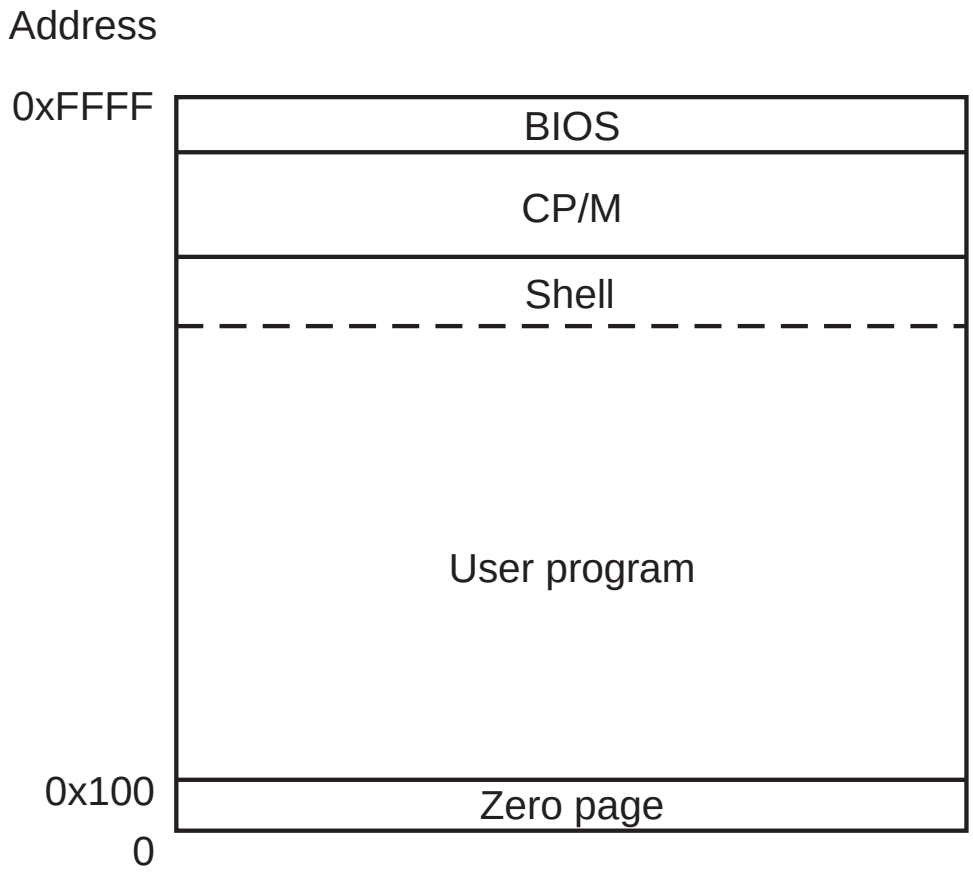
Esempi di File System

CP/M

- Il sistema CP/M può essere considerato l'antenato di MS-DOS
- CP/M era un sistema operativo per macchine con processori a 8 bit e 4KB di RAM e un singolo floppy disk di 8 pollici con capacità di 180 KB.
- CP/M era scarno e molto compatto e rappresenta un interessante esempio di sistema embedded
- Quando caricato nella RAM:
  - Il BIOS contiene una libreria di 17 chiamate di sistema (interfaccia harware)
  - Il sistema operativo occupa meno di 3584 byte ($<$ 4KB!); la shell occupa 2 KB

– Ultimi 256 byte: vettore interruzioni, buffer per la linea di comando



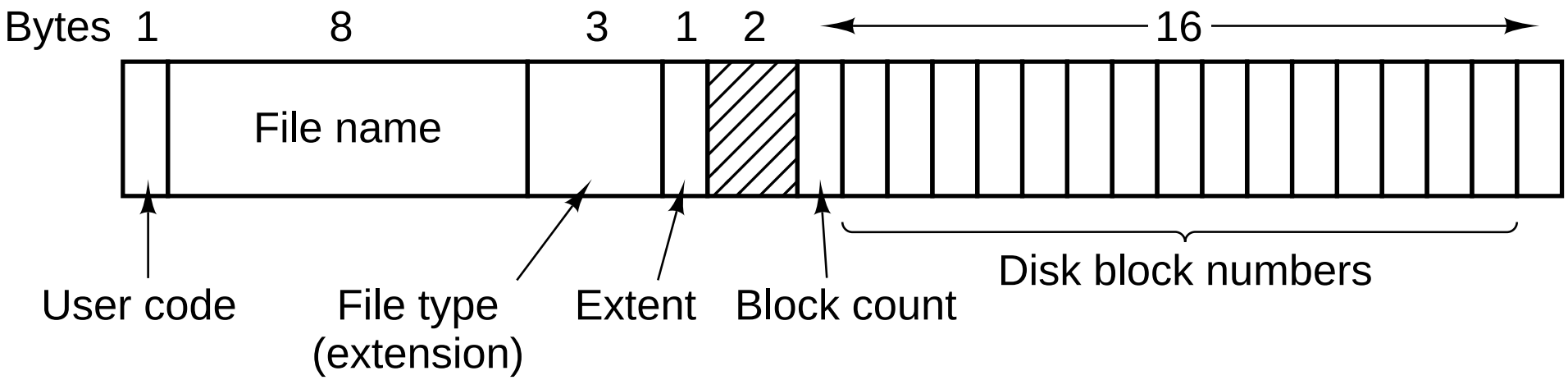
- Comandi vengono copiati nel buffer, poi CP/M cerca il programma da eseguire, lo scrive a partire dall'indirizzo 256 e gli passa il controllo

- Il programma può scrivere sopra la shell se necessario

File System in CP/M

- CP/M ha una sola directory (flat)
- Gli utenti si collegavano uno alla volta: i file avevano informazioni sul nome del proprietario
- Dopo l'avvio del sistema CP/M legge la directory e calcola una mappa di bit (di 23 byte per un disco da 180KB) per i blocchi liberi
- La mappa viene tenuta in RAM e buttata via quando si spegne la macchina

Elementi di directory in CP/M

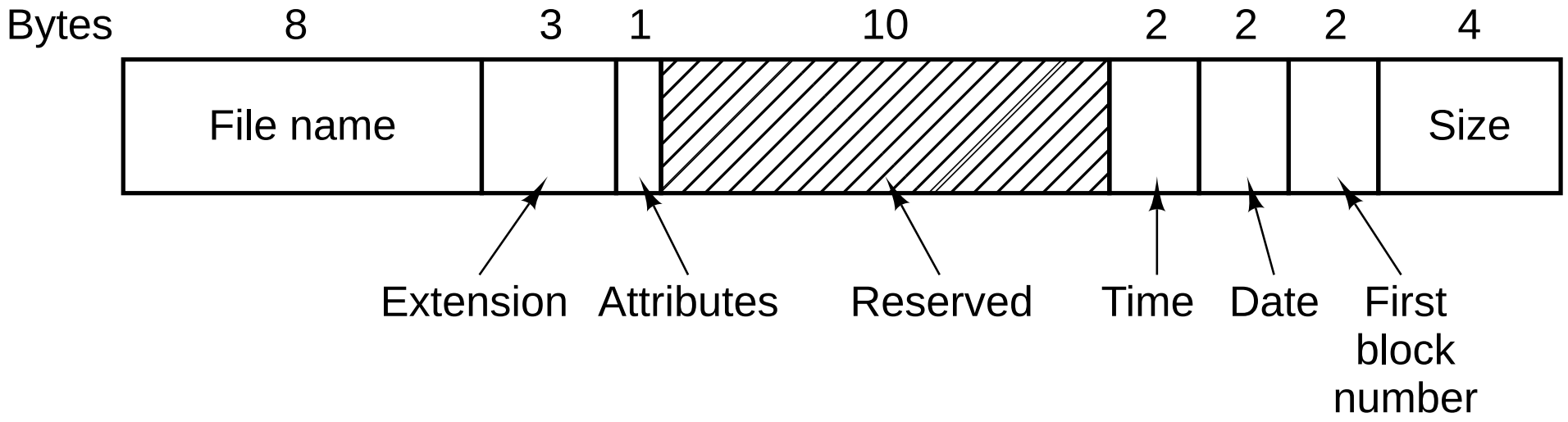


- Lunghezza del nome fissa: 8 caratteri + 3 di estensione
- Extent: serve per file con più di 16 blocchii: si possono usare più elementi di directory per lo stesso file, extent mantiene l'ordine con cui leggere i blocchi
- Contatore blocchi: numero complessivo di blocchi (non dimensione del file: serve EOF nell'ultimo blocco)
- Blocchi: dimensione da 1KB

File System in MS-DOS

- MS-DOS è una evoluzione di CP/M
- Le directory hanno struttura ad albero e non flat
- Si usa la File Allocation Table (FAT) per la mappa file blocchi e la gestione dei blocchi liberi (sono marcati in maniera speciale)
- Non è tuttavia multi utente

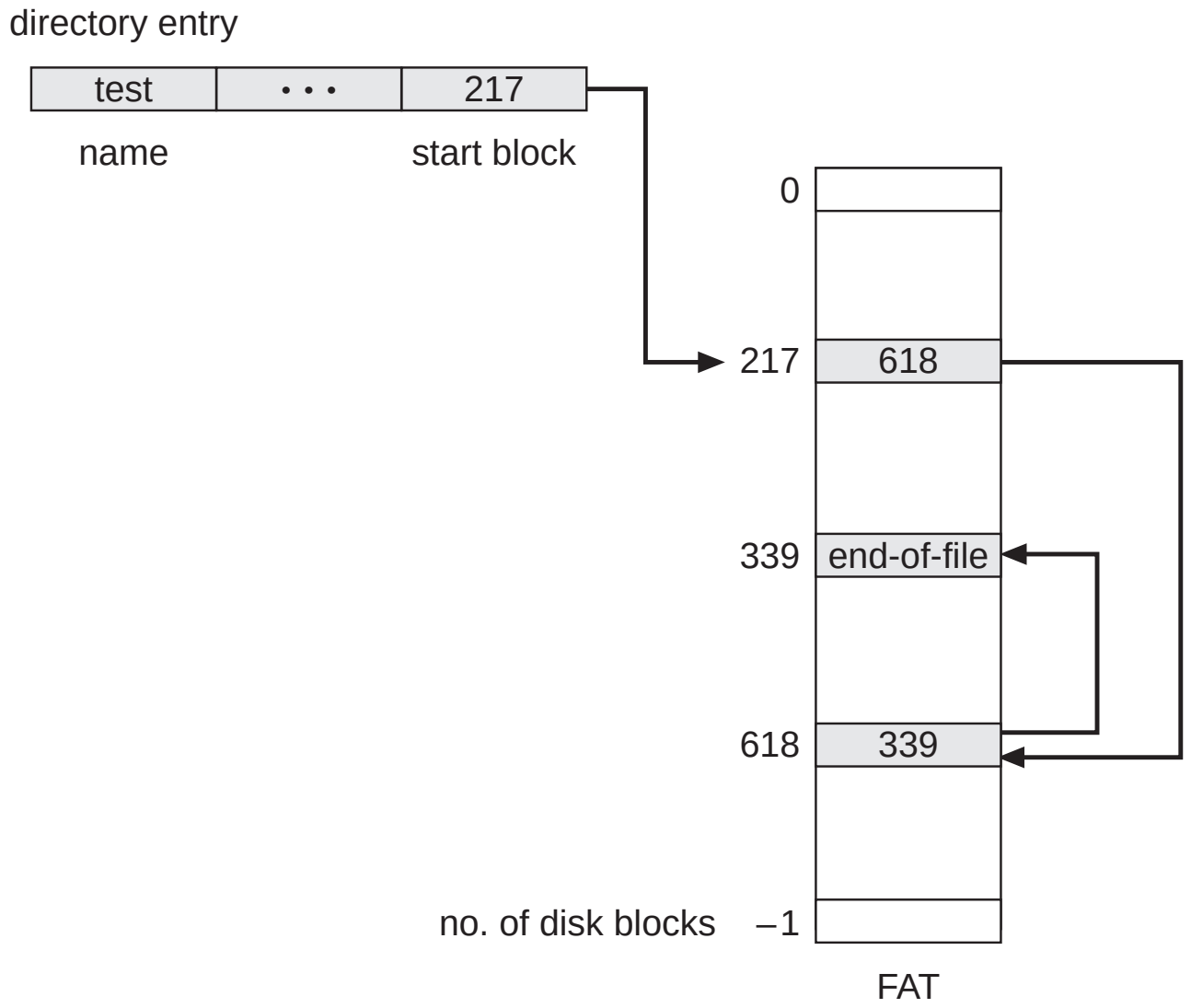
Directory in MS-DOS



- Lunghezza del nome fissa
- Attributi: read-only, system, archived, hidden
- Time: ore (5bit), min (6bit), sec (5bit)
- Date: giorno (5bit), mese (4bit), anno-1980 (7bit) (Y2108 BUG!)

File-allocation table (FAT)

Mantiene la linked list in una struttura dedicata, all'inizio di ogni *partizione* del disco



FAT12

- La prima versione di DOS usava una FAT-12 (cioé con indirizzi di disco di 12 bit) con blocchi da 512 byte
- Dimensione max di una partizione: $2^{12}x512$ byte ($2MB$)
- Dimensione della FAT: 4096 elementi da 2 byte
- Ok per floppy ma non per hard disk
- Con blocchi da 4KB si ottenevano partizioni da 16MB (con 4 partizioni: dischi da 64MB)

FAT16

- FAT-16 utilizza indirizzi di disco da 16 bit con blocchi da 2KB a 32KB
- Dimensione max di una partizione: $2GB$
- Dimensione della FAT: 128KB

FAT32

- A partire dalla seconda versione di Windows 95 si utilizzano indirizzi di 28 bit (non 32)

- Partizioni max: $2^{28} \times 32KB (= 2^{15})$ in realtà limitate a $2TB$ $(= 2048TB)$

- Vantaggi rispetto a FAT-16: un disco da 8GB può avere una sola partizione (su FAT 16 deve stare su 4 partizioni)

- Può usare blocchi di dimensione più piccola per coprire uguale spazio disco ($2GB$ FAT-16 deve usare blocchi da $32KB$ mentre FAT-32 può usare blocchi da $4KB$

FAT12, FAT16, FAT32

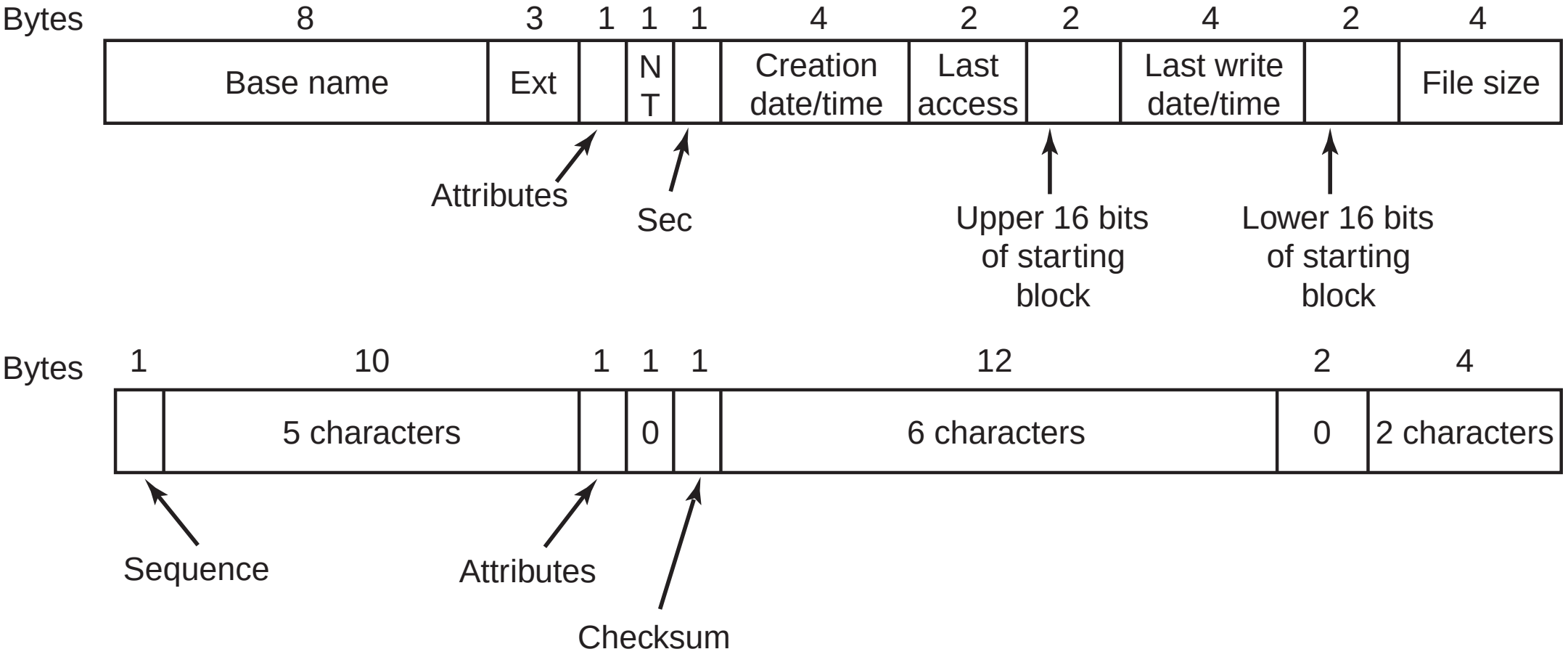
Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

In MS-DOS, tutta la FAT viene caricata in memoria.

Il block size è chiamato da Microsoft *cluster size*

Directory in Windows 98

Nomi lunghi ma compatibilità all'indietro con MS-DOS e Windows 3

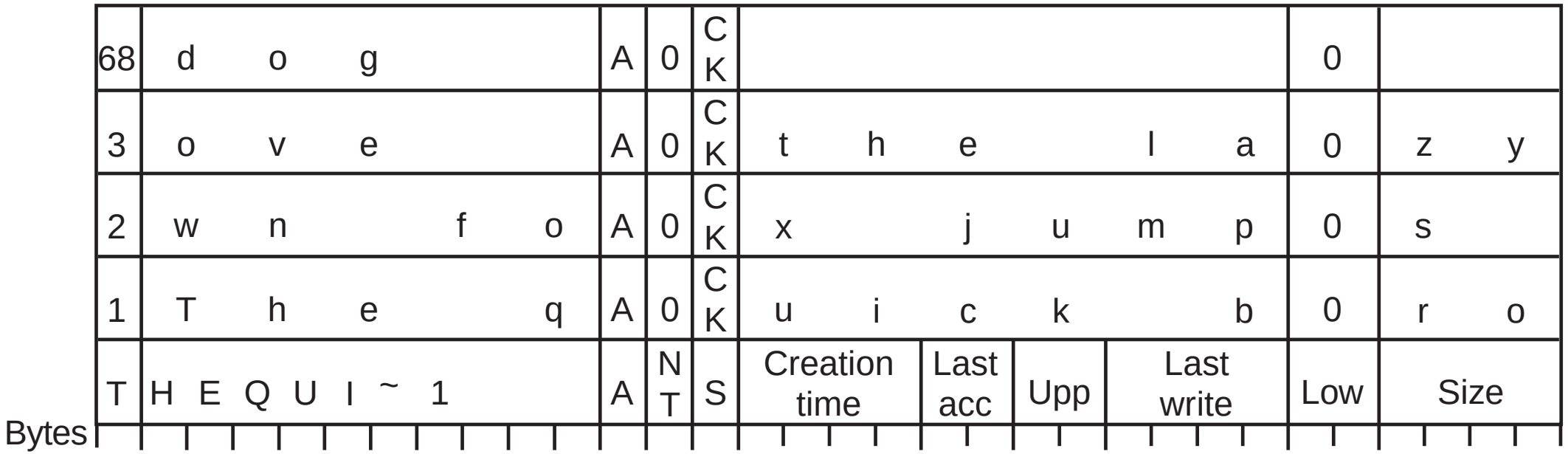


FAT32

- NT = compatibilita' con Windows NT (gestione lettere maiuscole/minuscole)
- Sec = bit aggiuntivi per ora
- I record per i nomi lunghi hanno 0x0F come valore di attributes. Questo valore e' invalido per MS-DOS. Cioe' questi record non vengono letti in DOS.
- CK= checksum per controllare consistenza tra nome corto e nome lungo (ad esempio in caso di cancellazione del file in DOS)

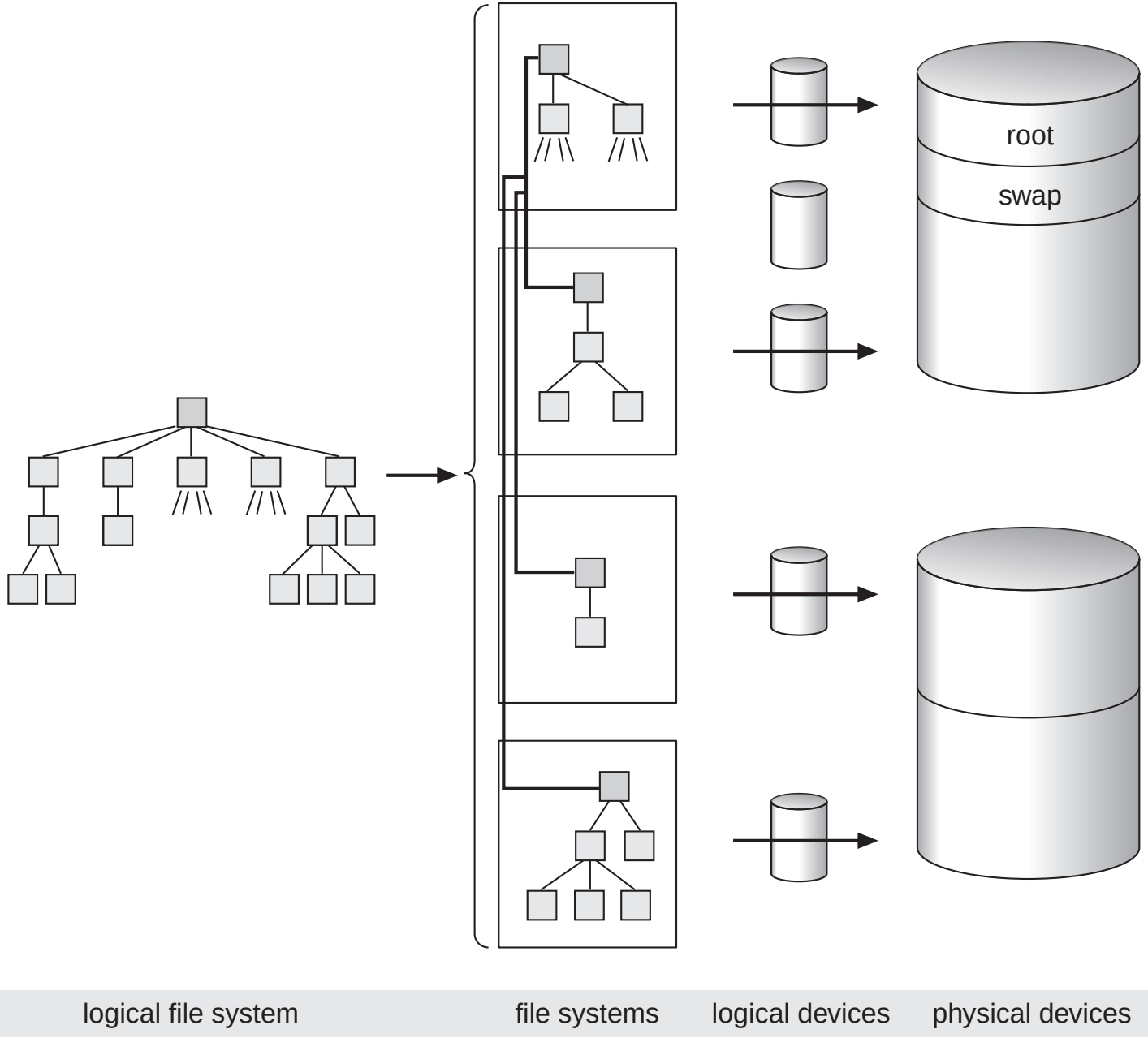
Esempio

```
$ dir
THEQUI~1         749 03-08-2000  15:38 The quick brown fox jumps over the...
```



UNIX: Il Virtual File System

Il file system *virtuale* che un utente vede può essere composto in realtà da diversi file system *fisici*, ognuno su un diverso dispositivo logico



Il Virtual File System (cont.)

- Il Virtual File System è composto da più file system fisici, che risiedono in dispositivi logici (*partizioni*), che compongono i dispositivi fisici (dischi)
- Il file system / viene montato al boot dal kernel
- gli altri file system vengono montati secondo la configurazione impostata
- ogni file system fisico può essere diverso o avere parametri diversi
- Il kernel usa una coppia <*logical device number, inode number*> per identificare un file
  - Il logical device number indica su quale file system fisico risiede il file
  - Gli inode di ogni file system sono numerati progressivamente

Il Virtual File System (cont.)

Il kernel si incarica di implementare una visione uniforme tra tutti i file system montati: operare su un file significa

- determinare su quale file system fisico risiede il file
- determinare a quale inode, su tale file system corrisponde il file
- determinare a quale dispositivo appartiene il file system fisico
- richiedere l'operazione di I/O al dispositivo

I File System Fisici di UNIX

- UNIX (Linux in particolare) supporta molti tipi di file system fisici (SysV, EXT2, EXT3 e anche MSDOS); quelli preferiti sono UFS (Unix File System, aka BSD Fast File System), EXT2 (Extended 2), EFS (Extent File System)

- Il file system fisico di UNIX supporta due oggetti:
  - file "semplici" (plain file) (senza struttura)
  - directory (che sono semplicemente file con un formato speciale)

- La maggior parte di un file system è composta da blocchi dati
  - in EXT2: 1K-4K (configurabile alla creazione)

Inodes

- Un file in Unix è rappresentato da un *inode* (nodo indice).
- Gli inodes sono allocati in numero finito alla creazione del file system
- Struttura di un inote in System V:

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

Inodes (cont)

- Gli indici indiretti vengono allocati su richiesta
- Accesso più veloce per file piccoli
- L'inode contiene puntatori diretti e indiretti (a 1, 2, e 3 livelli)
- N. massimo di blocchi indirizzabile: con blocchi da 4K, puntatori da 4byte

$$\begin{aligned} L_{max} &= 10 + 1024 + 1024^2 + 1024^3 \\ &> 1024^3 = 2^{30} blk \\ &= 2^{42} byte = 4\,TB \end{aligned}$$

  molto oltre le capacità dei sistemi a 32 bit.

Traduzione da file descriptor a inode

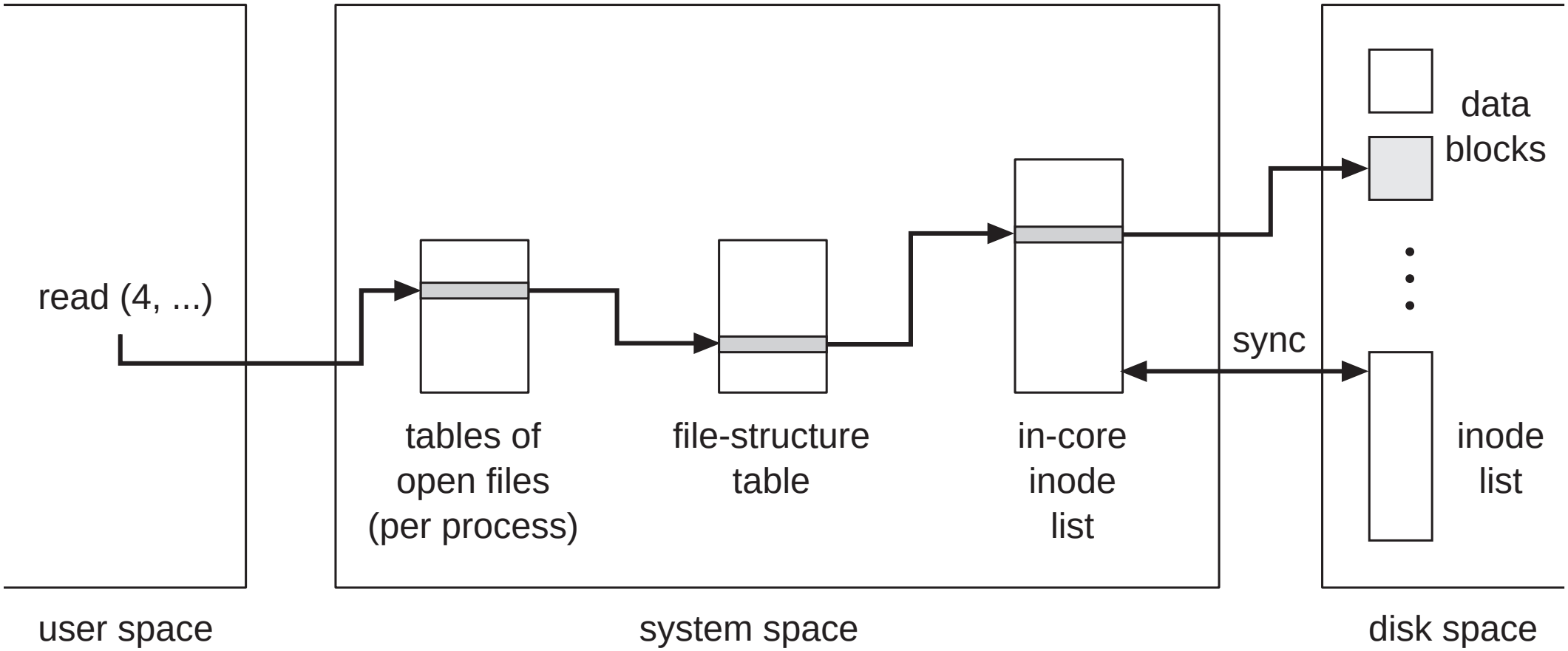
- Le system calls che si riferiscono a file aperti (read, write, close, ...) prendono un *file descriptor* come argomento

- Il file descriptor viene usato dal kernel per entrare in una tabella di file aperti del processo. Risiede nella U-structure.

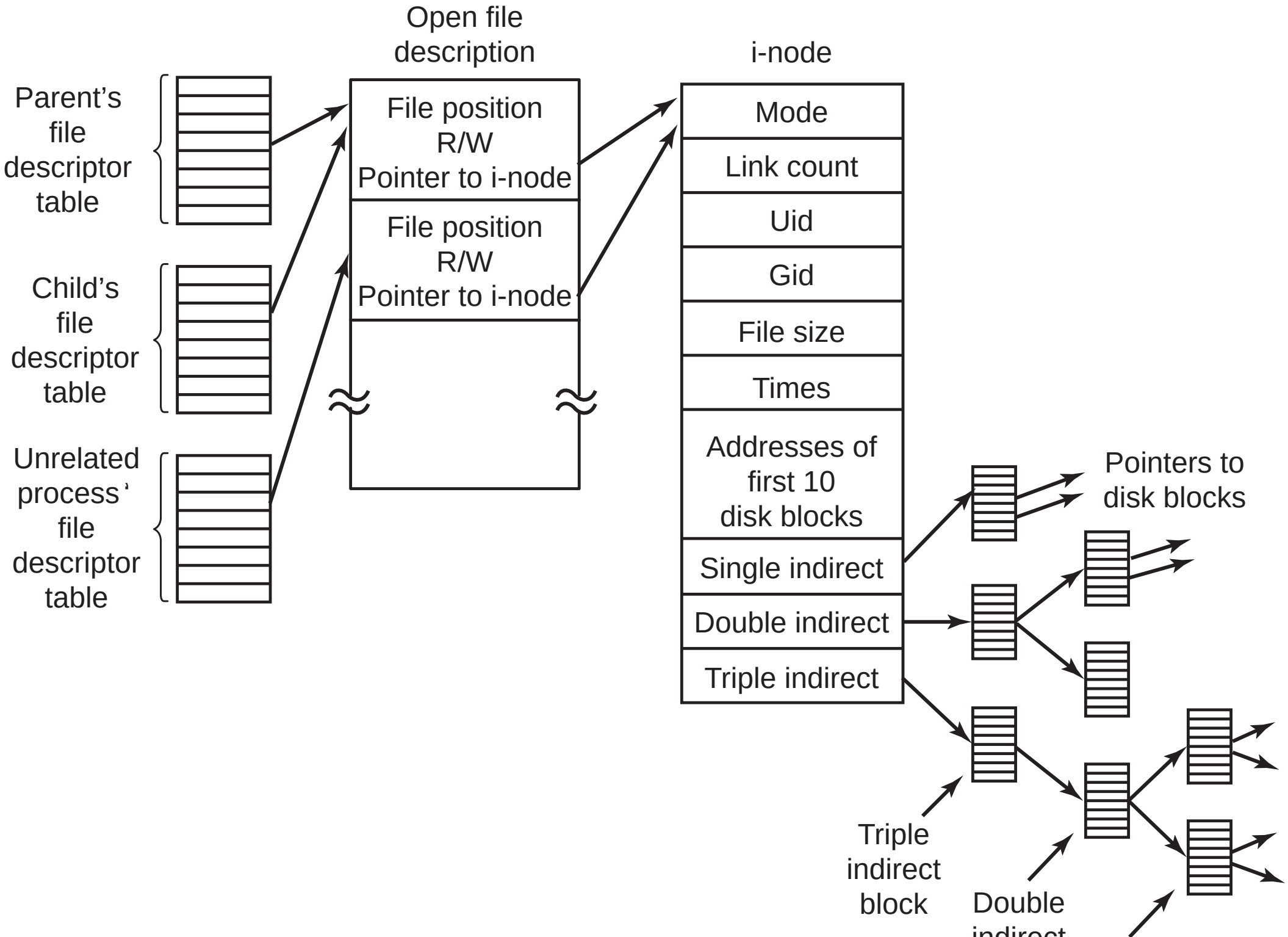
- Ogni entry della tabella contiene un puntatore ad una *file structure*, di sistema. Ogni file structure punta ad un inode (in un'altra lista), e contiene la posizione nel file.

- Ogni entry nelle tabelle contiene un contatore di utilizzo: quando va a 0, il record viene deallocato

File Descriptor, File Structure e Inode



La tabella intermedia è necessaria per la semantica della condivisione dei file tra processi


Open file description
i-node
Parent's file descriptor table
Child's file descriptor table
Unrelated process' file descriptor table
File position
R/W
Pointer to i-node
File position
R/W
Pointer to i-node
Mode
Link count
Uid
Gid
File size
Times
Addresses of first 10 disk blocks
Single indirect
Double indirect
Triple indirect
Pointers to disk blocks
Triple indirect block
Double indirect

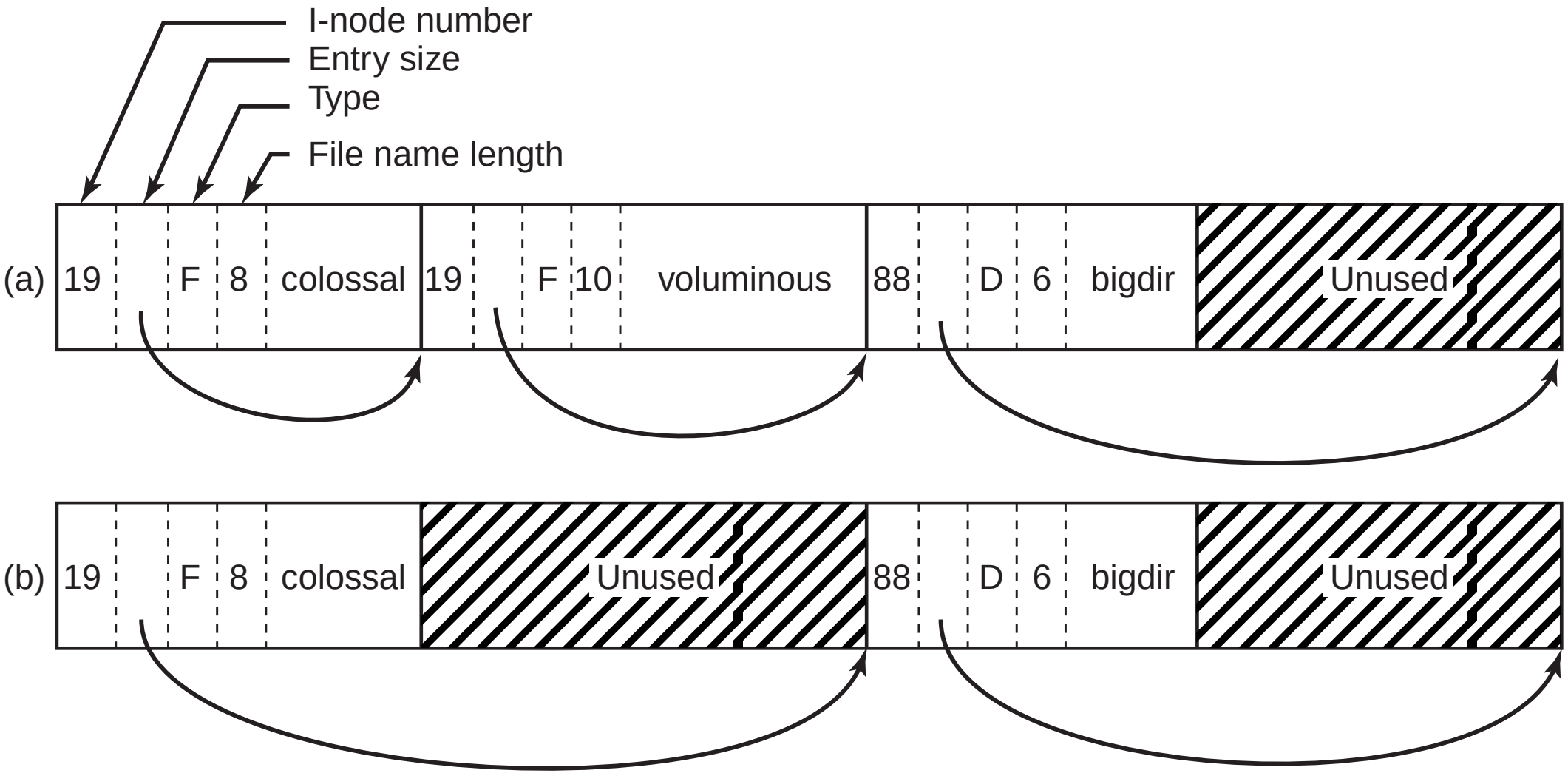
- Le chiamate di lettura/scrittura e la seek cambiano la posizione nel file

- Ad una *fork*, i figli ereditano (una copia de) la tabella dei file aperti dal padre $\Rightarrow$ condividono la stessa file structure e quindi la posizione nel file

- Processi che hanno aperto indipendentemente lo stesso file hanno copie private di file structure

Directory in UNIX

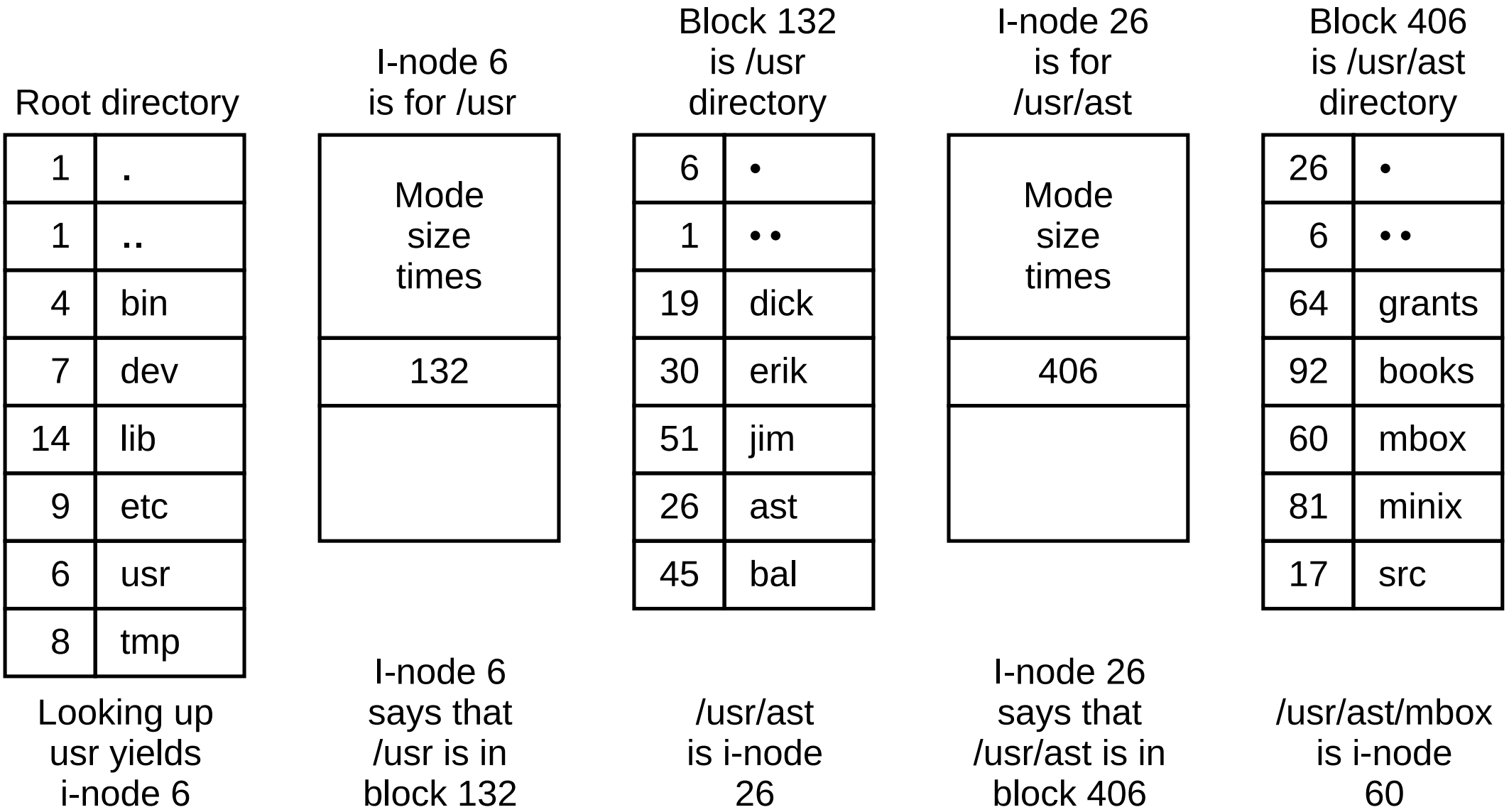
- Il tipo all'interno di un inode distingue tra file semplici e directory
- Una directory è un file con entry di lunghezza variabile. Ogni entry contiene
  - puntatore all'inode del file
  - posizione dell'entry successiva
  - lunghezza del nome del file (1 byte)
  - nome del file (max 255 byte)
- entry differenti possono puntare allo stesso inode (*hard link*)


I-node number
Entry size
Type
File name length
(a)
19
F
8
colossal
19
F
10
voluminous
88
D
6
bigdir
Unused
(b)
19
F
8
colossal
Unused
88
D
6
bigdir
Unused

Traduzione da nome a inode

L'utente usa i nomi (o path), mentre il file system impiega gli inode $\Rightarrow$ il kernel deve risolvere ogni nome in un inode, usando le directory

- Prima si determina la directory di partenza: se il primo carattere è "/", è la root dir (sempre montata); altrimenti, è la current working dir del processo in esecuzione
- Ogni sezione del path viene risolta leggendo l'inode relativo
- Si ripete finché non si termina il path, o la entry cercate non c'è
- Link simbolici vengono letti e il ciclo di decodifica riparte con le stesse regole. Il numero massimo di link simbolici attraversabili è limitato (8)
- Quando l'inode del file viene trovato, si alloca una *file structure* in memoria, a cui punta il *file descriptor* restituito dalla *open(2)*


Root directory
1 .
1 ..
4 bin
7 dev
14 lib
9 etc
6 usr
8 tmp
Looking up usr yields i-node 6
I-node 6 is for /usr
Mode size times
132
I-node 6 says that /usr is in block 132
Block 132 is /usr directory
6 •
1 ••
19 dick
30 erik
51 jim
26 ast
45 bal
/usr/ast is i-node 26
I-node 26 is for /usr/ast
Mode size times
406
I-node 26 says that /usr/ast is in block 406
Block 406 is /usr/ast directory
26 •
6 ••
64 grants
92 books
60 mbox
81 minix
17 src
/usr/ast/mbox is i-node 60

Esempio di file system fisico: UFS

- UFS (Unix File System) è un file system utilizzato in molti sistemi operativi Unix-like.

- È un derivato del Berkeley Fast File System, che fu a sua volta sviluppato a partire dal FS usato dalla prima versione di Unix realizzata presso i Bell Labs.

- Tutti i derivati di BSD, inclusi FreeBSD, NetBSD, OpenBSD, NEXTSTEP e Solaris, usano una variante di UFS.

- In Mac OS X è disponibile come alternativa a HFS.

- In Linux è disponibile un supporto parziale a UFS, mentre il suo filesystem nativo, EXT2, è anch'esso derivato da UFS.

Struttura di UFS

- In UFS i blocchi hanno due dimensioni: il *blocco* (4-8K) e il *frammento* (0.5-1K)
  - Tutti i blocchi di un file sono blocchi tranne l'ultimo
  - L'ultima parte del file è tenuta in frammenti
- Riduce la frammentazione interna e aumenta la velocità di I/O
- La dimensione del blocco e del frammento sono impostati alla creazione del file system:
  - se ci saranno molti file piccoli, meglio un frammenti piccoli, se ci saranno grossi file da trasferire spesso, meglio blocchi grandi
  - il rapporto max è 8:1. Tipicamente, 4K:512 oppure 8K:1K.

Partizioni UFS

- Una partizione di un filesystem UFS è composta dalle seguenti componenti:
  - un piccolo numero di blocchi posizionati all'inizio della partizione destinati al boot
  - un superblocco, contenente un magic number che identifica il filesystem come UFS ed altri parametri quali la geometria del filesystem
  - una serie di gruppi di cilindri (che corrispondono a gruppi consecutivi di cilindri del disco). Quando possibile, si allocano i blocchi nello stesso gruppo dell'inode.

UFS (Cont)

- Ogni gruppo di cilindri contiene:
  - Una copia di backup del superblocco
  - Un header per il gruppo di cilindri con informazioni quali lista dei blocchi liberi (bitmap) e statistiche sull'uso dei blocchi dati.
  - Lista di inode
  - Blocchi di dati
- I metadati di ogni gruppo sono memorizzati in posizioni diverse nei cilindri (formano una spirale) per evitare che il danneggiamento di una superficie/cilindro/traccia possa distruggere in un solo colpo tutte le copie di backup del superblocco

File System di Minix

- MINIX (Minimal Unix) è un sistema operativo open source, Unix-like sviluppato da Andrew S. Tanenbaum come supporto didattico al suo libro *Operating Systems: Design and Implementation* del 1987

- La prima versione di MINIX consisteva in 12.000 linee di codice C sviluppato per macchine IBM-compatibili dell'epoca.

- Le prime versioni di Linux adottavano il file system di MINIX

- In Minix i dati sono organizzati in *zone* per cercare di ottimizzare l'allocazione dei file su disco:
  1 zone $= 2^n$ blocchi, 1 blocco in un floppy $=$ 2 settori $=$ 1KB

Struttura del FS di Minix

Il file system di Minix ha la struttura seguente:

- Un disco viene diviso in partizioni, ogni partizione ha il suo file system
- Il primo blocco di una partizione contiene le informazioni per il boot
- Il secondo blocco è chiamato *superblocco* e contiene i metadati del file system:
  - numero di inode e zone, dimensione delle bitmap degli inode e delle zone, indirizzo del primo blocco di dati
  - inode/zone bitmap: tengono traccia di quali inode/zone sono liberi/e e quali occupati/e
  - lista di inode
  - blocchi dati

MINIX FS

- Gli inode sono di 32 byte e contengono 7 puntatori diretti (a zone) 1 indiretto, e 1 doppiamente indiretto

- I superblocchi di file system montati e i corrispondenti inode vengono mantenuti in memoria principale in apposite tabelle

- Si utilizza una disk cache in memoria principale (lista doppiamente linkata con accesso hash e rimpiazzamento LRU) per i blocchi utilizzati più recentemente

Limitazioni di Minix

- Le prime versioni di Linux utilizzavano un file system Minix
- Il filesystem di Minix aveva tuttavia grandi limitazioni:
  - indirizzi di blocco memorizzati in interi a 16 bit (dim. max = 64MB)
  - record di directory di dimensione fissa con nomi di al più di 14 caratteri

Extended File System (Ext)

- Il file system Ext del kernel standard di Linux è stato sviluppato per estendere il file system Minix e bypassare le sue limitazioni
- La dimensione massima di un filesystem Ext è di 2GB con nomi di lunghezza massima di 255 caratteri
- Ext ha ancora varie limitazioni e problemi: ad esempio utilizza linked list (non ordinate) per gestire blocchi liberi e inode con il rischio di frammentazione esterna del file system
- Ext è stato successivamente migliorato portando a versioni ormai standard quali Ext2 e Ext3 per sistemi operativi Linux-based

EXT2

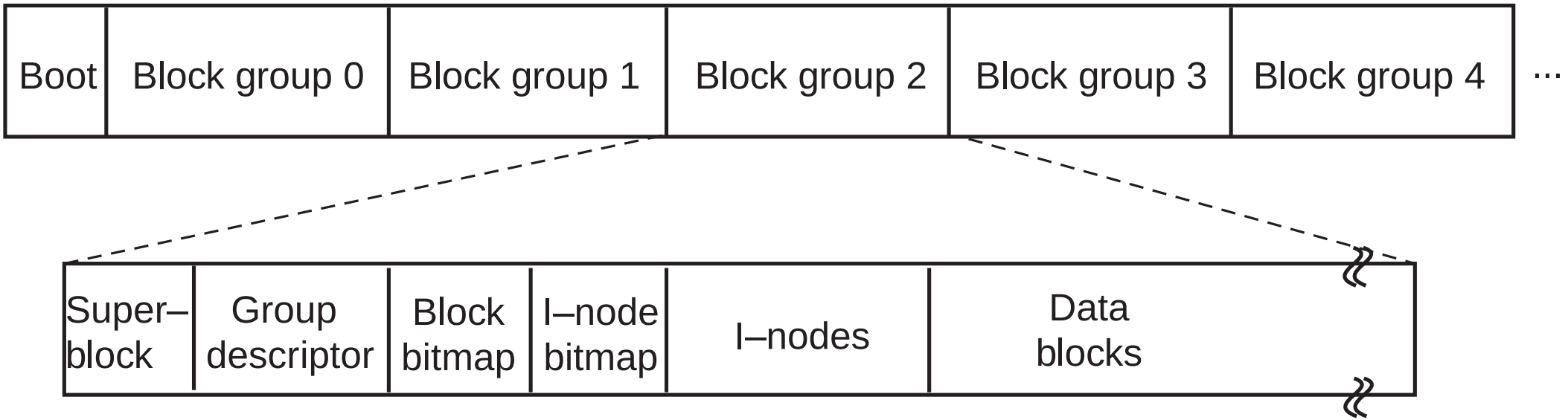
- Il file system Ext2 (second extended file system) è stato sviluppato migliorando e riorganizzando il codice di Ext (tenendo conto anche di sviluppi futuri del sistema operativo)

- Parallelamente a Ext2 è stato sviluppato anche il file system Xia sulla base del codice di Minix.

- Ext2 è stato il file system di default per molte distribuzioni di Linux (Red Hat Linux, Fedora Core and Debian Linux)

Struttura di EXT2

Una partizione Ext2 ha la seguente struttura:



- I blocchi hanno tutti della stessa dimensione (1K-4K)

- Il file system è suddiviso in in *gruppi* di 8192 blocchi

- Il *superblock* (blocco 0) contiene meta-informazioni sul file system (tipo di file system, primo inode, numero di gruppi, numero di blocchi liberi e inode liberi,. . .)

- Ogni gruppo ha una copia del superblock, la propria tabella di inode e tabelle di allocazione blocchi e inode

- Per minimizzare gli spostamenti della testina, si cerca di allocare ad un file blocchi dello stesso gruppo

- Le directory sono implementate come liste con entry di lunghezza variabile

Confronto tra Minix, Ext, Ext2, Xia

- Dim. massima file system:
  Minix=64MB , Ext=2GB , Ext2=4TB, Xia=2GB

- Dim. massima file:
  Minix=64MB , Ext=2GB , Ext2=2GB, Xia=64MB

- Dim. massima nome file:
  Minix=14/30, Ext=255 , Ext2=255, Xia=248 (caratteri)

Ext3 e Ext4

- Ext2 si è evoluto ulteriormente in
  - Ext3, un file system con caratteristiche aggiuntive quali il *journaling*, cioè con una gestione transazionale delle modifiche su disco,
  - Ext4, una versione che supporta volumi di capacità nell'ordine dei *petabyte* e una gestione separata (su due diverse partizioni) di directory e file

NTFS: File System di 2K, NT, XP

- Il file system NTFS è stato sviluppato *from scratch* per Windows NT
- Windows 2000 supporta sia FAT (-16 e -32) che NTFS
- NTFS supporta indirizzi di disco a 64 bit (ricordate che FAT-16 supporta indirizzi a 16 bit (max 2GB), e FAT-32 indirizzi a 28 bit (max 2TB))
- NTFS è un sistema molto più complesso del file system per MS-DOS

Caratteristiche di NTFS

- I nomi sono lunghi fino a 255 caratteri Unicode.
- I caratteri utilizzati sono in Unicode (2 byte per carattere)
- Si distinguono maiuscole e minuscole (ma le API Win32 no)
- A differenza di Unix e FAT-32, un file non è una sequenza lineare di file ma bensì si compone di attributi multipli (ad es. nome, flag, dati), ognuno rappresentato da una sequenza di byte
- L'idea di file come sequenza di attributi è stata introdotta dall'MacIntosh (MacOS)
- La lunghezza massima di una sequenza è 2^{64} byte (i.e. $18 exabyte \circ 18^{18}$)

Chiamate API Win32 del file system

- Le funzioni per il file system di API Win32 sono simili a quelle di Unix

- Ad esempio la chiamata `CreateFile` utilizzata per la creazione e apertura di un file restituisce un *gestore di file* (handle) come per il *file descriptor* di Unix

```
/* Open files for input and output. */
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
outhandle = CreateFile("newf", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, NULL);

/* Copy the file. */
do {
    s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
    if (s && count > 0) WriteFile(outhandle, buffer, count, &ocnt, NULL);
} while (s > 0 && count > 0);

/* Close the files. */
CloseHandle(inhandle);
CloseHandle(outhandle);
```

Implementazione di NTFS

- Diverse partizioni possono essere unite a formare un *volume logico*
- Lo spazio viene allocato in *cluster* (sequenze lineari di blocchi)
- Un blocco a dimensione variabile da 512byte e 64KB.
- La maggior parte dei dischi NTFS usa blocchi a 4KB
- La struttura principale in ogni volume è la Master File Table (MFT) che descrive file e directory

Master File Table (MFT)

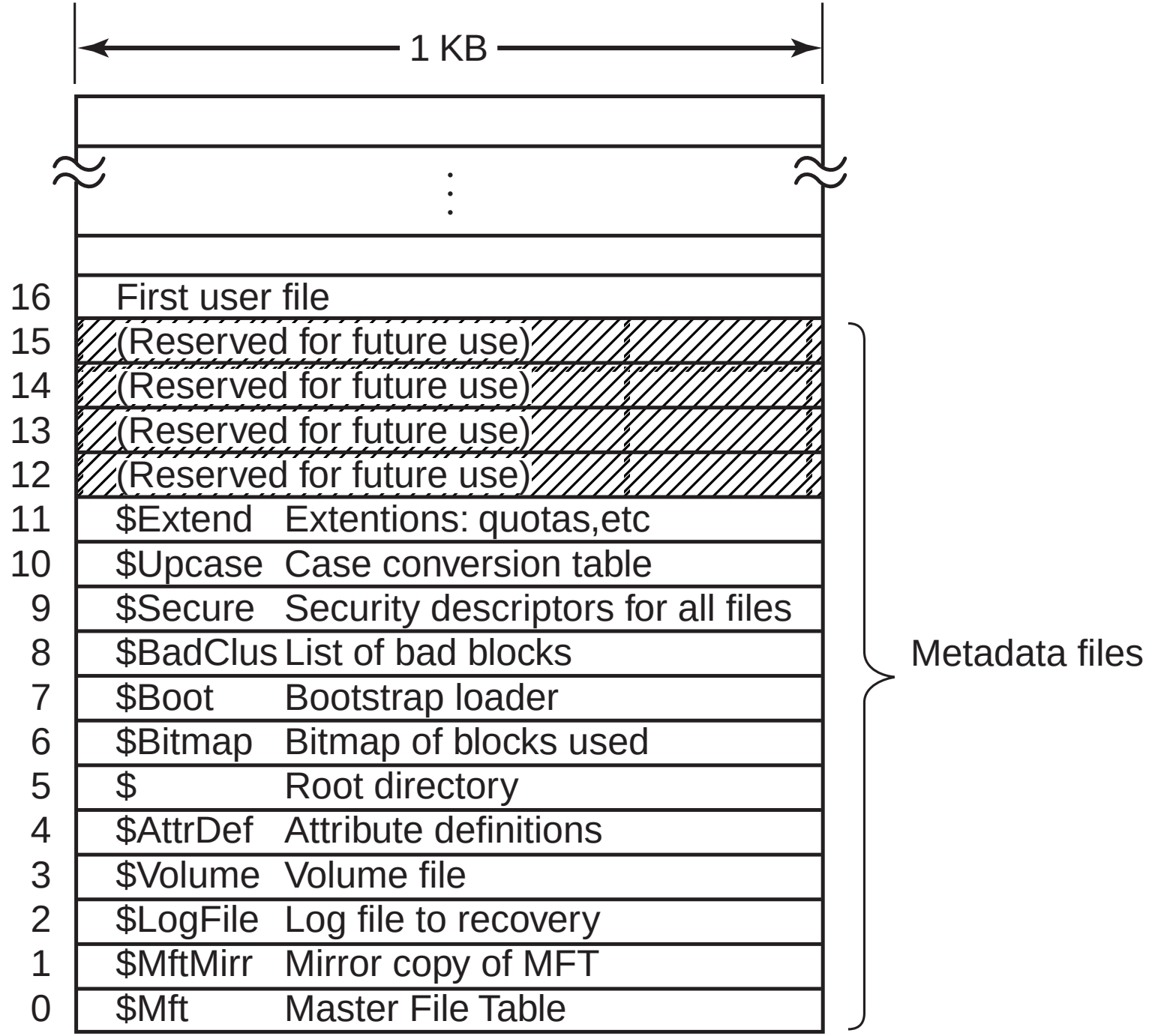
- È una sequenza lineare di record di 1KB
- Ogni record della MFT descrive un file o una directory e contiene attributi e la lista di indirizzi disco per quel file
- Per file grandi si possono usare più record per la lista dei blocchi (record base con puntatore ad altri record)
- Una mappa di bit tiene traccia dei record MFT liberi
- Una MFT è a sua volta vista come un file: può essere collocata ovunque su disco (evita il problema di settori imperfetti nella prima traccia), e può crescere fino a 2^{48} record

Struttura della MFT

- I primi 16 record descrivono l'MFT stesso e il volume (analogo al superblock di Unix).
  - Descrizione della MFT (primo record) e copia ridondante (secondo record): es. posizione dei blocchi della MFT
  - File di log: registra operazioni di aggiunta/rimozone/modifica al file system
  - Informazioni sul volume (etichetta, dimensione)
  - Informazioni sugli attributi dei file
  - Posizione della root dir (file anch'essa)
  - Attributi ed indirizzi di disco della bitmap per gestire lo spazio libero
  - Lista di blocchi malfunzionanti
  - Informazioni di sicurezza

  - Mappa dei caratteri maiuscoli (non sempre ovvia)
  - Informazioni su quote disco
  - Gli ultimi 4 record sono riservati ad usi futuri

- L'indirizzo del primo blocco della MFT viene memorizzato nel blocco d'avvio


1 KB
16 First user file
15 (Reserved for future use)
14 (Reserved for future use)
13 (Reserved for future use)
12 (Reserved for future use)
11 $Extend Extentions: quotas,etc
10 $Upcase Case conversion table
9 $Secure Security descriptors for all files
8 $BadClus List of bad blocks
7 $Boot Bootstrap loader
6 $Bitmap Bitmap of blocks used
5 $ Root directory
4 $AttrDef Attribute definitions
3 $Volume Volume file
2 $LogFile Log file to recovery
1 $MftMirr Mirror copy of MFT
0 $Mft Master File Table
Metadata files

Record della MFT

- Le informazioni sui file utente vengono memorizzate a partire dal record 16 della MFT

- Ogni record ha un'intestazione (header) seguita da una sequenza di coppie (intestazione di attributo e valore).

- Il record header contiene ad esempio uncontatore di riferimenti al file (come per i-node), ed il numero effettivo di byte usato nel record

- Ogni intestazione di attributo contiene il tipo dell'attributo e la locazione e la lunghezza del corrispondente valore

- I valori possono seguire il proprio header (*resident attribute*) o essere memorizzati in un record separato (*nonresident attribute*)

Attributi dei file NTFS

NTFS definisce 13 attributi standard quali

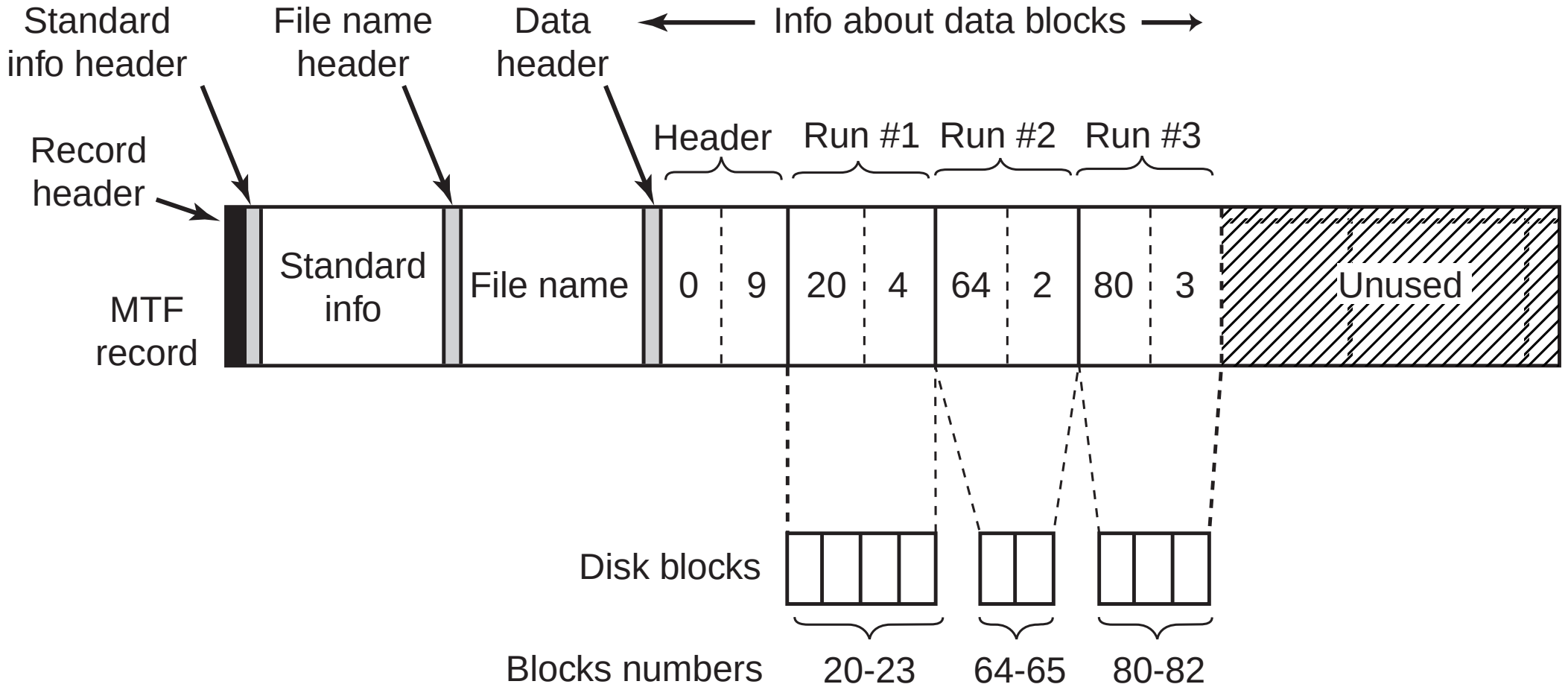
- Informazioni standard: proprietario, diritti, time-stamps, contatore di link fisici
- Nome del file (lunghezza variabile, Unicode)
- Identificatore di oggetto: nome unico per il file nel sistema
- Punti di analisi: per operazioni speciali durante il parsing del nome (ad es. montaggio)
- Dati: contiene gli indirizzi disco dei veri dati; se sono residenti, il file si dice "immediate"

Attribute	Description
Standard information	Flag bits, timestamps, etc.
File name	File name in Unicode; may be repeated for MS-DOS name
Security descriptor	Obsolete. Security information is now in $Extend$Secure
Attribute list	Location of additional MFT records, if needed
Object ID	64-bit file identifier unique to this volume
Reparse point	Used for mounting and symbolic links
Volume name	Name of this volume (used only in $Volume)
Volume information	Volume version (used only in $Volume)
Index root	Used for directories
Index allocation	Used for very large directories
Bitmap	Used for very large directories
Logged utility stream	Controls logging to $LogFile
Data	Stream data; may be repeated

File NTFS non residenti

I file non immediati si memorizzano a "run": sequenze di blocchi consecutivi

Nel record MFT corrispondente ci sono i puntatori ai primi blocchi di ogni run



- L'header del file contiene l'offset del primo blocco (e.g. 0) e il numero di blocchi totale (e.g. 9)

- Gli indirizzi dei run sono memorizzati a coppie (numero del primo blocco, numero dei blocchi del run)

- Nota: un file con n blocchi può essere memorizzato in numero di range che varia da 1 a n

- Un file descritto da un solo MFT record si dice *short* (ma potrebbe non essere corto per niente!)

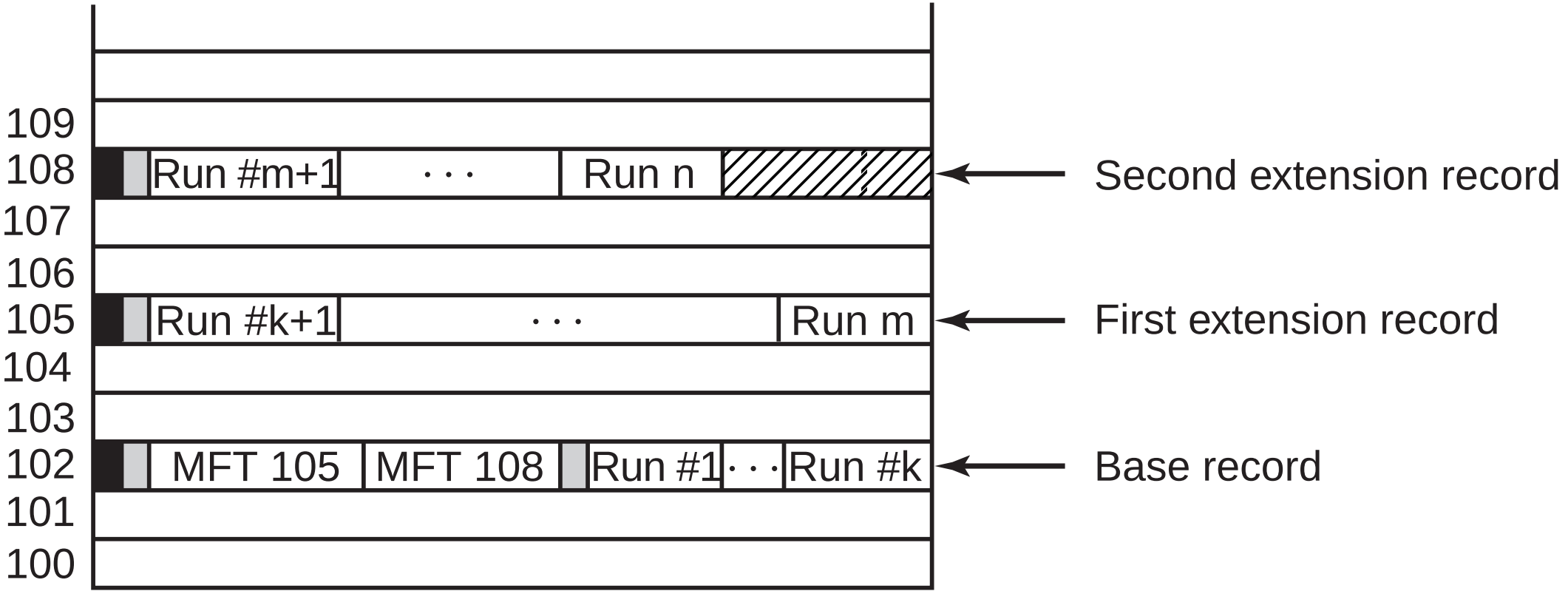
Dimensione File NTFS

- Non esiste un limite superiore alla dimensione di un file
- Ogni coppia richiede due numeri a 64 bit: 16 byte
- Una coppia può rappresentare più di un milione di blocchi disco consecutivi (ad es. 20 sequenze separate da 1 milione di blocchi da $1KB = 20KB$)
- Si usano metodi di compressione per memorizzate le coppie in meno di 16byte (si arriva fino a 4byte)

File "long"

Se il file è lungo o molto frammentato (es. disco frammentato), possono servire più di un record nell'MFT.

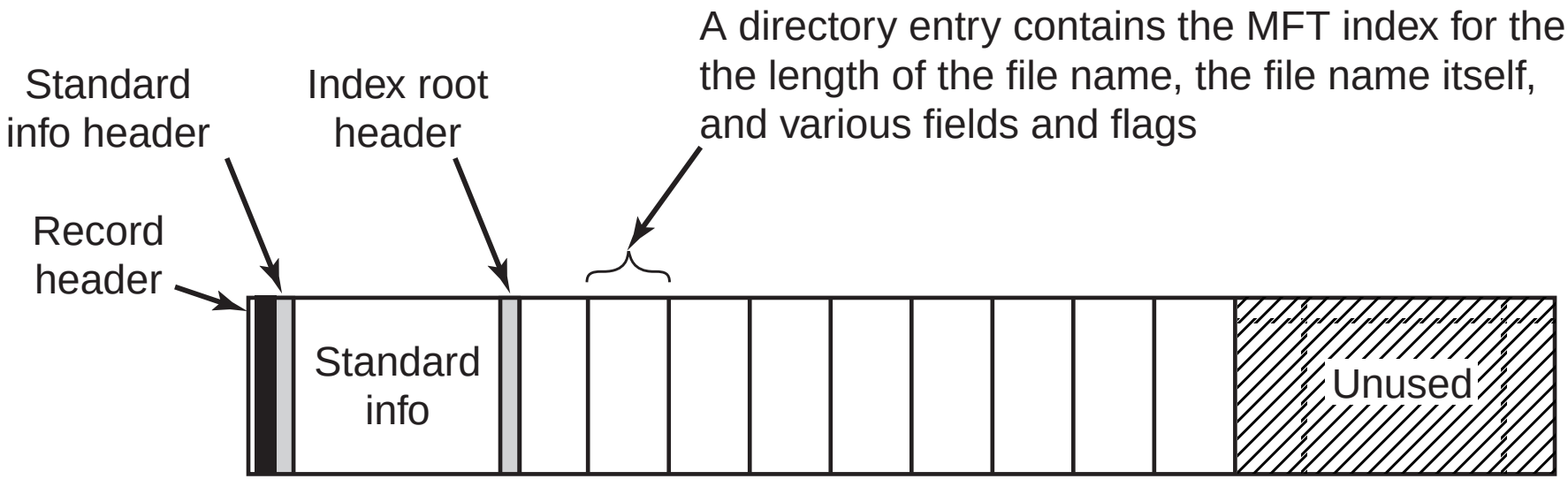
Prima si elencano tutti i record aggiuntivi, e poi seguono i puntatori ai run.



Se i puntatori ai record aggiuntivi non stanno su un solo MFT si memorizza la lista dei record con i blocchi su disco invece che nel MFT

Directory in NTFS

Le directory corte vengono implementate come semplici liste direttamente nel record MFT.



Directory più lunghe sono implementate come file nonresident strutturati a B+tree.

Navigazione in NTFS

- Consideriamo l'indirizzo

  C:\maria\web.html

- La directory radice \ contiene un puntatore alla lista di nomi di volumi logici (C, E, D, ecc)

- Il nome C: è un collegamento simbolico con la partizione del disco

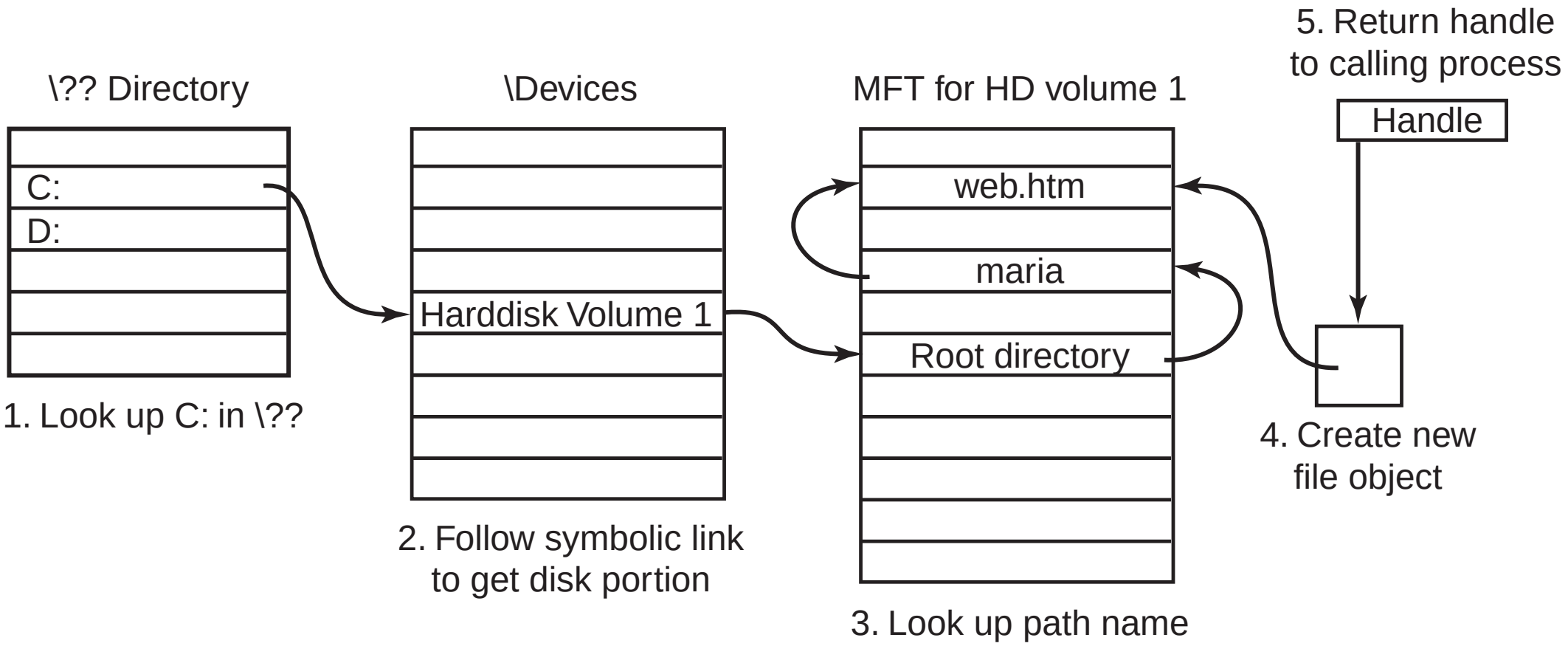
- Una volta identificata la partizione di può recuperare la corrispondente MFT

- Nel record 5 della MFT troviamo informazioni sulla directory radice del disco C

- La stringa *maria* viene cercata all'interno del record della directory radice, da tale ricerca otteniamo un indice nel MFT per la directory maria

- Quindi esaminiamo il record alla ricerca di *web.html*

- Se la ricerca ha successo si crea un nuovo *handle* (oggetto) che contiene l'indice del file nel MFT

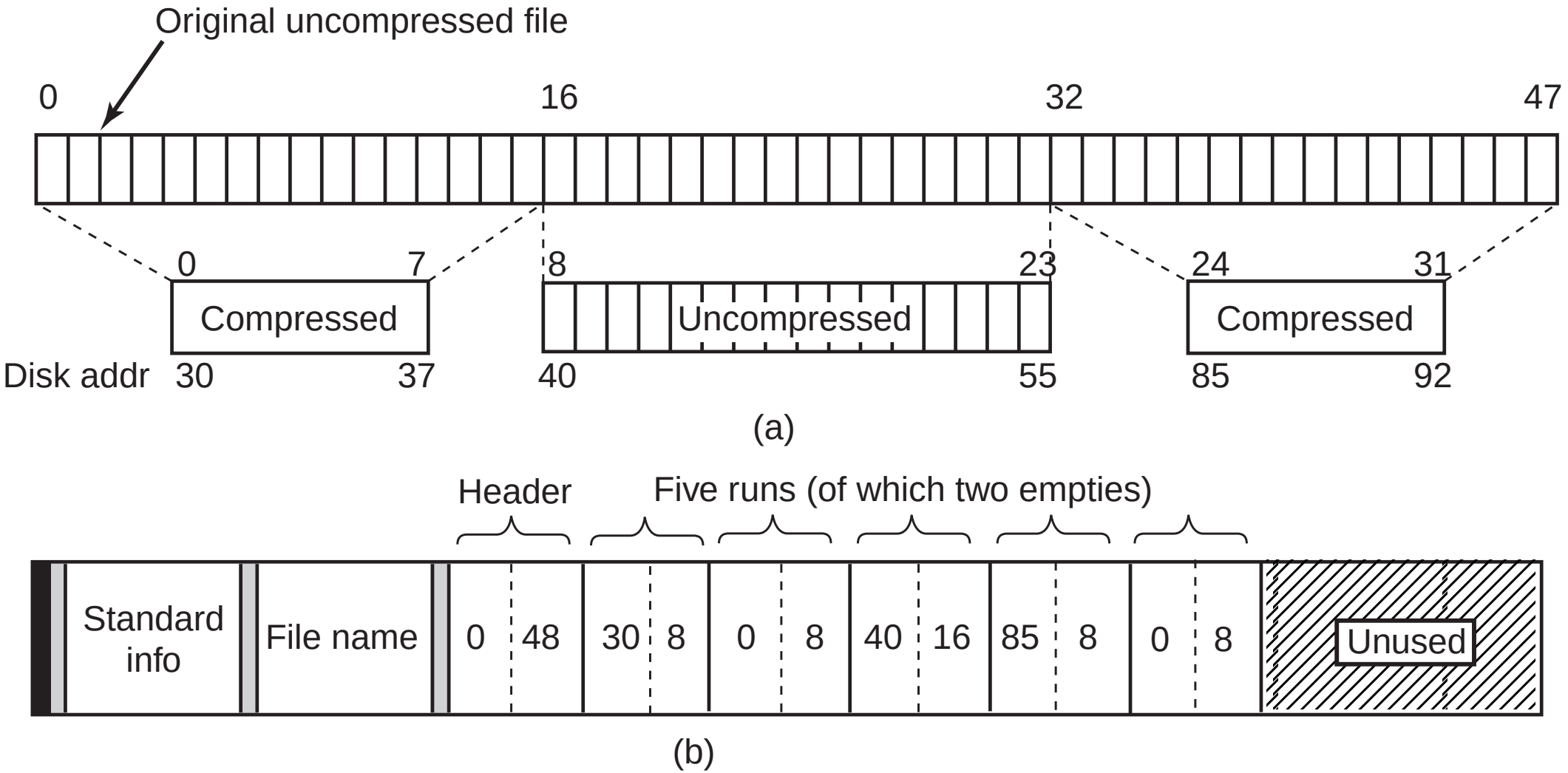

\?? Directory
C:
D:
1. Look up C: in \??
\Devices
Harddisk Volume 1
2. Follow symbolic link
to get disk portion
MFT for HD volume 1
web.htm
maria
Root directory
3. Look up path name
5. Return handle
to calling process
Handle
4. Create new
file object

Compressione file

- NTFS supporta la compressione trasparente dei file (cioè i file vengono compressi quando creati e decompressi in lettura)

- L'algoritmo di compressione lavora su gruppi di 16 blocchi: se si riescono a comprimere si scrivono i blocchi compressi e si memorizzano nel record MFT dei blocchi virtuali (con indirizzo di disco 0) per i blocchi mancanti;

- poi si prosegue con i successivi 16 blocchi



- Quando NFTS legge un file e trova due coppie consecutive $(n, m)(0, k)$ capisce che $m + k$ sono stati compressi in m blocchi e applica l'algorimo di decompressione a quella sotto-sequenza

File System CDRom

- File system particolarmente semplici in quanto progettati per supporti di sola lettura

- Ad esempio, non viene tenuta traccia dei blocchi liberi: i CDROM non vengono modificati

- Esistono vari tipi di file system: lo standard ISO 9660 specifica il tipo comunemente adottato dai supporti di lettura per CDROM

Organizzazione dei CDROM

- I CDROM hanno una spirale continua che contiene i bit in una sequenza lineare

- I bit lungo la spirale sono divisi in blocchi di 2532 byte:
  - 2048 byte sono di dati,
  - I byte rimanenti contengono header e codici di correzione

Standard ISO 9660

- File system leggibile da tutti i lettori e supportato dai principali sistemi operativi (anche MS-DOS)

- Lo standard definisce un file system per un'insieme di CDROM (fino a $2^{16} - 1$ CDROM), ognuno partizionato in volumi logici

- Ogni CDRom inzia con 16 blocchi la cui funzione non è definita dallo standard (vengono usati ad esempio per programmi di avvio)

- Di seguito si trova il descrittore di volume primario che contiene informazioni generali sul CDROM quali

  - identificatore di sistema (32 byte)

  - identificatore di volume (32 byte)

  - identificatore del distributore (128 byte)
  - identificatore del preparatore dei dati (128 byte)
  - nomi di tre file (riassunto, avviso di copyright ed informazioni bibliografiche)

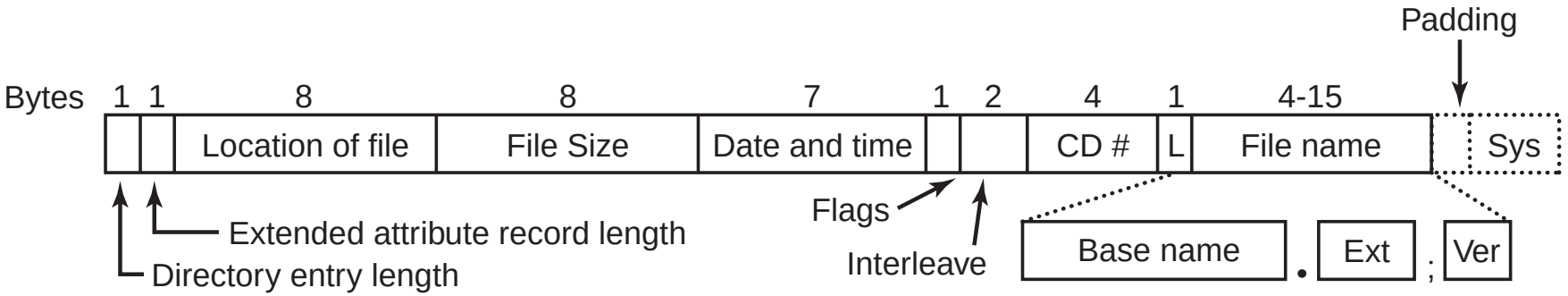
- Il descrittore contiene anche i seguenti numeri chiave
  - dimensione del blocco logico (normalmente 2048, in alcuni casi 4096, 8192)
  - numero di blocchi
  - date di creazione e scadenza

- Infine contiene un elemento di directory per la directory radice dei dati memorizzati

Directory in un CDROM

- Le directory sono composte di un numero variabile di elementi
- Gli elementi sono di dimensione variabile (tra 10 e 12 campi)
- La profondità di una directory è al più di 8 livelli, mentre non c'è limite al numero di elementi in una directory
- I primi due elementi indicano la directory corrente e la directory genitore

Elementi di Directory in un CDROM



- Il primo byte identifica la lunghezza dell'elemento di directory
- Il secondo byte la lunghezza del record di eventuali attributi estesi
- Poi seguono
  - Posizione del file: i file sono identificati tramite la coppia posizione del primo blocco e lunghezza (i file sono memorizzati come blocchi contigui)
  - Data di registrazione (range 1900-2155)

- Flag: contiene bit per distinguere file da directory, bit per nascondere file, bit per abilitare uso di attributi estesi, bit per marcare l'ultimo elemento di directory
- Interlacciamento: usato solo in versioni avanzate
- Numero del CDROM sul quale è posizionato il file (l'elemento potrebbe far riferimento ad un file su un'altro CD dell'insieme)
- Dimensione del nome del file in byte
- Nome del file: nome (8 caratteri), punto, estensione (3 caratteri)
- Padding: usato per far si che ogni elemento di directory sia formato da un numero pari di byte (per allineamento)
- System use: utilizzato in modo speciale dai diversi sistemi operativi

Livelli in ISO 9660

- Lo standard definisce tre livelli
  - Livello 1: nomi con 8+3 caratteri e file memorizzati in blocchi contigui
  - Livello 2: nomi fino a 31 caratteri
  - Livello 3: nomi come nel livello 2, file formati da diverse sezioni di blocchi contigui; le sezioni possono essere condivise tra diversi file, o comparire più volte nello stesso file

Estensione Rock Ridge

- L'estensione Rock Ridge permette di rappresentare file system Unix in un CDROM

- Le informazioni aggiuntive vengono memorizzate nel campo *system use* e sono ad esempio:
  - Bit Unix per i diritti sui file, bit SETUID e SETGID
  - Nome alternativo: nome UNIX per il file
  - Campi per la rilocazione di una directory (per superare il limite di 8 livelli)
  - Time stamp contenuti negli inode di un file Unix (creazione, ultima modifica, ultimo accesso)

Estensione Joliet

- L'estensione Joliet permette di rappresentare file system Windows in un CDROM

- Le informazioni aggiuntive vengono memorizzate nel campo *system use* e specificano ad esempio:
  - Nome di file lungo (fino a 64 caratteri)
  - Insiemi di caratteri Unicode per i nomi (caratteri unicode occupano 2 byte, quindi nomi di al più 128 byte)
  - Profondità della struttura della directory maggiore di 8 livelli
  - Nomi di directory con estensione (non usato per ora)