Agenda

File System Design

Example File Systems

- CP/M
- MS-DOS
- UNIX ext2

File system design

File system design questions

1. What is the blocksize?
2. What information will be stored in the superblock?
3. What attributes do files have?
4. Where are file attributes stored?
5. How are filenames stored? How long can they be?
6. How are file paths implemented? Where is the root directory stored?
7. How is free memory managed? Linked-lists? tables?
8. FAT or inodes? What is the max number of files? What is the largest file size that your system can support?

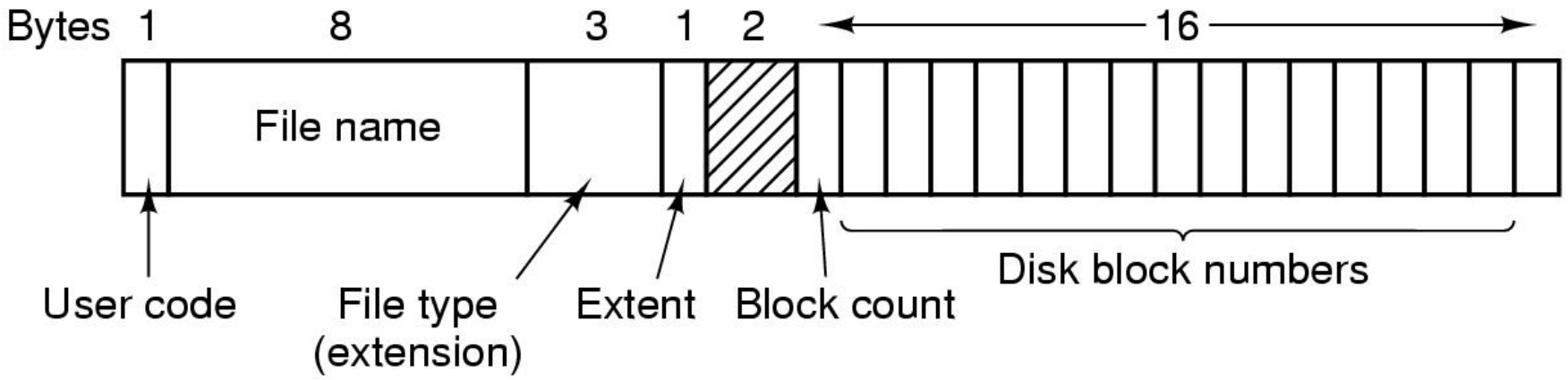
Example File System: CP/M

Dominated the microcomputer world in the early 80s

- Created by Gary Kildall, Digital Research
- Ran on 16KB of RAM. (Entire OS under 4KB)

Design:

- Only one directory. All files stored as entries in a single large table.
- Attributes are stored with the file
- Block size 1KB. Max of 16 blocks per file
- Fixed-length filenames



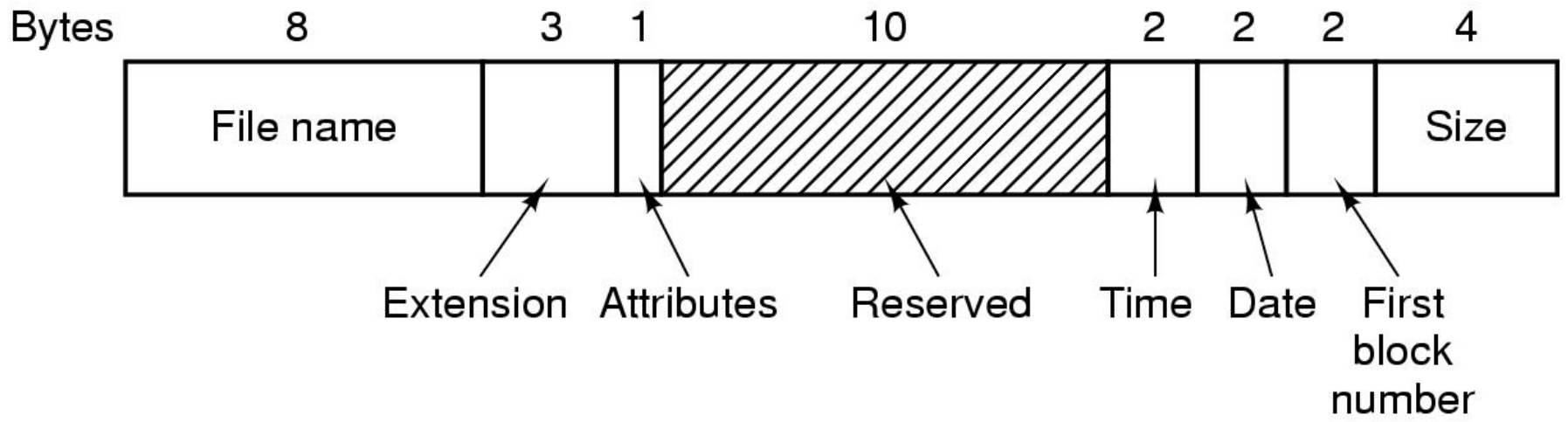
Example File System: MS-DOS

Microsoft Disk Operating System (MS-DOS)

Originally invented as a method for storing data on floppy disks, first version has single directory

Design

- Blocks managed with a file allocation table (FAT)
- Directories contain 32-bit entries
- Fixed-length filenames
- Directories store file attributes



MS-DOS: Limitations

The FAT table puts a limit on the max size of a partition.

- Exercise: Suppose the FAT table had 64 entries. How many blocks can be stored? If the blocksize is 1KB, how many bytes of data can we store?

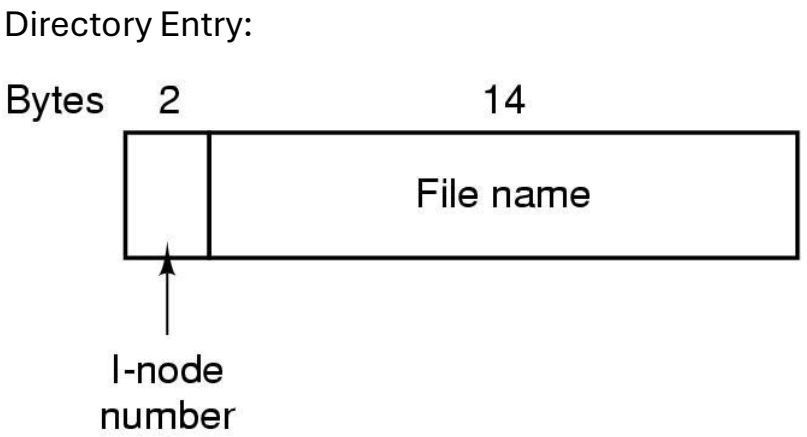
Larger/variable blocksizes help but because most files are small, using larger blocks wastes a lot of space (e.g. a 16 KB file might need to be stored in a 32-KB block)

NTFS fixes these limitations for modern computers

But many simple systems, such as cameras, still use MS-DOS

Example File System: UNIX V7

Pre-cursor to modern UNIX/Linux systems. Grew out of MULTICS



Design:

- Inode-based approach
- Fixed-length filenames
- File attributes stored with inodes
- Supports large files via indirect nodes
- Supports links
- Directories are files. Directories contain a table mapping names to inode numbers.

Example File Systems: ext2

Improves on UNIX V7. Core ideas are in wide-use today. ext3 adds features such as journaling.

Design:

- Inode-based design
- Variable-length directory names
- File attributes stored with inodes
- Supports large files via indirect nodes
- Supports links
- Directories are files. Directories contain a table mapping names to inode numbers.
- Free data managed with bitmaps

Incorporates several interesting ideas, such as

- Improves speed by trying to keep directories and their files close together within the file system
- Pre-allocates blocks to a file to avoid fragmentation as the file grows

Example File Systems: ext2

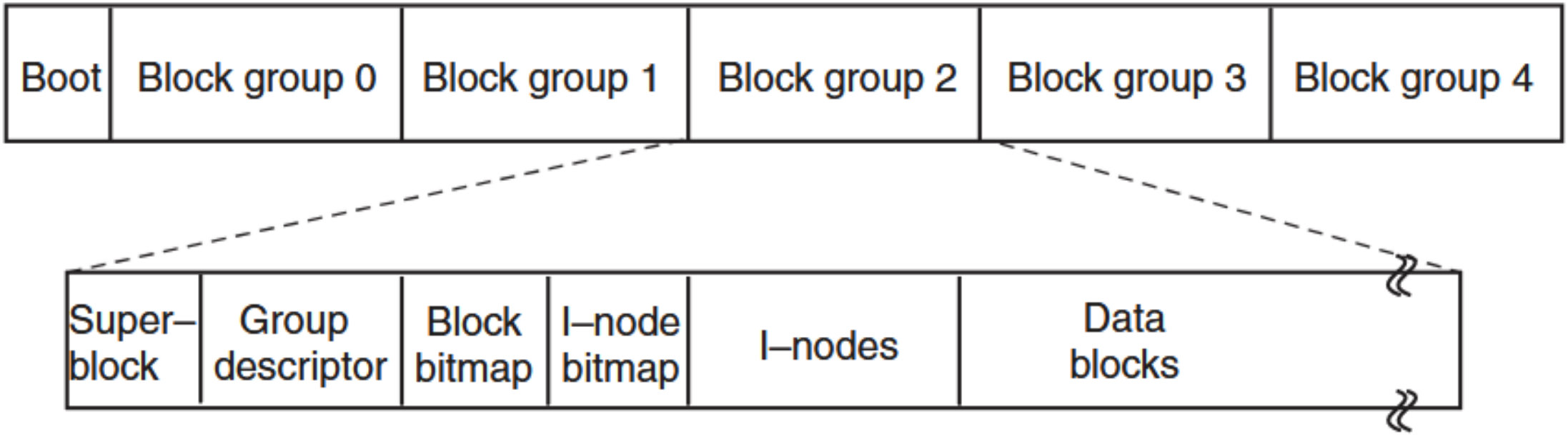


Figure 10-31. Disk layout of the Linux ext2 file system.

Example File Systems: ext2

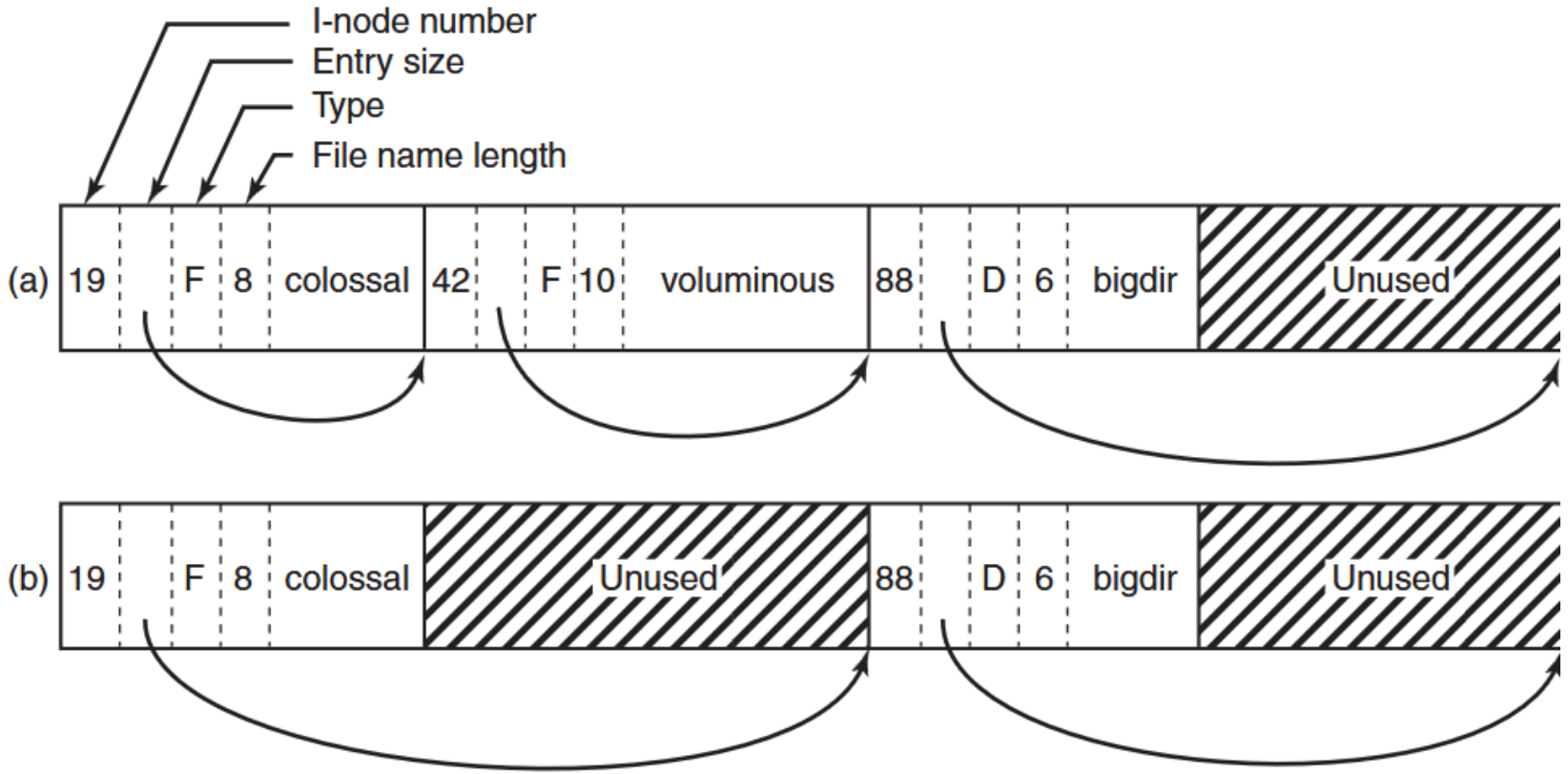


Figure 10-32. (a) A Linux directory with three files. (b) The same directory after the file *voluminous* has been removed.

Directory entries support variable length filenames.

Files are not sorted within a directory.

Caching can be integrated to make finding files with paths faster.

Example File Systems: ext2

File attributes are stored with the inode

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	60	Address of first 12 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)

Figure 10-33. Some fields in the i-node structure in Linux.

Example File System: ext2

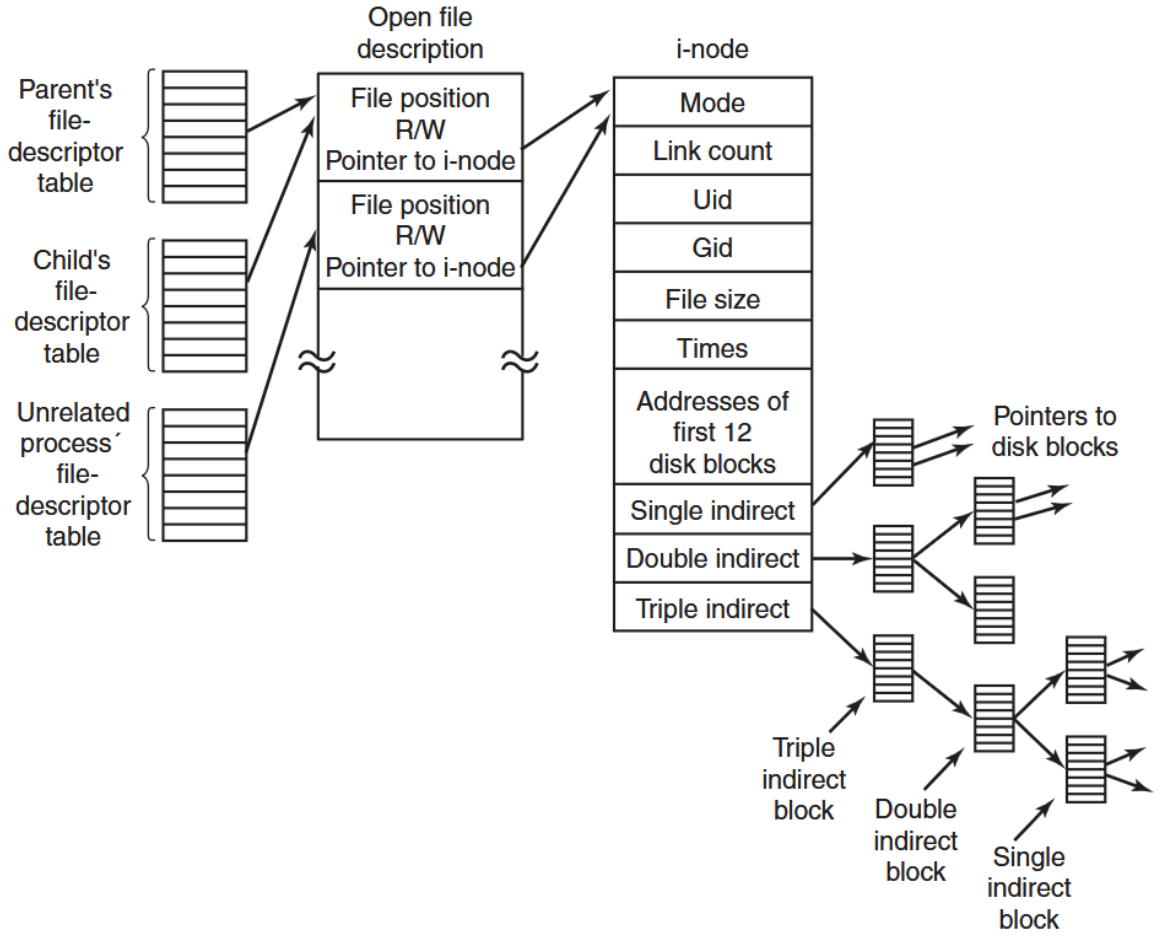


Figure 10-34. The relation between the file-descriptor table, the open-file-description-table, and the i-node table.

Open file descriptors for each process are kept in a table.

Each process needs its own file position marker to access the file's data

Large files are supported with indirect blocks.

Exercise: Tiny blocks

Suppose we have an ext2-based file system design

- Blocksize: 16 bytes
- 10 direct blocks
- 4 indirect blocks
- 1 double-indirect block
- 1 triple-indirect block
- Block ids are 4-byte integers
- sizeof(inode_ = 120 bytes

How many block ids can we store in an indirect block?

Exercise: Tiny blocks

Blocksize: 16 bytes
10 direct blocks
4 indirect blocks
1 double-indirect block
1 triple-indirect block
Block ids are 4-byte integers

What is the maximum filesize that we can store in this file system using direct blocks?

Exercise: Tiny blocks

Blocksize: 16 bytes
10 direct blocks
4 indirect blocks
1 double-indirect block
1 triple-indirect block
Block ids are 4-byte integers

What is the maximum filesize that we can store in this file system using indirect blocks?

Exercise: Tiny blocks

Blocksize: 16 bytes
10 direct blocks
4 indirect blocks
1 double-indirect block
1 triple-indirect block
Block ids are 4-byte integers

What is the maximum filesize that we can store in this file system using double-indirect blocks?

Exercise: Tiny blocks

Blocksize: 16 bytes
10 direct blocks
4 indirect blocks
1 double-indirect block
1 triple-indirect block
Block ids are 4-byte integers

What is the maximum filesize that we can store in this file system using triple-indirect blocks?

Exercise: Tiny blocks

Blocksize: 16 bytes
10 direct blocks
4 indirect blocks
1 double-indirect block
1 triple-indirect block
Block ids are 4-byte integers

Suppose we have a 300 byte file. Sketch how these blocks would be stored in the inode.