北京邮电大学软件学院

School Of software Engineering Of BUPT

Operating Systems

Lecture 10 File system implementation

Jinpengchen

Email: jpchen@bupt.edu.cn

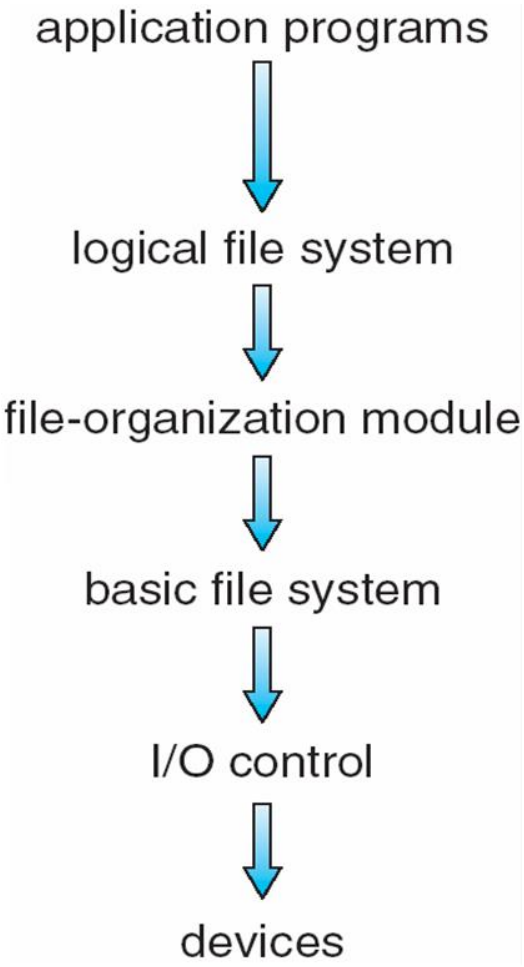
Catalog Description

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related Information
- FS resides on secondary storage(disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- File system organized into layers
- File control block – storage structure consisting of information about a file

Layered File System



File name

Logical block address

Physical block address

Issue commands to I/O to retrieve physical block

Hardware-specific instructions

Catalog Description

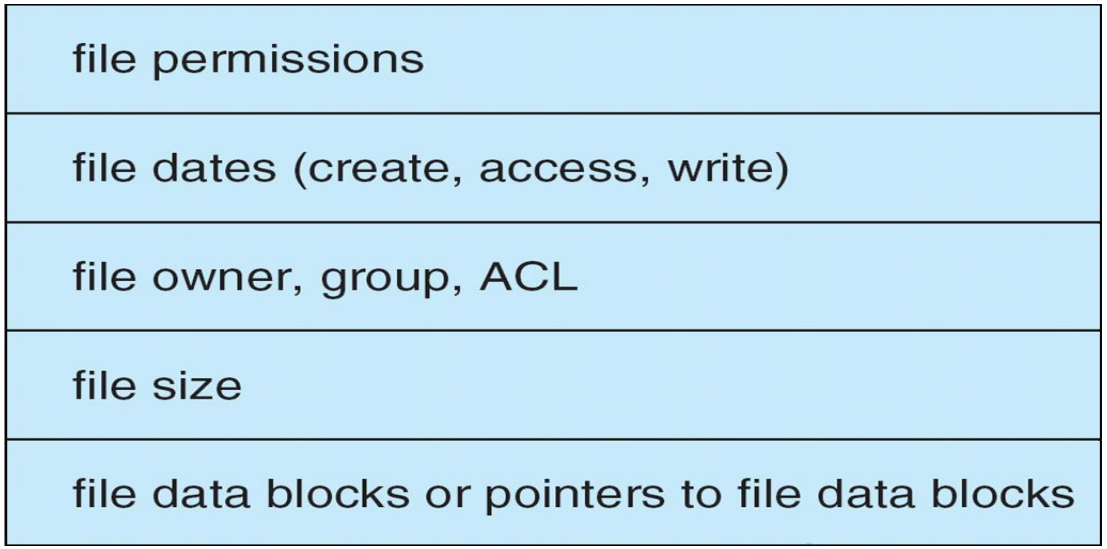
- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

FS Implementation

- Structures and operations used to implement file system operation, OS- & FS-dependent
  - On-disk structures
  - In-memory structures

FS Implementation

- On-disk structures
  - Boot control block: contains info needed by system to boot OS from that volume
    - ✓ To boot an OS from the partition (volume)
    - ✓ If empty, no OS is contained on the partition
  - Volume control block: contains volume details
  - Directory structure organizes the files
  - Per-file File Control Block (FCB) contains many details about the file

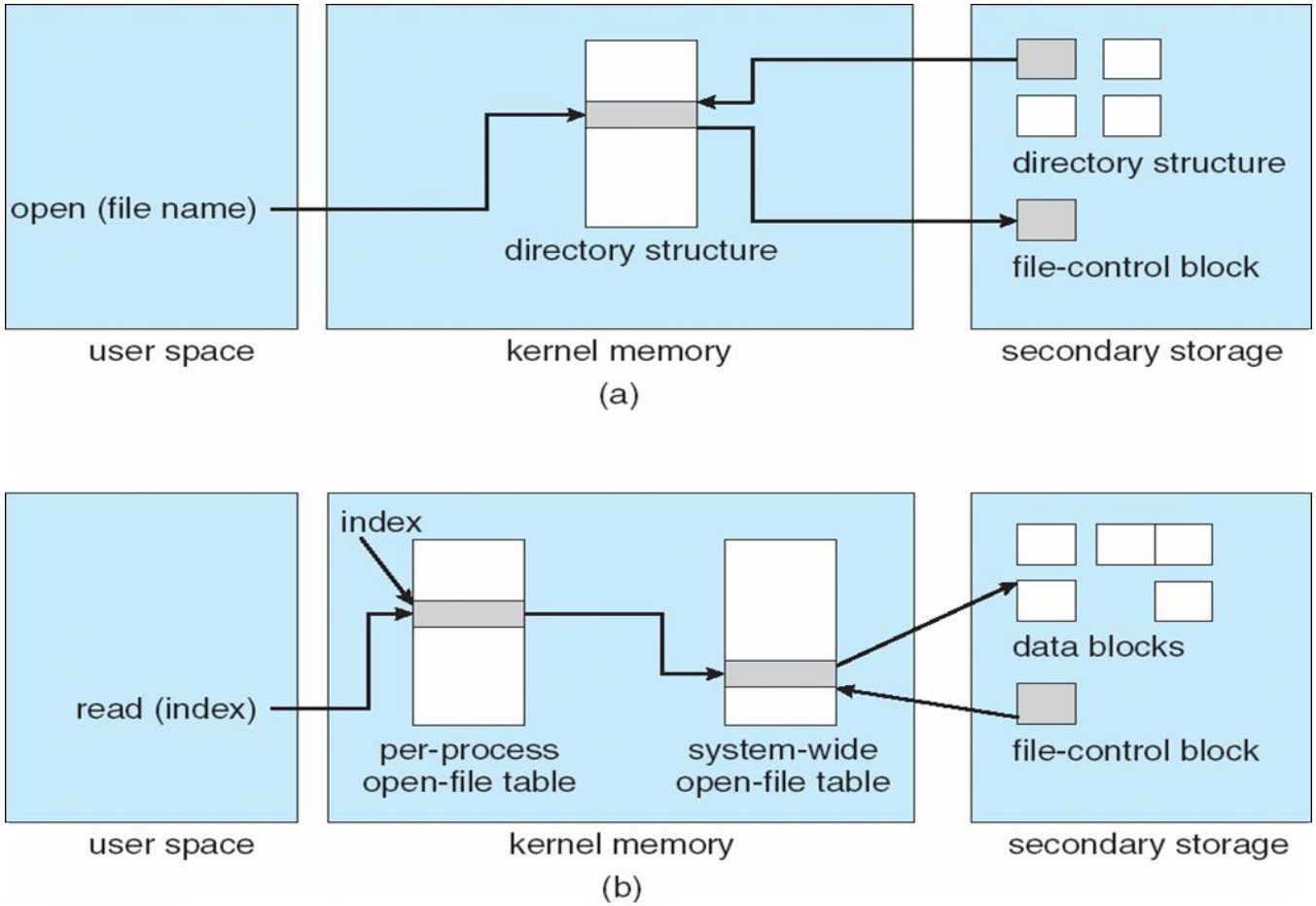


A typical file control block

FS Implementation

- In-memory structures: for both FS management and performance improvement via caching
  - Data are loaded at mount time and discarded at dismount
  - Structures include:
    - ✓ system-wide open-file table
    - ✓ per-process open-file table
    - ✓ …
- The following figure illustrates the necessary file system structures provided by the operating systems

In-Memory File System Structures



(a) refers to opening a file. (b) refers to reading a file.

Catalog Description

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

Directory Implementation

- Linear list of file names with pointer to the data blocks.
  - Simple to program
  - Time-consuming to execute
- Hash Table – linear list with hash data structure.
  - Decreases directory search time
  - Collisions – situations where two file names hash to the same location

Catalog Description

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

Allocation Methods (分配方法)

- An allocation method refers to how disk blocks are allocated for files so that disk space is utilized effectively & files can be accessed quickly
  - Contiguous allocation (连续分配)
  - Linked allocation (链接分配)
  - Indexed allocation (索引分配)
  - Combined (组合方式)

Contiguous Allocation (连续分配)

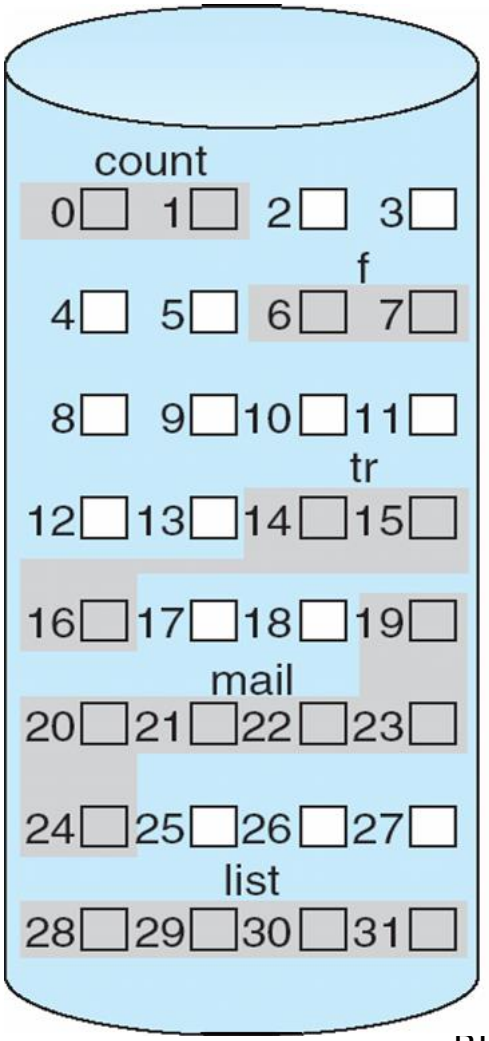
- Each file occupies a set of contiguous blocks on the disk
- Simple – each directory entry only needs
  - starting location (block #)
  - & length (number of blocks)
- Mapping from logical to physical

  LogicalAddress/512 = Q. . . . . R

  Block to be accessed = Q + starting address

  Displacement into block = R

Contiguous Allocation of Disk Space



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Contiguous Allocation (连续分配)

- Advantages:
  - Support both random & sequential access
    - ✓ Start block: b;
      Logical block number: i
      ⇒ physical block number: b + i
    - ✓ Fast access speed, because of short head movement
- Disadvantages:
  - External fragmentation
    - ✓ Wasteful of space (dynamic storage-allocation problem).
    - ✓ Files cannot grow,
      or File size must be known in advance.

Contiguous Allocation (连续分配)

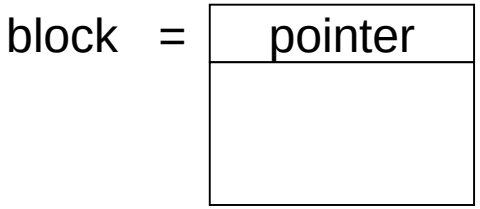
- Extent-Based Systems
  - Many newer file systems (I.e.Veritas File System) use a modified contiguous allocation scheme
  - Extent-based file systems allocate disk blocks in extents
  - An extent is a contiguous block of disks
    - ✓ Extents are allocated for file allocation
    - ✓ A file consists of one or more extents.

Linked Allocation（链接分配）

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Two types
  - Implicit（隐式链接）
  - Explicit（显式链接）

Linked Allocation (链接分配)

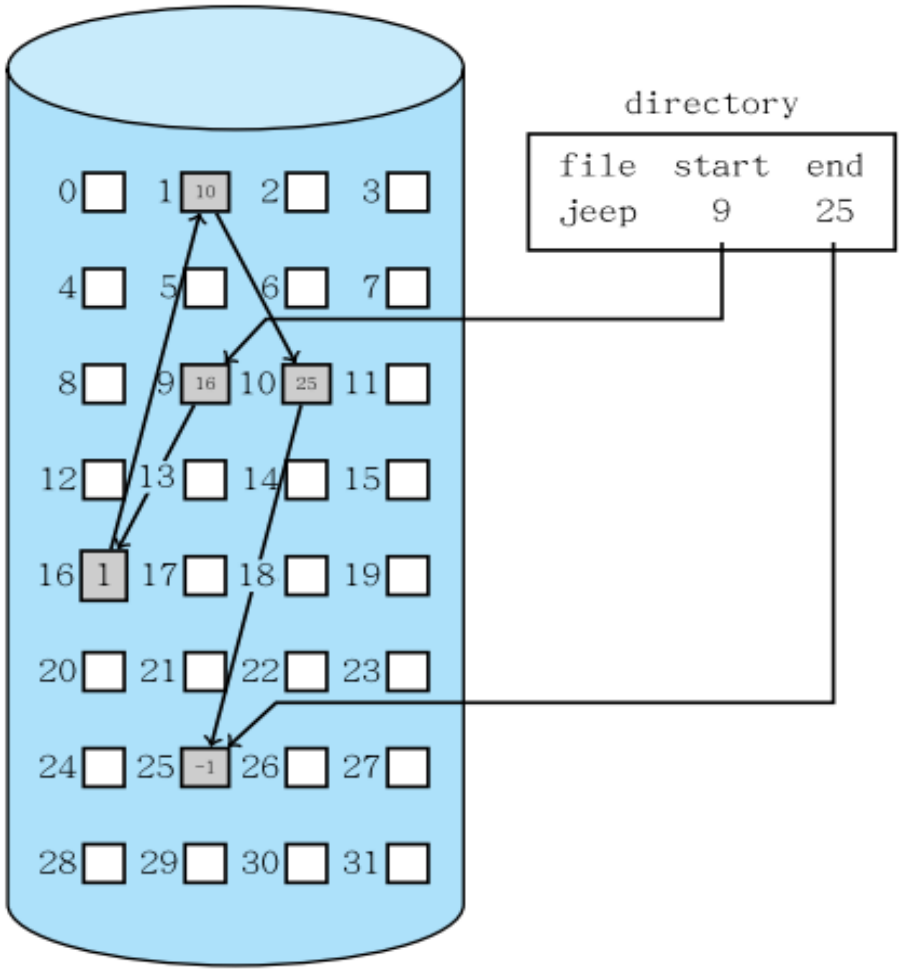
- Implicit (隐式链接)
  - Directory contains a pointer to the first block & last block of the file.
  - Each block contains a pointer to the next block.



  - Allocate as needed, link together
    - ✓ Simple -- only need starting address
    - ✓ Free-space management system -- no waste of space

Linked Allocation（链接分配）

- Implicit（隐式链接）



Linked Allocation (链接分配)

- Implicit (隐式链接)
  - Disadvantage:
    - ✓ No random access
    - ✓ Link pointers need disk space
      E.g.: 512 per block, 4 per pointer ⇒0.78%
      Solution: clusters
      ⇒ disk throughput ↑
      But internal fragmentation ↑

Linked Allocation (链接分配)

- Implicit (隐式链接)
  - Mapping:

    Suppose
    - ✓ block size = 512B,
    - ✓ block pointer size =1B, using the first byte of a block
    - ✓ Logical addr in the file to be accessed = A

    we have
    - ✓ Data size for each block = 512 − 1 = 511
    - ✓ A/511 = Q . . . . . R

    then
    - ✓ Block to be accessed is the Qth block in the linked chain of blocks representing the file.
    - ✓ Displacement into block = R + 1
  - How to reduce searching time?

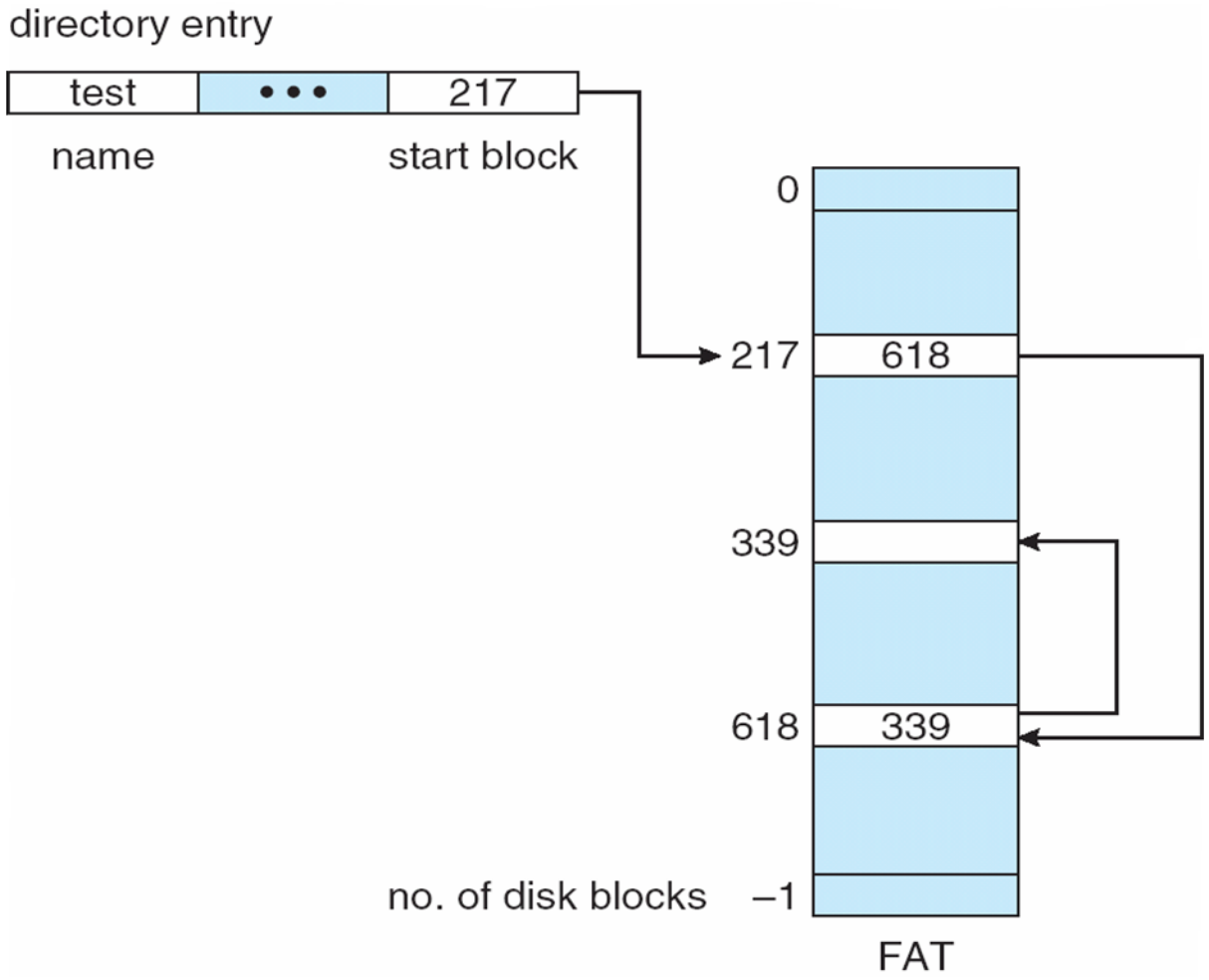
Linked Allocation (链接分配)

- Explicit linked allocation:
  - File Allocation Table, FAT

    Disk-space allocation is used by MS-DOS and OS/2
  - A section of disk at the beginning of each partition is set aside to contain the FAT
    - ✓ The FAT has one entry for each disk block and is indexed by block number
    - ✓ The entry contains
      - (1) the index of the next block in the file
      - (2) end-of-file, for the last block entry
      - (3) 0, for unused block
  - Directory entry contains the first block number

Linked Allocation（链接分配）

- File-Allocation Table



Linked Allocation (链接分配)

- Explicit linked allocation:
  - Now random-access time is improved, but is still not very efficient
  - May result in a significant disk head seeks.
    Solution: Cached FAT
  - How to compute FAT size?
    Suppose
    - ✓ Disk space = 80 GB, Block size = 4 KB

    Then

    $$\text{Total block number} = 80 \times 2^{30}/2^{12} = 5 \times 2^{22}$$
    $$4 \times 2^{22} = 2^{24} < 5 \times 2^{22} < 8 \times 2^{22} = 2^{25}$$

    - ✓ Length of each FAT entry? (25bits? 28bits? 32bits?)
    - ✓ Length of FAT? $(5 \times 2^{22} \times 4\text{B} = 80\text{MB} = 80\text{GB}/2^{10})$

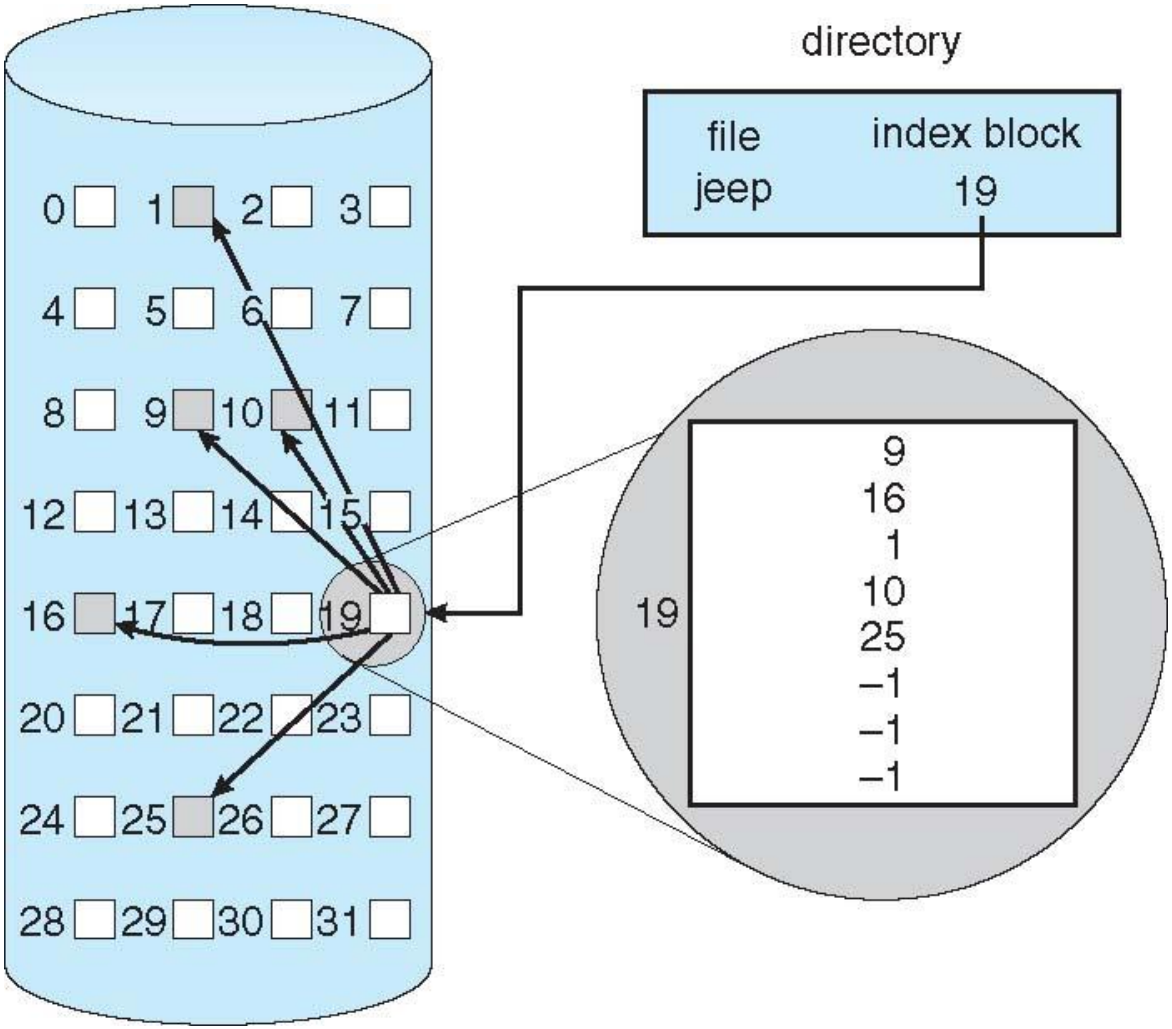
Indexed Allocation (索引分配)

- Indexed Allocation (索引分配):Brings all pointers together into one location -- the index block.
- Each file has its own index block
- Directory entry contains the index block address
- Each index block: An array of pointers (an index table)
  - ✓ Logical block number i
    = the i th pointer

Indexed Allocation (索引分配)

- Advantage:
  - Random access
  - Dynamic access without external fragmentation
- Disadvantage:
  - have overhead of index block.
  - File size limitation, since one index block can contains limited pointers

Example of Indexed Allocation



Indexed Allocation（索引分配）

- Mapping from logical to physical
- Suppose

  (1) Block size = 1KB

  (2) Index size = 4B

  Then for logical address LA, we have LA/1K = Q…R

  (3) Q = the index of the pointer

  (4) R = displacement into block

  We also have Max file size = $2^{10}/4 \times 1KB = 256KB$

Indexed Allocation (索引分配)

- How to support a file of unbounded length?
  - multi-level index scheme
  - combined scheme

Indexed Allocation (索引分配)

- multi-level index scheme
  - Mapping

Suppose

(1) Block size = 1KB

(2) Index or link pointer size = 4B

  - Example: Two-level index (maximum file size is?)

We have

LA/ (1K × 1K/4) = Q1 . . . R1

(1) Q1 = index into outer-index

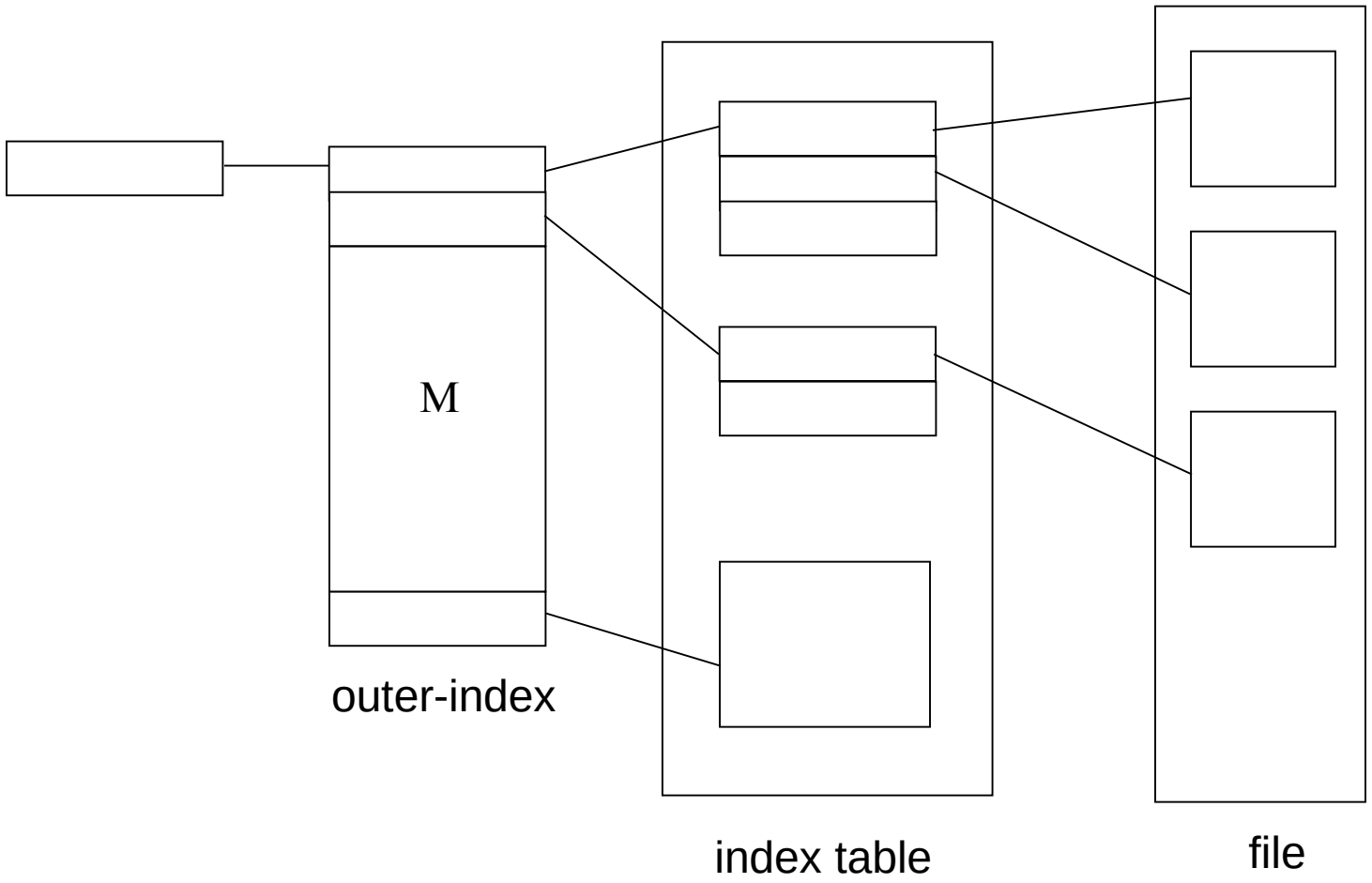
(2) R1 is used as follows:

R1 /1KB = Q2 . . . R2

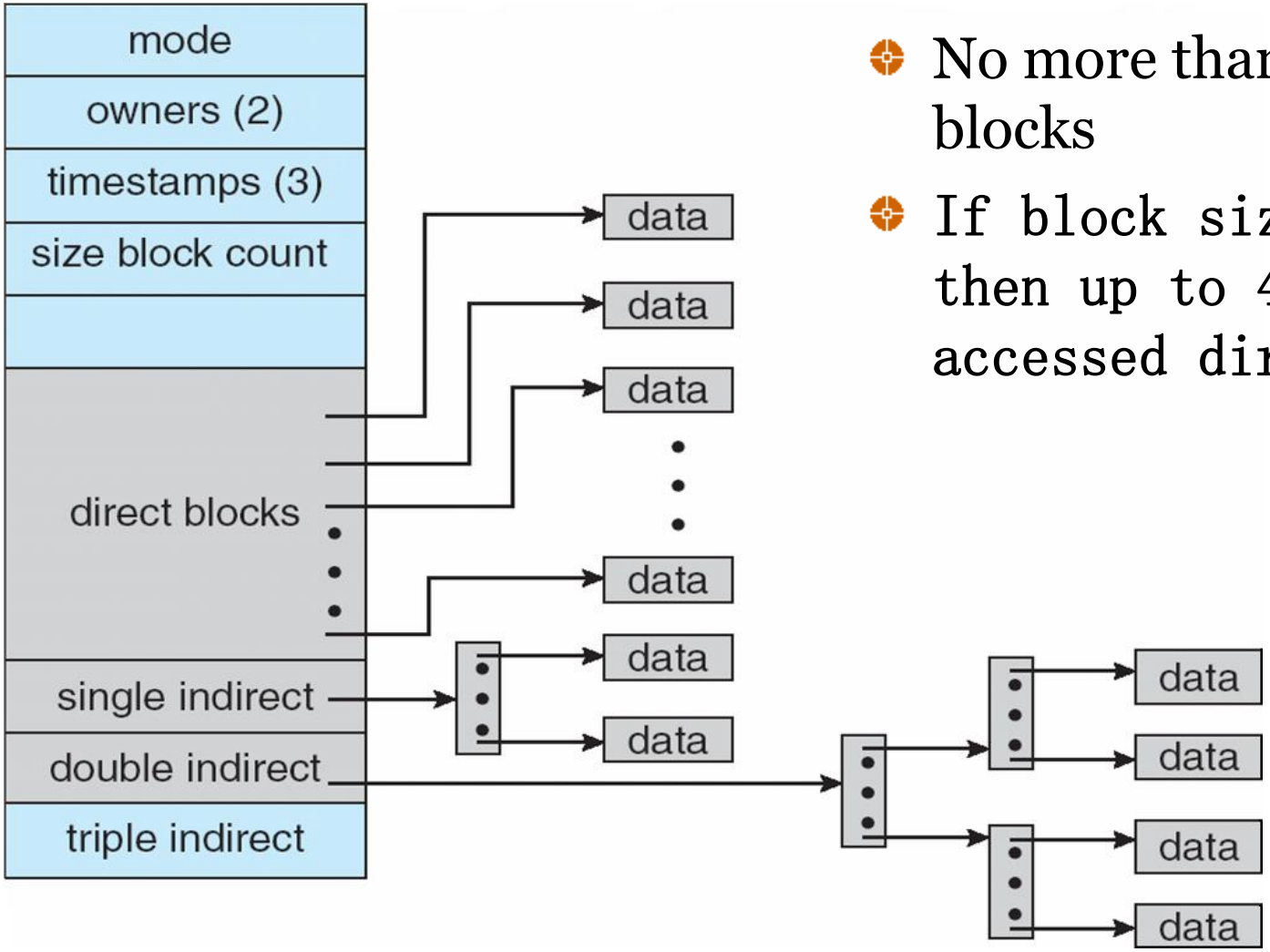
(3) Q2 = displacement into block of index table

(4) R2 = displacement into block of file

Indexed Allocation – Mapping



Combined Scheme (组合方式): UNIX (4K bytes per block)



- No more than 12 direct blocks
- If block size is 4KB, then up to 48KB can be accessed directly

Combined Scheme (组合方式): UNIX (4K bytes per block)

- if 4KB per block, and 4B per entry

Maximum file size = ?

Direct blocks = 12 × 4KB = 48KB

Number of entries per block = 4KB/4B = 1K

Single indirect = 1K × 4KB = 4MB

Double indirect = 1K × 4MB = 4GB

Triple indirect = 1K × 4GB = 4TB

Catalog Description

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

Free-Space Management

- Disk Space: limited
- Free space management: To keep track of free disk space
- Algorithms
  - Bit vector
  - Linked list
  - Counting

Free-Space Management

- Bit vector
  - Free-space list is implemented as a bit map or bit vector
    - ✓ 1 bit for each block

      1=free;

      0=allocated
    - ✓ Example:

      a disk where blocks 2,3,4,5,8,9,10,11,12,13,17,18,25,26,27 are free and the rest blocks are allocated. The bitmap would be

      0011 1100 1111 1100 0110 0000 0111 0000 0...
    - ✓ Bit map length.

      For n blocks, if the base unit is word, and the size of word is 16 bits, then bit map length = (n + 15)/16

      U16 bitMap[bitMaptLength];

Free-Space Management

- Bit vector
  - How to find the first free block or n consecutive free blocks on the disk?
    - ✓ To find the first free block:

      Suppose: base unit = word (16 bits) or other

      (1) find the first non-0 word

      (2) find the first 1 bit in the first non-0 word
    - ✓ If first K words is 0, & (K + 1)th word > 0, the first (K + 1)th word's first 1 bit has offset L, then

      first free block number N = K × 16 + L

Free-Space Management

- Bit vector
  - Simple
  - Must be kept on disk
    Bit map requires extra space,
    Example:
    block size = 2^{12} bytes
    disk size = 2^{30} bytes (1 gigabyte)
    n = $2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
  - Easy to get contiguous files

Free-Space Management

- Bit vector
  - Efficient to get the first free block or n consecutive free blocks, if we can always store the vector in memory.
    - ✓ But copy in memory and disk may differ.
      E.g. bit[i] = 1 in memory & bit[i] = 0 on disk
    - ✓ Solution:
      Set bit[i] = 1 in memory.
      Allocate block[i]
      Set bit[i] = 1 in disk

Free-Space Management

- Linked Free Space List on Disk
  - Link together all the free disk blocks
    - ✓ First free block
    - ✓ Next pointer
  - Cannot get contiguous space easily
  - No waste of space

Free-Space Management

- Counting
  - Assume:

    Several contiguous blocks may be allocated or freed simultaneously
  - Each entry = first free block number & a counter (number of free blocks)
  - Shorter than linked list at most time, generally counter > 1

Catalog Description

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency （空间） and Performance（时间）
- Recovery

Efficiency（空间）

- Efficiency in usage of disk space is dependent on:
  - Disk allocation and directory algorithms
  - Various approaches
    - ✓ Variable cluster size
    - ✓ Types of data kept in file's directory entry
    - ✓ Large pointers provides larger file length, but cost more disk space

Performance (时间)

- Performance: other ways
  - disk cache - on disk controllers, large enough to store entire tracks at a time.
  - buffer cache - separate section of main memory for frequently used blocks
  - free-behind and read-ahead - techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

Catalog Description

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency （空间） and Performance（时间）
- Recovery

Recovery

- Consistency checking (一致性检查)
  - compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - UNIX: fsck
  - MS-DOS: chkdsk
- Backup & restore
  - Use system programs to back up data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
  - Recover lost file or disk by restoring data from backup
  - A typical backup schedule may be:

    Day1: full backup;

    Day2: incremental backup;

    ...

    DayN: incremental backup. Then go back to Day1.

End of Chapter 10