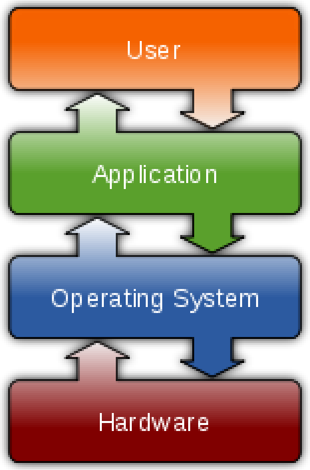
ENCE360
Operating Systems



File Systems

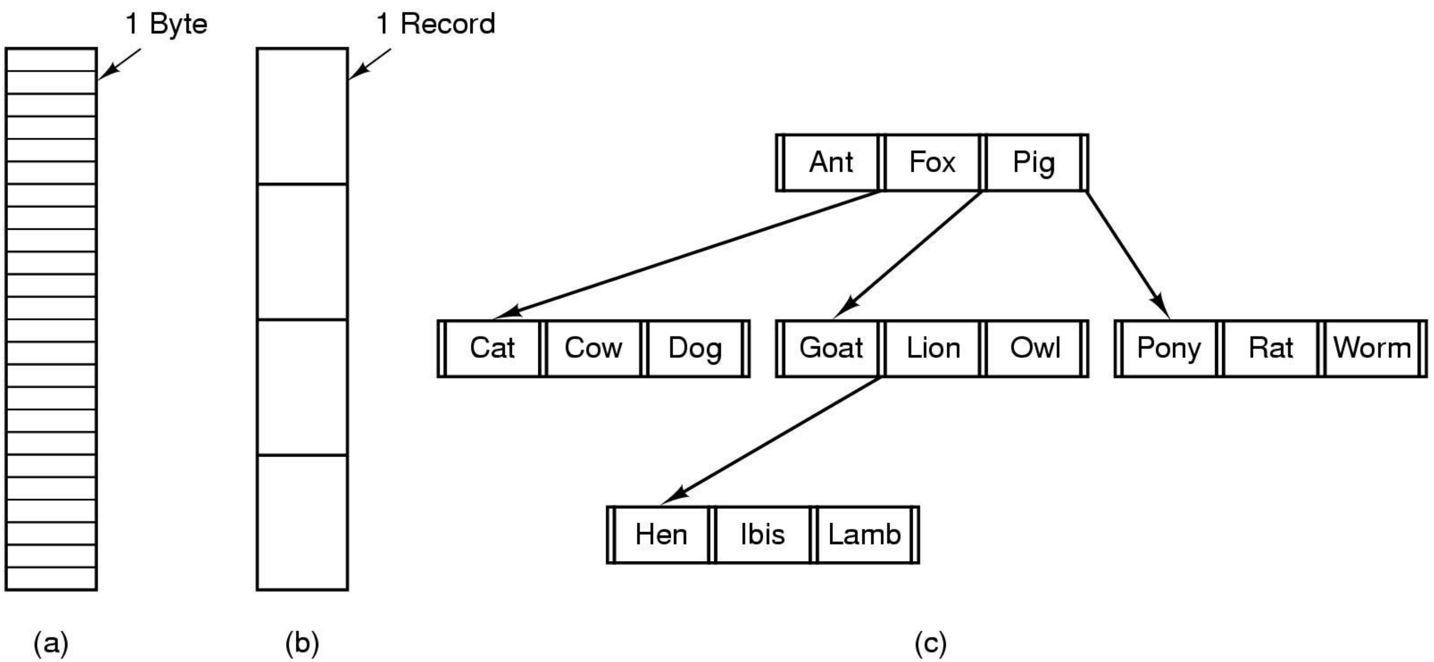
- Files
- Directories
- File system implementation
- Example file systems

MOS Ch 6

Long-term Information Storage

1. Must store large amounts of data
2. Information stored must survive the termination of the process using it
3. Multiple processes must be able to access the information concurrently

File Structure



- Three kinds of files
  - byte sequence
  - record sequence
  - tree

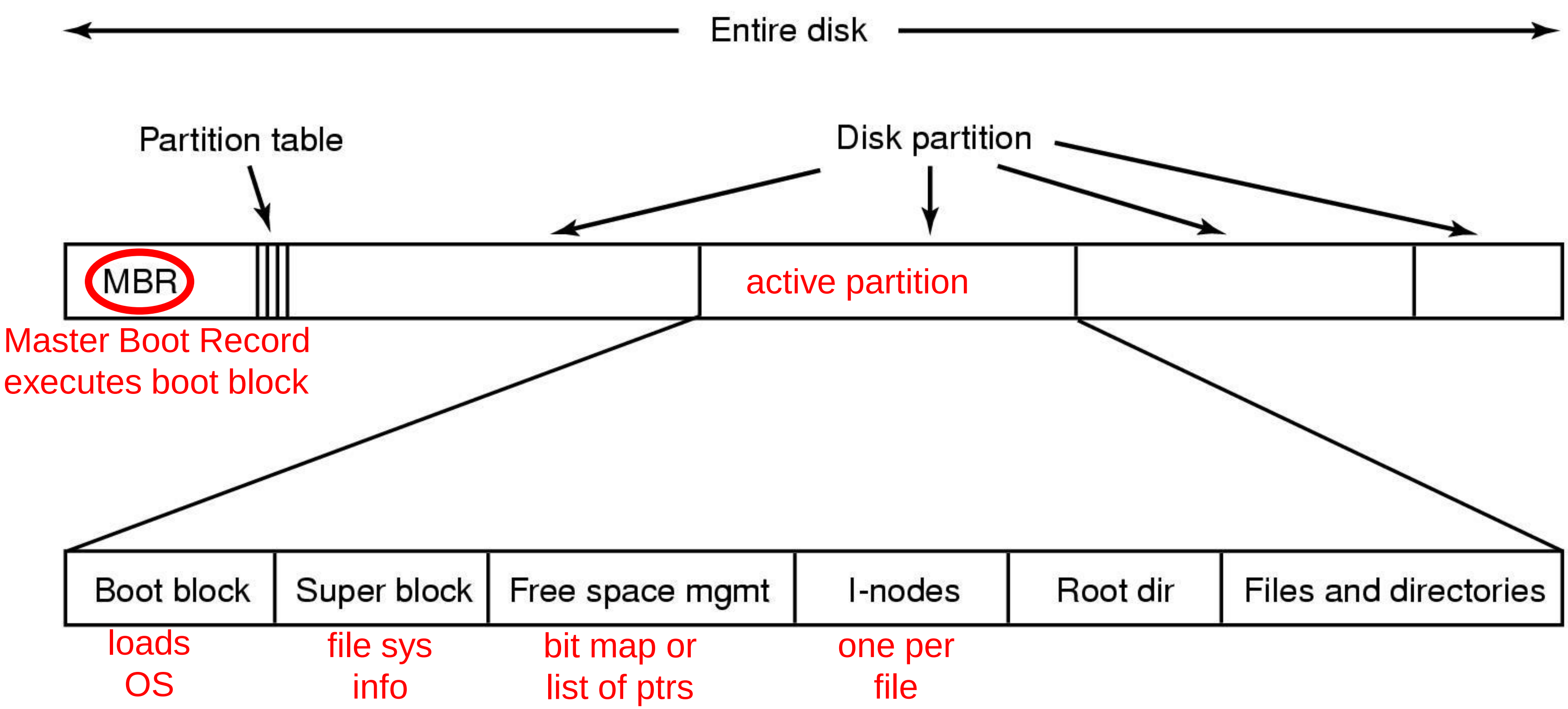
File Access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was mag tape
- Random access
  - bytes/records read in any order
  - essential for data base systems
  - read can be …
    - move file marker (seek), then read or …
    - read and then move file marker

File Operations

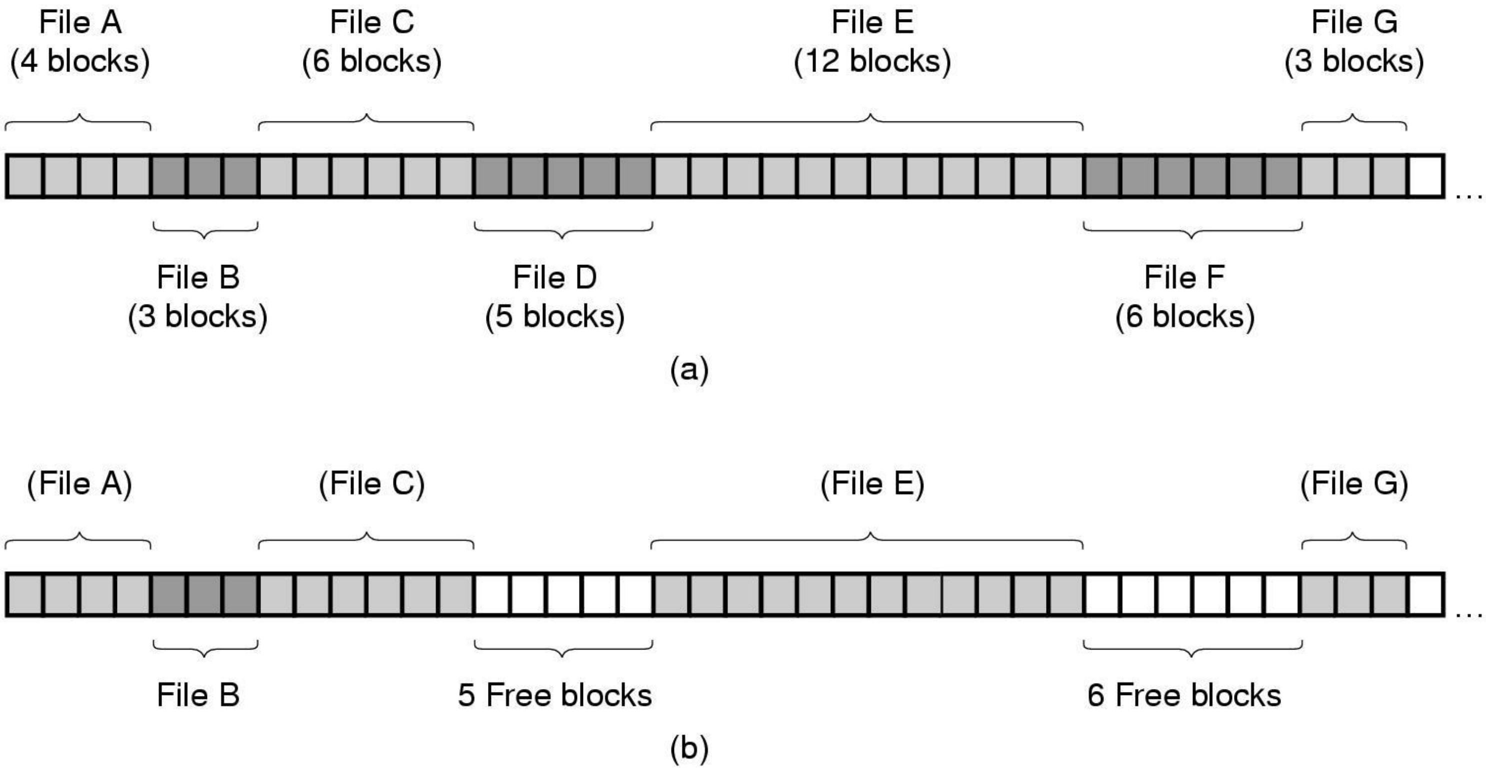
1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename

File System Implementation



A possible file system layout

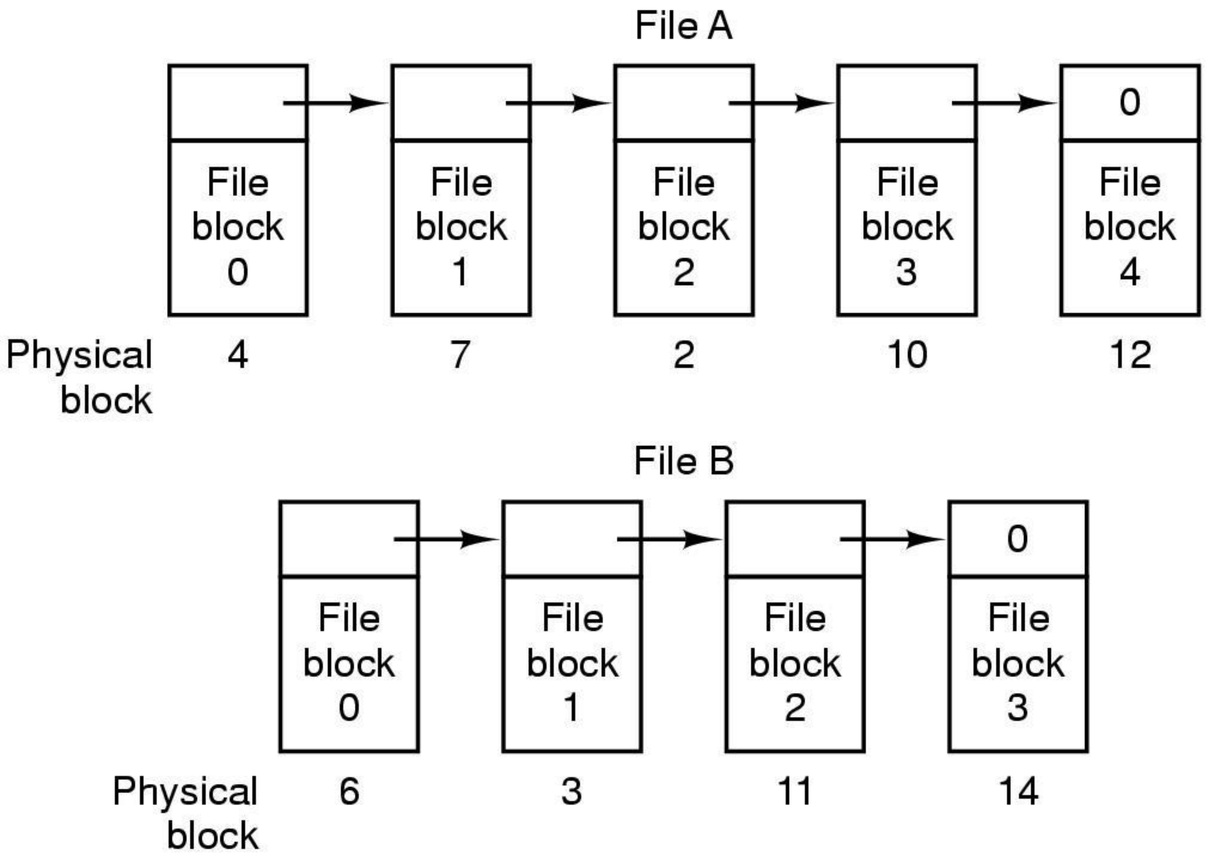
Implementing Files (1)



(a) Contiguous allocation of disk space for 7 files

(b) Fragmened state of the disk after files *D* and *E* have been removed

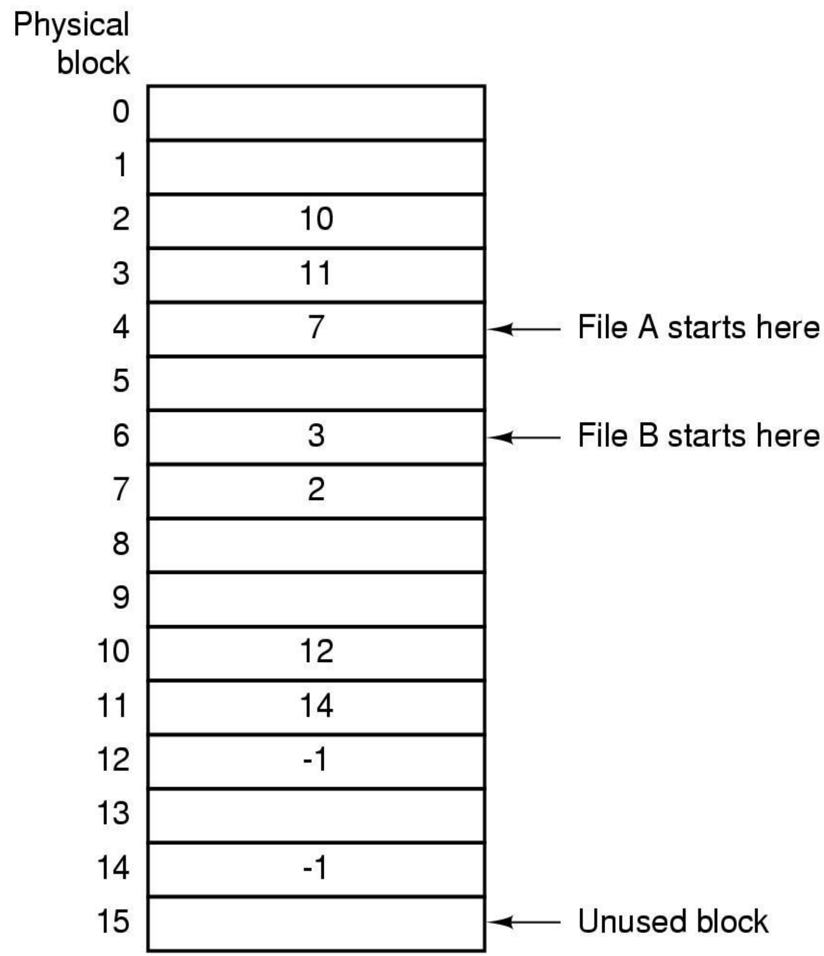
Implementing Files (2)



Storing a file as a linked list of disk blocks

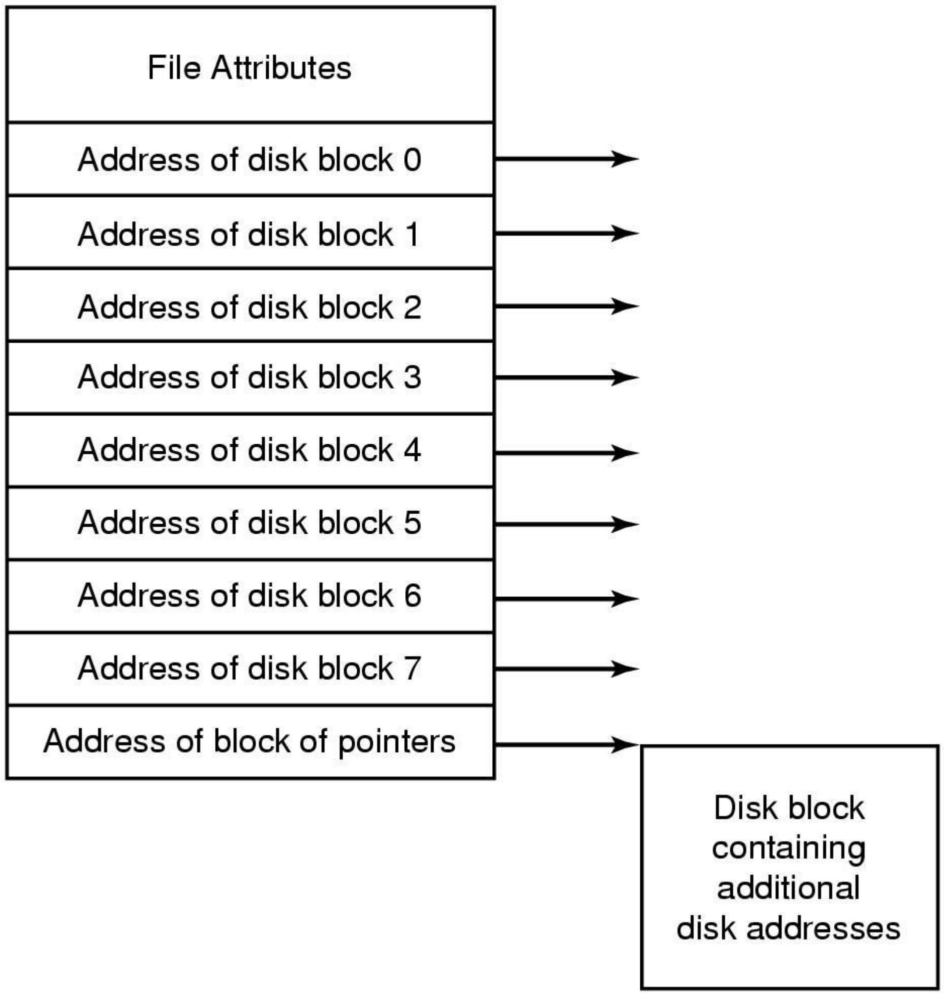
But slow sequential reads and unusual block sizes

Implementing Files (3)



Linked list allocation using a file allocation table (FAT) in RAM
But 20GB disk with 1KB blocks with 4byte entries (pointers) = 80MB

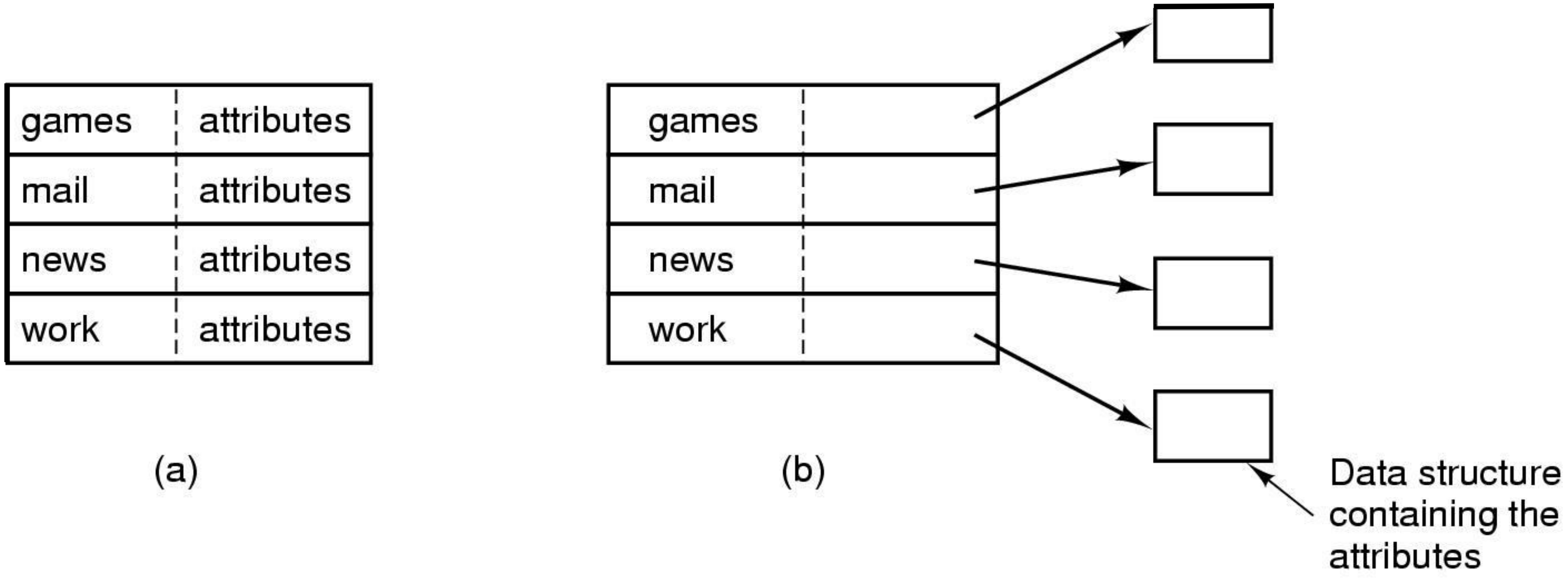
Implementing Files (4)



An example i-node

Only needs to be in memory when the file is open

Implementing Directories (1)



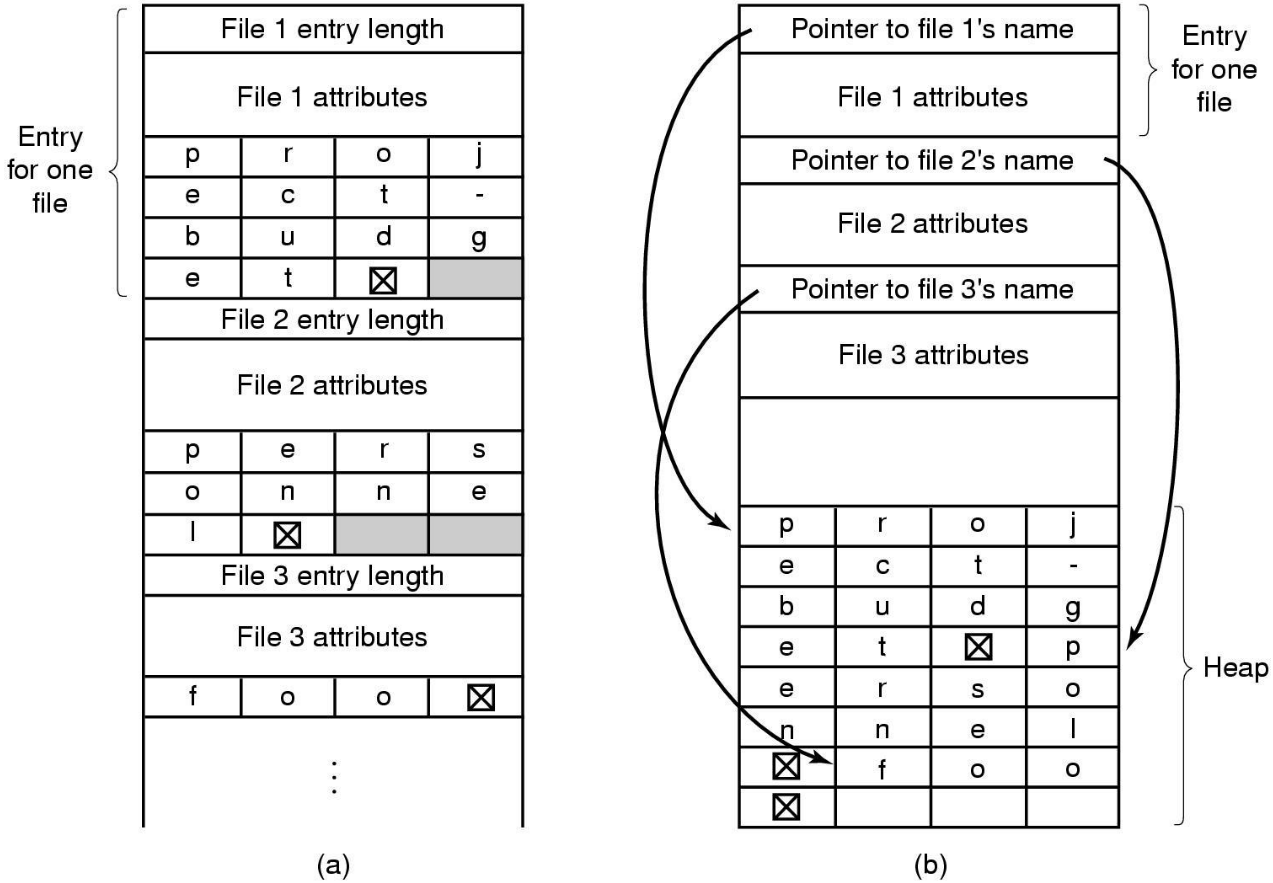
(a) A simple directory

fixed size entries

disk addresses and attributes in directory entry

(b) Directory in which each entry just refers to an i-node

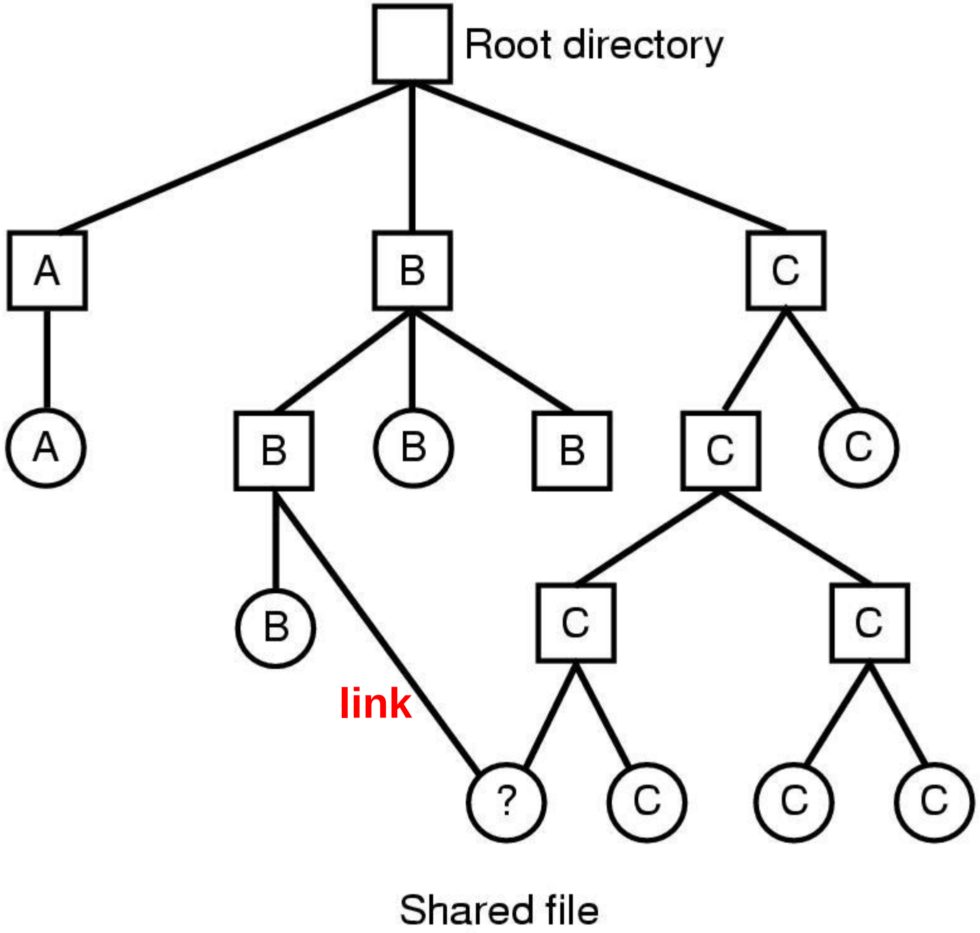
Implementing Directories (2)



- Two ways of handling long file names in directory
  - (a) In-line (becomes fragmented) – slow linear search
  - (b) In a heap (need to manage heap) – slow linear search
- Two more ways for a faster lookup:

  (c) Hash table (d) Cache search history

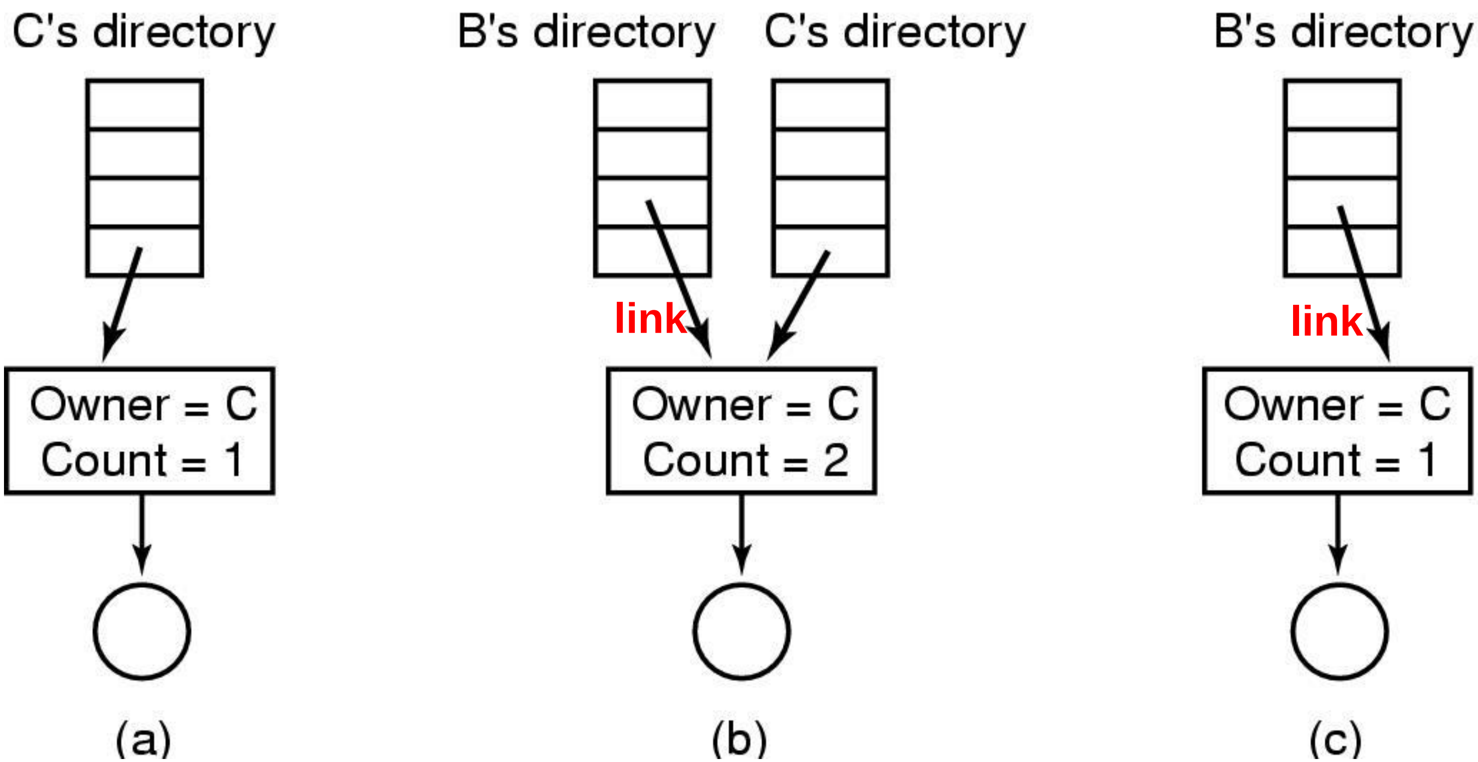
Shared Files (1)



File system containing a shared file

(Now a Directed Acyclic Graph (DAG) instead of a tree)

Shared Files (2)

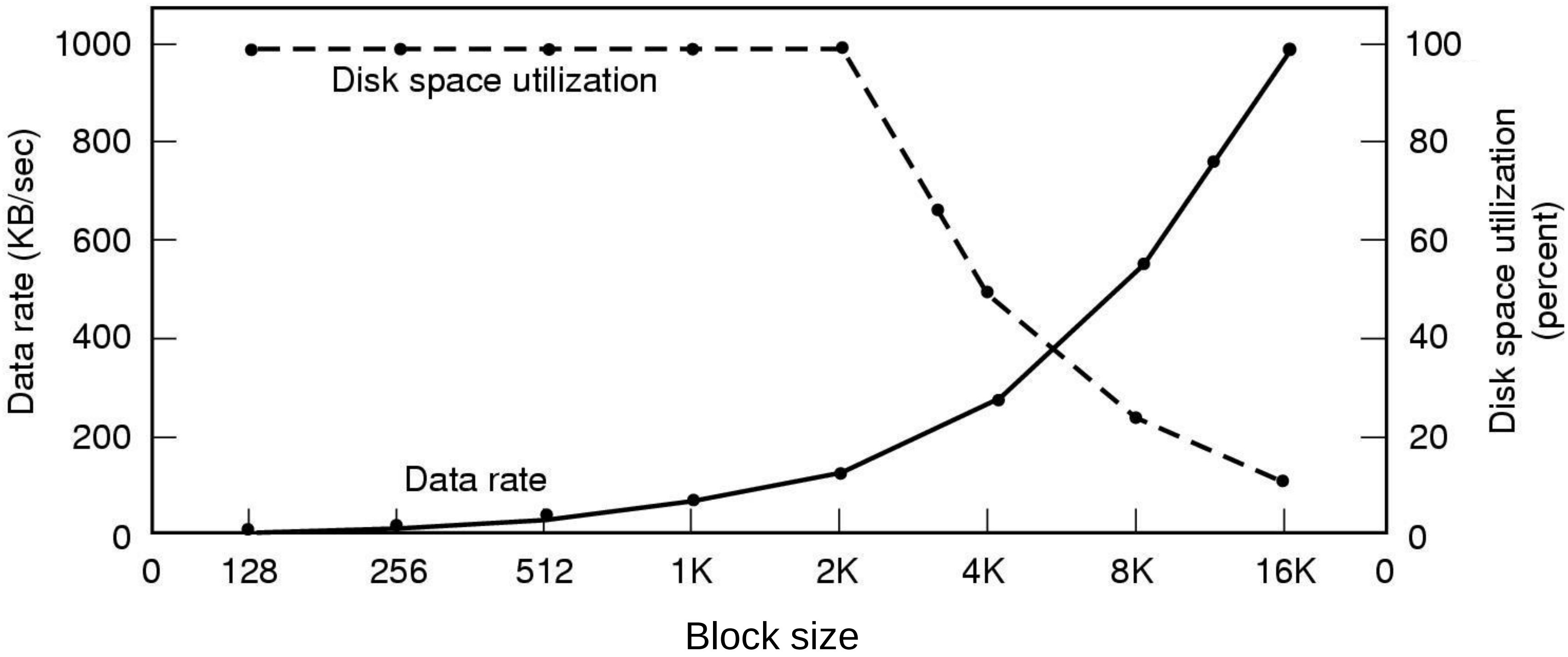


(a) Situation prior to linking

(b) After the link is created

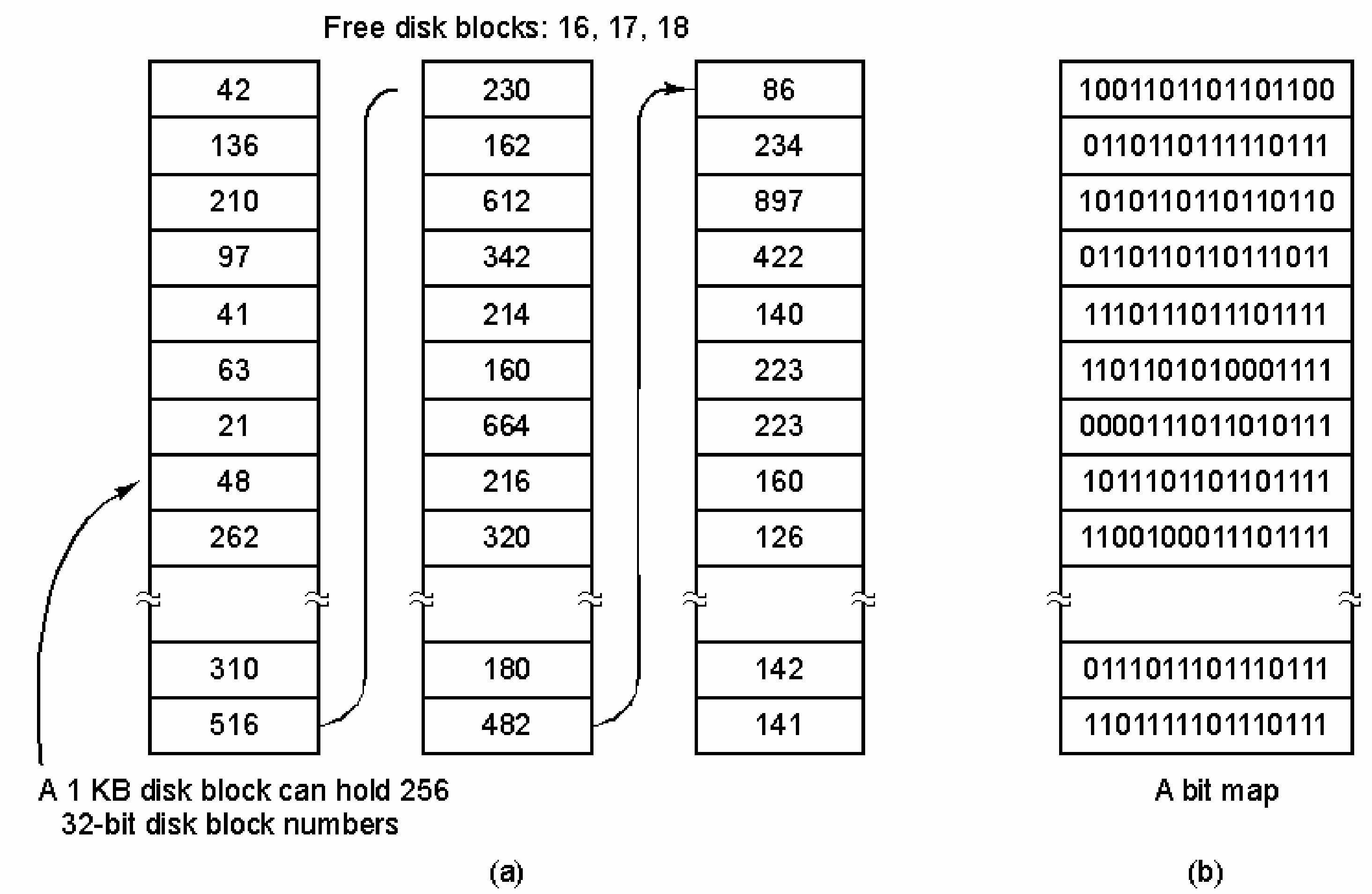
(c) After the original owner removes the file - problem

Disk Space Management (1)



- Dark line (left hand scale) gives data rate of a disk
- Dotted line (right hand scale) gives disk space efficiency
- All files 2KB

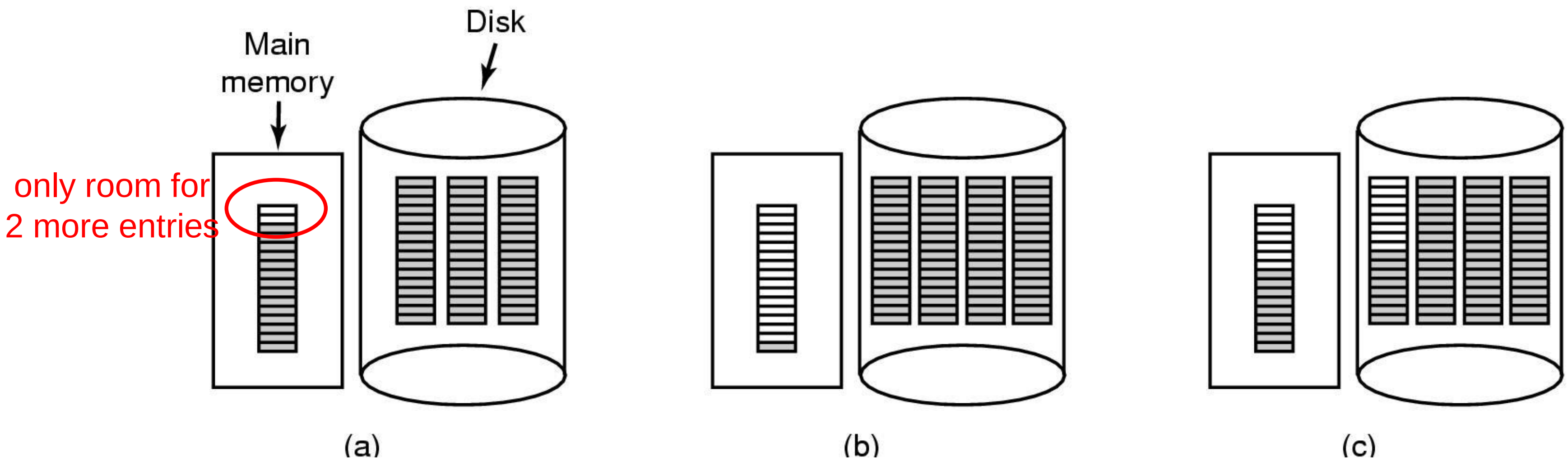
Disk Space Management (2)



(a) Storing the free list on a linked list **(only better if disk nearly full)**

(b) A bit map - one bit per free block

Disk Space Management (3)



shaded entries are pointers to free disk blocks

(a) Almost-full block of pointers to free disk blocks in RAM

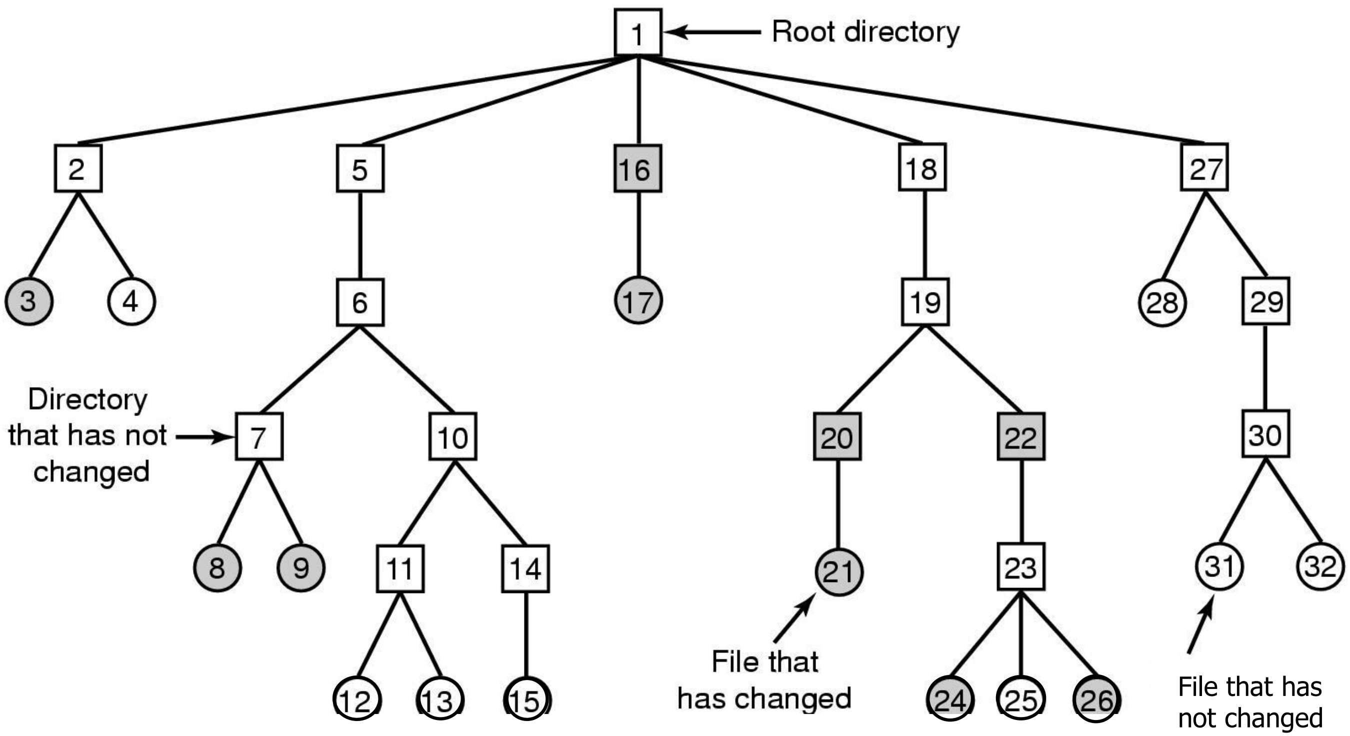
- three blocks of pointers on disk

(b) Result of freeing a 3-block file

(c) Alternative strategy for handling 3 free blocks

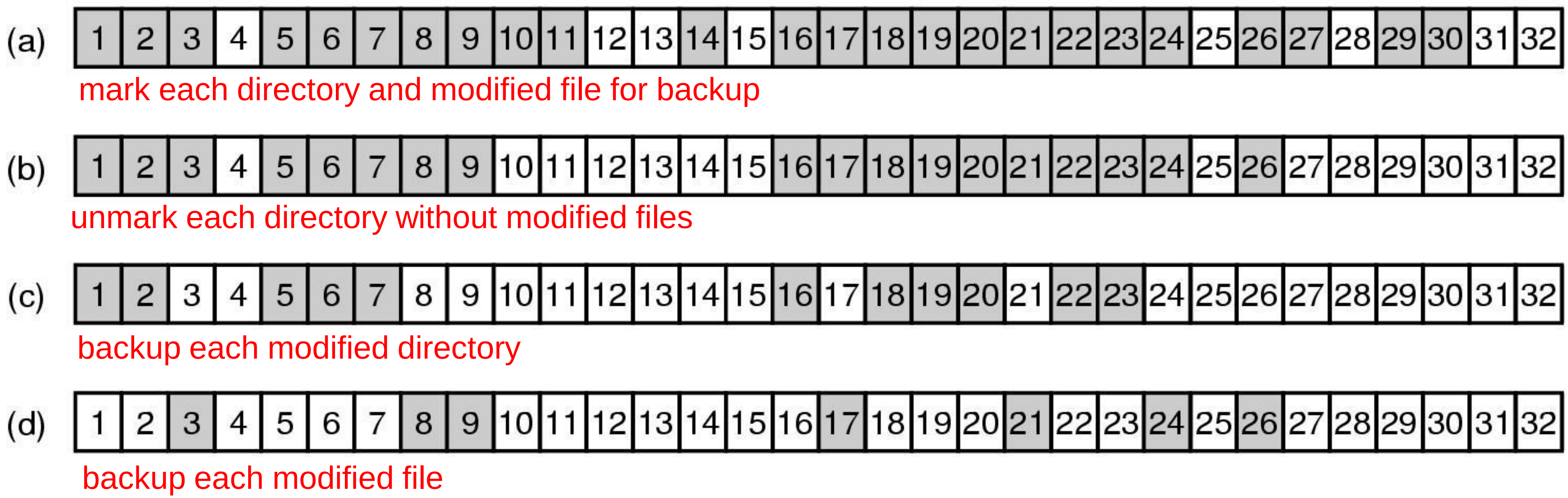
- keep the one in memory about half full so it can handle file create and remove without disk I/O on the free list

File System Reliability (1)



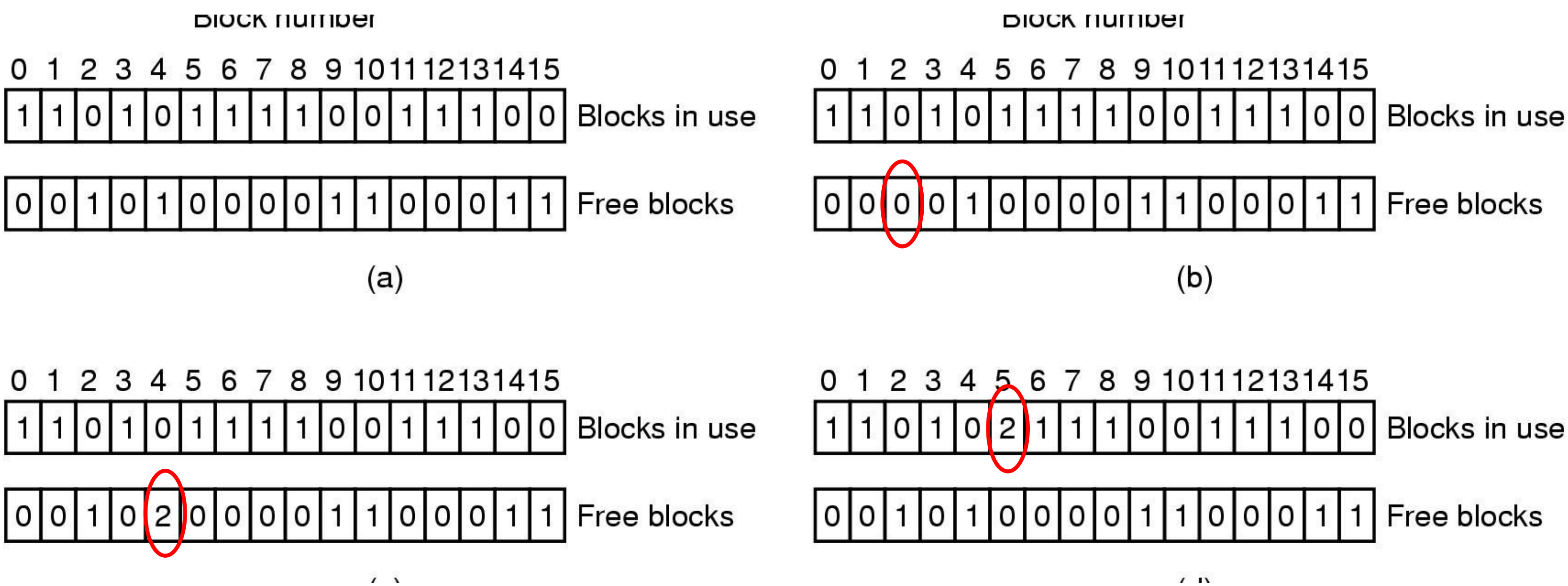
- A file system to be backed up
  - squares are directories, circles are files
  - shaded items, modified since last backup
  - each directory & file labeled by i-node number

File System Reliability (2)



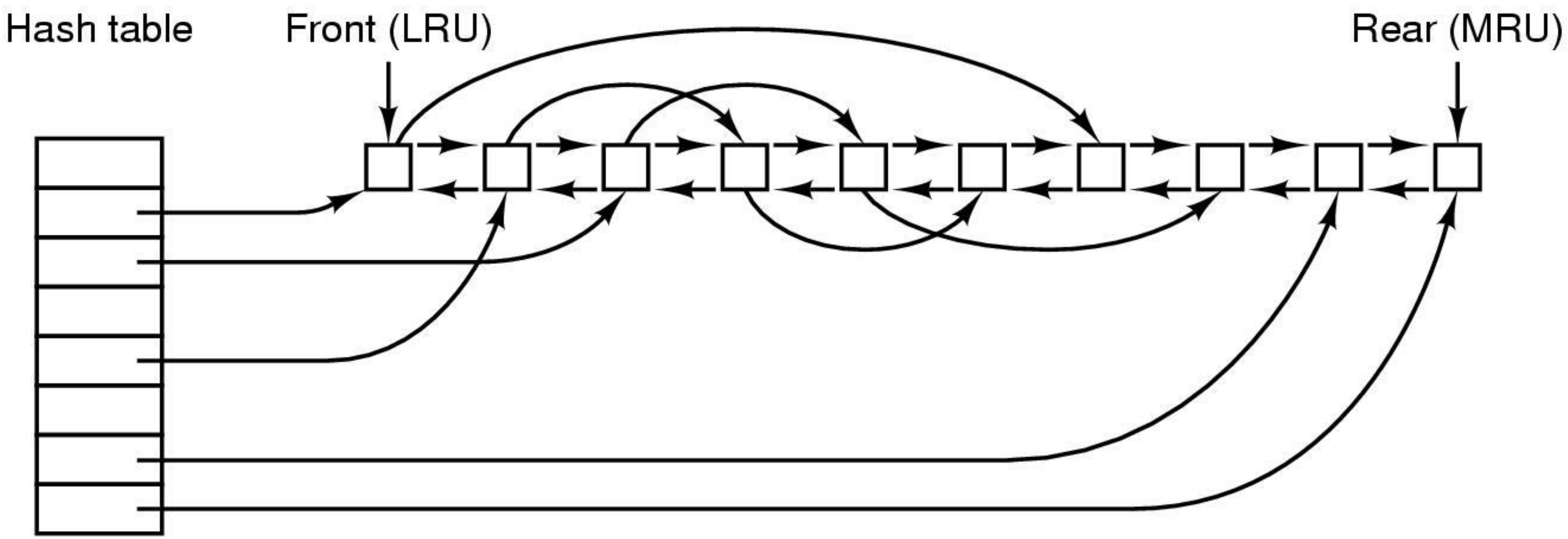
Bit maps used by the logical backup algorithm

File System Reliability (3)



- File system states
  - (a) consistent
  - (b) missing block (e.g. after system crash)
  - (c) duplicate block in free list
  - (d) duplicate data block (worst case)

File System Performance (1)

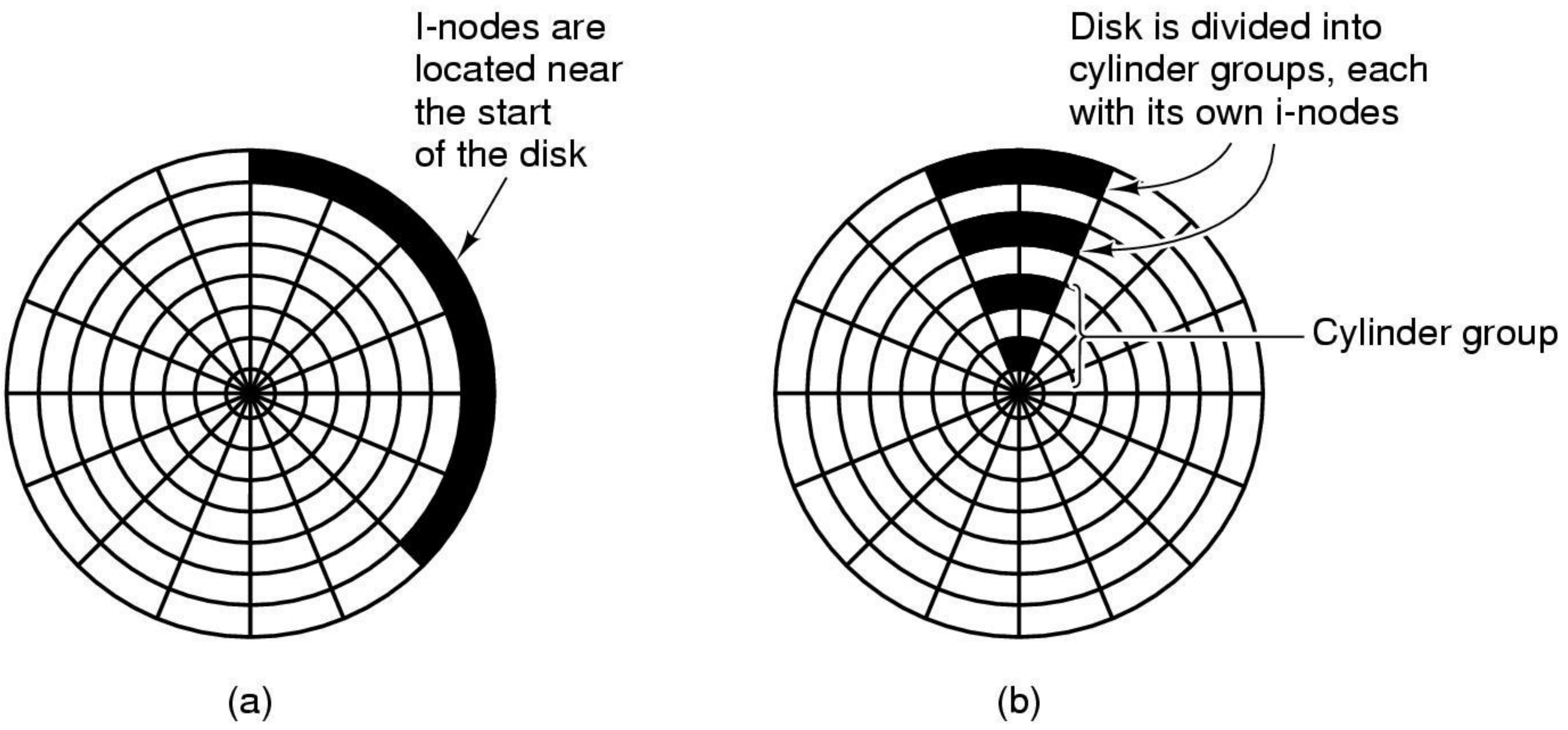


The block cache data structures

(all blocks with the same hash value are linked into a collision chain)

(cache from *cacher – to hide – French)*

File System Performance (2)



- I-nodes placed at the start of the disk
- Disk divided into cylinder groups
  - each with its own blocks and i-nodes

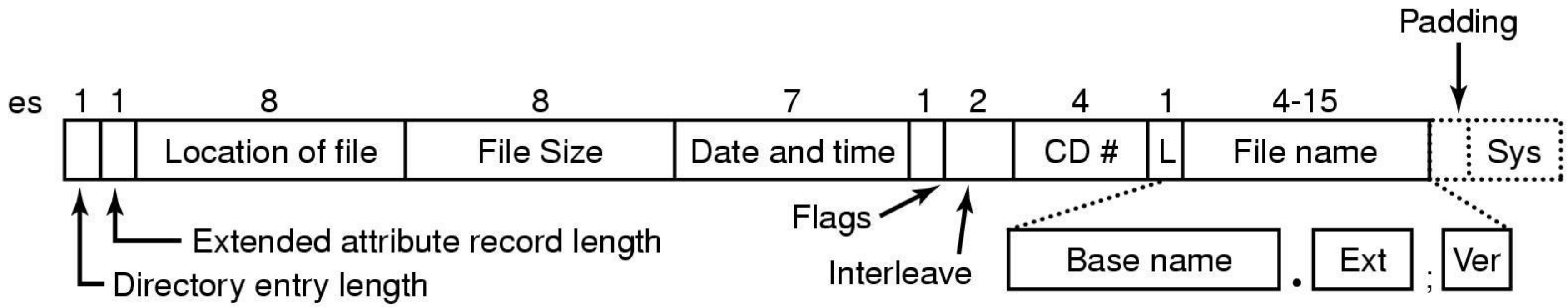
Log-Structured File Systems (LFS)

- Log-structured File System (LFS) Strategy structures the entire disk as a log
  - have all writes initially buffered in memory
  - periodically write these to the end of the disk log
  - when file opened, locate i-node, then find blocks

- With CPUs faster, memory larger
  - disk caches can also be larger
  - increasing number of read requests can come from cache
  - thus, most disk accesses will be writes

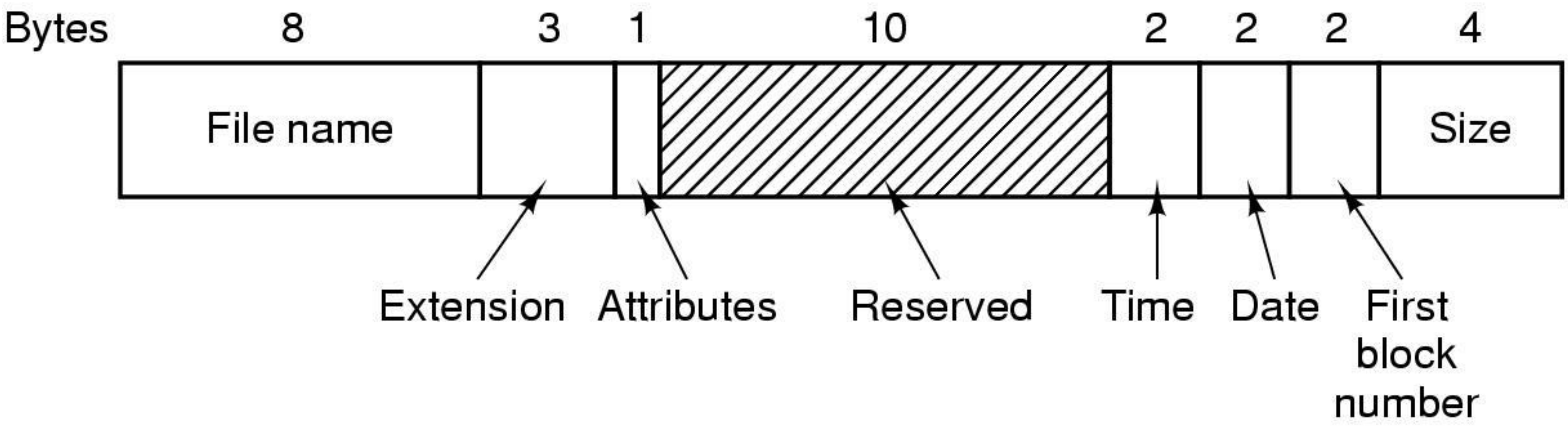
Example File Systems

CD-ROM File Systems



The ISO 9660 directory entry

The MS-DOS File System (1)



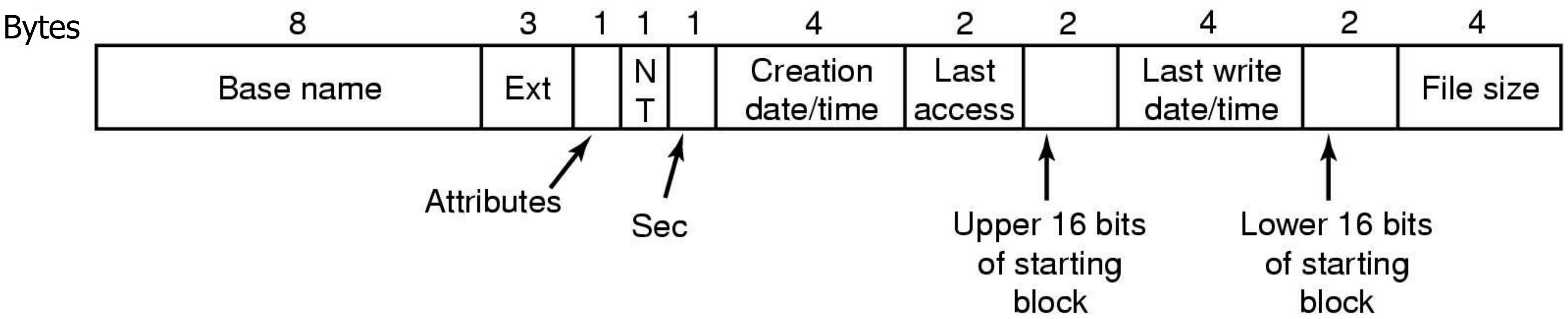
The MS-DOS directory entry

The MS-DOS File System (2)

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

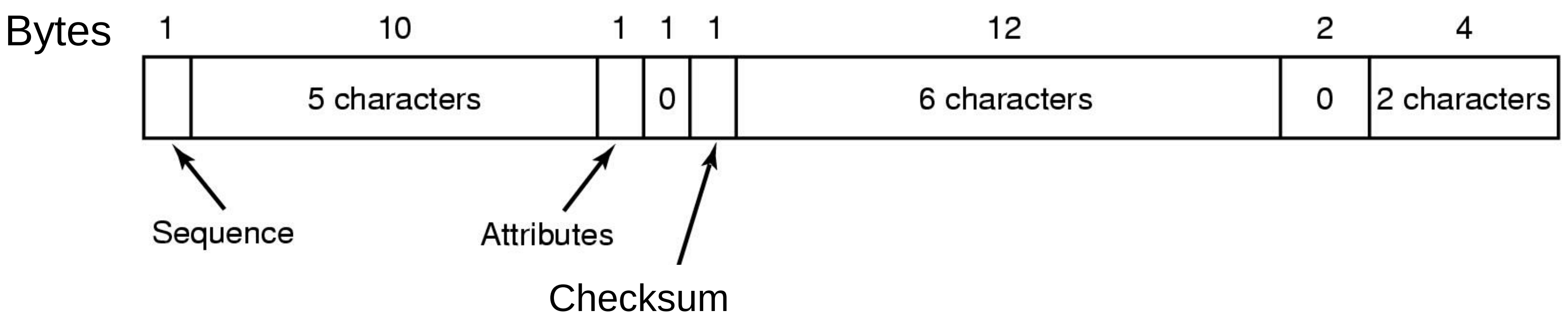
- Maximum partition for different block sizes
- The empty boxes represent forbidden combinations

The Windows File System (1)



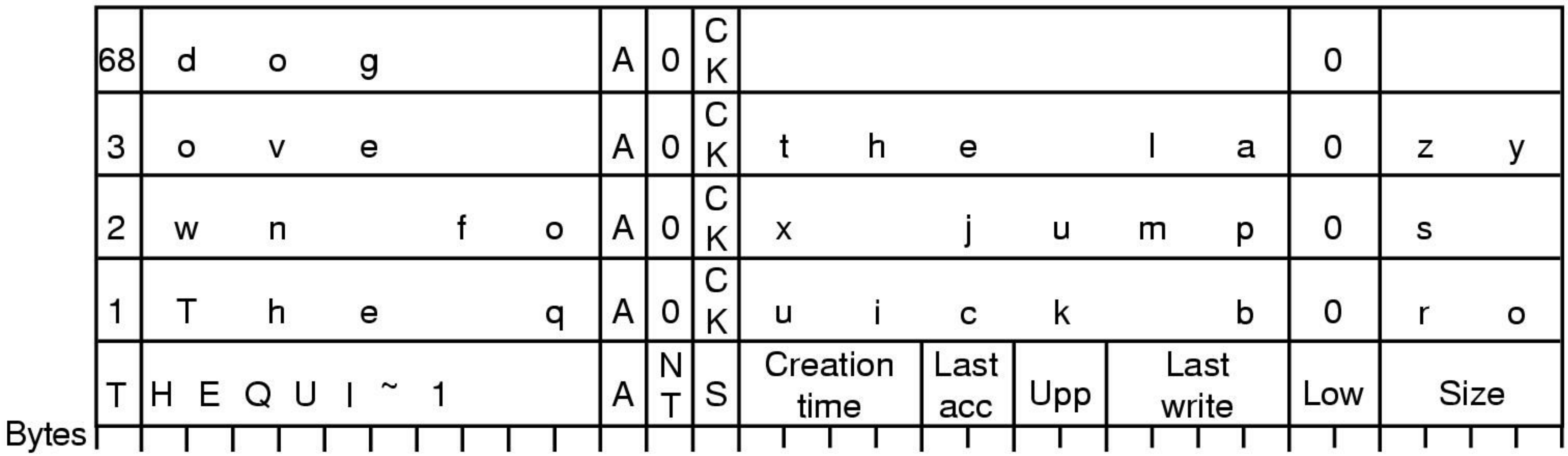
The extended MOS-DOS directory entry used in Windows

The Windows File System (2)



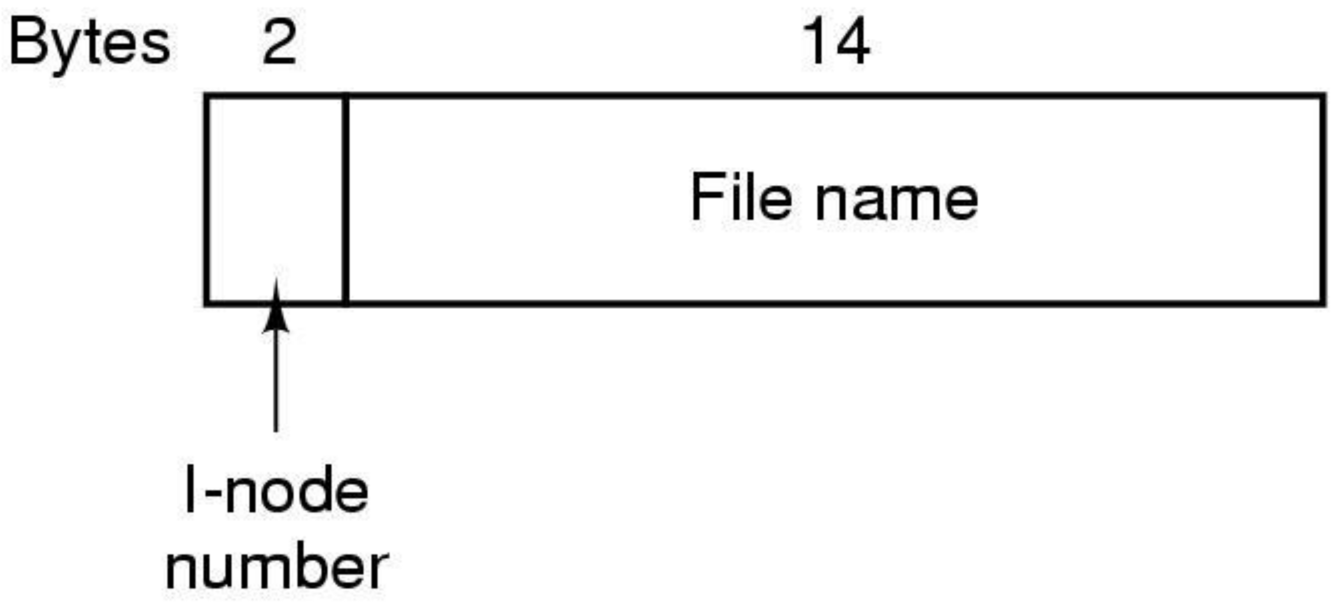
An entry for (part of) a long file name in Windows

The Windows File System (3)



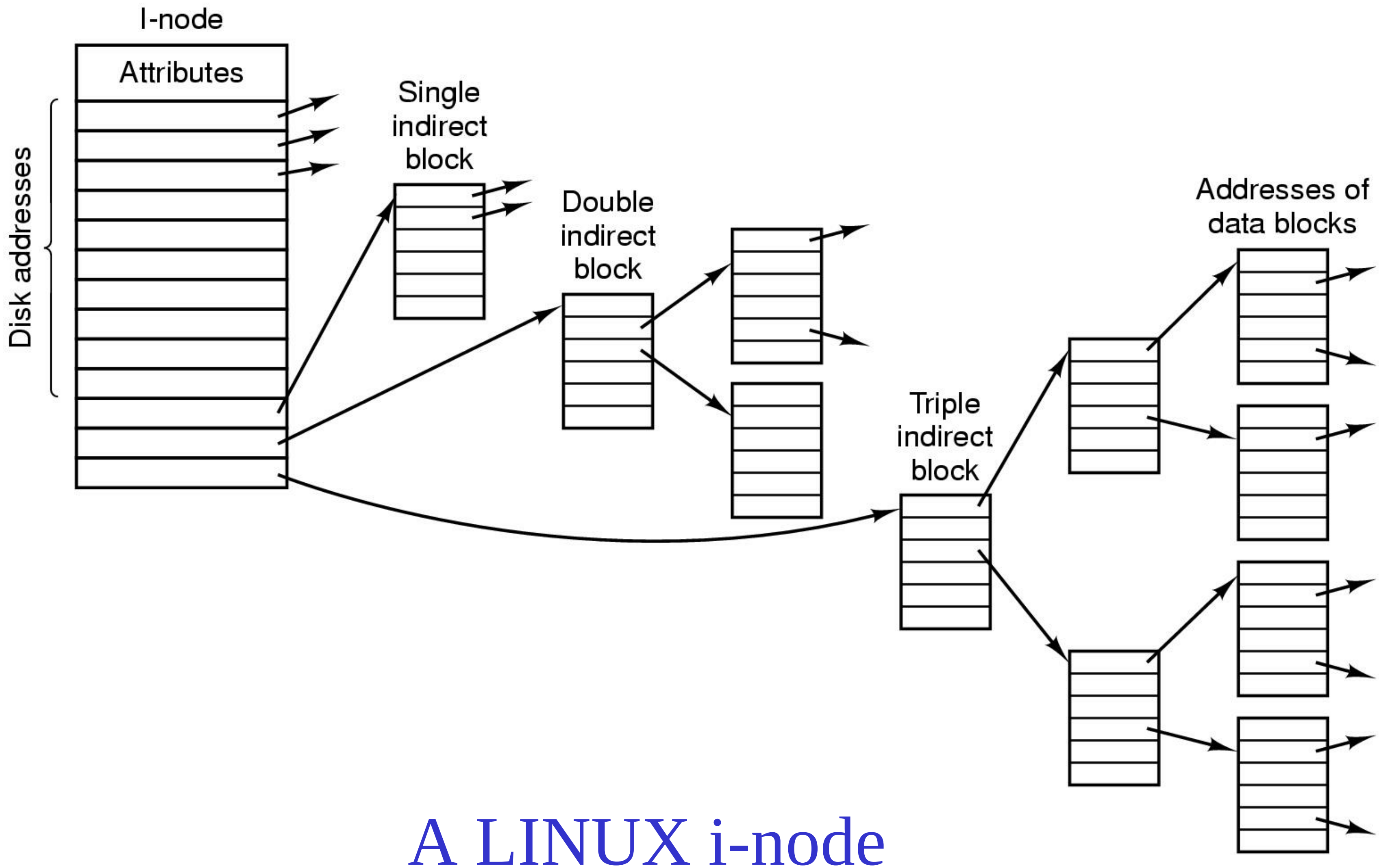
An example of how a long name is stored in Windows

The LINUX File System (1)



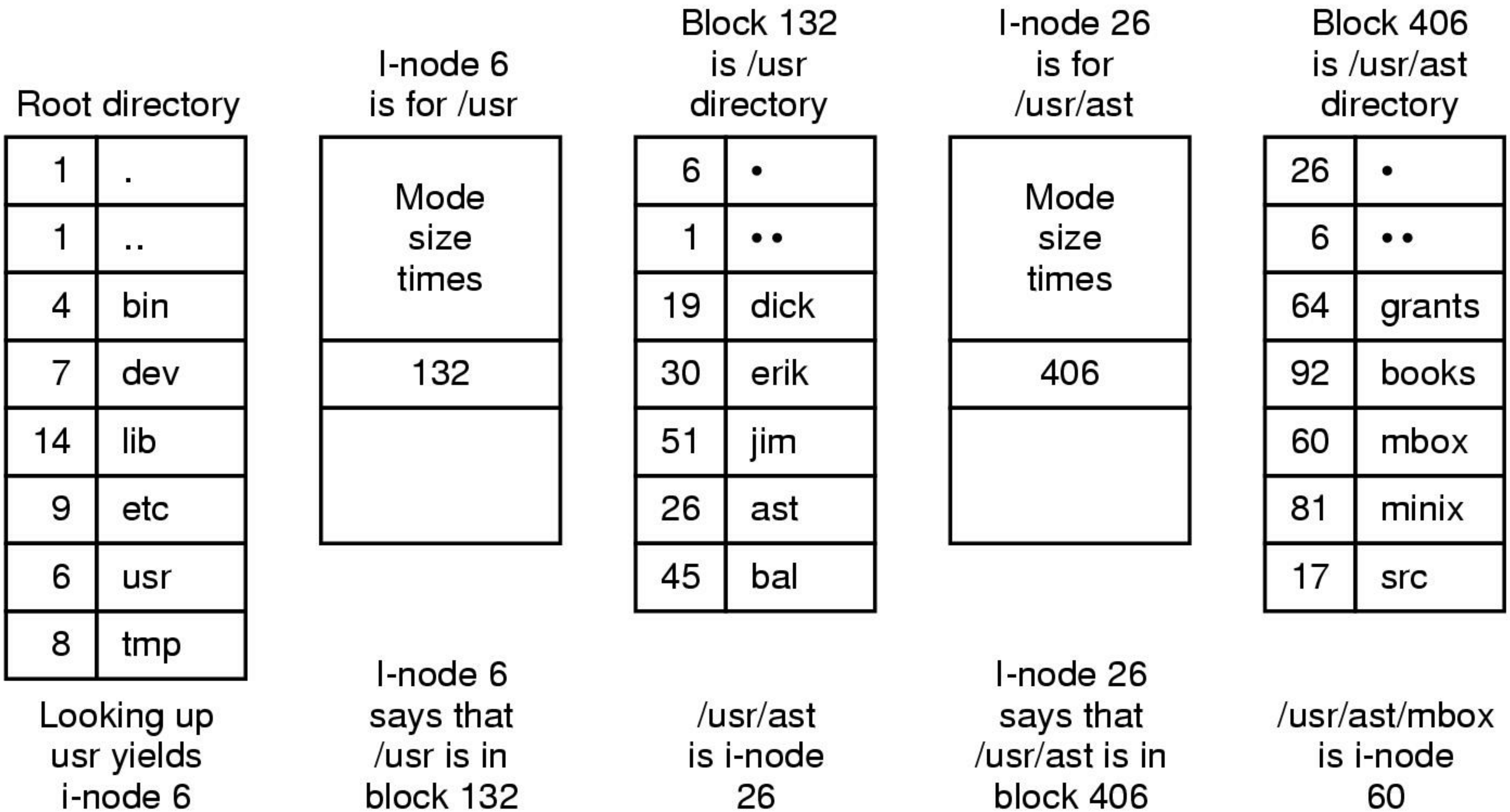
A LINUX directory entry

The LINUX File System (2)



A LINUX i-node

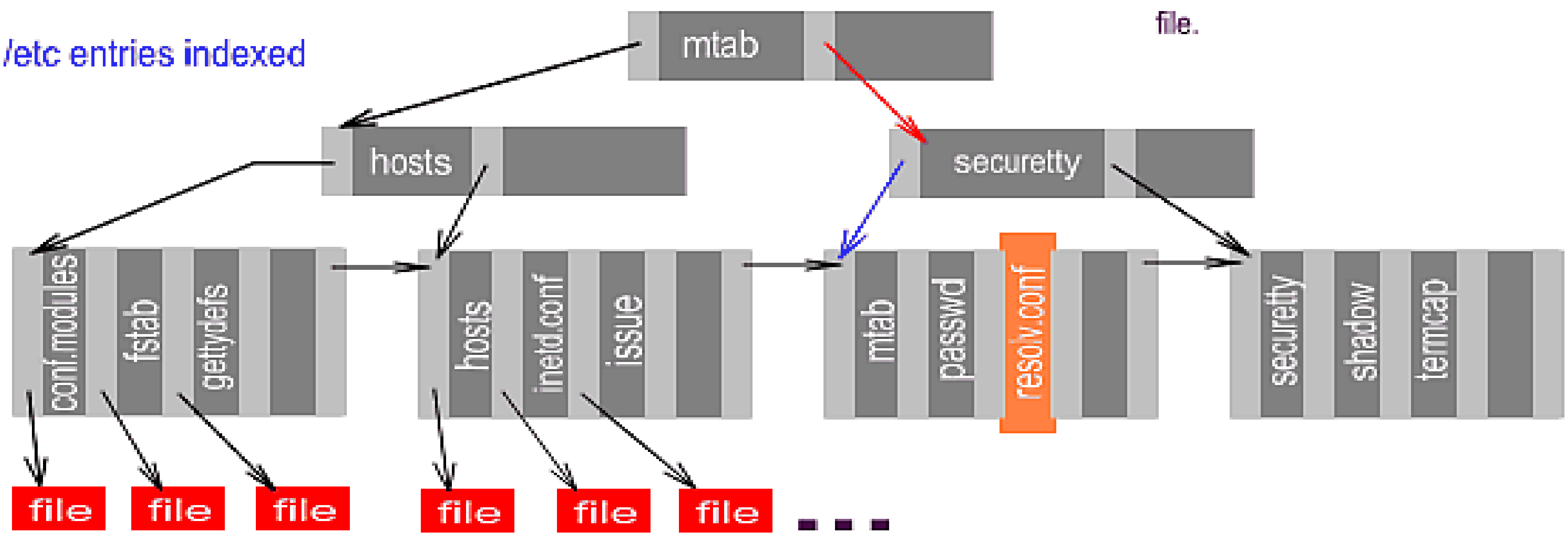
The LINUX File System (3)



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

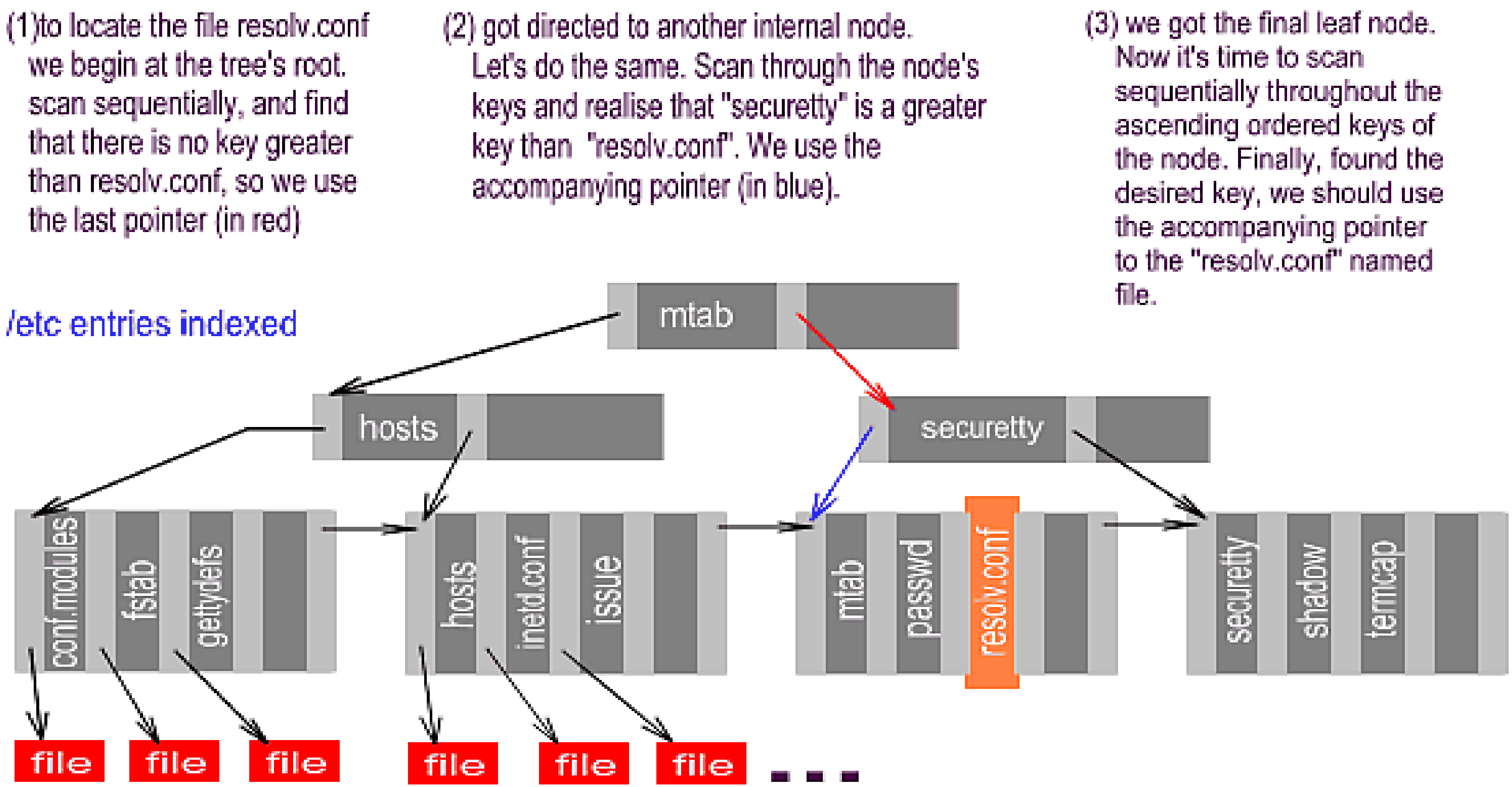
Btree – balanced tree file system

length of path from root to any leaf node should always be the same
nodes must contain a minimum number of pairs



The leaf node's keys are ordered within the tree improving scan times, since the scan is no longer sequential. Leaf nodes are chained using pointers to each other.

Btree – balanced tree file system



The keys are file names. The bottom row above the red boxes contains a key for every file in the directory: these are the leaf nodes. Above these are the internal nodes, keys that have been chosen by the system to make finding other keys faster