Operating System: Chap11 File System Implementation

National Tsing Hua University

2022, Fall Semester

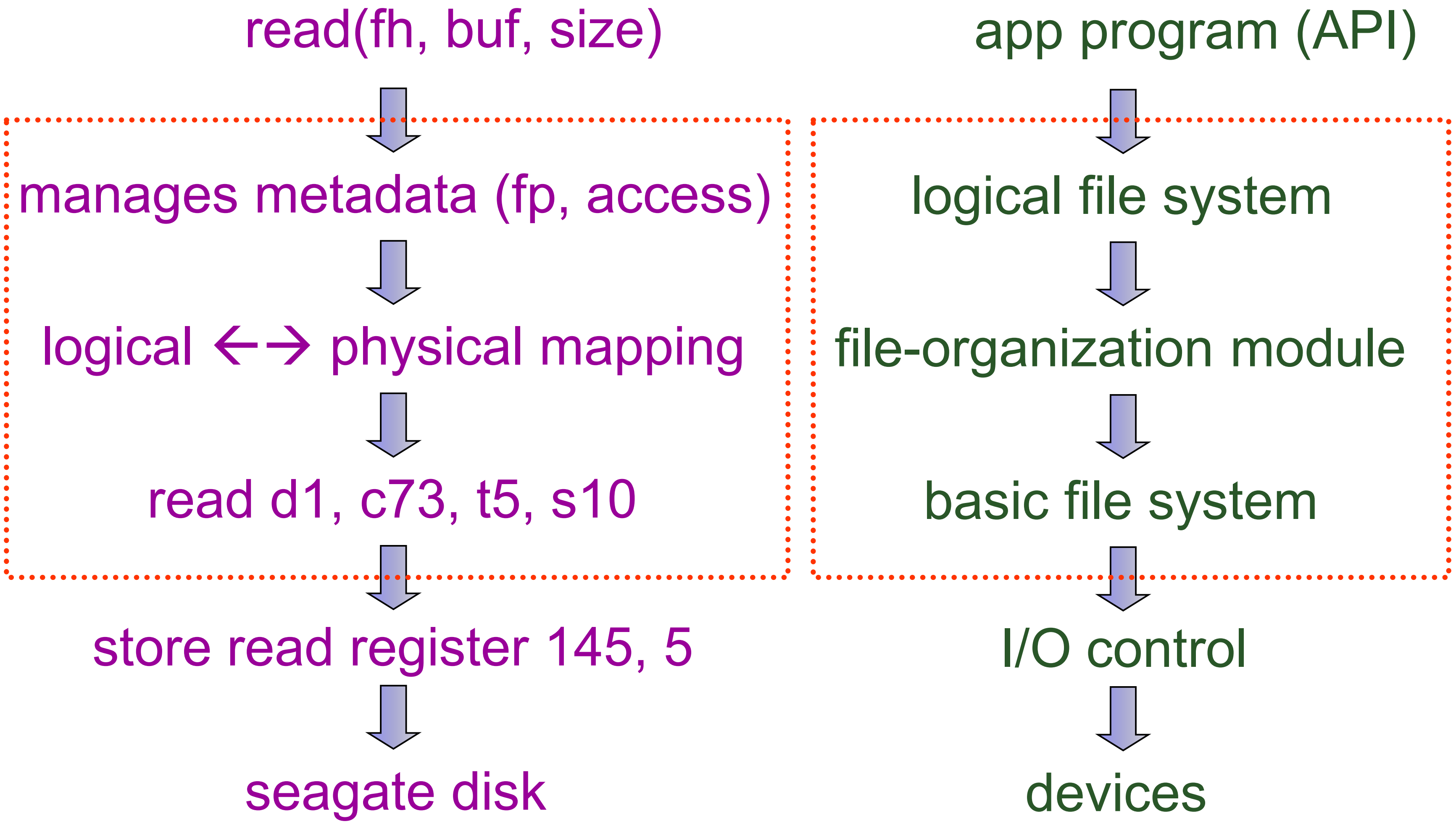
Overview

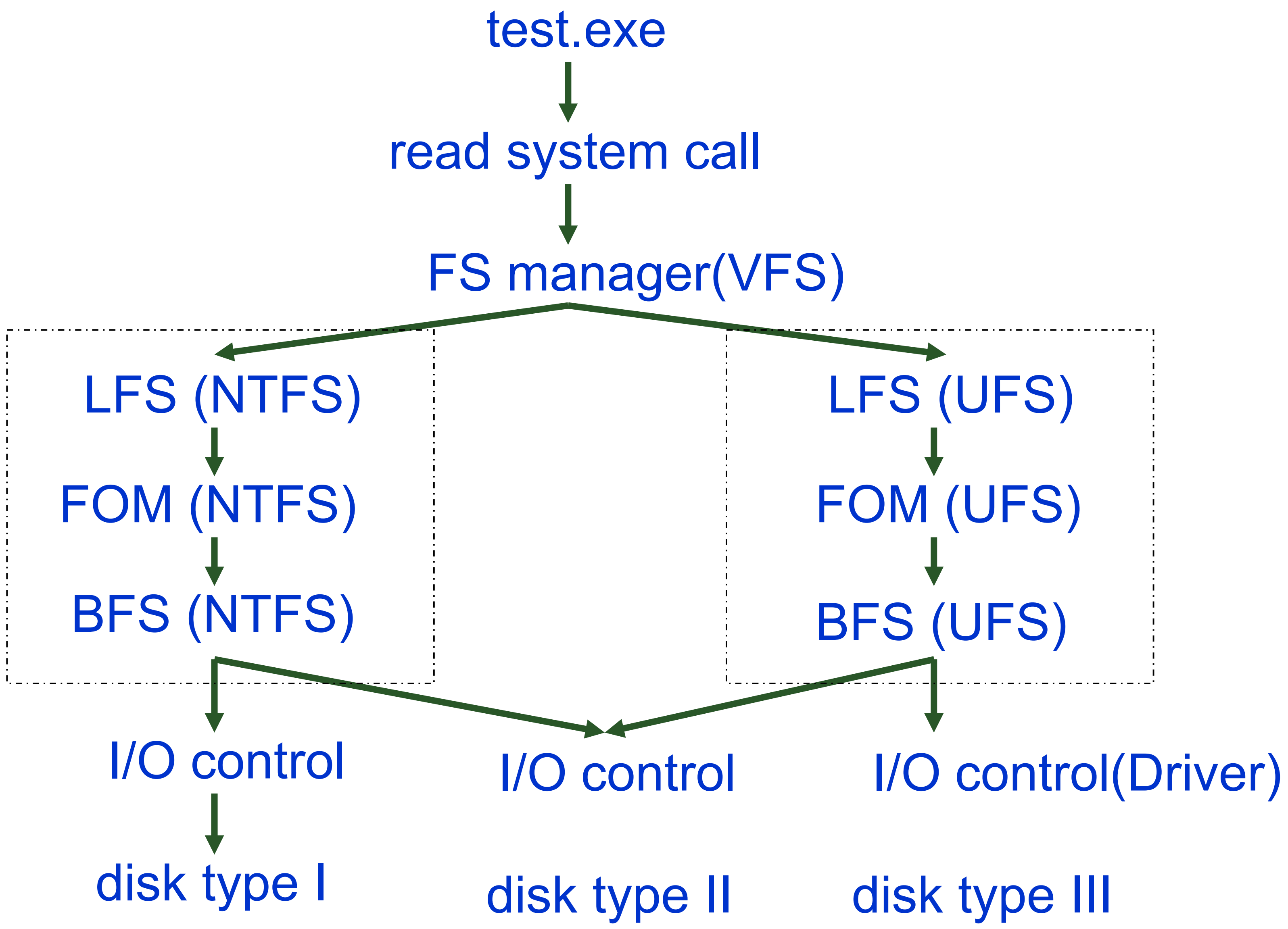
- File-System Structure
- File System Implementation
- Disk Allocation Methods
- Free-Space Management

File-System Structure

- I/O transfers between memory and disk are performed in units of **blocks**
  - one block is one or more **sectors**
  - one sector is usually 512 bytes
- One OS can support more than 1 FS types
  - NTFS, FAT32
- Two design problems in FS
  - interface to user programs
  - interface to physical storage (disk)

Layered File System




test.exe
read system call
FS manager(VFS)
LFS (NTFS)
FOM (NTFS)
BFS (NTFS)
LFS (UFS)
FOM (UFS)
BFS (UFS)
I/O control
I/O control
I/O control(Driver)
disk type I
disk type II
disk type III

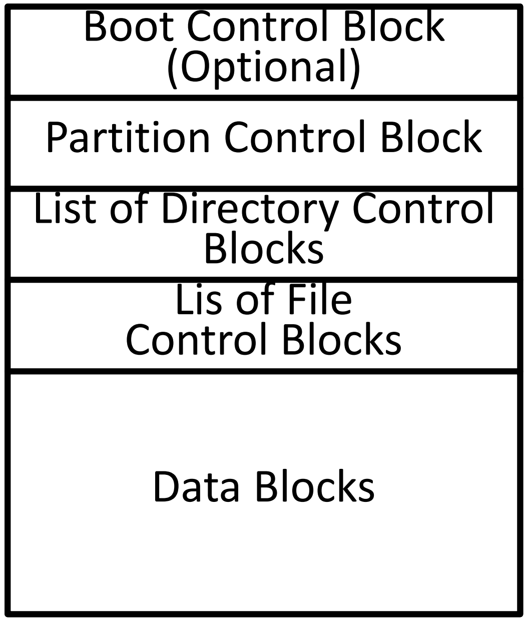
File-System Implementation

On-Disk Structure

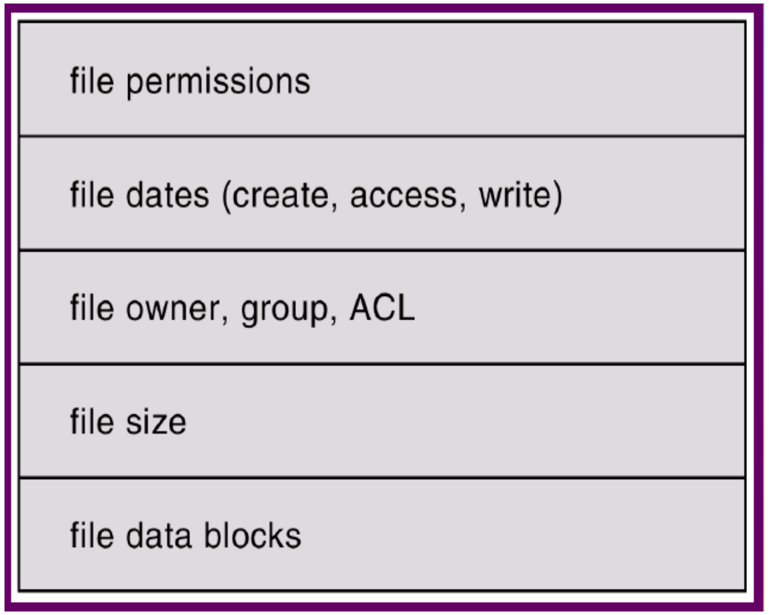
- **Boot control block** (per partition): information needed to boot an OS from that partition
  - typical the first block of the partition (empty means no OS)
  - UFS (Unix File Sys.): **boot block**, NTFS: partition boot sector
- **Partition control block** (per partition): partition details
  - details: # of blocks, block size, free-block-list, free FCB pointers, etc
  - UFS: **superblock**, NTFS: Master File Table
- **File control block** (per file): details regarding a file
  - details: permissions, size, location of data blocks
  - UFS: **inode**, NTFS: stored in MFT (relational database)
- **Directory structure** (per file system): organize files

On-Disk Structure

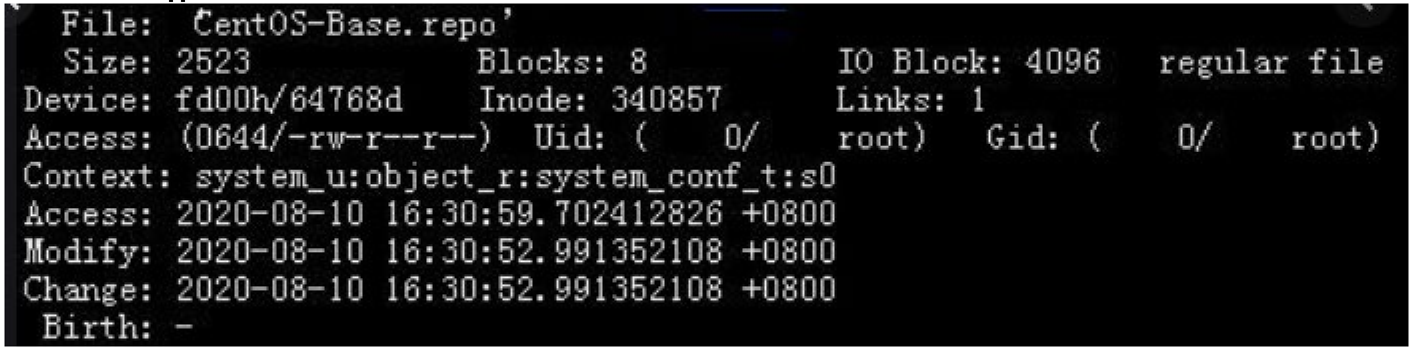
Partition


Boot Control Block (Optional)
Partition Control Block
List of Directory Control Blocks
Lis of File Control Blocks
Data Blocks

File Control Block (FCB)



stat <filename || dirname>: show the FCB info in Linux

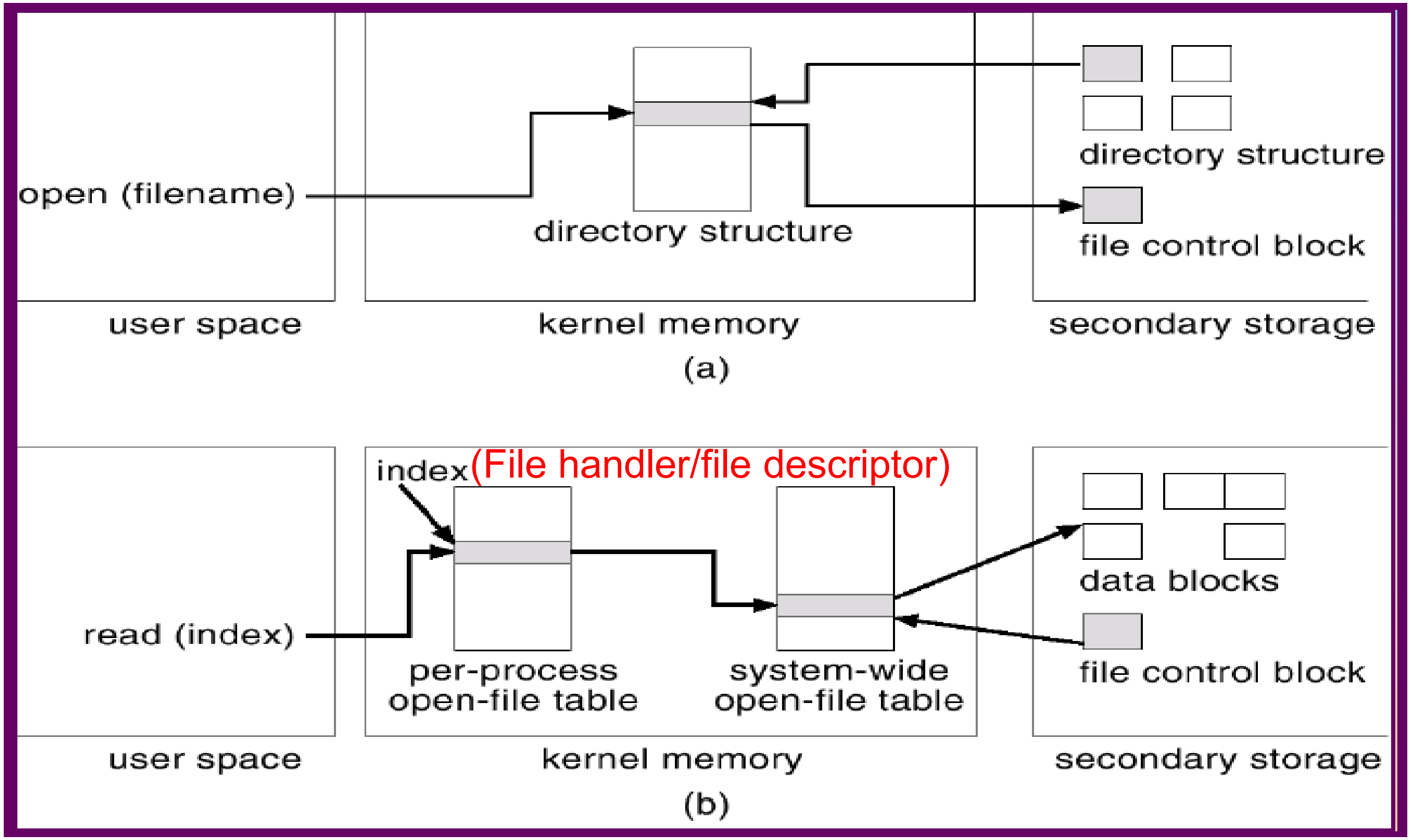


```
  File: `CentOS-Base.repo'
  Size: 2523            Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 340857      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Context: system_u:object_r:system_conf_t:s0
Access: 2020-08-10 16:30:59.702412826 +0800
Modify: 2020-08-10 16:30:52.991352108 +0800
Change: 2020-08-10 16:30:52.991352108 +0800
 Birth: -
```

In-Memory Structure

- in-memory partition table: information about each **mounted partition**
- in-memory directory structure: information of **recently accessed directories**
- system-wide open-file table: contain a copy of each **opened file’s FCB**
- per-process open-file table: **pointer (file handler/descriptor)** to the corresponding entry in the above table

File-Open & File-Read

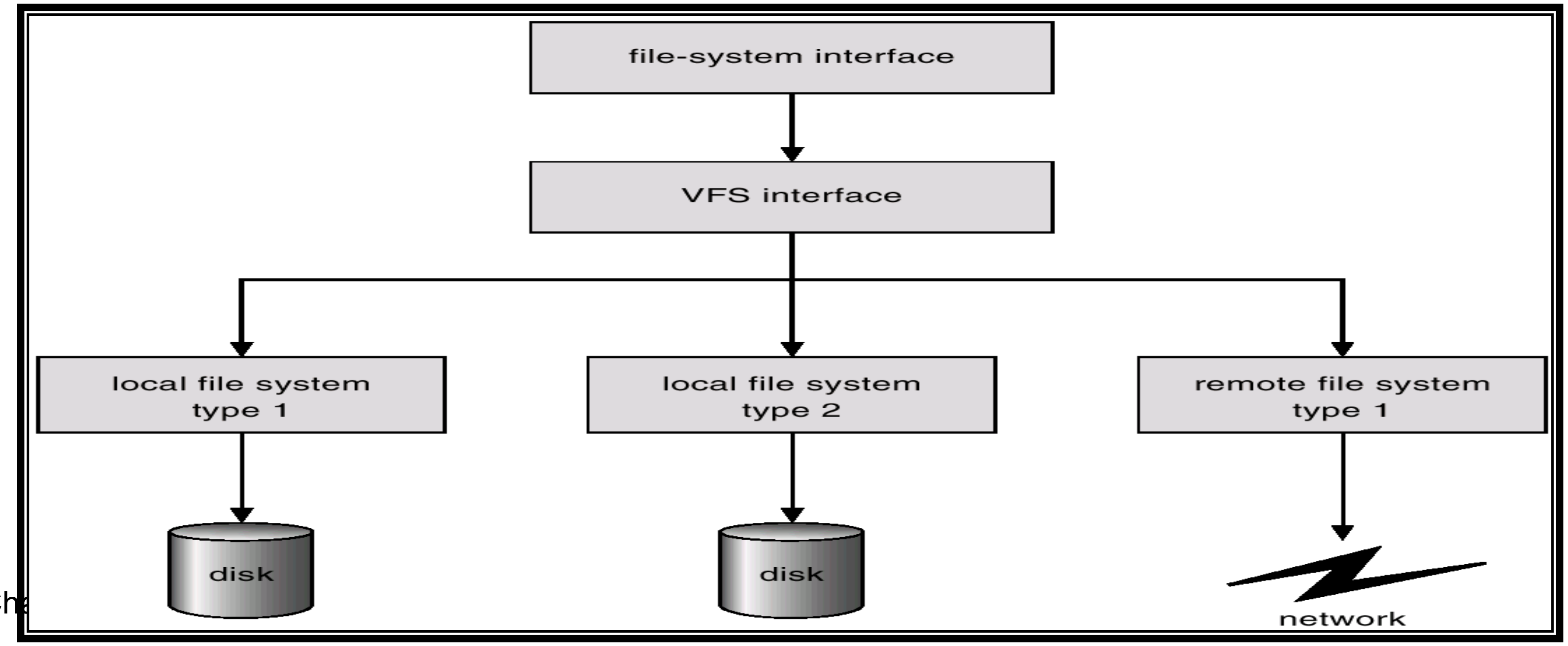


File Creation Procedure

1. OS allocates a new **FCB**
2. Update **directory structure**
    1. OS reads in the corresponding directory structure into memory
    2. Updates the dir structure with the new file name and the FCB
    3. (After file being closed), OS writes back the directory structure back to disk
3. The file appears in user's dir command

Virtual File System

- VFS provides an object-oriented way of implementing file systems
- VFS allows the same **system call interface** to be used for different types of FS
- VFS calls the appropriate FS routines based on the partition info



Virtual File System

- Four main object types defined by Linux VFS:
  - inode ➔ an individual file
  - file object ➔ an open file
  - superblock object ➔ an entire file system
  - dentry object ➔ an individual directory entry
- VFS defines a set of operations that must be implemented (e.g. for file object)
  - int open(…) ➔ open a file
  - ssize_t read() ➔ read from a file

Directory Implementation

- Linear lists
  - list of file names with pointers to data blocks
  - easy to program but poor performance
    - insertion, deletion, searching
- Hash table – linear list w/ hash data structure
  - constant time for searching
  - linked list for collisions on a hash entry
  - hash table usually has fixed # of entries

Review Slides (I)

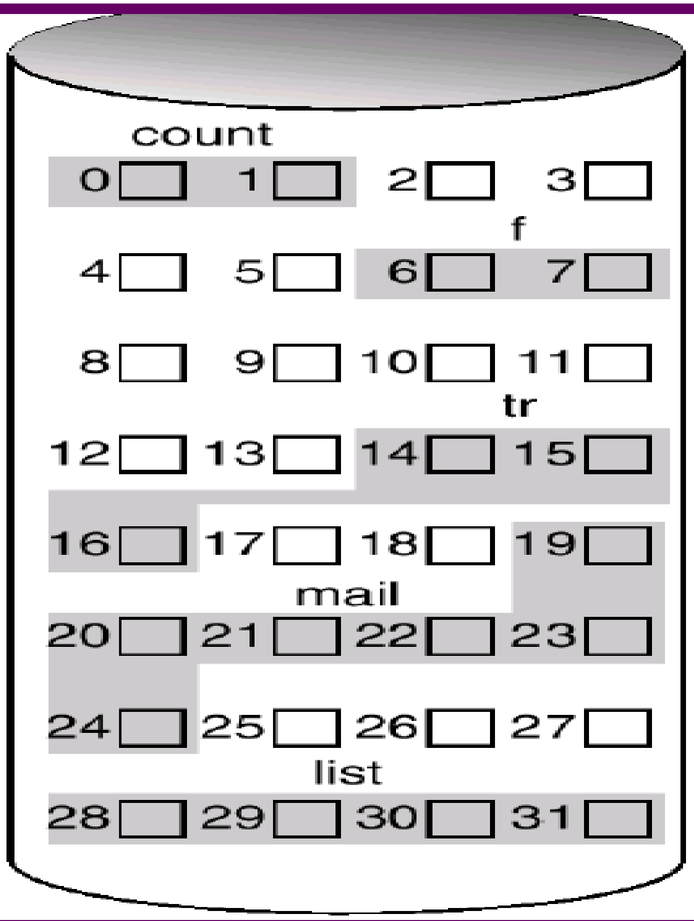
- Transfer unit between memory and disk?
- App ➔ LFS ➔ FOM ➔ BFS ➔I/O Control ➔Devices
- On-disk structure
  - Boot control block, Partition control block
  - File control block, Directory structure
- In-memory
  - Partition table, Directory structure
  - System-wide open-file table
  - Per-process open-file table
- Steps to open file, read/write file and create file?
- Purpose of VFS?

Allocation Methods

Outline

- An allocation method refers to how **disk blocks** are allocated for **files**
- Allocation strategy:
  - **Contiguous allocation**
  - **Linked allocation**
  - **Indexed allocation**

Contiguous Allocation



directory

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

Contiguous Allocation

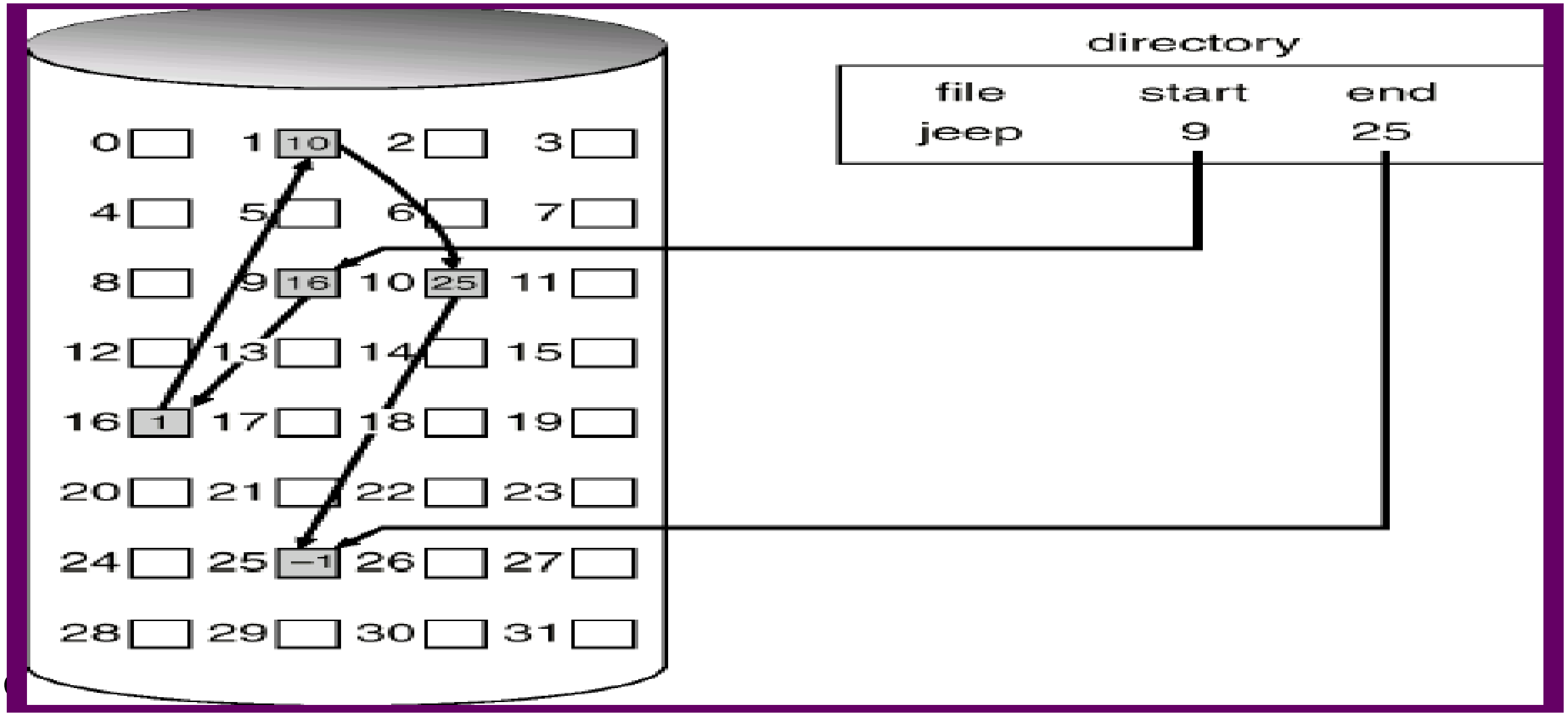
- Each file occupies a set of contiguous blocks
  - # of disk seeks is minimized
  - The dir entry for each file = (starting #, size)
- Both sequential & random access can be implemented efficiently
- Problems
  - External fragmentation ➔ compaction
  - File cannot grow ➔ extend-based FS

Extent-Based File System

- Many newer file system use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An extent is a **contiguous blocks** of disks
  - A file contains one or more extents
  - An extent: (starting block #, length, **pointer to next extent**)
  - ☹ Random access become more costly
  - ☹ Both internal & external fragmentation are possible

Linked Allocation

- Each file is a linked list of blocks
  - Each block contains a pointer to the next block
  - ➔data portion: block size – pointer size
- File read: following through the list



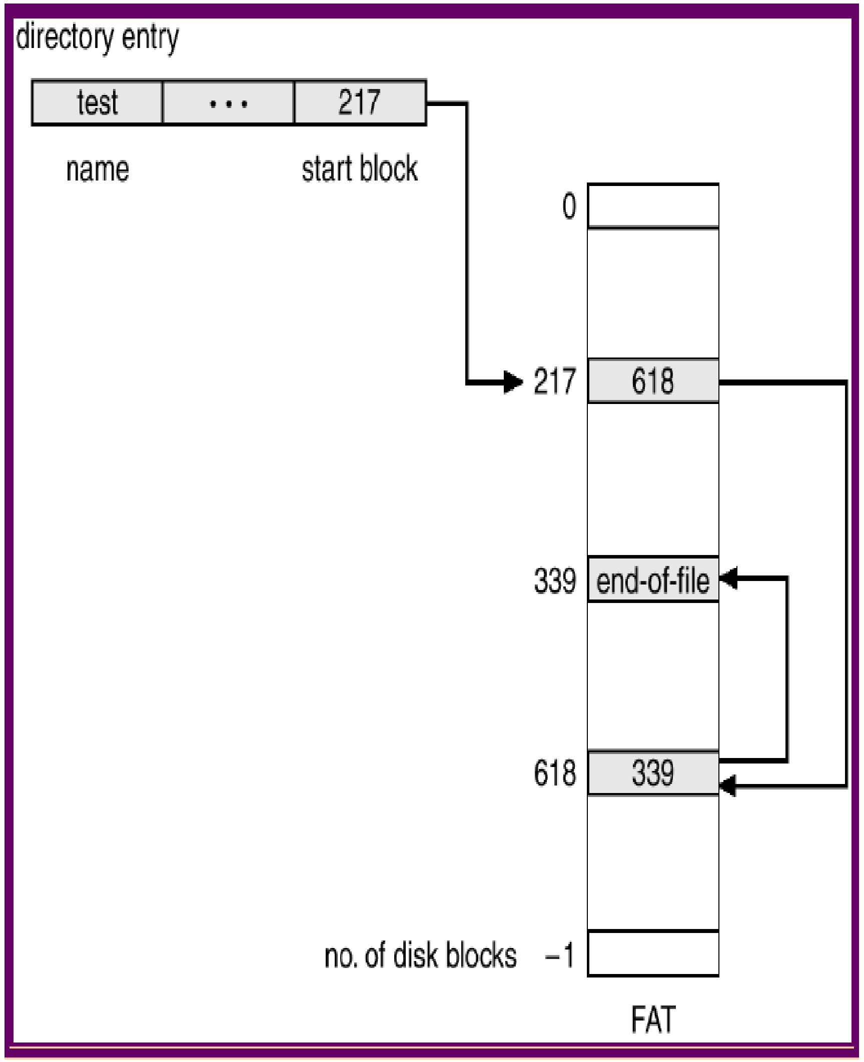
Linked Allocation

- Problems
  - Only good for **sequential-access** files
    - Random access requires traversing through the link list
    - Each access to a link list is a disk I/O (because link pointer is stored inside the data block)
  - Space required for pointer (4 / 512 = 0.78%)
    - solution: unit = cluster of blocks

      → internal fragmentation
  - Reliability
    - One missing link breaks the whole file

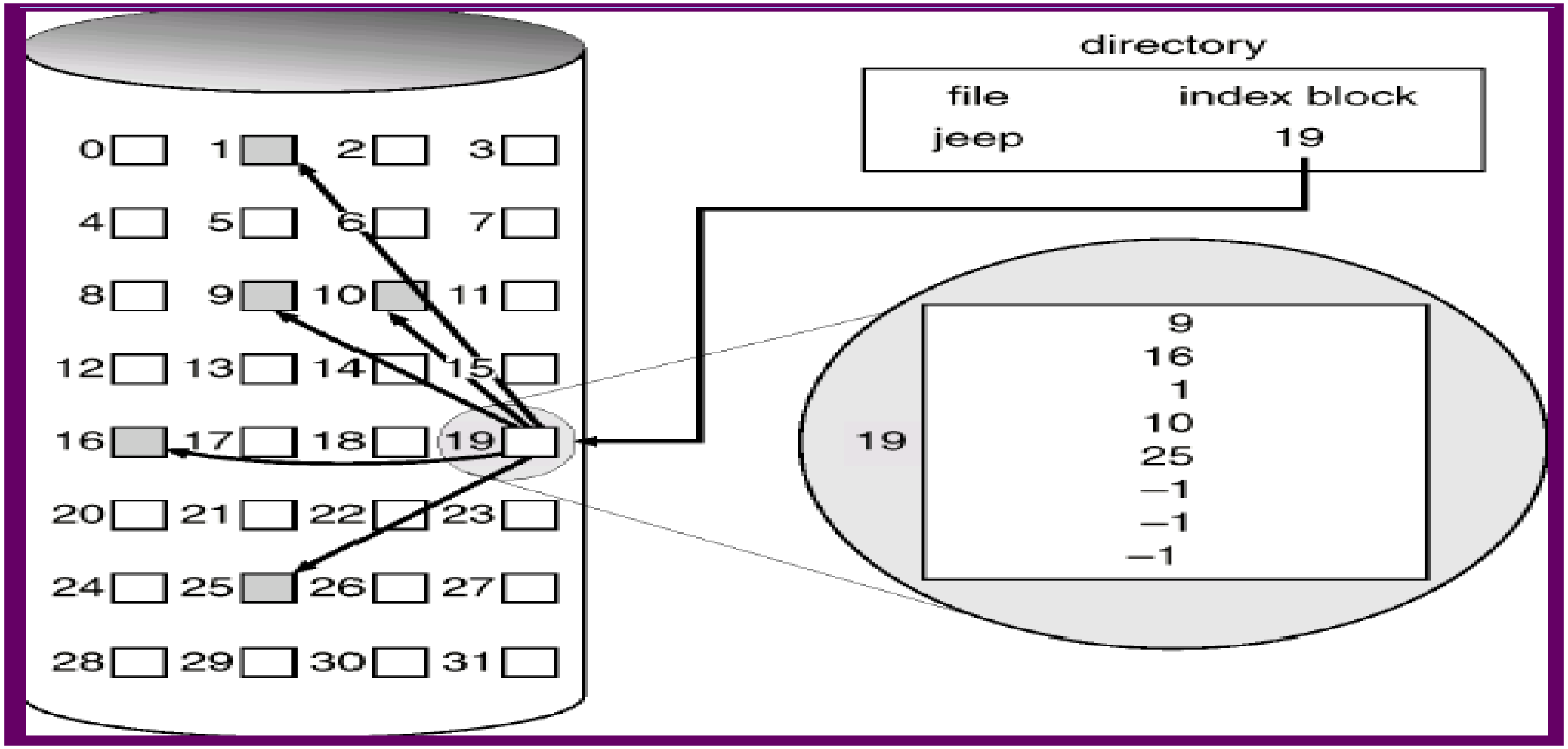
FAT (File Allocation Table) file system

- FAT32
  - Used in MS/DOS & OS/2
  - Store all links in a table
  - 32 bits per table entry
  - located in a section of disk at the beginning of each partition
- FAT(table) is often cached in memory
  - Random access is improved
  - Disk head find the location of any block by reading FAT



Indexed Allocation Example

- The directory contains the address of the file index block
- Each file has its own index block
- Index block stores block # for file data



Indexed Allocation

- Bring all the pointers together into one location: the index block (one for each file)

☺: 1. Implement **direct and random access efficiently**

2. No external fragmentation

2. Easy to create a file (no allocation problem)

☹: 1. Space for index blocks

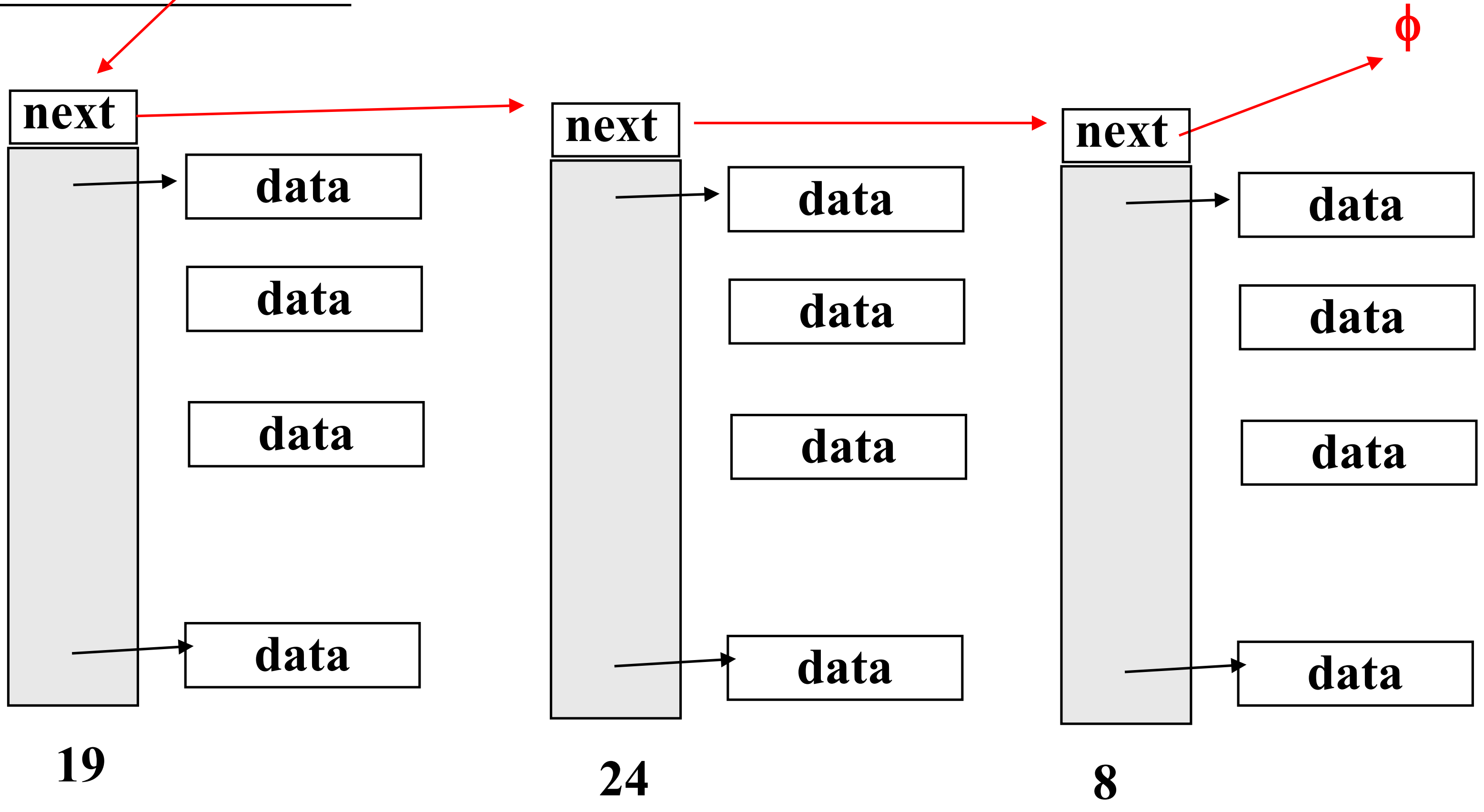
2. How large the index block should be ?

- **linked scheme**
- **multilevel index**
- **combined scheme** (inode in BSD UNIX)

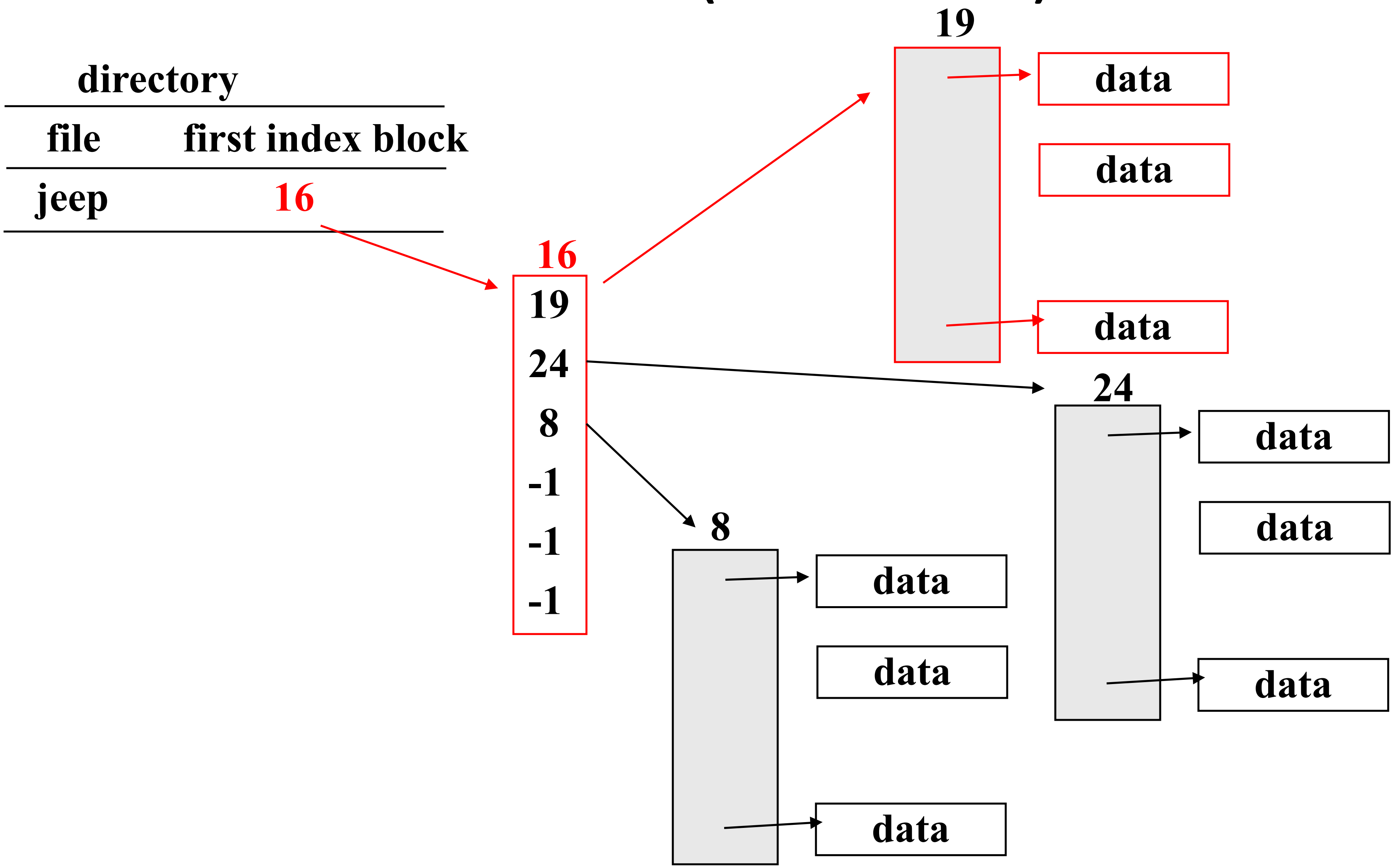
Linked Indexed Scheme

directory

file	first index block
jeep	19

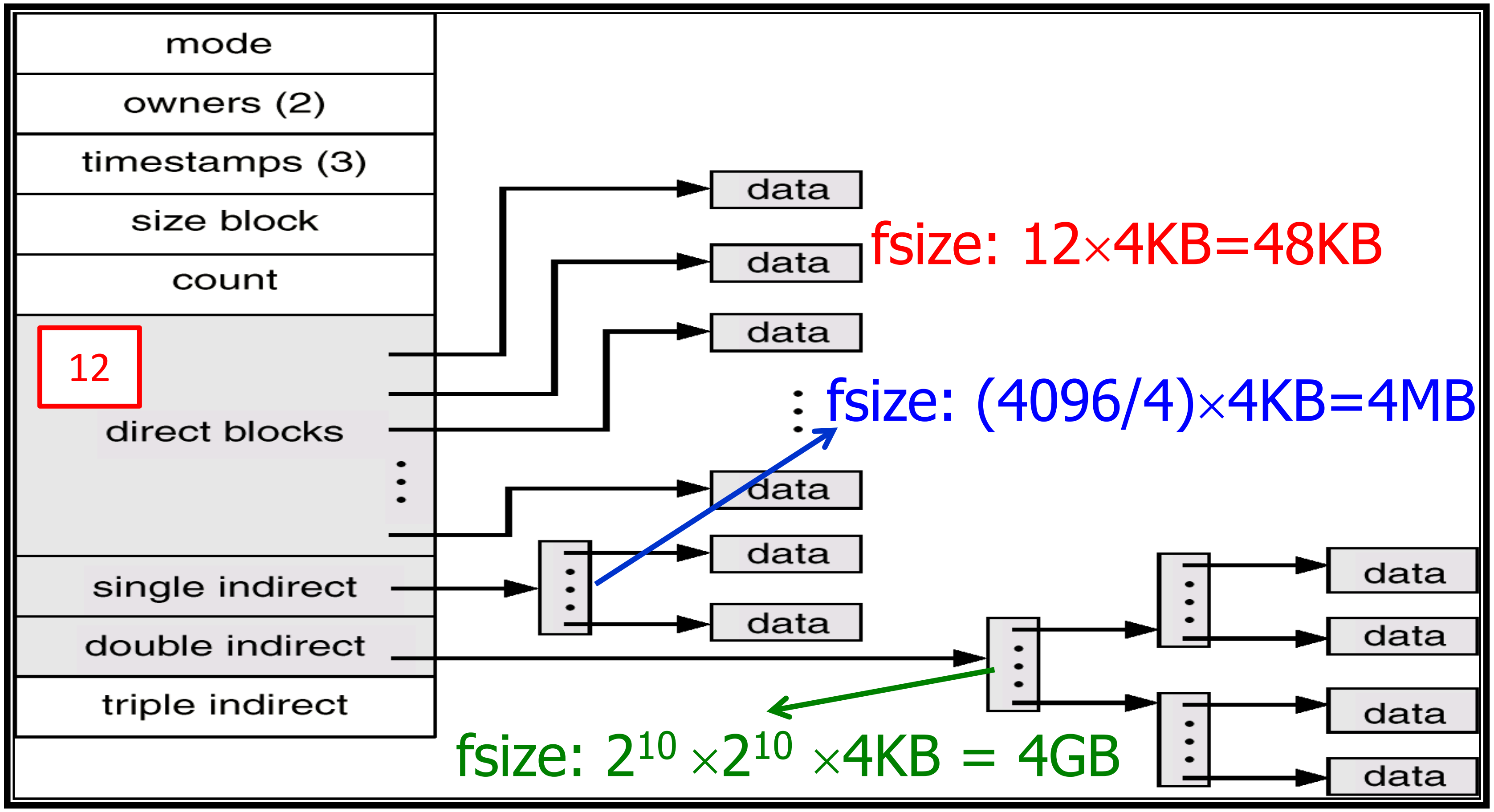


Multilevel Scheme (two-level)



Combined Scheme: UNIX inode

- File pointer:4B (32bits)➔reach only 4GB (2^{32}) files
- Let each data/index block be 4KB



Free-Space Management

Free Space

- Free-space list: records all free disk blocks
- Scheme
  - Bit vector
  - Linked list (same as linked allocation)
  - Grouping (same as linked index allocation)
  - Counting (same as contiguous allocation)
- File systems usually manage free space in the same way as a file

Bit vector

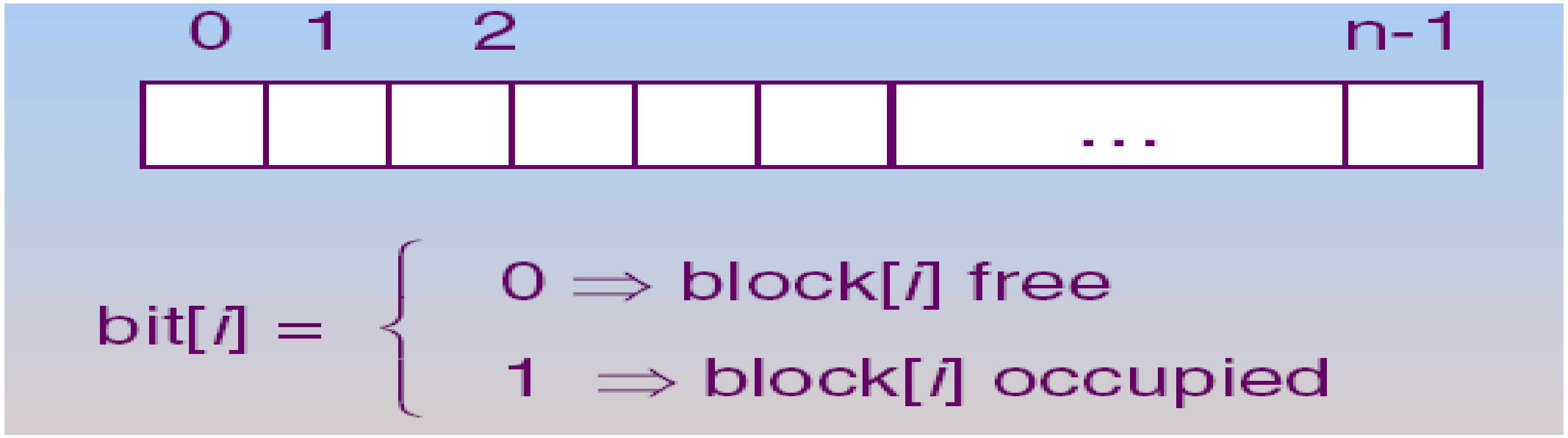
- Bit Vector (bitmap): one bit for each block
  - e.g. 001111001111111100111001100000.....

☺: simplicity, efficient

(HW support bit-manipulation instruction)

☹:bitmap must be cached for good performance

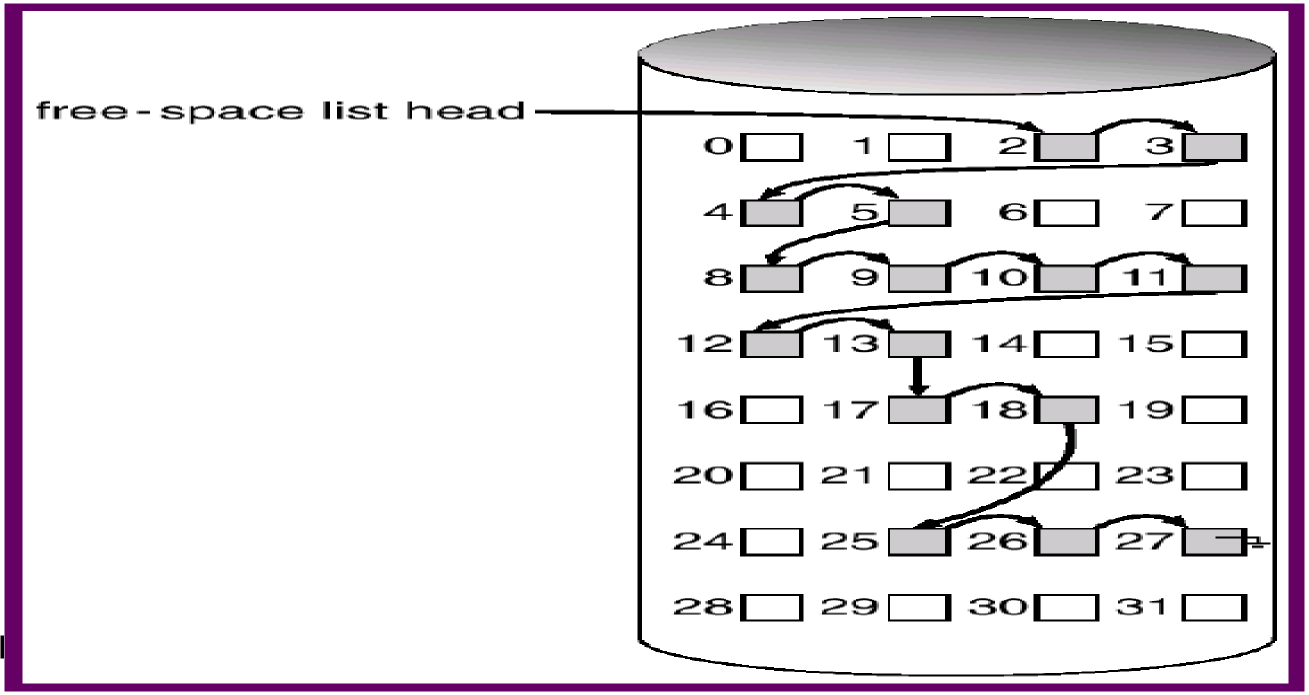
- A 1-TB(4KB block) disk needs 256MB (i.e., $2^{(40-12)}$) bitmap



$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ free} \\ 1 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

Linked List

- Link together all free blocks (same as linked allocation)
- Keep the first free block pointer in a special location on the disk and caching in memory
- Traversing list could be inefficient
  - No need for traversing; Put all link-pointers in a table(FAT)



Grouping & Counting

- Grouping (Same as linked-index allocation)
  - store address of n free blocks in the 1st block
  - the first (n-1) pointers are free blocks
  - the last pointers is another grouping block
- Counting (Same as contiguous allocation)
  - keep the address of the first free block and # of contiguous free blocks

Review Slides (II)

- Allocation:
  - Contiguous file allocation? Extent-based file system?
  - Linked allocation?
  - Indexed allocation?
    - Linked scheme
    - multilevel index allocation
    - Combine scheme
- Free space:
  - Bit vector, linked list, counting, grouping

Textbook Questions

- 11.1: Consider a file system that uses a modified contiguous-allocation scheme with support for extents. A file is a collection of extents, with each extent corresponding to a contiguous set of blocks. A key issue in such systems is the degree of variability in the size of the extents. What are the advantages and disadvantages of the following schemes?
  a. All extents are of the same size, and the size is predetermined.
  b. Extents can be of any size and are allocated dynamically
  c. Extents can be of a few fixed sizes, and these sizes are predetermined.
- 11.2: Contrast the performance of the three techniques for allocating disk blocks (contiguous, linked, and indexed) for both sequential and random file access.

Textbook Questions

- 11.3: What are the advantages of the variant of linked allocation that uses a FAT to chain together the blocks of a file?
- 11.4: Consider a system where free space is kept in a free-space list.
  a. Suppose that the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain your answer.
  b. Consider a file system similar to the one used by UNIX with indexed allocation. How many disk I/O operations might be required to read the contents of a small local file at /a/b/c? Assume that none of the disk blocks is currently being cached.
  c. Suggest a scheme to ensure that the pointer is never lost as a result of memory failure.

Textbook Questions

- 11.7: Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of the three allocation strategies (contiguous, linked, and indexed), answer these questions:
   a. a. How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long.)
   b. If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?
- 11.8: Consider a file system that uses inodes to represent files. Disk blocks are 8-KB in size and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, plus single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system?