- A File System is a software component, which builds a logical structure for storing files on storage devices (hard disks etc.)
- It appears as a **hierarchical structure** in which files and folders can be stored. The top of the hierarchy of each File System is usually called "root"
- A File System specifies naming conventions for naming files and folders
- "Path" in a File System represents location, where specific file is stored

File system

- Mount-on
  - A directory is covered by the mounted file system.
  - mount table & vfs list

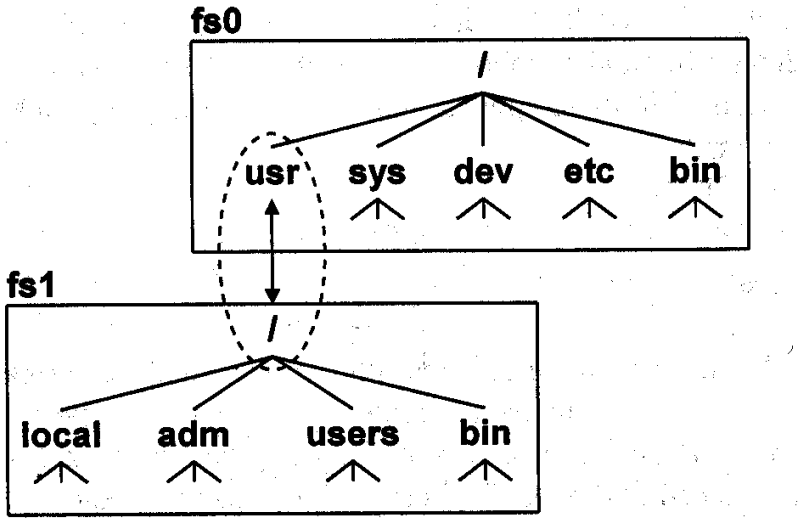


Figure 8-5. Mounting one file system onto another.

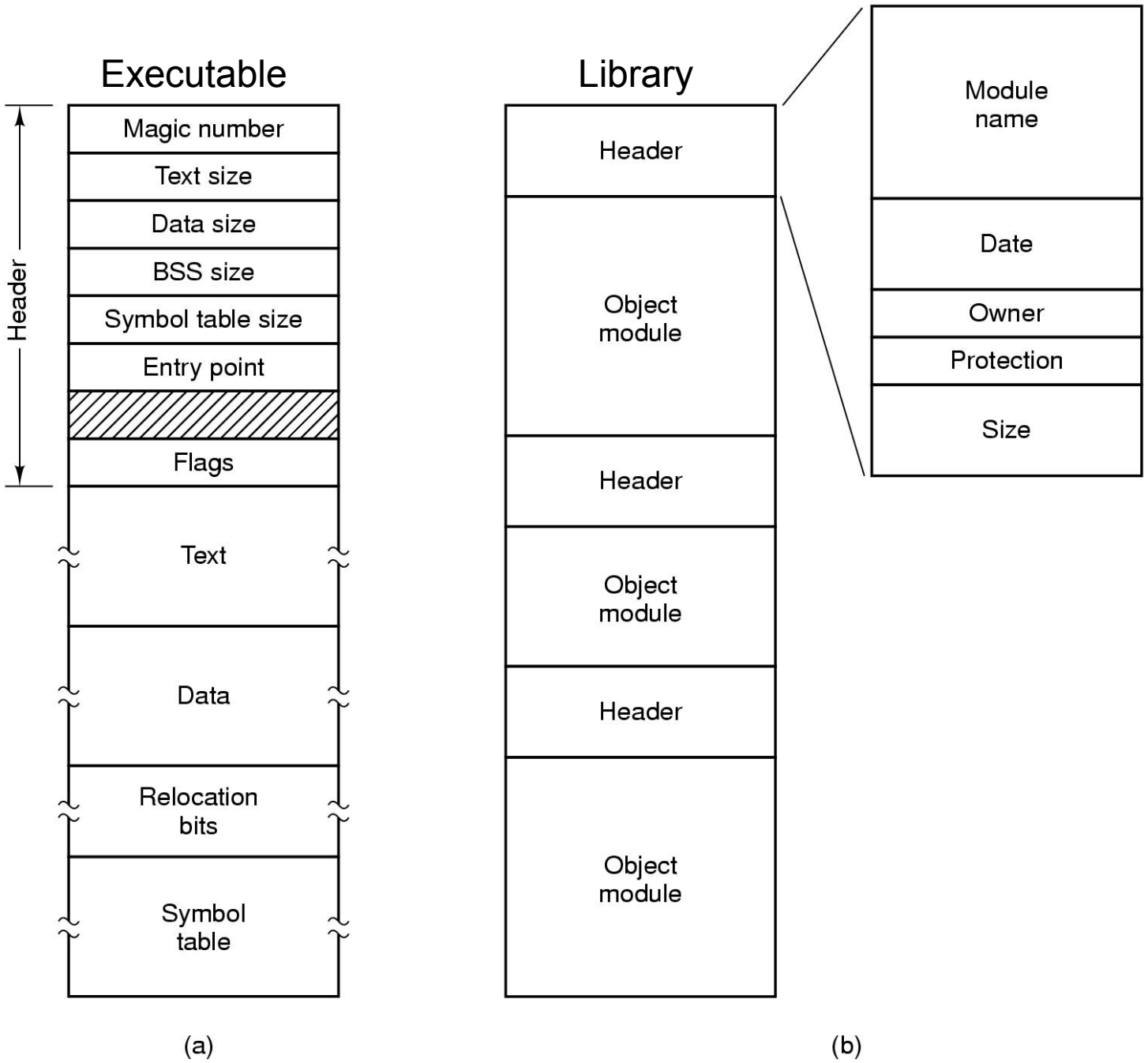
File Structure

- O.S. systems such as UNIX and Windows impose no structure to ensure maximum flexibility. They consider a file as a stream of bytes , and user processes define any structure that they want
- I/O is usually performed in units of ONE physical Block and all blocks have the same size that is related to the page size in paging scheme.

File Types

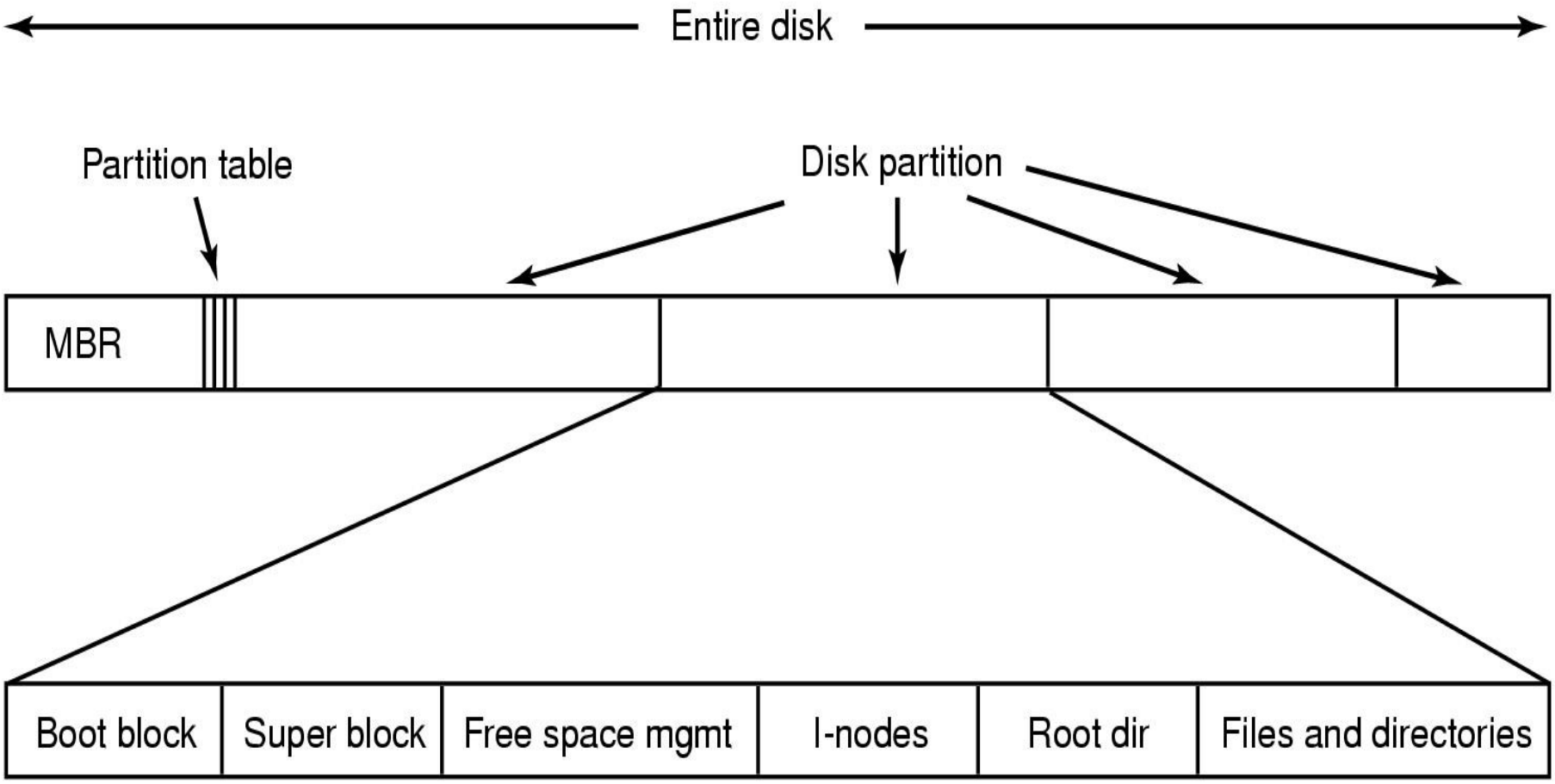
Some of the file types are

- Regular files: User files (ASCII files or binary files)
- Directory files: System files used to maintain directory structure
- Device files : Special system files dedicated to I/O
- Executable files: O.S. usually expects special structure for these files. For example in Unix they must start with Magic Number.


Executable
Header
Magic number
Text size
Data size
BSS size
Symbol table size
Entry point
Flags
Text
Data
Relocation bits
Symbol table
(a)
Library
Header
Object module
Header
Object module
Header
Object module
Module name
Date
Owner
Protection
Size
(b)

File System Layout

- Most disks divided up into one or more partitions, with independent file systems on each partition.
- On a PC, sector 0 of disk is called MBR (Master Boot Record) and contains partition table that contains start and ending address for each partition
- The layout of a disk partition depends on its file system. For example after its first block (i.e., boot block) it may contain super block that contains administrative information such as magic numbers to identify file types. (see next slide)


Entire disk
Partition table
Disk partition
MBR
Boot block
Super block
Free space mgmt
I-nodes
Root dir
Files and directories

Implementing the Files

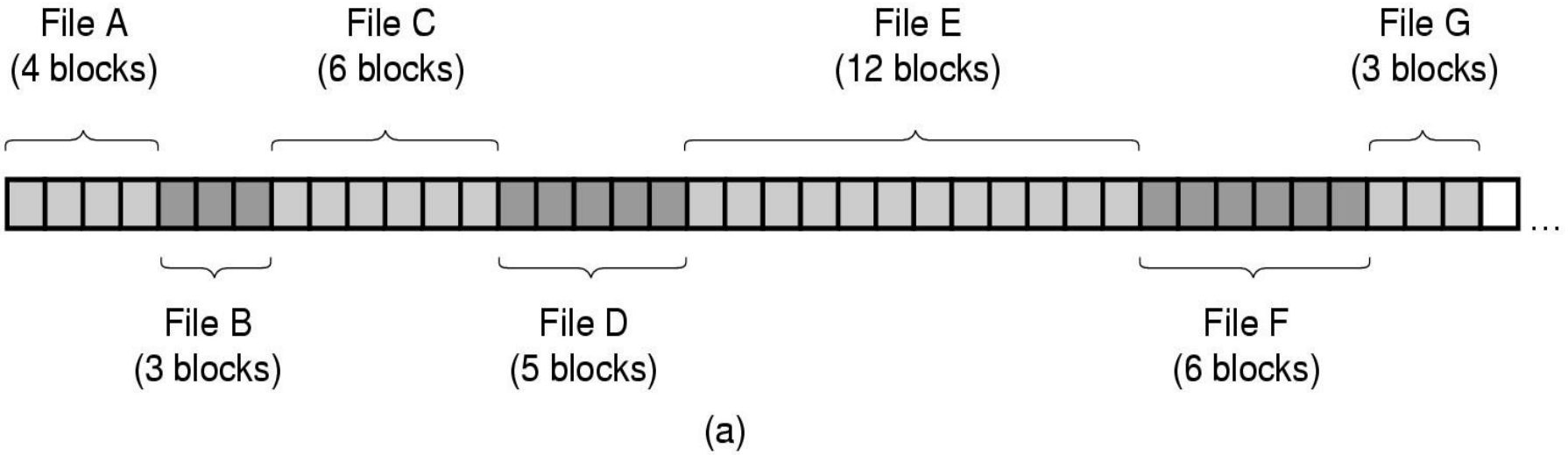
Various methods are used in different O.S. for implementing the files:

- Contiguous Allocation: Each file is stored on consecutive disk blocks. For example for a disk with 4K block size a 20K file is stored on 5 consecutive blocks. (see next slide)

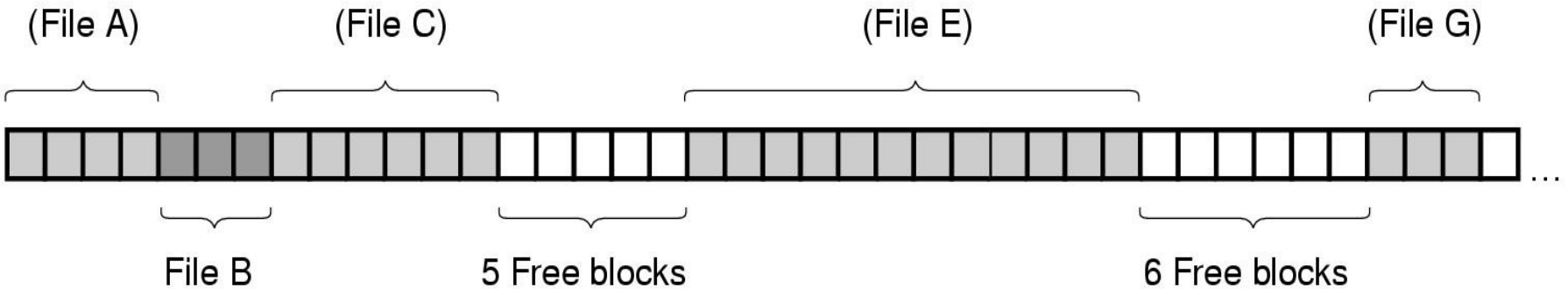
Advantages:

- simple to implement because we need to know only disk address of the first block and number of blocks
- The read performance is excellent because we need only one disk operation to read the entire file.

Contiguous Allocation



(a)



(b)

Contiguous Allocation

The disadvantages of Contiguous allocation are:

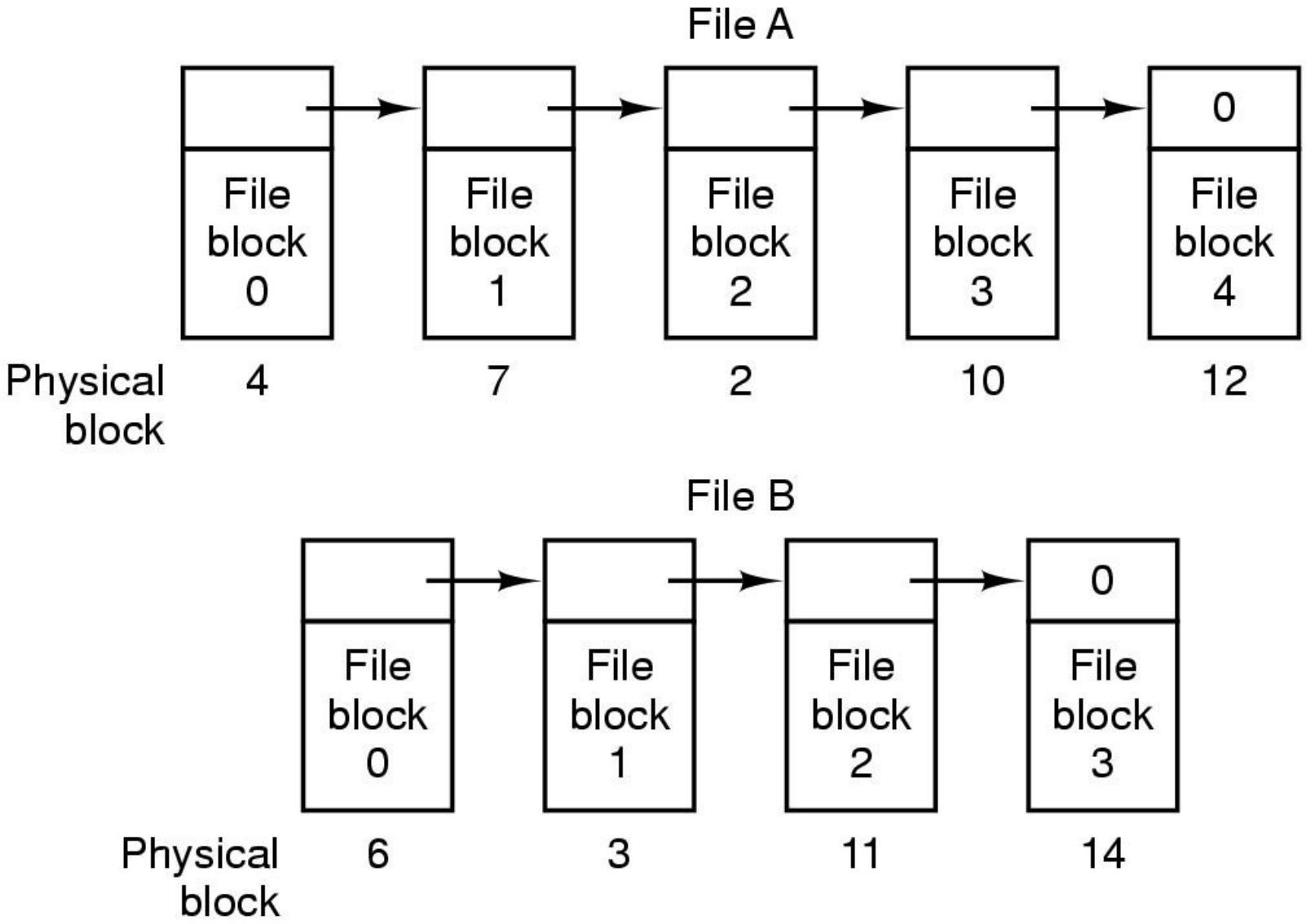
- Disk fragmentation: happens when the files are removed. Compaction is difficult because all the blocks following the holes should be copied. It is worse when the disk filled up.
- Needs to know the final size of new file to be able to choose the correct hole to place it. That is also difficult

Consecutive allocation is good for write once medias such as CD-ROMS and DVDs

Linked List Allocation

- A linked list of disk blocks (first word is pointer) is kept in this method
- Every disk blocks can be used (except for internal fragmentation)
- The sequential read for the blocks of the file is easy but random access to each block is hard because we have to read all the blocks of a file before that block
- Because of pointer the amount of data stored in each block is not a power of two

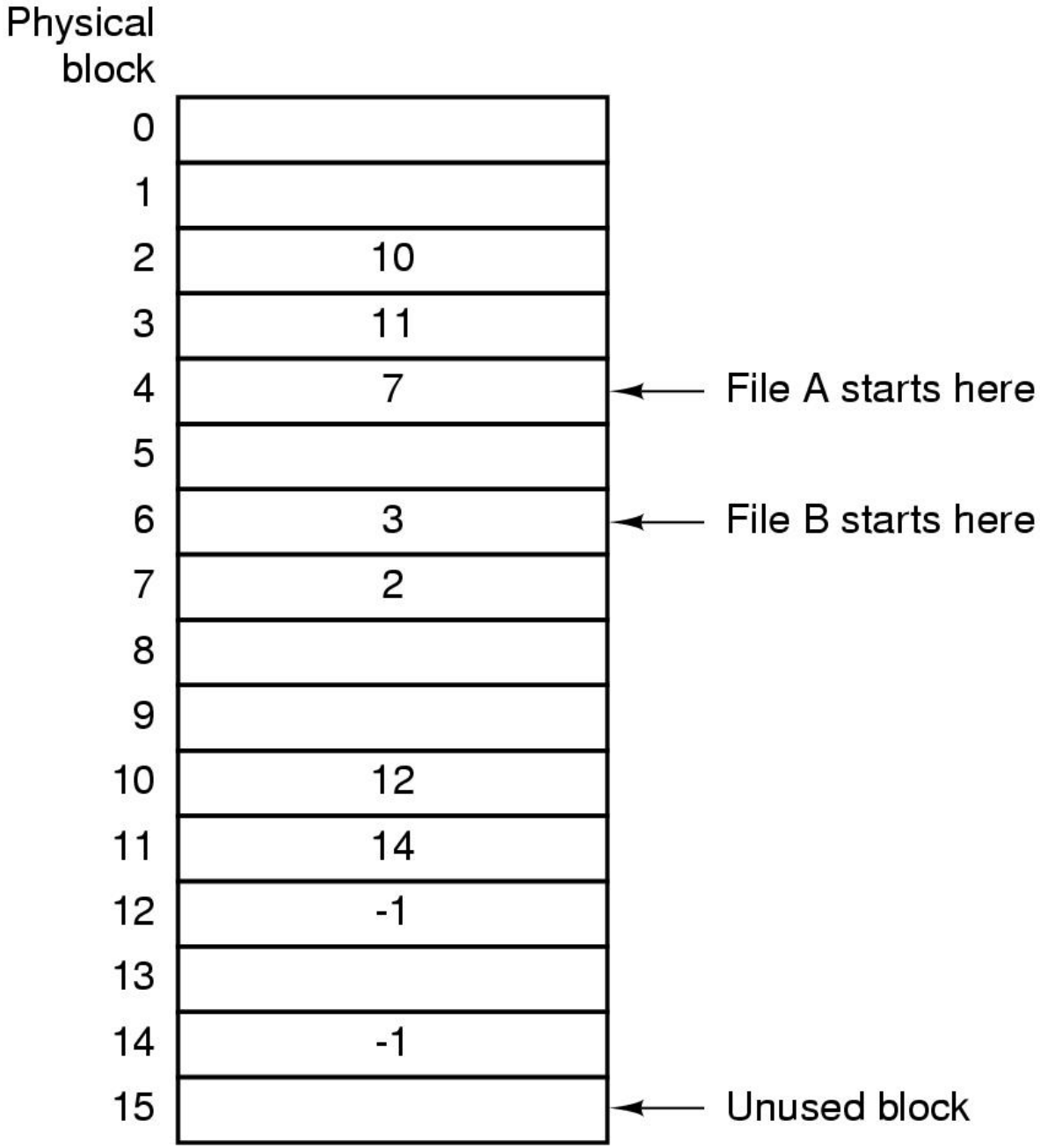
Linked List Allocation



Linked List Allocation using a Table in Memory

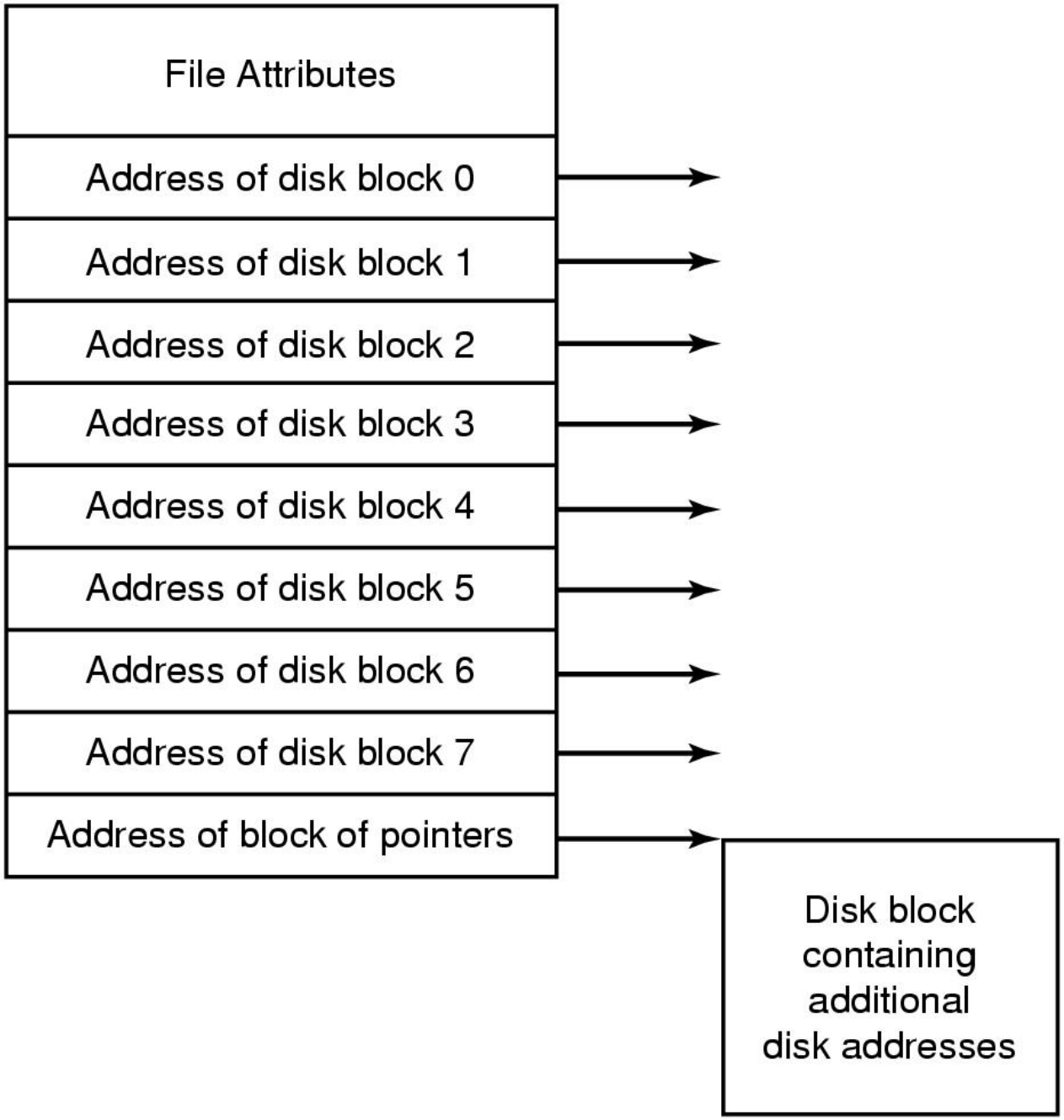
- Both of disadvantages of the linked list allocation can be eliminated by keeping the table of pointer to the blocks (FAT) in the memory. MSDOS uses that.
- Random access to blocks is easy because there is no disk reference involved. We need only the starting block number.
- The problem is for 20 GB disk, and a 1 KB block size table needs 20 million entries if each be 4 bytes, table will take approximately 80 MB .

File Allocation Table



I-nodes

- To solve the problem of the large file table we can use i-node
- In this method for each file there is a table contains attributes and disk address of the blocks of that file.
- Thus memory consumption depends on open files and not disk size

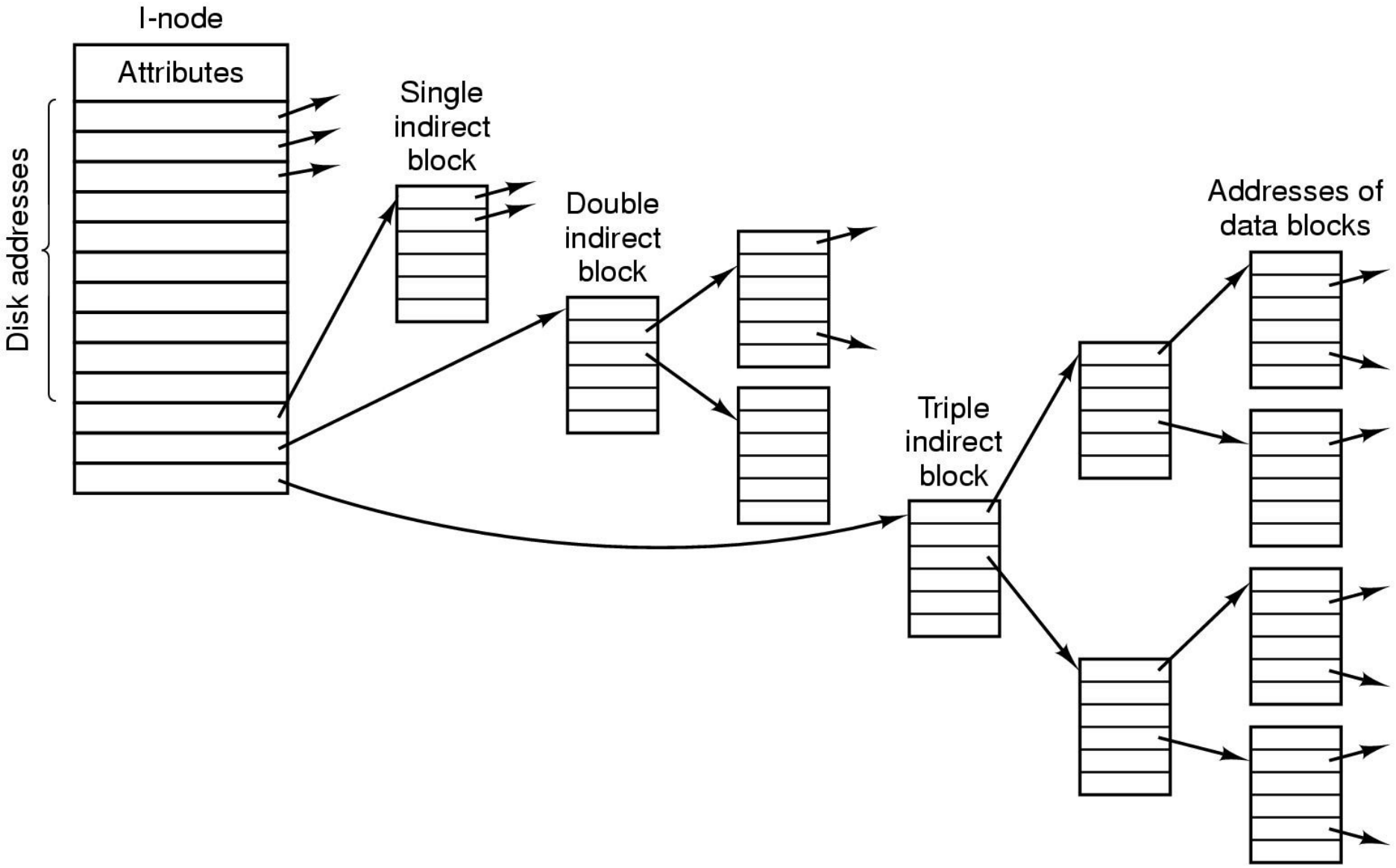

File Attributes
Address of disk block 0
Address of disk block 1
Address of disk block 2
Address of disk block 3
Address of disk block 4
Address of disk block 5
Address of disk block 6
Address of disk block 7
Address of block of pointers
Disk block containing additional disk addresses

I-node in Unix

i-node in UNIX has

- Initial 10 disk addresses.
- Single indirect blocks keeps address of file more blocks for larger files.
- Double indirect block that holds address of the blocks each contains a list of single indirect block
- Triple indirect block has the address of block each is double indirect block

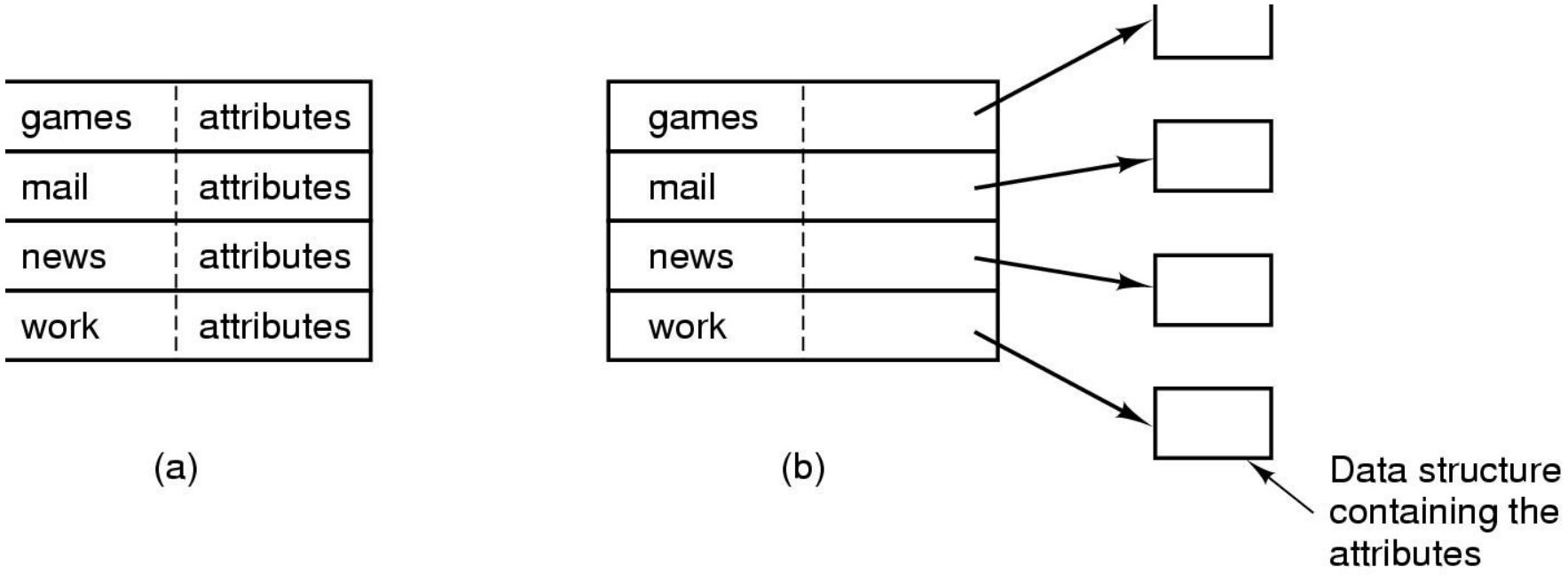
I-node in Unix



Implementing Directories

- Basically, a directory is a file that contains an entry for each file or subdirectory in that directory
- When a file is opened, O.S. uses the path name to locate directory entry
- Each directory entry contain the file information
- Each file information can be stored:
  - directly in directory entry (a in the next slide)
  - in i-node and each directory entry refers to i-node (b in the next slide)

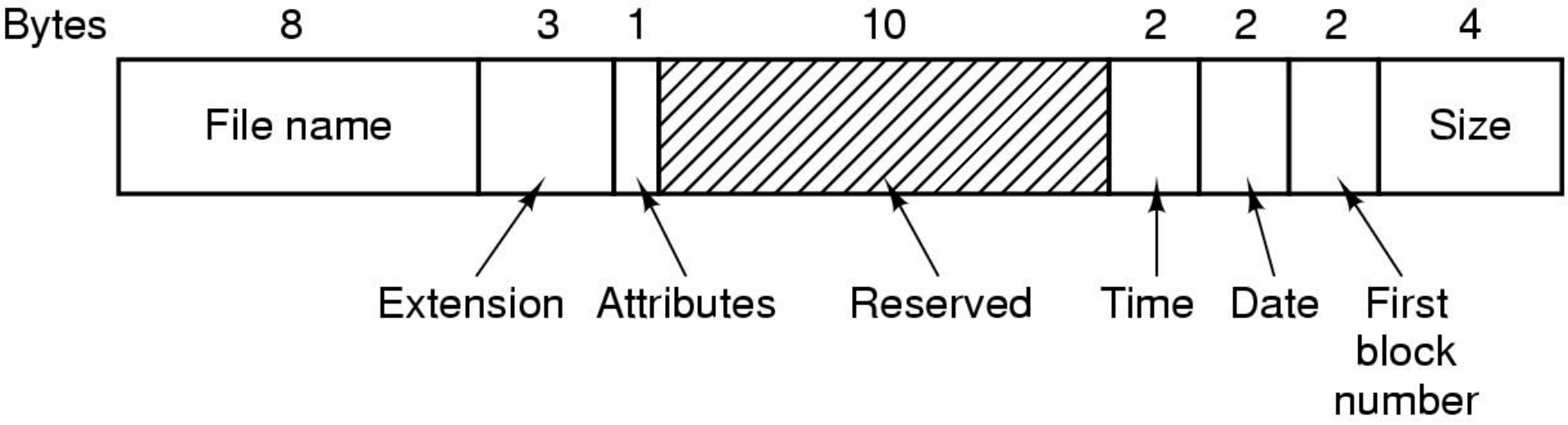
Implementing Directories



Directories in MS-DOS

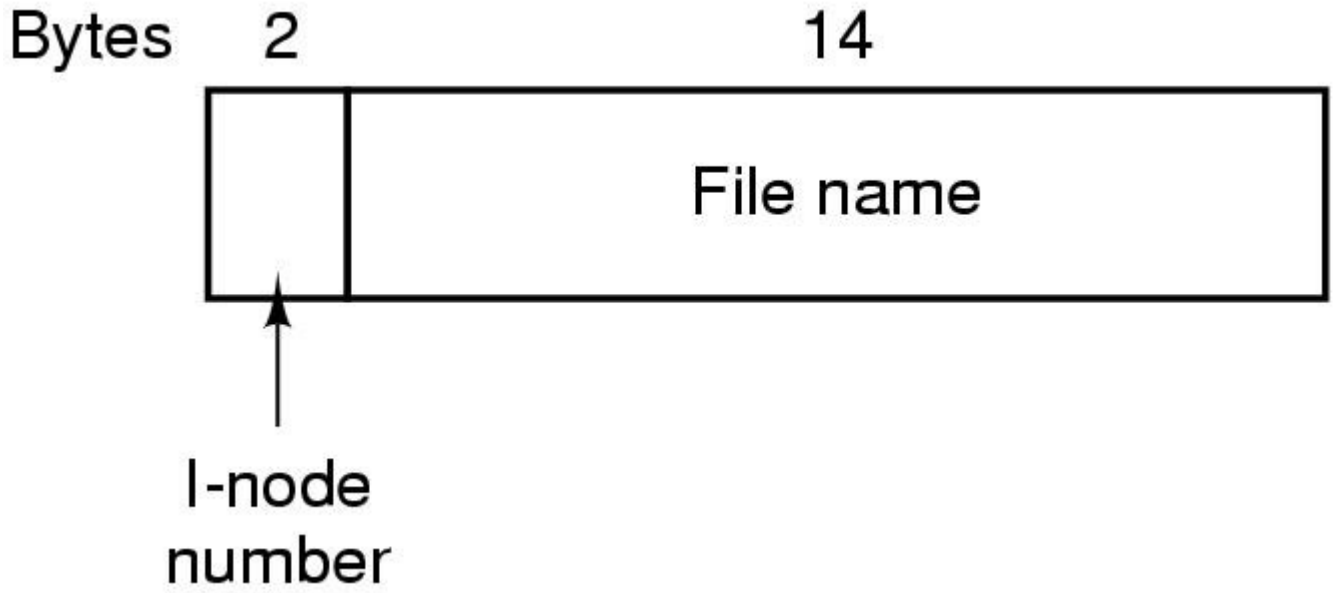
- Same as CP/M directory entries
- 32 bytes each
- The extension is for a large file size that requires more than one directory entry.
- First block number is the physical block number address of the file

Directories in MS-DOS



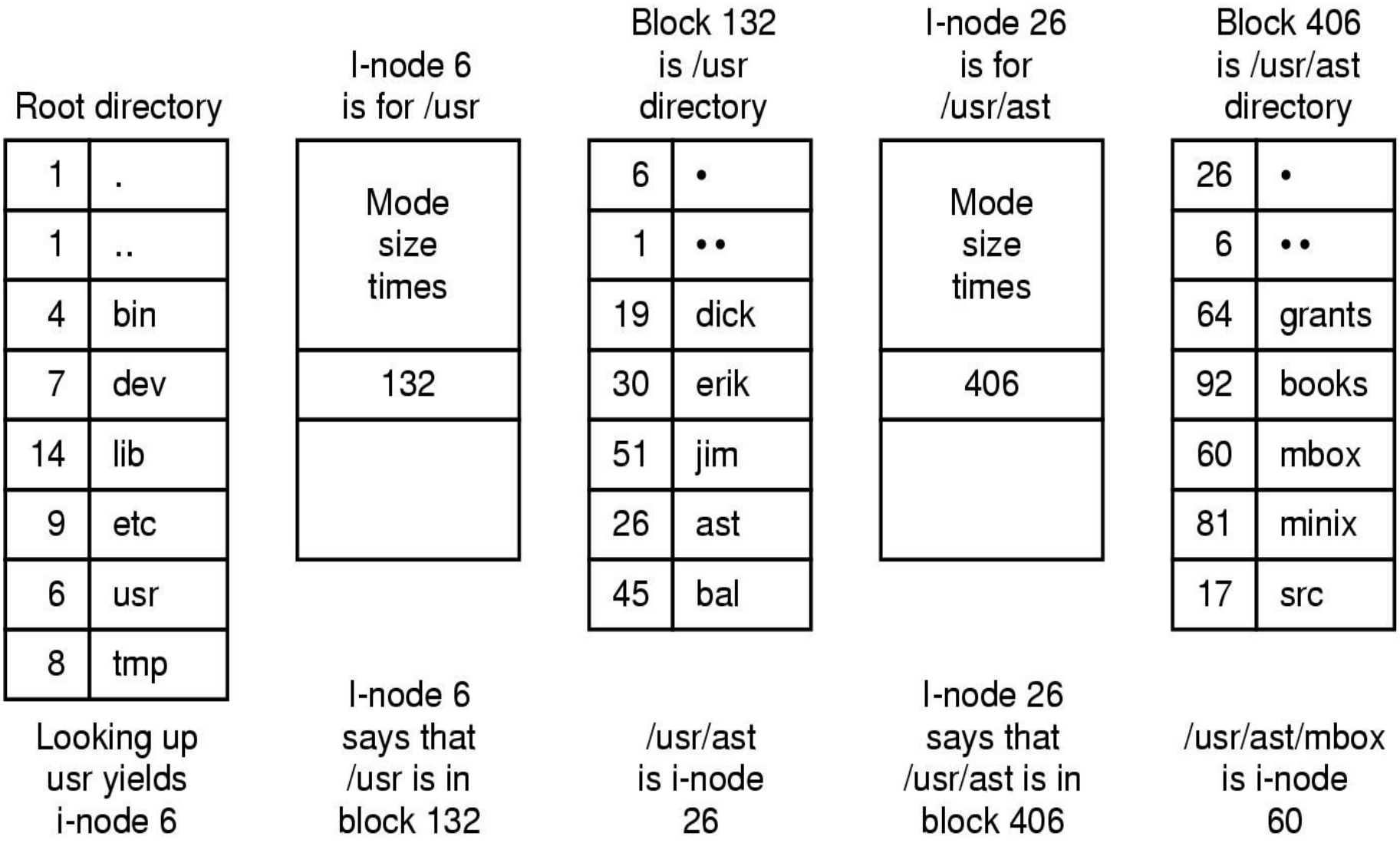
Directories in UNIX

- Each directory entry contains file name and i-node number



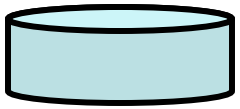
Directories in UNIX

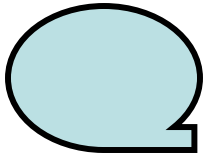
- Directory lookup in Unix and all hierarchical system is same
- First file system locates the root directory.
- Then it looks up the first component of the path and its i-node
- From the i-node system looks up the block address of next component and it works in the same way until the file can be found. For example next slide shows the steps in looking up /usr/ast/mbox


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

z/OS

- **I Dati sono contenuti su**
  - **dischi magnetici ad accesso Diretto (**DASD**) ,**
  - nastri **magnetci**
  - **altri** dispositivi Ottici**.**
- **Usualmente i dati possono essere reperiti in maniera**
  - Sequenziale
  - con Accesso Casuale**.**
- **Si possono usare I DASD suddividendoli**
  - **in strutture** fisiche **o**
  - **In strutture** logiche **dette VOLUMI.**
- **All'interno di un volume si possono conservare**
  - dati**,**
  - programmi eseguibili **, tra i quali lo stesso SO**
  - aree di lavoro temporaneo**. ecc**
- **Le allocazioni possono essere**
  - statiche
  - dinamiche,
  - manuali
  - automaticamente gestite **da un componente di Sistema detto Storage Management System (DFSMS)**





- **Si definisce** data set **una collezione di** strutture **di dati logiche dette** records **e contenute in un** disco **magnetico o in un altro dispositivo in grado di contenere dati in maniera non volatile.**
- **IL contenuto di un data set puo' essere:**
  - **Un** programma
  - **Un componente di** System library **of** macro**s**
  - **Un file (**archivio**) di dati usabili dall'utente.**
- **Il record e' l'unita' di base delle informazioni contenute sul Sistema z/OS .**
- **Lo spazio sul DASD puo' essere usato per molti archivi detti 'data sets' .Essi possono essere**
  - allocati,
  - riservati,
  - scritti
  - letti
  - cancellati**, lo spazio puo' essere subito riutilizzato.**

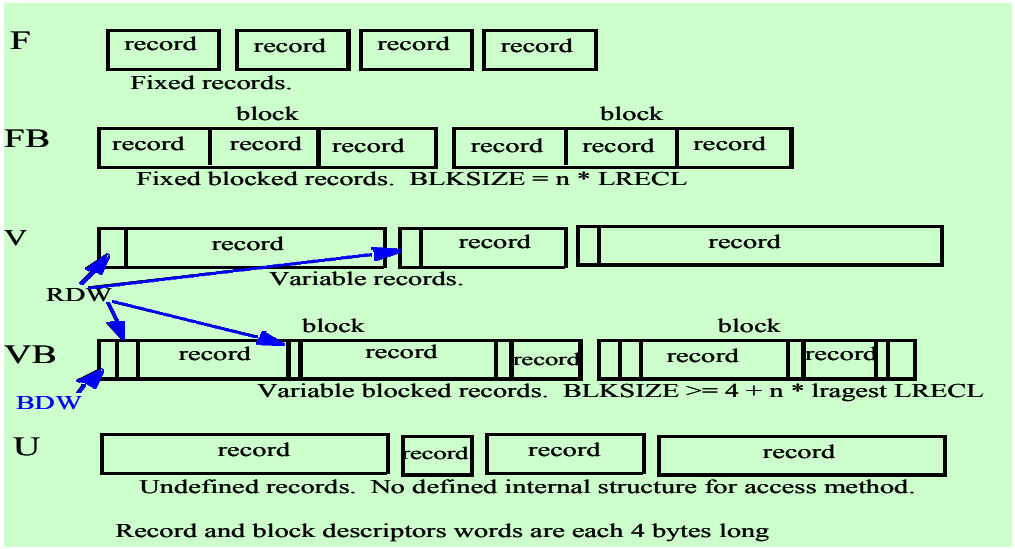
Metodi di Accesso e Formati dei Record

Il metodo di accesso :

- **definisce la tecnica per scrivere e reperire i dati sul Sistema.**
- **include una serie di programmi e di utilities per trattare i dati**
- **i metodi di accesso piu' comuni sono :**
  - **VSAM,**
  - **QSAM,**
  - **BSAM,**
  - **BDAM,**
  - **BPAM.**
- **Il** formato dei records **puo' avere varie implementazioni**
  - **F= Fisso**
  - **FB= Fisso Bloccato**
  - **V=Variabile**
  - **VB=Variabile Bloccato**
  - **U=indefinito**

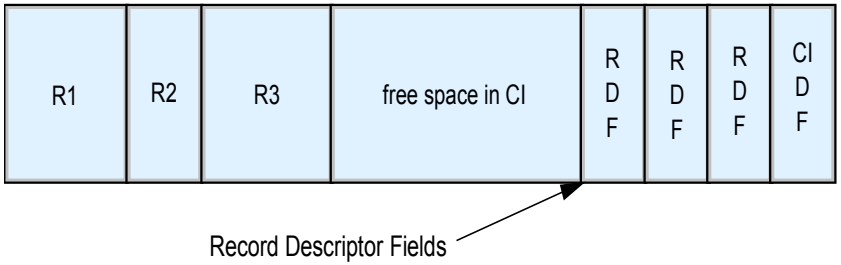
Tipi di Data Set:

- Sequenziali
  Semplici Raccolte di records
- Partizionati o librerie **(PDS/PDSE)**
  Archivi con un organizzazione a 2 livelli ,
  Data set name e member name tipici delle librerie di programi
- VSAM
  Utilizzano il metodo di accesso VSAM. Rappresentano la maggior parte dei DATI contenuti nel Sistema



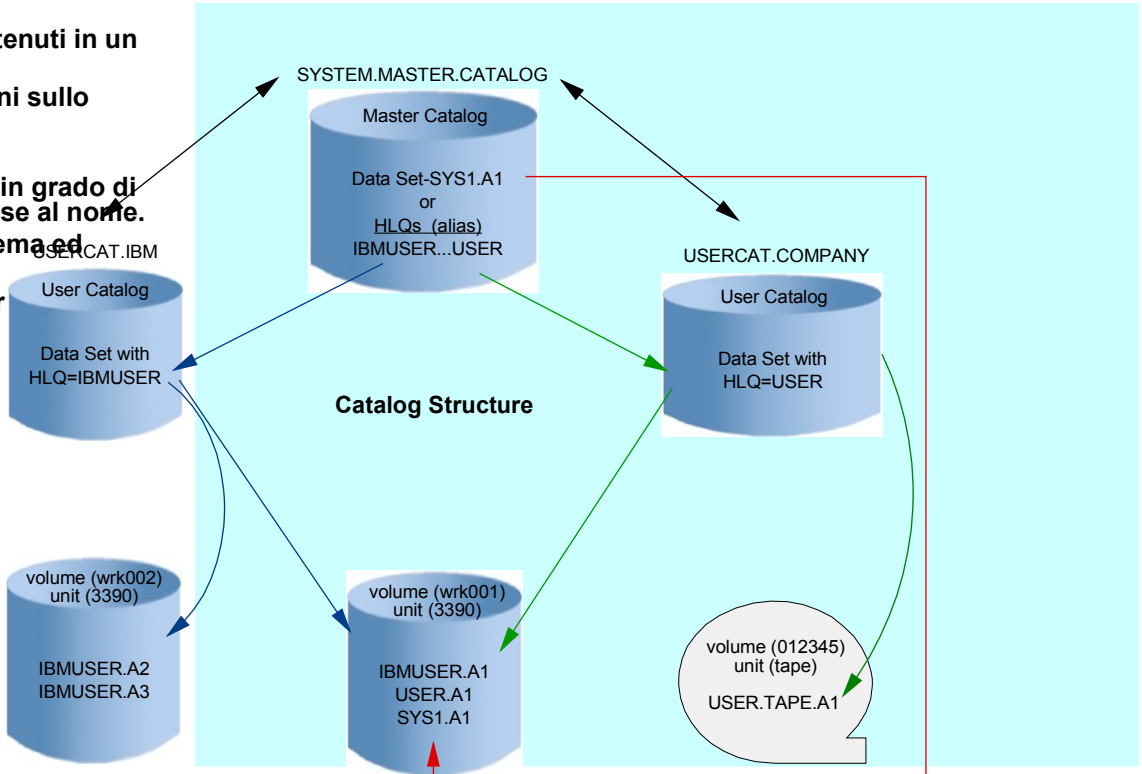
VSAM

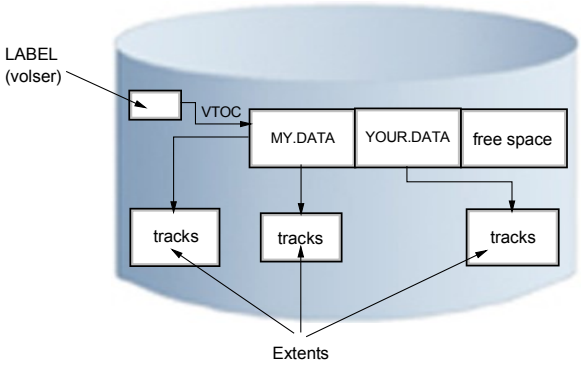
- VSAM **indica un** Virtual Storage Access Method
- **VSAM fornisce una serie di** funzioni **che vanno ben oltre il metodo di accesso (ad esempio** indici, Gestione Spazi ,Utilities**)**
- VSAM record formats**:**
  - **Key Sequence Data Set (**KSDS)
  - **Entry Sequence Data Set (**ESDS**)**
  - **Relative Record Data Set (**RRDS**)**
  - **Linear Data Set (**LDS**)**



Catalogs and VTOCs

- VTOC (volume table of contents):
  - **E' la lista dei data sets contenuti in un volume .**
  - **Contiene anche informazioni sullo spazio vuoto del volume.**
- Catalog
  - **E' una funzione di Sistema in grado di localizzare un dataset in base al nome.**
  - **Esistono Cataloghi di Sistema ed Utente.**
- **z/OS usa entrambe le tecniche per i dati**





Logical Disks

- A logical disk is a storage abstraction that the kernel sees as a linear sequence of fixed sized, randomly accessible blocks.
  - Looks like a *regular* disk to OS
- Advanced Topics:
  - Logical Volume Managers
  - RAID
    - Striping
      - Stripe sets
    - Mirroring

Device I/O

- Two type of devices in Unix:
  - Block
  - Character
- The interface to the kernel are different but basic framework is the same
- Each device is required to support a standard set of operations

Character Device I/O

- Operations are encapsulated in a:

```
struct {
  int (*d_open)();
  int (*d_close)();
  int (*d_read)();
  int (*d_write)();
} cdevsw[];
```

- Major & minor device number:
  - indexes in the device table

read() system call

1) Use the file descriptor to get to the open file object;
2) Check the entry to see if the file is open for read;
3) Get the pointer to the in-core inode from this entry;
4) Lock the inode so as to serialize access to the file;
5) Check the inode mode field and find that the file is a character device file.
6) Use the major device number to index into a table of character devices and obtain the cdevsw entry for this device;
7) From the cdevsw, obtain the pointer to the d_read routine for this device;
8) Invoke the d_read operation to perform the device-specific processing of the read request.

```
result=(*(*cdevsw[major].d_read))(…);
```

9) Unlock the inode and return to the user.

read() system call

- Much of the processing is independent of the specific device
- `cdevsw[]` fields define an *abstract* interface
- E.g. each device implements `d_read` through specific functions:
  - `lpread()`
  - `ttread()`
- The major device number translates the generic `read()` operation

Operations on Files

- One of the most frequently used functions is the “path name traversal”
- Translate a pathname and returns a pointer to the *vnode* of the file
- Starting point can be relative or absolute
- `lookuppn()` locks the starting *vnode* and then parses each component of the pathname

Operations on Files

- lookuppn():
  1. v_type
  2. ".." & system root -> move to next component
  3. ".." & a mounted system root -> access the mount point
  4. VOP_LOOKUP
  5. Not found, last one & SLASHOK -> success (mkdir) else -> ENOENT
  6. A mount point -> follow the pointer to mount point
  7. A symbolic link -> translate it (and restart)
  8. Release the directory
  9. If not at end, go back to 4
  10. Terminate, do not release the reference and return a pointer to the final vnode

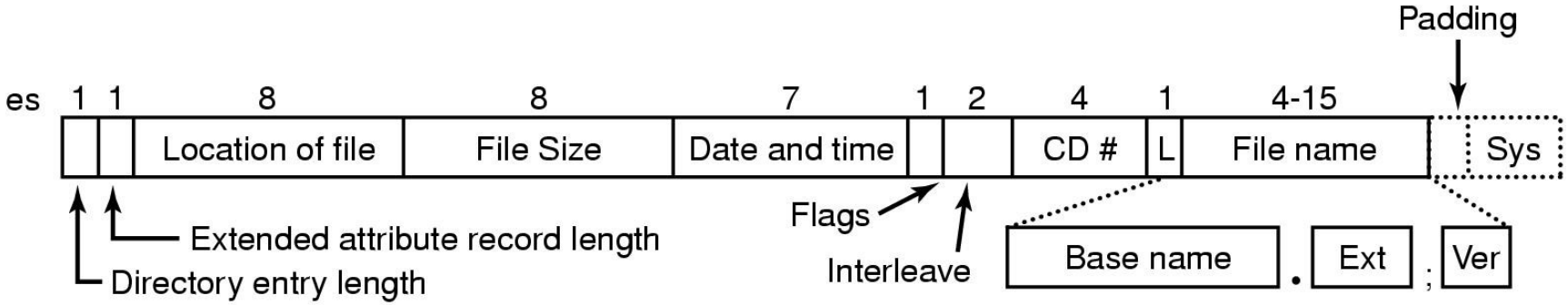
VOP_LOOKUP

- Interface to the filesystem specific function
  - first searches the lookup cache
  - then lookup in the parent directory
    - local search
    - remote search (request to the server)
  - check if the vnode is in memory
  - or allocate and initialize a new vnode, if needed
  - add an entry to the lookup cache

Opening a file

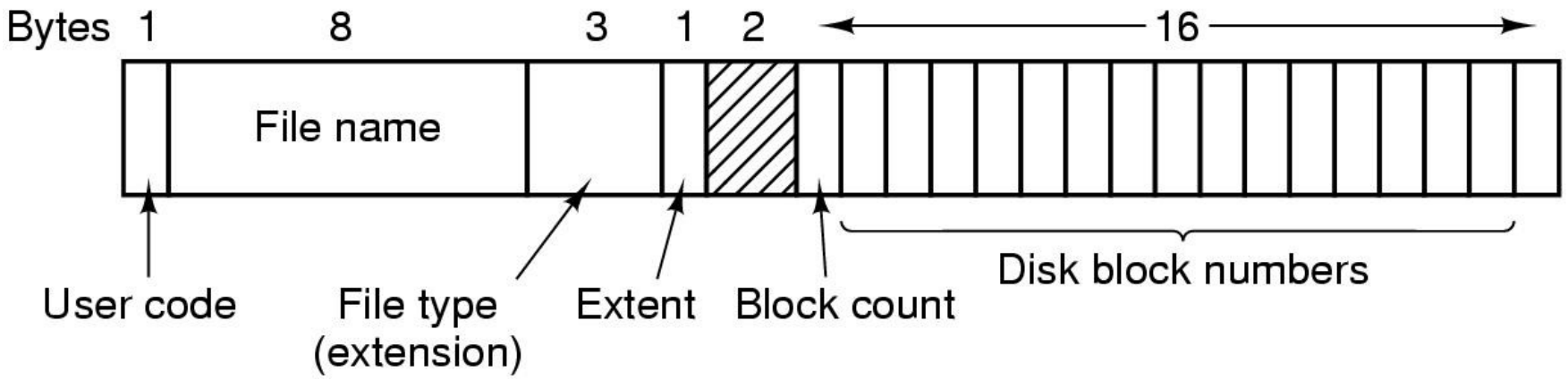
- fd = open(pathname, mode)
  1. Allocate a descriptor
  2. Allocate an open file object
  3. Call lookuppn()
  4. Check the vnode for permissions
  5. Check for the operations
  6. Not exist && O_Creat? VOP_CREAT:ENOENT
  7. VOP_OPEN
  8. O_TRUNC? VOP_SETATTR
  9. Initialize
  10. Return the index of the dexcriptor

Example File Systems
CD-ROM File Systems



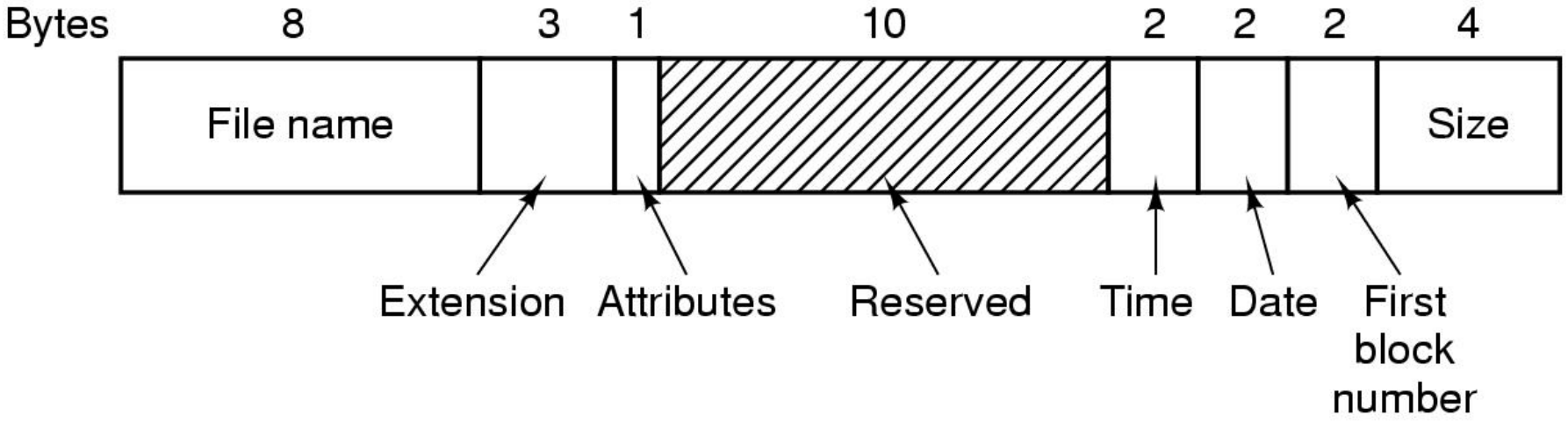
The ISO 9660 directory entry

The CP/M File System



The CP/M directory entry format

The MS-DOS File System



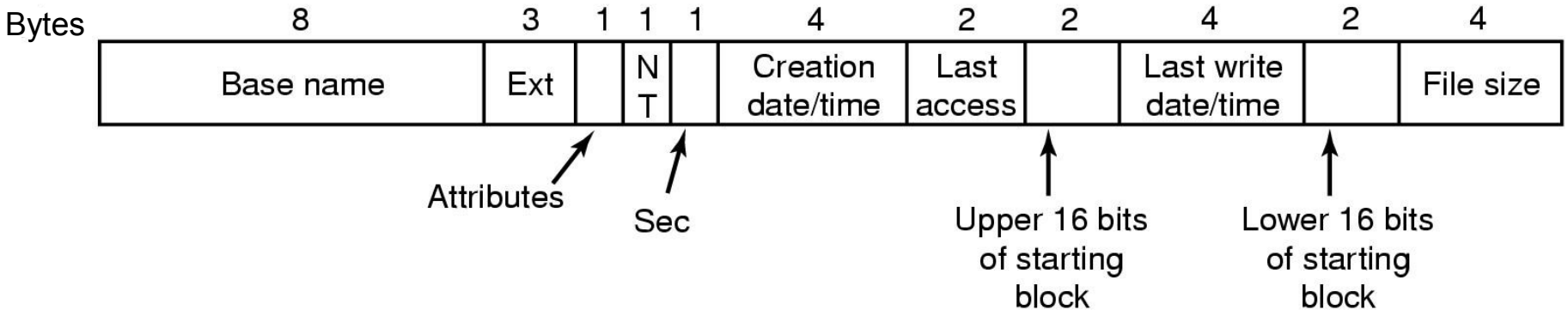
The MS-DOS directory entry

The MS-DOS File System

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

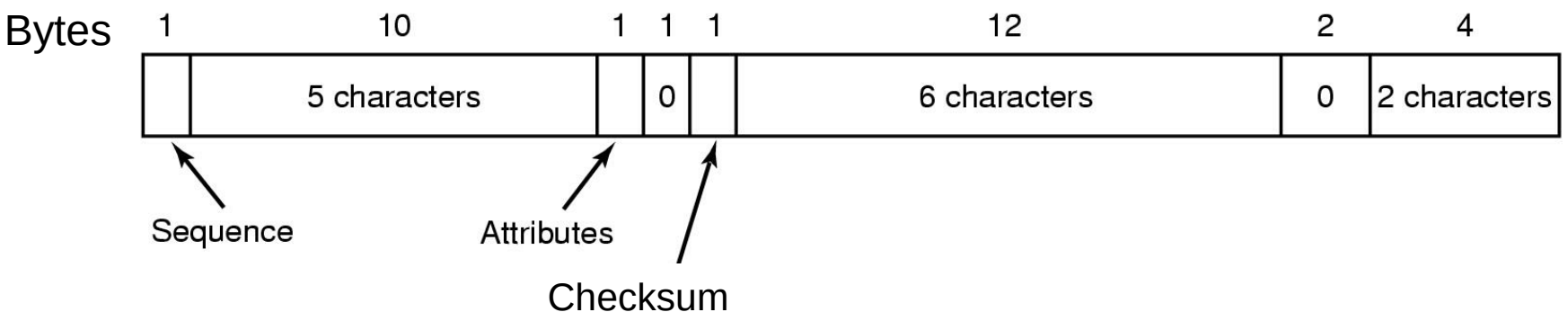
- Maximum partition for different block sizes
- The empty boxes represent forbidden combinations

The Windows 98 File System



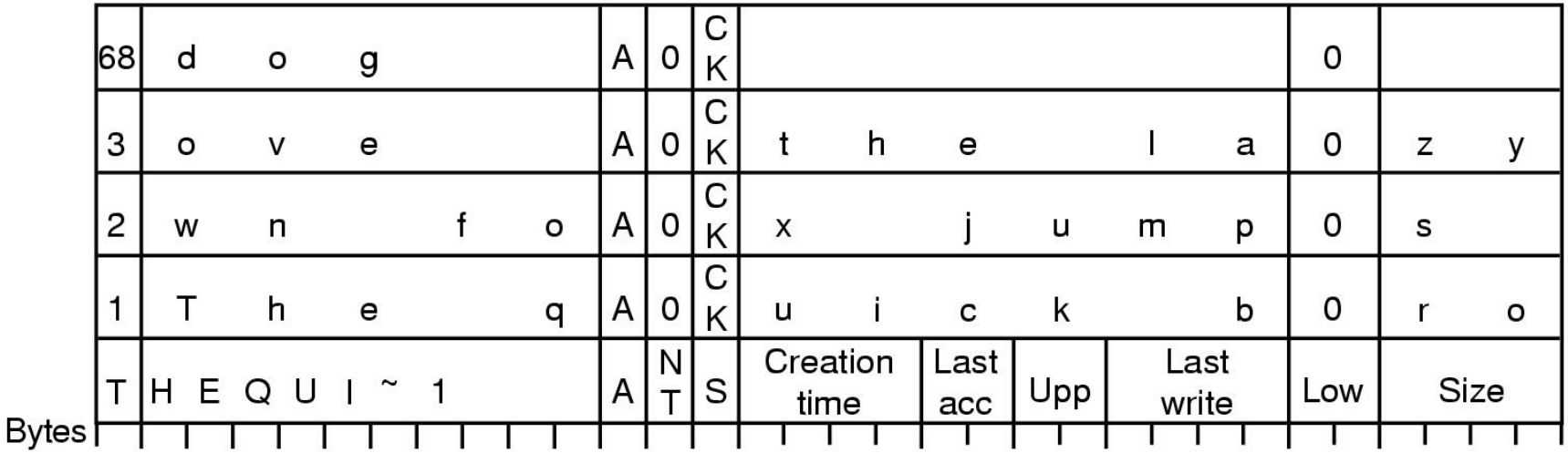
The extended MS-DOS directory entry used in Windows 98

The Windows 98 File System



An entry for (part of) a long file name in Windows 98

The Windows 98 File System



An example of how a long name is stored in Windows 98

The System V File System(s5fs)

- The layout of s5fs partition:

B	S	inode list	data blocks

- Directories:
  - s5fs directory is a special file containing a list of files and subdirectories.

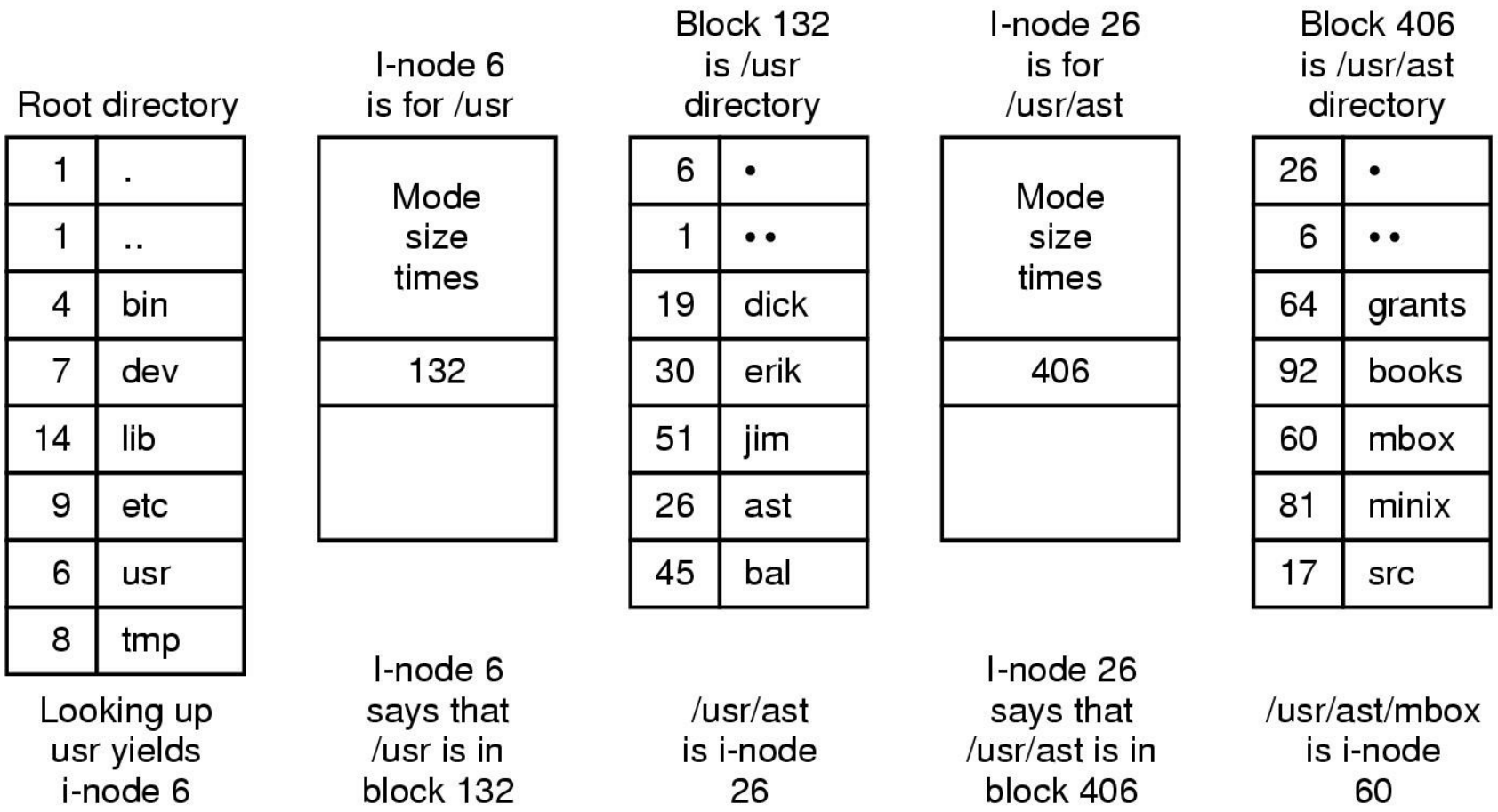
73	.
38	..
9	file1
0	deletedfile
110	subdirectory 1
65	archana

Figure 9-2. *s5fs* directory structure.

inode & Data

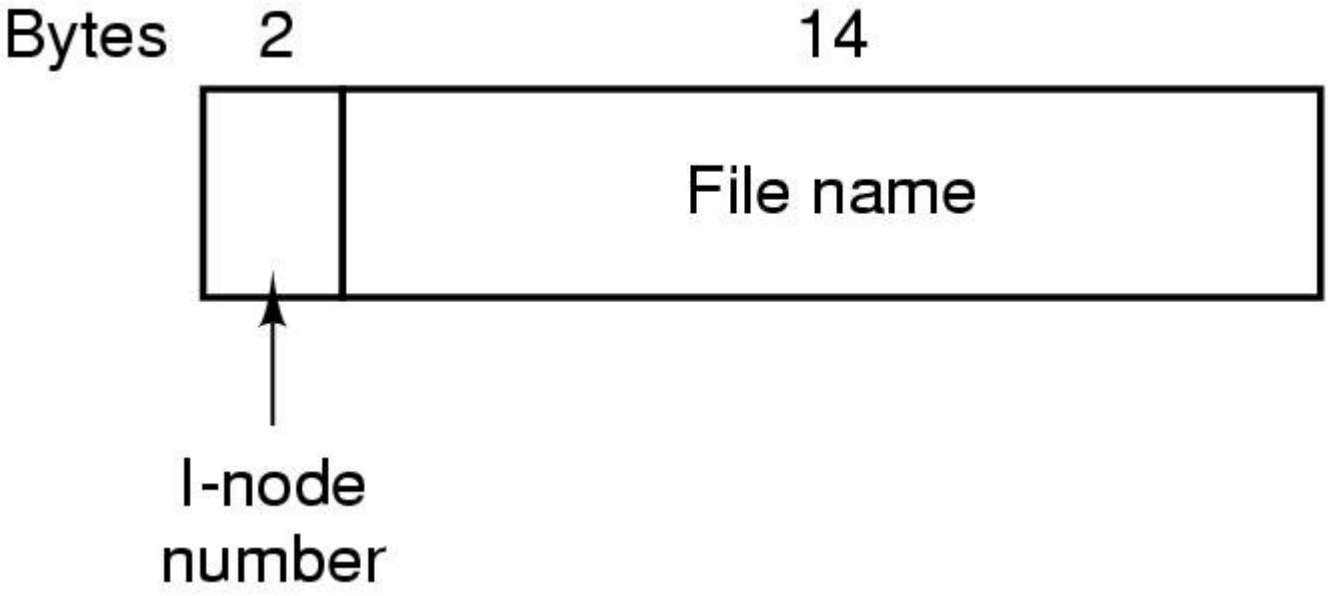
- linear array of inodes
  - inode identified by its index into the list
  - inode is 64 bytes
  - several inodes are stored in one disk block
- Data region - area for allocating data blocks and indirect blocks

The UNIX V7 File System



The steps in looking up */usr/ast/mbox*

The UNIX V7 File System



A UNIX V7 directory entry

Inodes

- The inode contains administrative information,or meta data.
  - The node list contains all the inodes.
  - On-disk inode
  - In-core inode

Inode Fields

Table 9-1. Fields of `struct dinode`

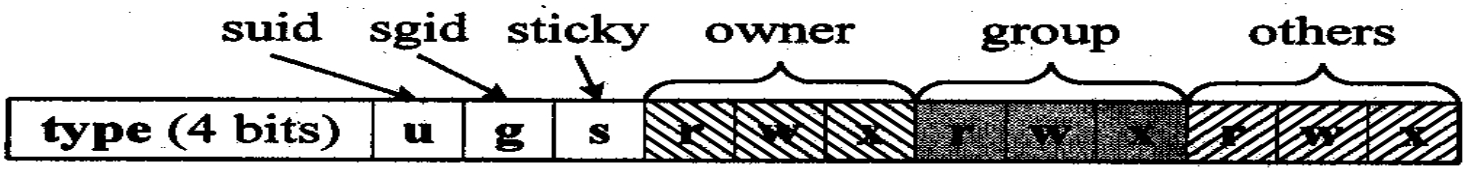
Field	Size (bytes)	Description
`di_mode`	2	file type, permissions, etc.
`di_nlinks`	2	number of hard links to file
`di_uid`	2	owner UID
`di_gid`	2	owner GID
`di_size`	4	size in bytes
`di_addr`	39	array of block addresses
`di_gen`	1	generation number (incremented each time inode is re-used for a new file)
`di_atime`	4	time of last access
`di_mtime`	4	time file was last modified
`di_ctime`	4	time inode was last changed (except changes to `di_atime` or `di_mtime`)

di_mode

MGT - [/etc]

~/MasterINFN /etc

```
-rw-r--r--     1 root     root          66 Nov 13   2001 shells
drwxr-xr-x     3 root     root        4096 Jun 26 11:53 skel
drwxr-xr-x     2 root     root        4096 Nov 13   2001 smrsh
drwxr-xr-x     2 root     root        4096 Jul  1 13:50 snmp
drwxr-xr-x     3 root     root        4096 Jun  7 16:12 sound
drwxr-xr-x     2 root     root        4096 Jul  8 08:34 squid
drwxr-xr-x     2 root     root        4096 Jul  1 13:36 ssh
-r--r-----     1 root     root         417 Dec 22   2001 sudoers
-r--r-----     1 root     root         580 Jan 14   2002 sudoers.rpmnew
drwxr-xr-x     7 root     root        4096 Jun  2 08:01 sysconfig
-rw-r--r--     1 root     root         173 Dec  7   2001 sysctl.conf
-rw-r--r--     1 root     root         693 May  8 12:48 syslog.conf
-rw-r--r--     1 root     root      737535 Jul 20   2001 termcap
-rw-r--r--     1 root     root        8818 Aug 22   2001 timidity.cfg
-rw-r--r--     1 root     root        2600 Jan  8   2002 tux.mime.types
-rw-r--r--     1 root     root         140 Jun 25   2001 updatedb.conf
drwxr-xr-x     3 root     root        4096 Feb 14 09:26 uucp
lrwxrwxrwx     1 root     root          34 Nov 13   2001 vfontcap -> ../usr/share/VFlib/2.2
5.1/vfontcap
lrwxrwxrwx     1 root     root          37 Nov 13   2001 vfontcap.ja -> ../usr/share/VFlib/
2.25.1/vfontcap.ja
drwxr-xr-x     3 root     root        4096 Jul  1 16:55 vfs
drwxr-xr-x     2 root     root        4096 May 20 07:28 vga
drwxr-xr-x     3 root     root        4096 Nov 18   2001 vmware
-rw-r--r--     1 root     root         289 Sep  5   2001 warnquota.conf
-rw-r--r--     1 root     root       23910 Oct 24   2001 webalizer.conf
-rw-r--r--     1 root     root        3956 Sep  5   2001 wgetrc
-rw-------     1 giorgio  giorgio      929 May 18 19:08 wvdial.conf
-rw-r--r--     1 root     root        1333 Dec 25   2001 xcdroast.conf
drwxr-xr-x     2 root     root        4096 Nov 16   2001 ximian
-rw-r--r--     1 root     root         289 Aug 29   2001 xinetd.conf
drwxr-xr-x     2 root     root        4096 Mar 12 18:18 xinetd.d
-rw-r--r--     1 root     root         361 Nov 13   2001 yp.conf
-rw-r--r--     1 root     root        1398 Aug 28   2001 ypserv.conf
[giorgio@gastone etc]$
```



E.g.
Regular file, directory, symbolic link

Figure 9-3. Bit-fields of di_mode.

Block array of inode, di_addr

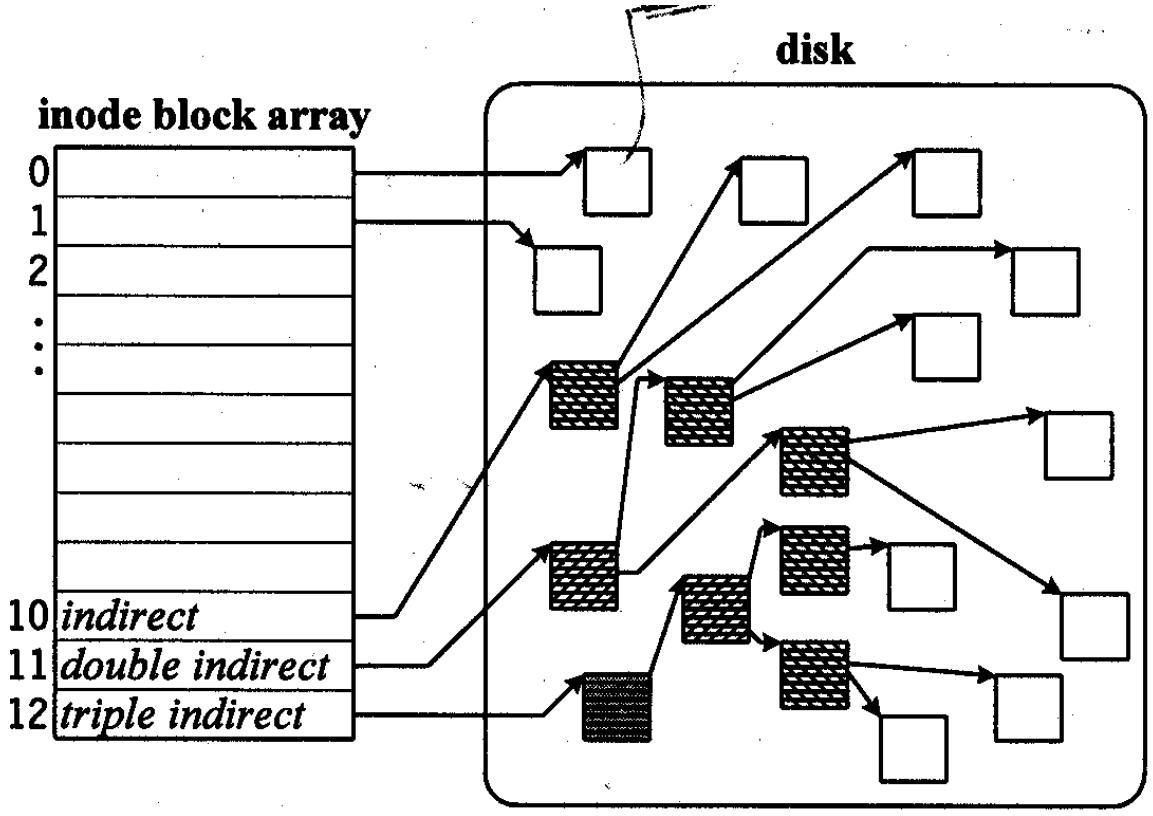


Figure 9-4. disk block array in *s5fs* inode.

Holes

- Files may contain *holes*
  - E.g. create a file, seek to arbitrary offset, write
  - Reading before the offset will return 0s
- Corresponding elements of *di_addr* are set to 0
- Beware of consequences:
  - *cp* might create larger entities
  - *tar* or *cpio* could have problems restoring filesystems

The superblock

- Total size
- Size of Inode list (I-list)
- Header of free block list
- Partial list of free inodes
- Modified flag
- Read-only flag
- Number of free blocks and free inodes

Free Block List

- Superblock stores up to n free disk blocks
- The last of these blocks contains pointers to additional free blocks in a linked list
- Most requests obtain free blocks from superblock
- When last block is consumed a disk read must be done to fetch up the next n blocks from the last block

Free block list

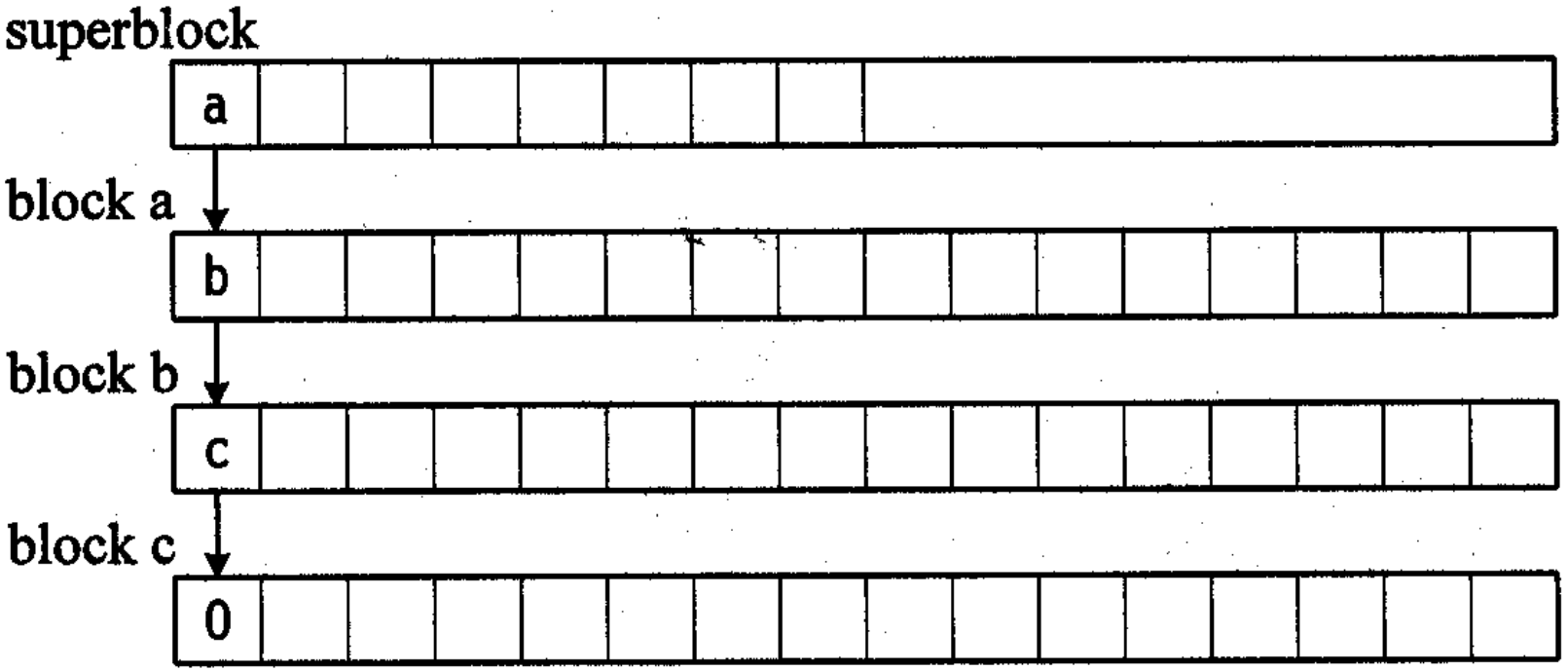
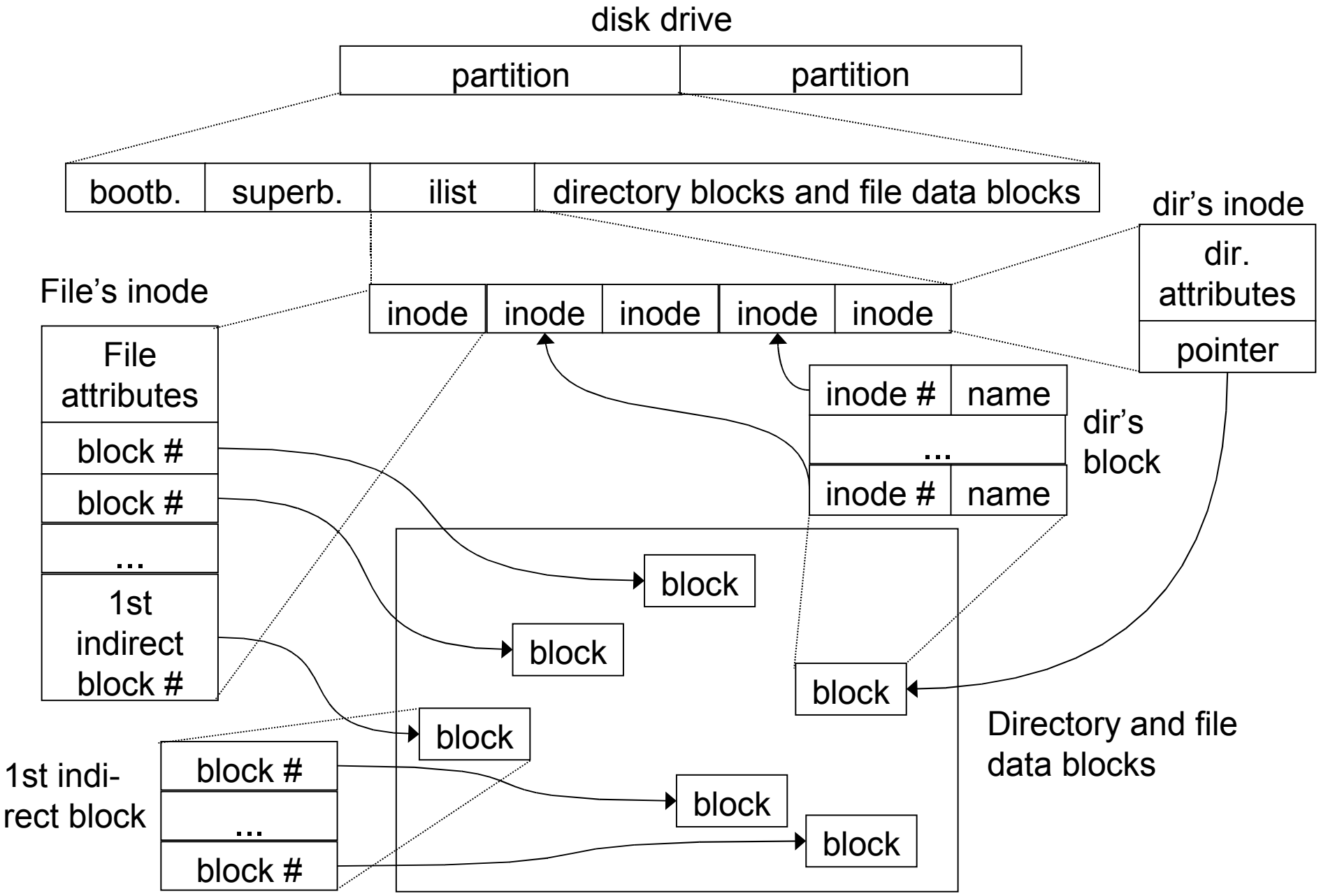


Figure 9-5. free block list in *s5fs*.

Free inode list

- Superblock maintains partial list of free inodes (cache)
- I-list contains (on disk) the complete list of inodes
- When free inode is needed, its index is taken from this cache
- When superblock cache becomes empty, kernel scans the I-list on disk to find free inodes to replenish the list (di_mode==0)

Unix file system “the big picture”



s5fs Kernel Organization

- In-core nodes (*struct inode*)
  - Same fields as *struct dinode* plus:
    - vnode: contains the vnode of the file
    - Device ID of the partition containing the file
    - Inode number of the file
    - Flags for synchronization and cache management
    - Pointers to keep the inode on a free list
    - Pointers to keep the inode on a hash queue
    - Block number of last block read

Working with directories (Lookup)

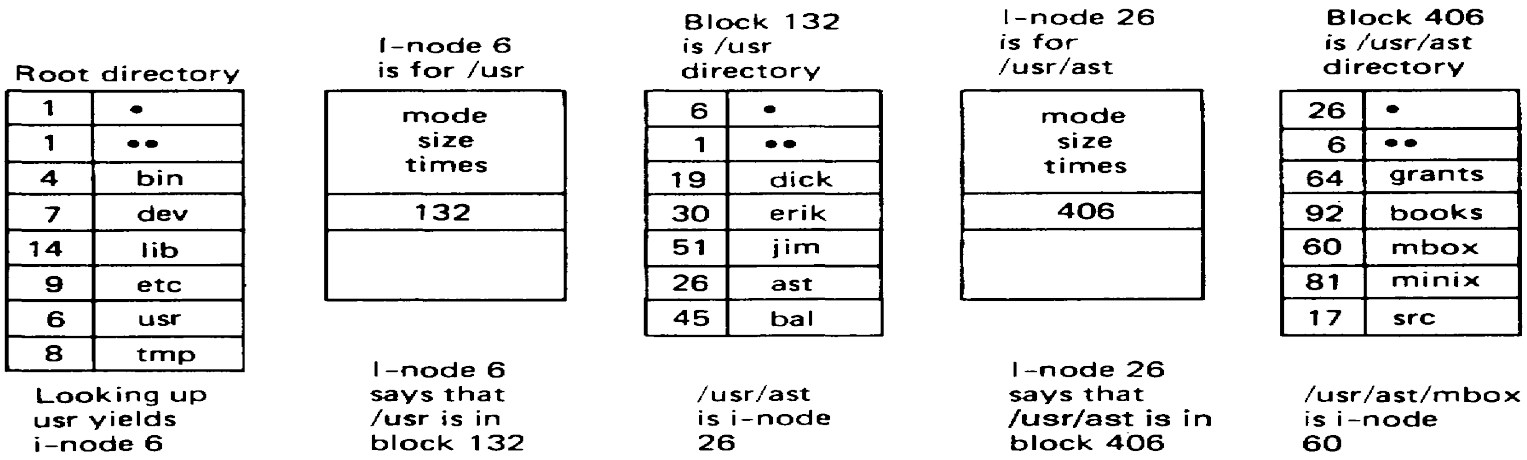


Fig. 4-16. The steps in looking up */usr/ast/mbox*.

- A directory is a table of entries:
  - 2 bytes — inumber
  - 14 bytes — file name (improved in BSD 4.2 and later)
- Search to find the file begins with either root, or the current working directory
  - Inode 2 points to the root directory (“ / ”)
  - Example above shows lookup of /usr/ast/mbox

Inode lookup

- s5lookup()
  - Checks the directory name lookup cache
  - Directory name lookup cache Miss? Reads the directory one block at a time, searching the entries for the specified file name
  - If the file is in the directory, get the inode number, use iget() to locate the inode
  - Inode in the table? No: allocate a new inode, initialize, copy, put in the hash queue, also initialize the vnode(v_ops, v_data, vfs)
  - Return the pointer to the inode

File I/O (1)

- Read(to a user buffer address)
  - Fd-> the open file object, verify mode-> vnode-> get the rw-lock->call s5read()
  - Offset -> block number & the offset
  - uiomove() -> copyout() (one page at atime)
  - page fault? ->s5getpage() -> bmap() (convert logical to physical block#)
  - search vnode's page list, not in? allocates a free page and call the disk driver to read the file
  - sleep until the I/O completes (before copy to user data space, verify the user has access) -> resume copyout()
  - s5read() returns, unlock, advance the offset, return the number of bytes read

File I/O (2)

- write():
  - Similar to read() but with a few differences:
    - Blocks might not immediately written to disk
    - May increase the file size, and require allocation of blocks (data or indirect blocks)
    - If only a parto of a block must be written, then read the entire block, write relevant data, write back the whole block

Allocating and Reclaiming Inodes

- Inodes remain active as long as the vnode has a nonzero reference count
- SVR2:
  - Free inodes are marked as “invalid”, so they have to be read back from disk, if needed
  - Inefficient
- SVR3:
  - LRU replacement
  - Suboptimal (certain inodes are likely to be referenced more often than others)

Allocating and Reclaiming Inodes

- SVR4:
  - Inode table(LRU) containing the active inodes
  - When reference count of a vnode ==0, mark the inode as free (leaving it on the hash list)
- iget():
  - Get 1st inode from list
  - If it has pages in memory, then allocate a new one

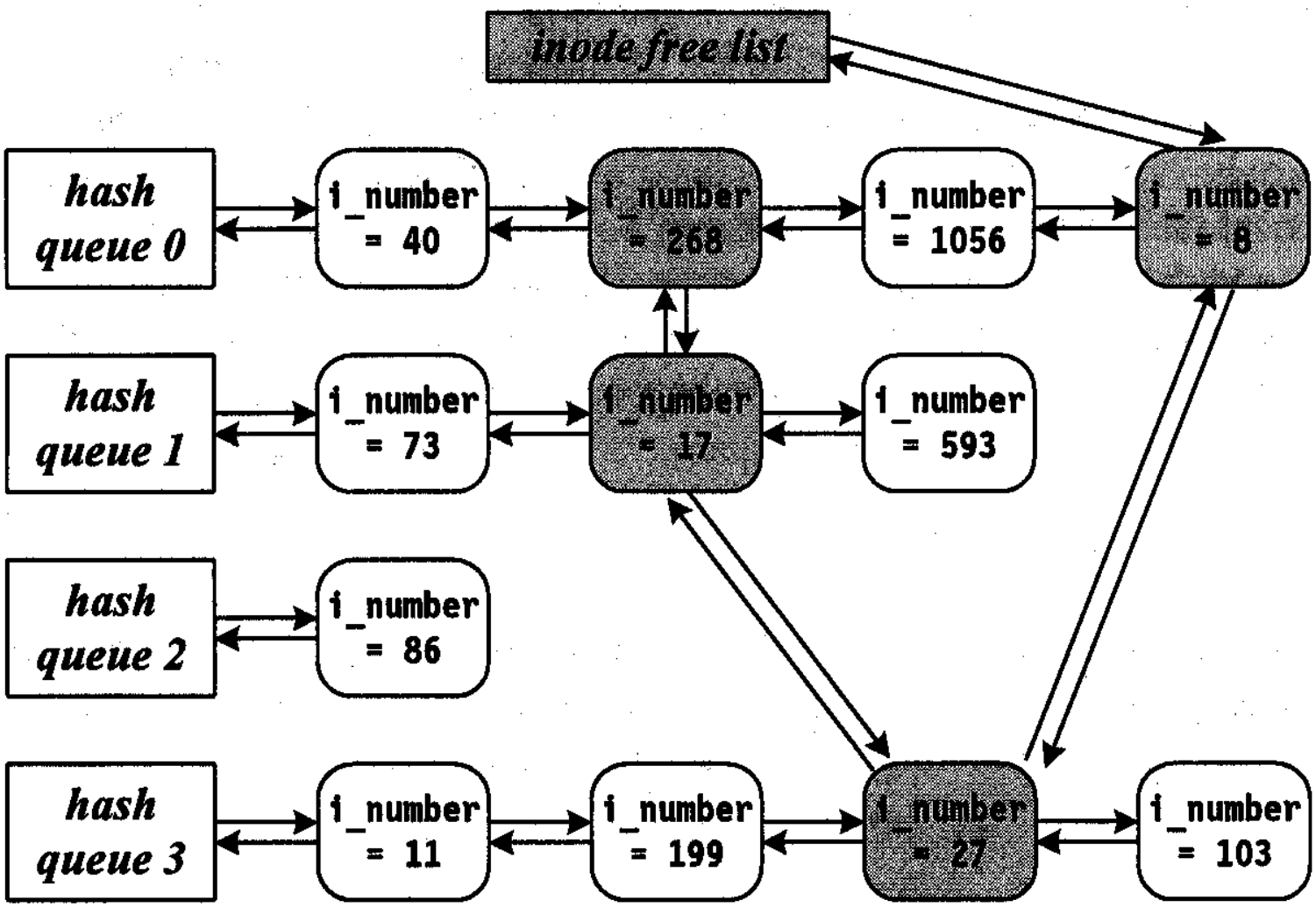


Figure 9-6. Organization of in-core inodes.

Analysis of s5fs

- Advantages:
  - Simple
- Problems:
  - Reliability, Performance, Functionality

Analysis of s5fs

- Reliability
  - Superblock corruption
- Performance
  - Grouping problems:
    - Inodes grouped together followed by data blocks
    - Reading a file means accessing an inode then a long seek to the data blocks

Analysis of s5fs

- Performance
  - Disk block allocation:
    - mkfs configures the free block list optimally so that blocks are allocated in a rotationally consecutive order but as files are created and freed blocks are returned in random order
    - after a while, as blocks become free, list becomes random
  - Disk block size
    - SVR2 uses block size of 512 bytes
    - SVR3 goes to 1024 bytes
    - less overhead on I/O
    - more disk storage wasted
- Functionality
  - File names restricted to 14 characters
  - Limit of 65535 inodes per file system is too restrictive

The Berkeley Fast File System

- Improves performance, reliability and functionality
- Provides all functionality of s5fs, system call handling algorithms and kernel data structures
- Difference in disk layout, on disk structures and free block allocation methods

Berkeley FFS

- On disk organization
  - Disk partition comprises of set of consecutive cylinders on disk
  - FFS further divides the partition into one or more cylinder groups (consecutive cylinders)
- Traditional superblock is divided into two structures
  - FFS superblock contains information like number, size and location of cylinder group, block size, inodes etc.
  - Superblock does not change unless file system is rebuilt
  - Every cylinder group has information about the group including free inodes, free block lists etc
  - Each group has a copy of superblock

FFS Headers

- Boot block: first few sectors
  - Typically all of cylinder 0 is reserved for boot blocks, partition tables, etc.
- Superblock: file system parameters, including
  - Size of partition (note that this is dangerously redundant)
  - Location of root directory
  - Block size
- Cylinder groups, including
  - Data blocks
  - List of inodes
  - Bitmap of used blocks and fragments in the group
  - Replica of superblock (not always at start of group)

Berkeley FFS

- Sector size is 512 bytes
- Unix view of disk is linear array of blocks
- Number of sectors/block = 2^n, n is small number
- Device driver translates block number to logical sector number and the physical track, head and sector number
- Each cylinder contains a sequential set of block numbers
- Head seek time, rotation latency

FFS File Tracking

- Directory: file containing variable-length records
  - File name
  - I-node number
- Inode: holds metadata for one file
  - Located by number, using info from superblock
  - Integral number of inodes in a block
  - Includes
    - Owner and group
    - File type (regular, directory, pipe, symbolic link, …)
    - Access permissions
    - Time of last i-node change, last modification, last access
    - Number of links (reference count)
    - Size of file (for directories and regular files)
    - Pointers to data blocks

FFS Inodes

- Inode has 15 pointers to data blocks
  - 12 point directly to data blocks
  - 13th points to an *indirect block*, containing pointers to data blocks
  - 14th points to a *double* indirect block
  - 15th points to a *triple* indirect block
- With 4K blocks and 4-byte pointers, triple indirect block can address 4 terabytes (2^{42} bytes) of data
- Data blocks might not be contiguous on disk
- But OS tries to *cluster* related items in cylinder group:
  - Directory entries
  - Corresponding inodes
  - Their data blocks

Berkeley FFS

- Blocks and fragments:
  - Advantage & disadvantage of block size
- FFS divides blocks in to fragments:
  - Block size is 2^n , min = 4096, larger then s5fs (512/1024 bytes)
  - Replacing the free block list with a bitmap that track each fragment
- Useful for small files
- Lower bound of fragments = 512 bytes
- File has complete disk blocks except last, which may contain one or more *consecutive* fragments
- First block should be a single block not set of fragments
- Occasional recopying of data when file grows
  - FFS controls this by allowing only direct block to contain fragments

Berkeley FFS

- Allocation Policies:
  - In s5fs, free inode and block lists locate blocks randomly, except after the file system creation
  - FFS aim to collocate related information on the disk to optimize sequential access
  - FFS places inodes of all the files of a single directory into same cylinder group
  - Create new directory in a different cylinder group from the parent (for uniform distribution)
  - Place data blocks of file in the same cylinder group as inodes
  - Change cylinder group when the file reaches 48KB size and again at every MB
  - Allocate sequential blocks of a file at rotationally optimal positions

FFS Free-Space Management

- Free space managed by bitmaps
  - One bit per block
  - Makes it easy to find groups of contiguous blocks
- Each cylinder group has own bitmap
  - Can find blocks that are physically nearby
  - Prevents long scans on full disks
- Prefer to allocate block in cylinder group of last previous block
  - If can’t, pick group that has most space
  - Heuristic tries to maximize number of blocks close to each other

FFS Fragmentation

- Blocks are typically 4K or 8K
  - Amortizes overhead of reading or writing block
  - On average, wastes 1/2 block (total) per file
- FFS divides blocks into 4-16 *fragments*
  - Free space bitmap manages fragments
  - Small files, or tails of files are placed in fragments
  - This turned out to be terrible idea
    - Greatly complicates OS code
    - Didn't foresee how big disks would get
- Linux EXT2 uses smaller block size (typically 1K) instead of fragments

Berkeley FFS

- FFS Functional enhancements
  - Long file names- 255 characters and variable directory entry length
  - Symbolic links: a file that points to another file
    - Type field of the inode identifies the file as symbolic link

Berkeley FFS

- Analysis
  - Read throughput increases from 29KB/s in s5fs to 221 KB/s in FFS (VAX/750 … 1983!)
  - CPU utilization increases from 11% to 43%
  - Write throughput increases from 48KB/s to 142 KB/s
  - On the average, it is lost half block per file in s5fs and half fragment per file in FFS
    - Same when fragment size equals block size
  - Overhead to maintain fragments