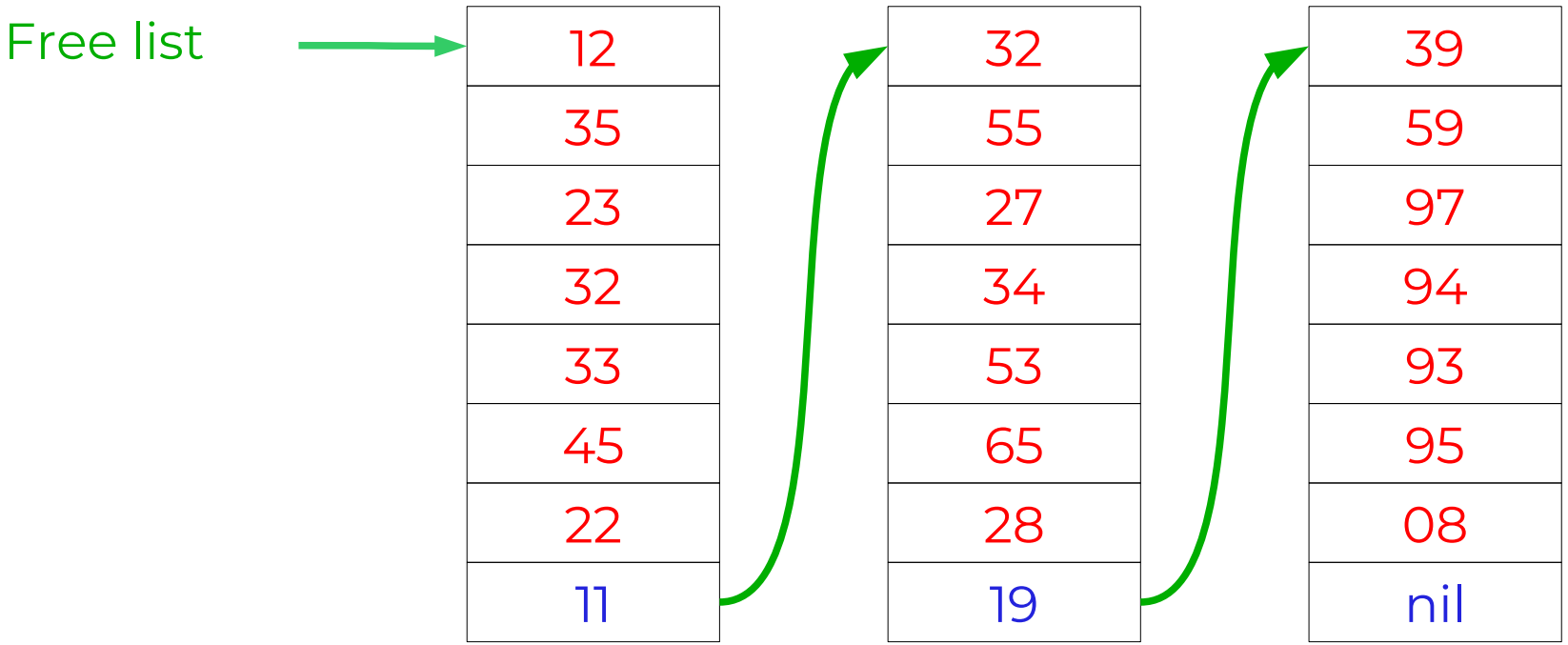


Gestione spazio libero - Lista concatenata

- **Vantaggi**
  - richiede poco spazio in memoria centrale
- **Svantaggi**
  - l'allocazione di un'area di ampie dimensioni è costosa
  - l'allocazione di aree libere contigue è molto difficoltosa

Gestione spazio libero - Lista concatenata (blocchi)

- **Descrizione**
  - è costituita da una lista concatenata di blocchi contenenti puntatori a blocchi liberi



Gestione spazio libero - Lista concatenata (blocchi)

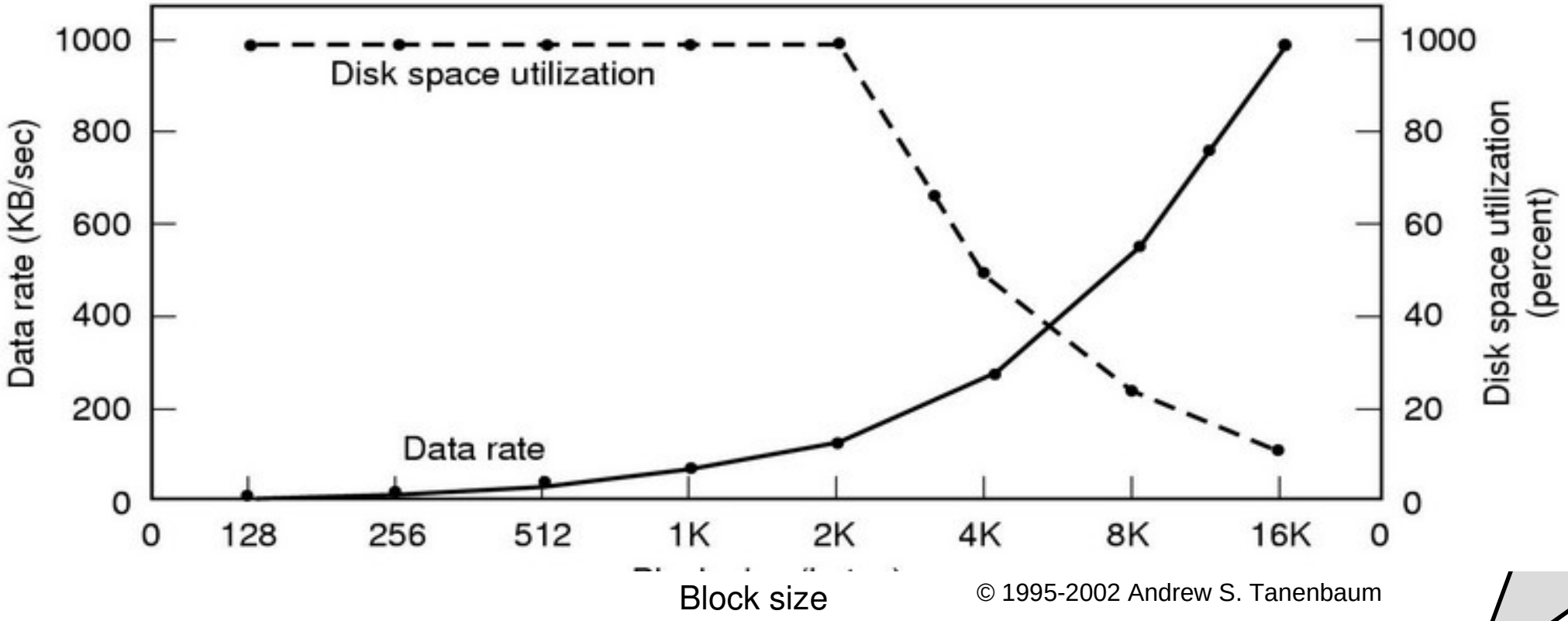
- **Vantaggi**
  - ad ogni istante, è sufficiente mantenere in memoria semplicemente un blocco contenente elementi liberi
  - non è necessario utilizzare una struttura a dati a parte; i blocchi contenenti elenchi di blocchi liberi possono essere mantenuti all'interno dei blocchi liberi stessi
- **Svantaggi**
  - l'allocazione di un'area di ampie dimensioni è costosa
  - l'allocazione di aree libere contigue è molto difficoltosa

Gestione spazio libero: un po' di conti

- **Premesse**
  - blocchi: 1K
  - dimensione partizione: 16GB
- **Mappa di bit**
  - 2^{24} bit = 2^{21} byte = 2048 blocchi = 2MB nella bitmap
- **Lista concatenata (FAT)**
  - 2^{24} puntatori = 2^{24}*3 byte = 48MB nella FAT
- **Lista concatenata (blocchi)**
  - spazio occupato nei blocchi liberi

Gestione spazio libero: un po' di conti

- **Scegliere la dimensione di un cluster**
  - cluster grandi: velocità di lettura alta, frammentazione interna
  - cluster piccoli: minore frammentazione, velocità più bassa
  - dati ottenuti su file system reali, lunghezza mediana: 2Kb



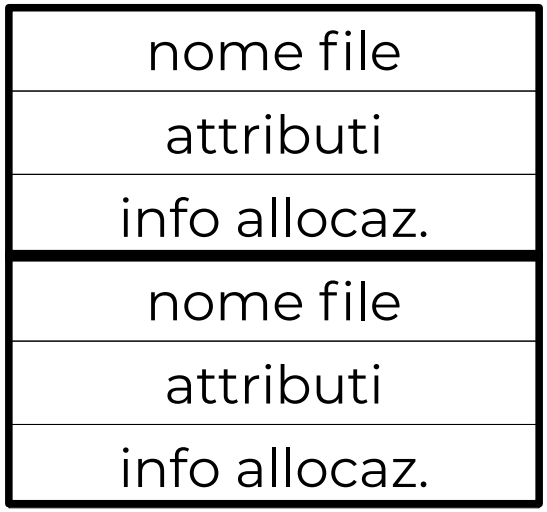
© 1995-2002 Andrew S. Tanenbaum

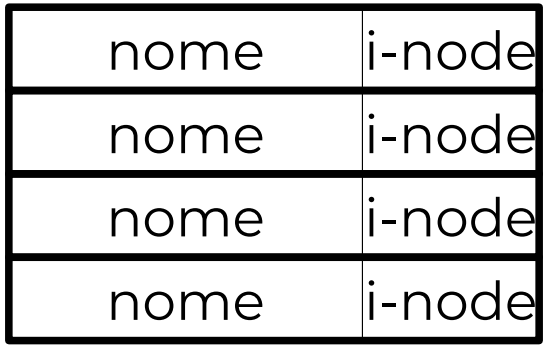
Implementazione delle directory

- **Una directory**
  - è un *file speciale* contenente informazioni sui file contenuti nella directory
  - una directory è suddivisa in un certo numero di *directory entry*
  - ogni directory entry deve permettere di accedere a tutte le informazioni necessarie per gestire il file
    - nome
    - attributi
    - informazioni di allocazione
- **Scelte implementative da considerare**
  - attributi contenuti nelle directory entry oppure nell'i-node
  - lista lineare (array) vs hash table

Implementazione delle directory

- **Informazioni nelle directory entry**
  - la directory entry contiene tutte le informazioni necessarie associate ad un file
  - utilizzata da MS-DOS
- **Informazione negli i-node**
  - le informazioni sono contenute negli inode; una directory entry contiene un indice di inode
  - utilizzata in UNIX

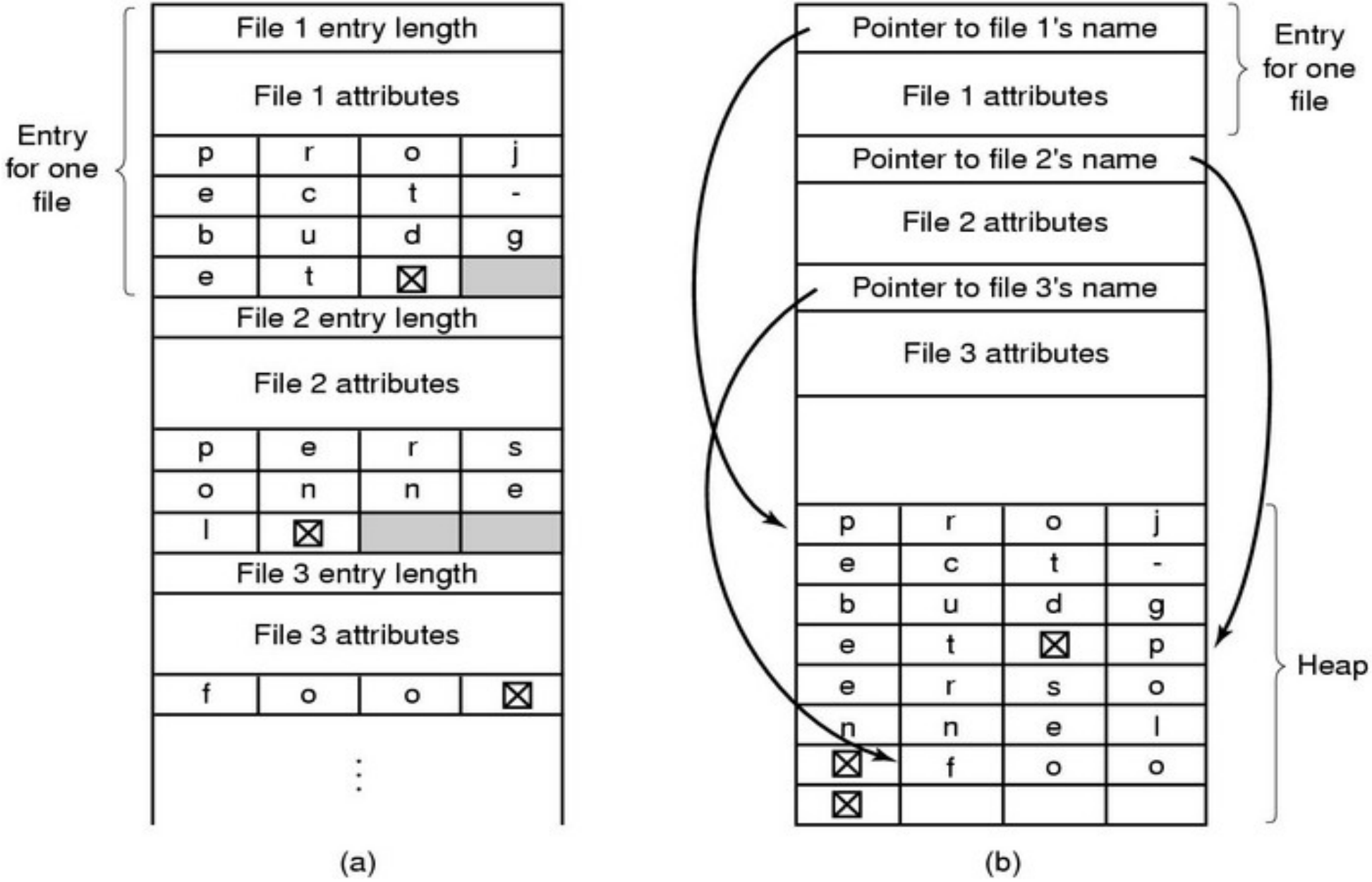




Implementazione delle directory

- **Problema della lunghezza dei nomi**
  - memorizzare nomi a lunghezza variabile nelle directory comporta una serie di problemi
- **Lunghezza fissa**
  - è il meccanismo più semplice...
  - ...ma presenta un trade-off:
    - spazio riservato molto grande, spreco di memoria
    - spazio riservato troppo piccolo, nomi troppo brevi
- **Lunghezza variabile**
  - la struttura dati diviene più complessa
  - due possibili esempi nel lucido successivo

Implementazione della directory



Implementazione delle directory

- **Lista lineare**
  - semplice da implementare
  - inefficiente nel caso di directory di grandi dimensioni
- **Tabella hash**
  - occorre stabilire la dimensione della tabella di hash e il metodo di gestione delle collisioni
  - la gestione può essere inefficiente se ci sono numerose collisioni

Directory strutturata a grafo aciclico

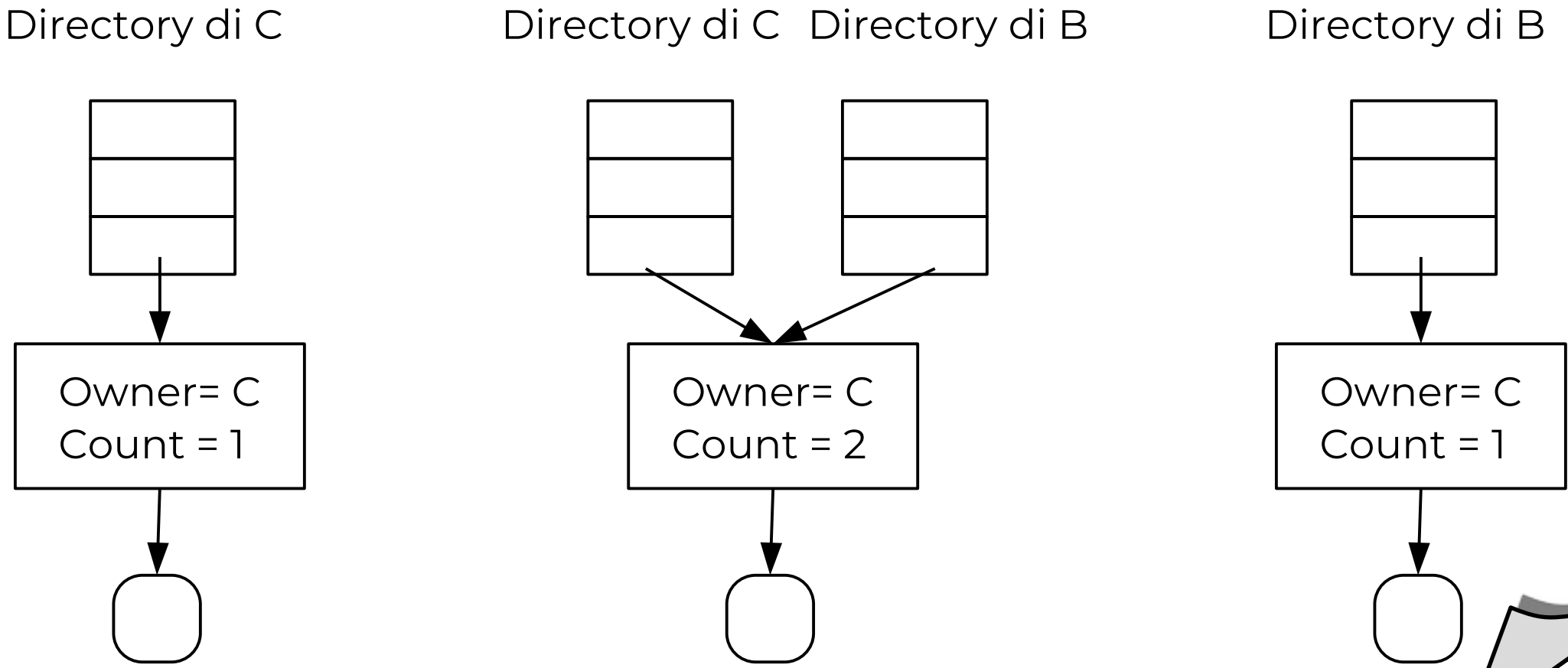
- **Due implementazioni possibili**
  - link simbolici
  - hard link
- **Link simbolici**
  - viene creato un tipo speciale di directory entry, che contiene un riferimento (sotto forma di cammino assoluto) al file in questione
  - quando viene fatto un riferimento al file
    - si cerca nella directory
    - si scopre che si tratta di un link
    - viene *risolto* il link
      (ovvero, viene utilizzato il cammino assoluto registrato nel file

Directory strutturata a grafo aciclico

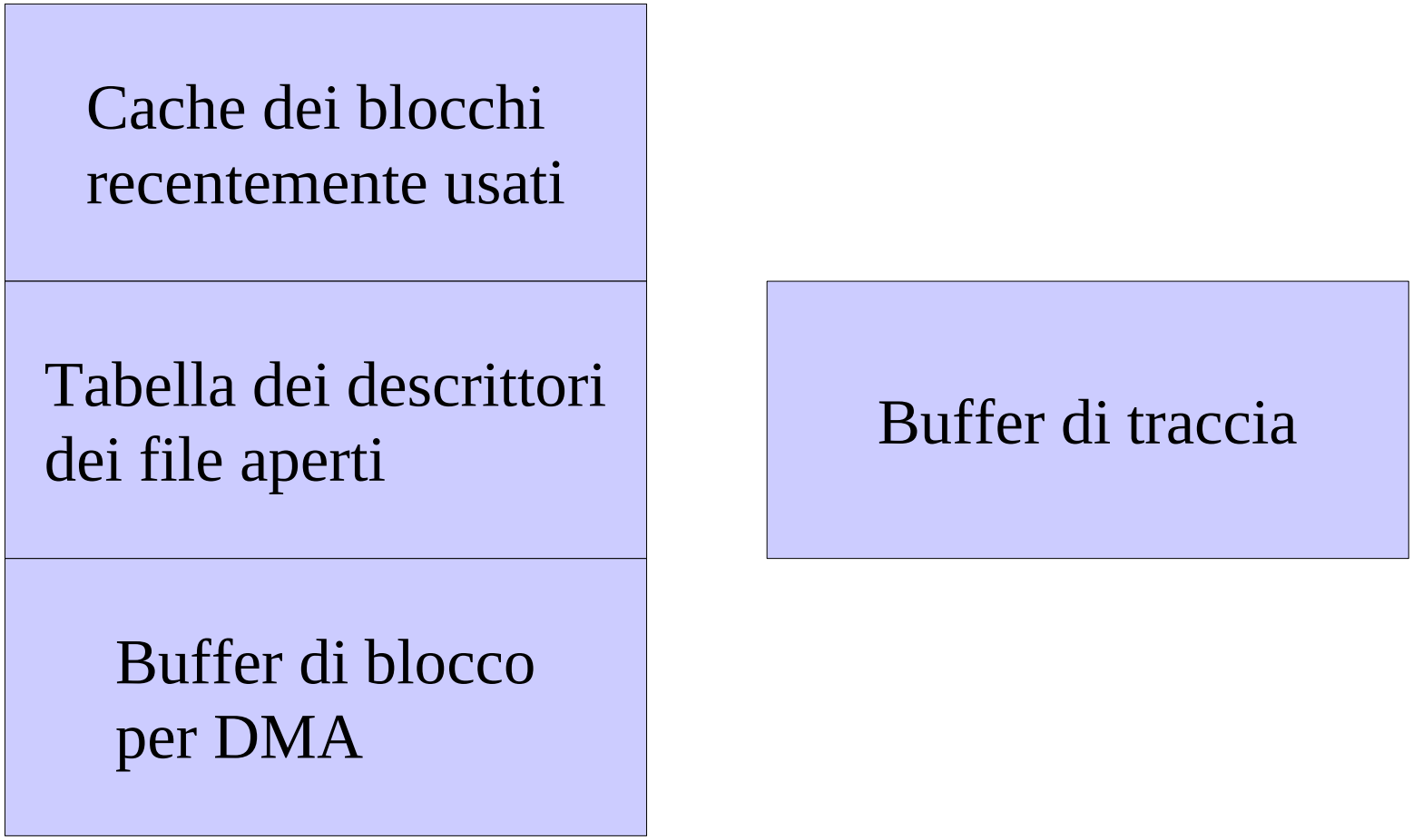
- **Hard link**
  - le informazioni relative al file sono presenti, uguali, in entrambe le directory
  - non è necessario una doppia ricerca nel file system
  - è impossibile distinguere la copia dall'originale
- **Considerazioni**
  - una struttura a grafo diretto aciclico è più flessibile di un albero, ma crea tutta una serie di problemi nuovi
  - non tutti i file system sono basati su DAG; esempio MS-DOS non li supporta

Hard-link

- **Implementazione**
  - E' necessario utilizzare la tecnica degli i-node
  - gli i-node devono contenere un contatore di riferimenti



- **Tecniche per migliorare le performance dei file system**



Tecniche per garantire la coerenza

- **Problema**
  - i meccanismi di caching possono causare inconsistenze nel file system
    - ad esempio, a causa di interruzioni di corrente
    - il problema è particolarmente critico per zone del disco contenenti FAT, bitmap, i-node, directory
- **Due possibili soluzioni:**
  - curare (file system checker)
    - fsck, scandisk
  - prevenire (journaling file system)
    - ext3, reiserfs

Controlli di coerenza: fsck

- **Phase 1: Check Blocks and Sizes**
  - Scandisce la tabella degli I-node: controlla le incoerenze
- **Phase 2: Check Path-Names - checks directory**
  - Controlla dir: devono puntare a inode legali
- **Phase 3: Check Connectivity**
  - Scandisce l'albero per vedere se tutti i file sono raggiungibili (L+F)
- **Phase 4: Check Reference Counts**
  - Verifica il numero di riferimenti ad ogni file
- **Phase 5: Check Cylinder Group**
  - Verifica i-node e blocchi liberi-occupati
- **Phase 6: Salvage Cylinder Groups**
  - Aggiorna le tabelle per salvare i cambiamenti

File system basati su log

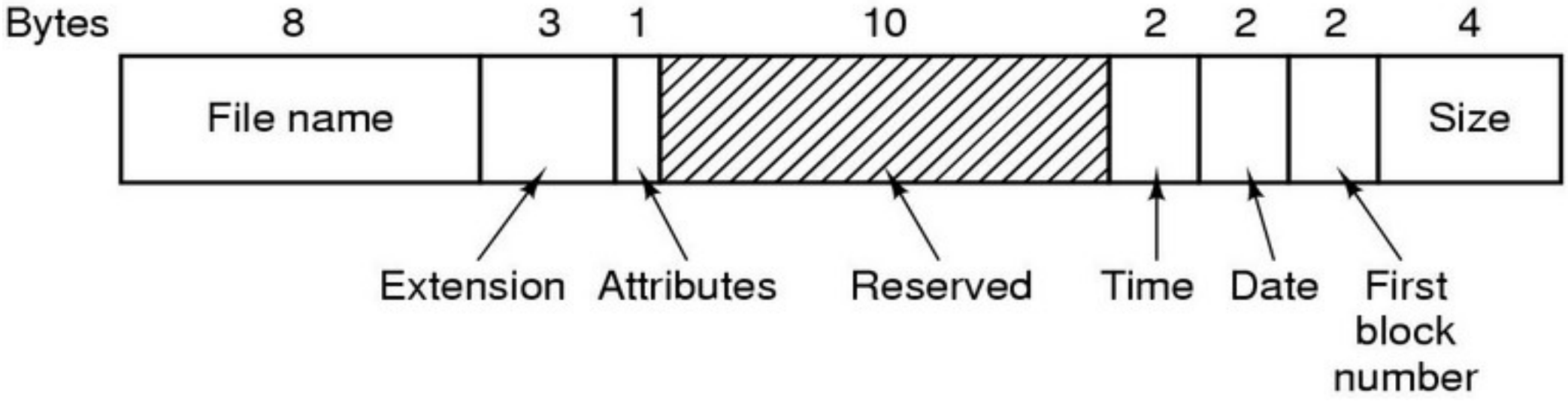
- **File system basati su log (o di tipo journaling)**
  - ogni aggiornamento al file system è trattato come una *transazione*
  - una transazione è un'operazione che viene eseguita in modo atomico: o tutto, o niente
- **Consente di ripristinare rapidamente uno stato coerente**

File system basati su log

- **Log**
  - tutte le transazioni vengono memorizzate in un log
  - una transazione
    - si considera completata (committed) quando è stata memorizzata nel log
    - tuttavia, il file system può non essere ancora aggiornato
  - periodicamente, le transazioni nel log vengono effettuate nel file system
  - alla modifica del file system, la transazione viene rimossa dal log
  - In caso di errore (e.g. mancanza di corrente) sistema, tutte le transazioni nel log devono essere ripetute

Esempio: MS-DOS - FAT

- **Nomi file:**
  - 8+3 caratteri
  - indici di blocco a 12-16 bit (MS-DOS, W95), 32 bit (W98)

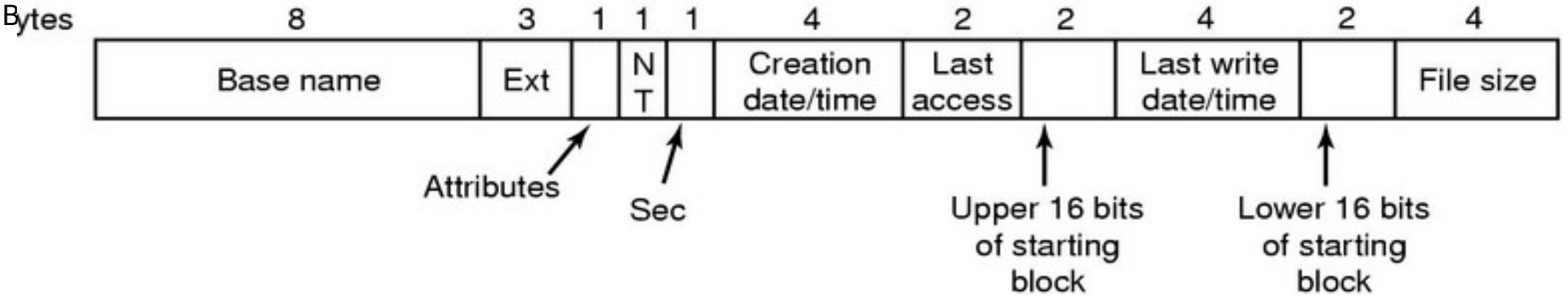


Esempio: MS-DOS

Dimensione blocco/cluster	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8MB	128 MB	
4 KB	16MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Esempio: Windows 9x

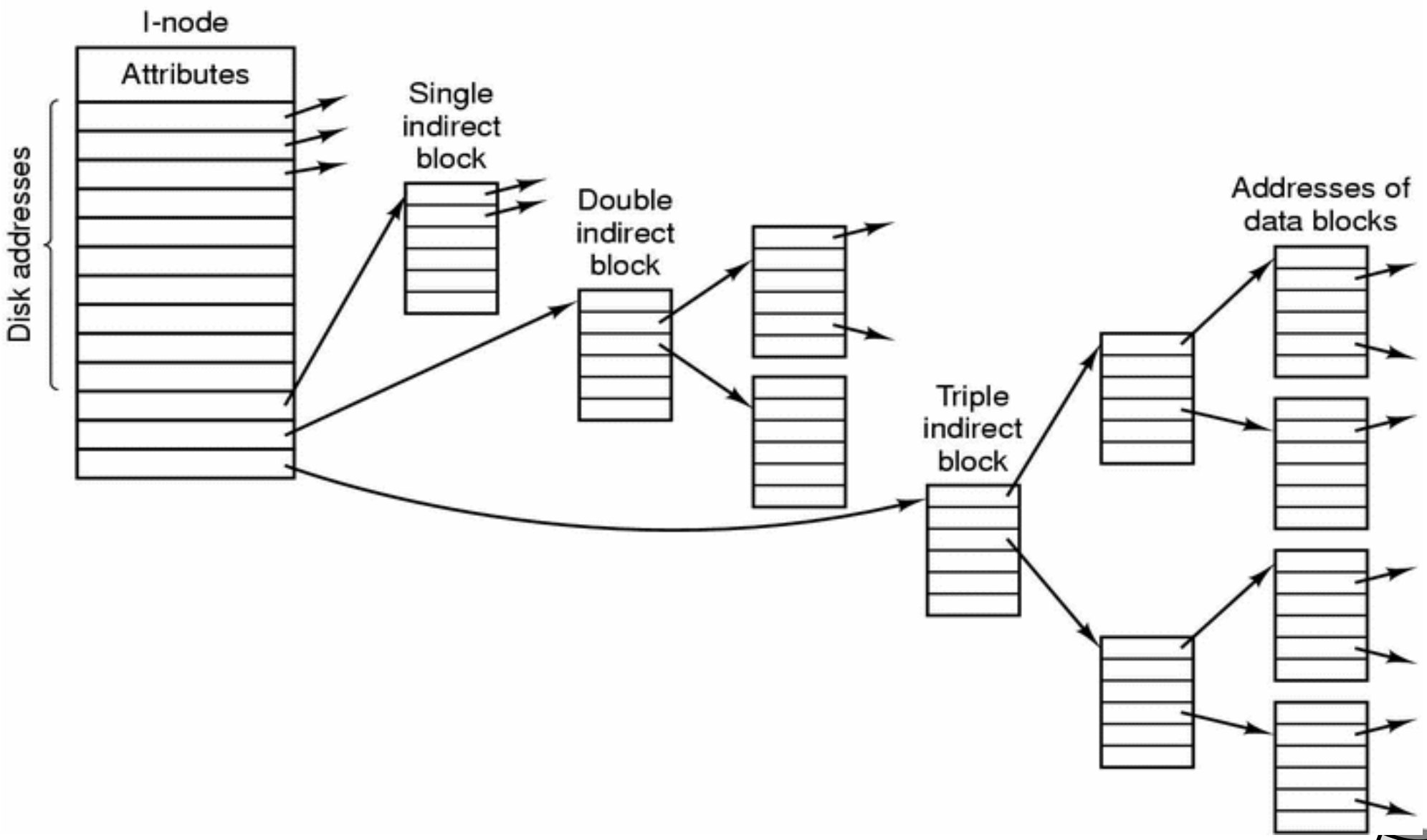
Struttura di una directory entry "compatibile con MSDOS"



68	d o g			A	0	CK					0		
3	o v e			A	0	CK	t h e		l a		0	z	y
2	w n	f o		A	0	CK	x	j u m p			0	s	
1	T h e	q		A	0	CK	u i c k	b			0	r o	
T	H E Q U I ~ 1			A	NT	S	Creation time		Last acc	Upp	Last write	Low	Size

Bytes

Esempio: UNIX V7



Esempio: UNIX V7