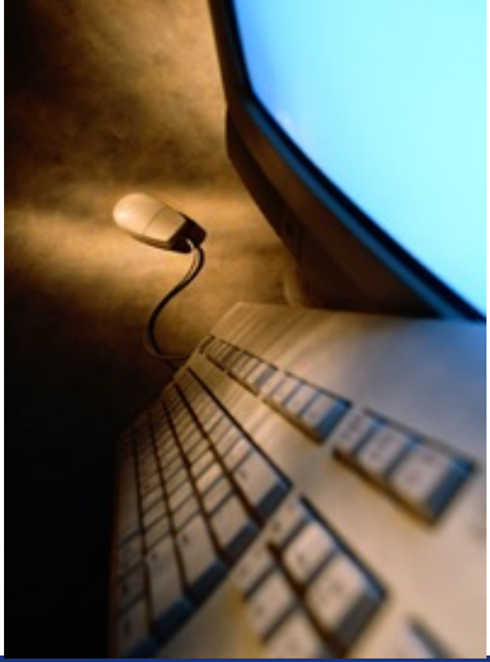
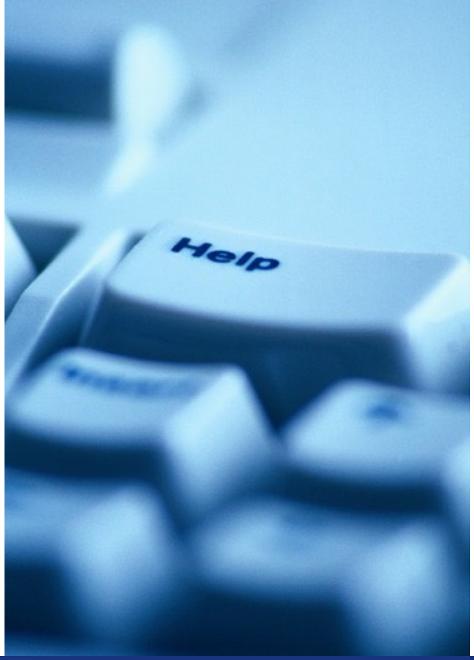




操作系统原理

Operating Systems Principles

陈鹏飞
计算机学院

第十四讲 — 文件系统实现

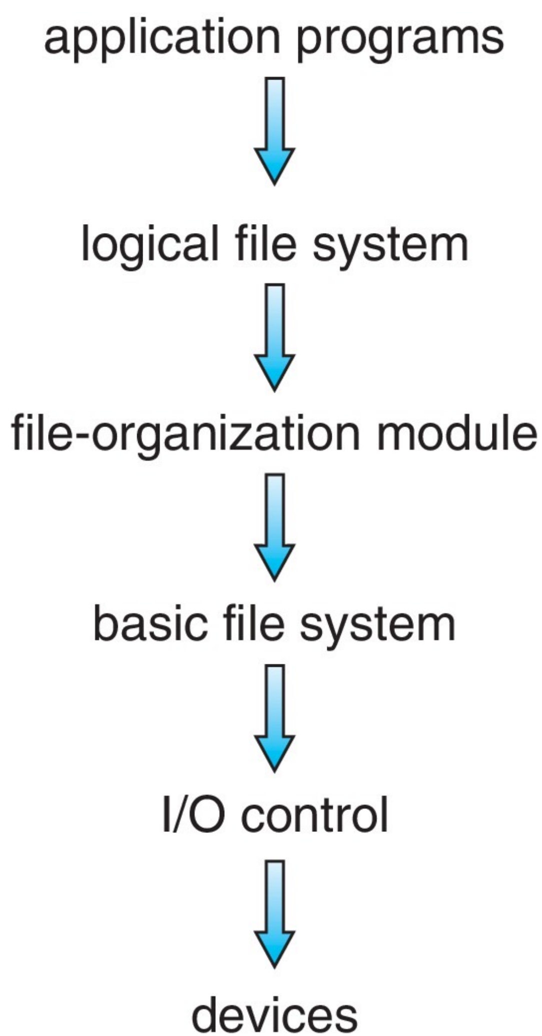
目标

- 本地文件系统和目录结构的实现细节；
- 远程文件系统；
- 块分配与空闲块的算法和权衡；

文件系统结构

- **File structure**
  - Logical storage unit
  - Collection of related information
- **File system resides on secondary storage (disks)**
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- **Disk provides in-place rewrite (原地重写)and random access**
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block (FCB) – storage structure consisting of information about a file**
- **Device driver controls the physical device**
- **File system organized into layers**

分层设计的文件系统


application programs
logical file system
file-organization module
basic file system
I/O control
devices

分层设计的文件系统

- **Device drivers manage I/O devices at the I/O control layer**

  Given commands like

  read drive1, cylinder 72, track 2, sector 10, into memory location 1060

  Outputs low-level hardware specific commands to hardware controller

- **Basic file system given command like “retrieve block 123” translates to device driver**
- **Also manages memory buffers and caches (allocation, freeing, replacement)**
  - Buffers hold data in transit
  - Caches hold frequently used data
- **File organization module understands files, logical address, and physical blocks**

- Translates logical block # to physical block #
- Manages free space, disk allocation

分层设计的文件系统

- **Logical file system (逻辑文件系统) manages metadata information**
  - Translates file name into file number, file handle, location by maintaining file control blocks（文件控制块） (**inodes** in UNIX)
  - Directory management
  - Protection
- **Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance**
- **Logical layers can be implemented by any coding method according to OS designer**

分层设计的文件系统

- **Many file systems, sometimes many within an operating system**
  - Each with its own format:
  - CD-ROM is ISO 9660;
  - Unix has **UFS**, FFS;
  - Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray,
  - Linux has more than 130 types, with **extended file system** ext3 and ext4 leading; plus distributed file systems, etc.)
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, **FUSE**

分层设计的文件系统

- **We have system calls at the API level, but how do we implement their functions?**
  - On-disk and in-memory structures
- **Boot control block contains info needed by system to boot OS from that volume**
  - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table) contains volume details**
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- **Directory structure organizes the files**
  - Names and inode numbers, master file table

文件系统操作

- **We have system calls at the API level, but how do we implement their functions?**
  - On-disk and in-memory structures
- **Boot control block (引导控制块) contains info needed by system to boot OS from that volume**
  - Needed if volume contains OS, usually first block of volume
- **Volume control block （卷控制块）(superblock, master file table) contains volume details**
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- **Directory structure organizes the files**
  - Names and inode numbers, master file table

文件控制块

- **OS maintains FCB per file, which contains many details about the file，包含唯一标识号**
  - Typically, inode number, permissions, size, dates
  - Example

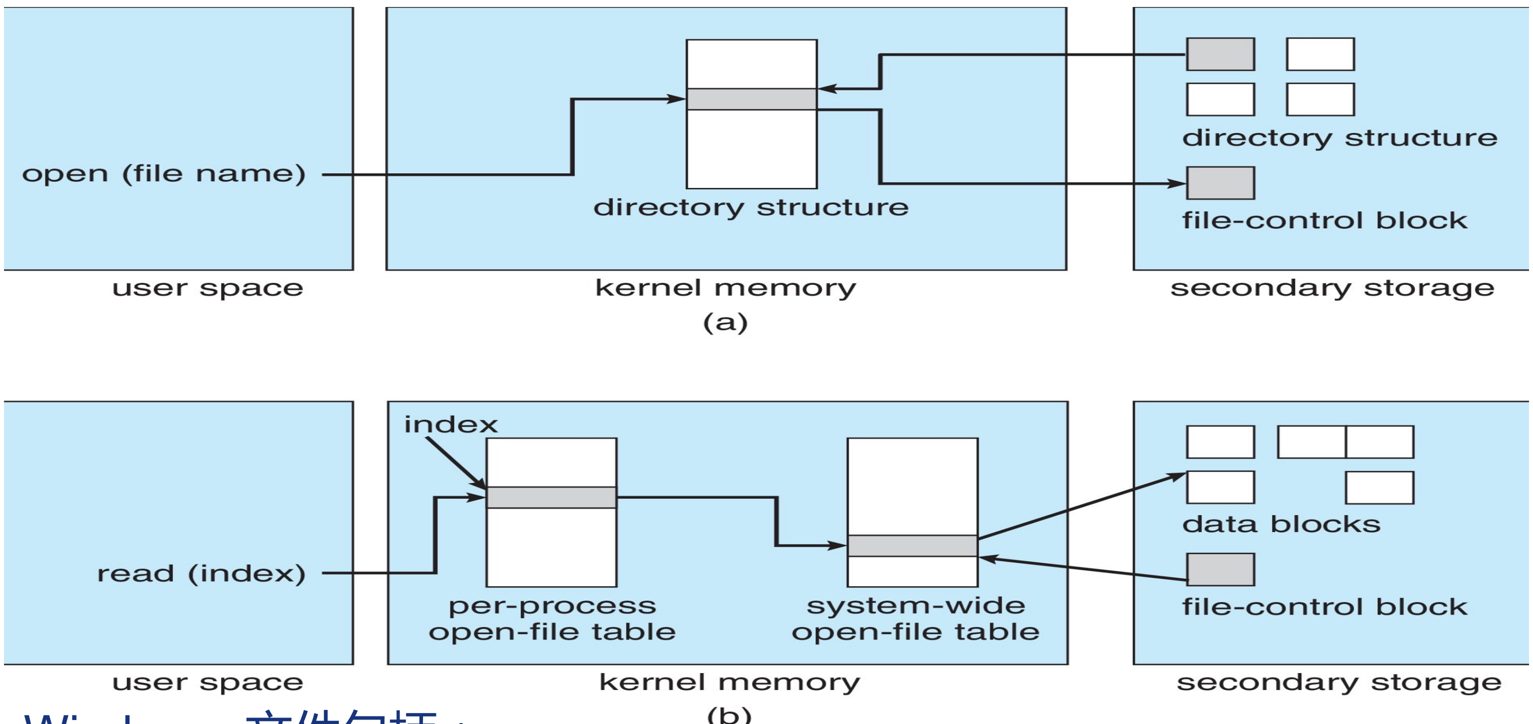
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

内存中的文件结构

- **Mount table （安装表、挂载表）storing file system mounts, mount points, file system types**
- **System-wide open-file table contains a copy of the FCB of each file and other info**
- **Per-process open-file table contains pointers to appropriate entries in system-wide open-file table as well as other info**

内存中的文件结构

- Figure 12-3(a) refers to opening a file
- Figure 12-3(b) refers to reading a file



Windows: 文件句柄；
Unix：文件描述符

目录实现

- **Linear list of file names with pointer to the data blocks**
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table – linear list with hash data structure**
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

分配方法

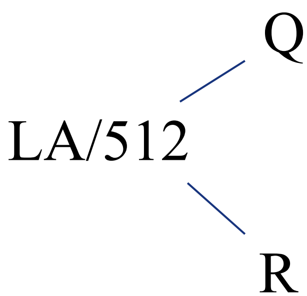
- **An allocation method refers to how disk blocks are allocated for files:**
  - Contiguous
  - Linked
  - File Allocation Table (FAT)
  - 索引分配（Indexed allocation）

连续分配方法

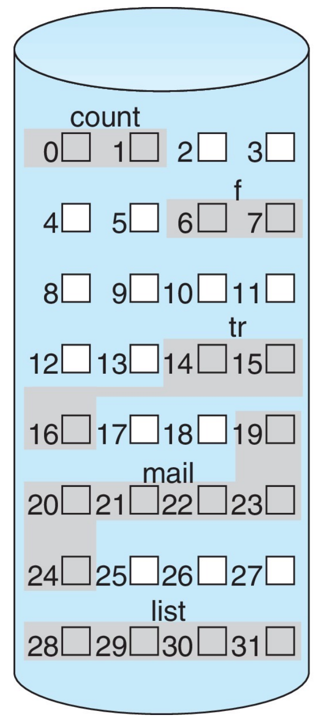
- **An allocation method refers to how disk blocks are allocated for files:**
- **Each file occupies set of contiguous blocks**
  - Best performance in most cases (寻道数量最小， 寻道时间最小)
  - Simple – only starting location (block #) and length (number of blocks) are required
  - 支持顺序访问和直接访问；
  - Problems include:
    - Finding space on the disk for a file,
    - Knowing file size,
    - External fragmentation, need for **compaction off-line (downtime)** or **on-line**

连续分配方法

- Mapping from logical to physical (block size =512 bytes)



- Block to be accessed = starting address + Q
- Displacement into block = R



directory

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

基于扩展的系统

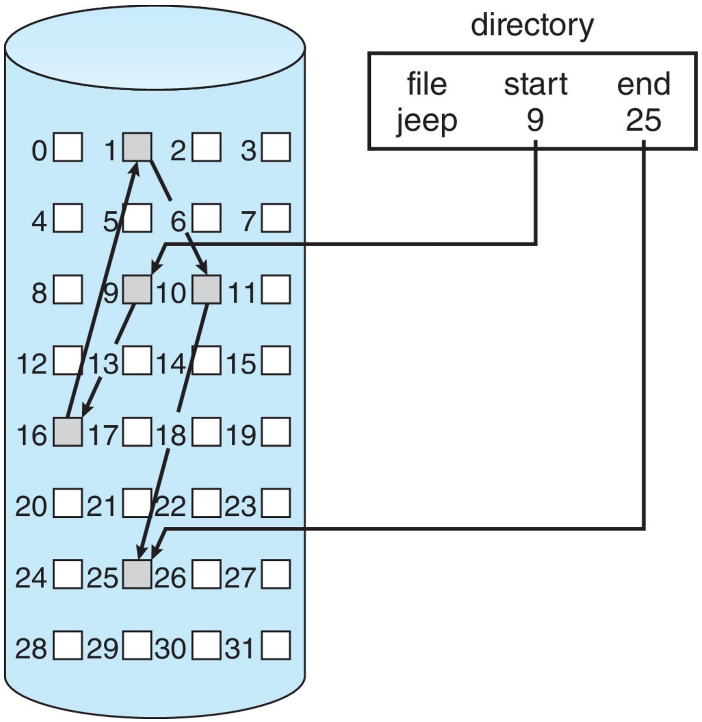
- **Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme**
- **Extent-based file systems allocate disk blocks in extents**
- **An extent is a contiguous block of disks**
  - Extents are allocated for file allocation
  - A file consists of one or more extents
  - 文件块的位置记录为：地址、块数、下一个扩展的首块的指针；

链接分配

- Each file is a linked list of blocks
- File ends at nil pointer
- No external fragmentation
- Each block contains pointer to next block
- No compaction, external fragmentation
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem
- Locating a block can take many I/Os and disk seeks

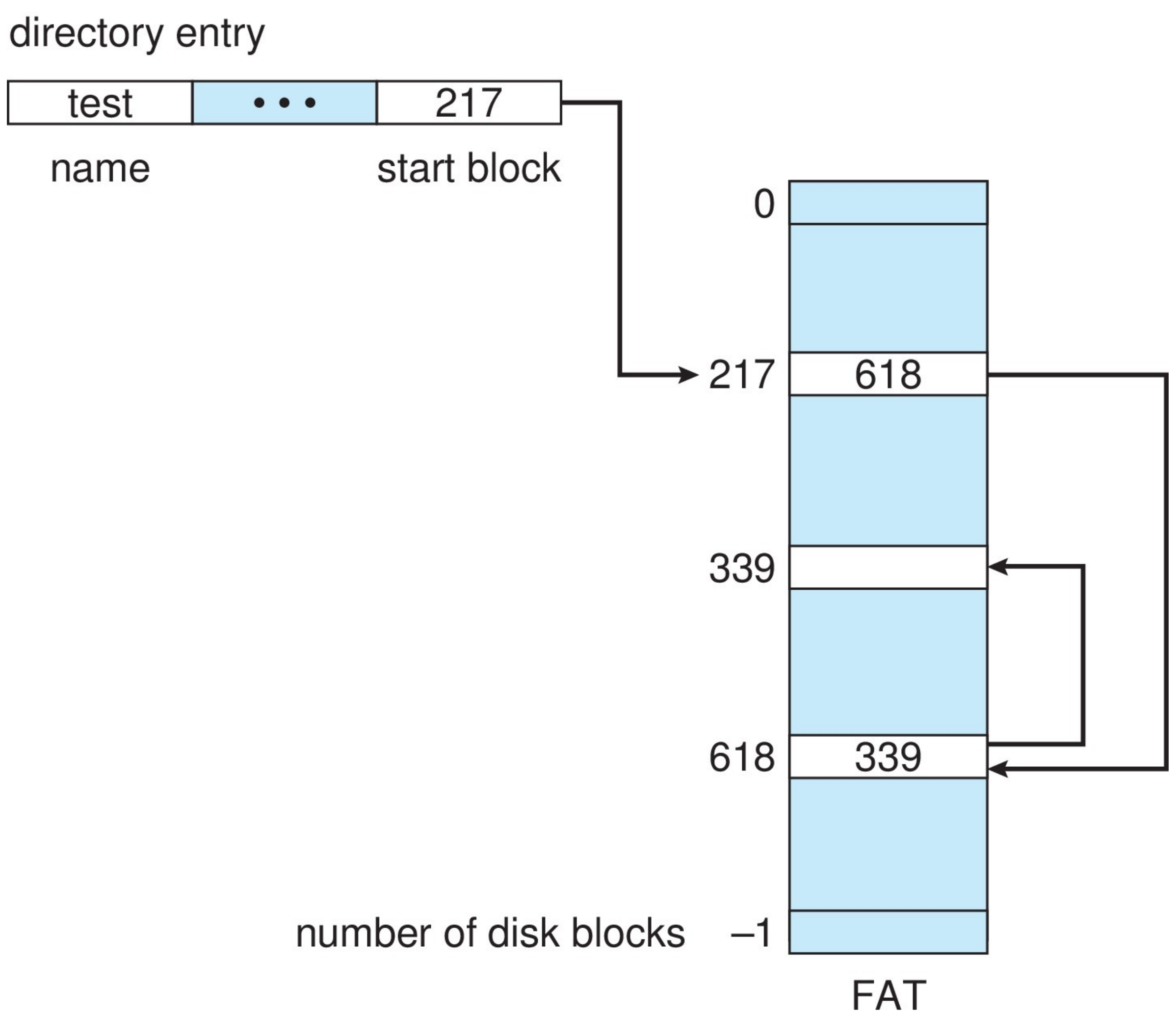
链接分配

- **Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk**
- **Scheme**



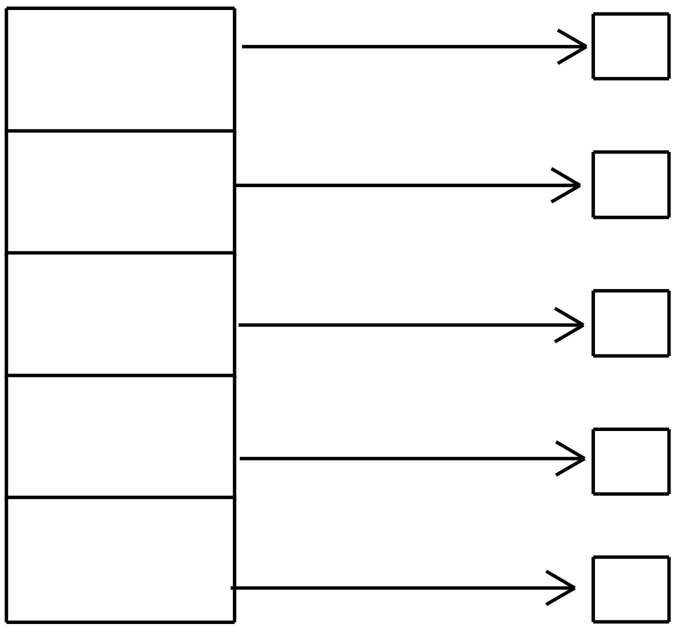
无外部碎片；但是有效用于顺序访问，指针占空间，可靠性也是一个问题

文件分配表


directory entry
test
• • •
217
name
start block
0
217
618
339
618
339
number of disk blocks
–1
FAT

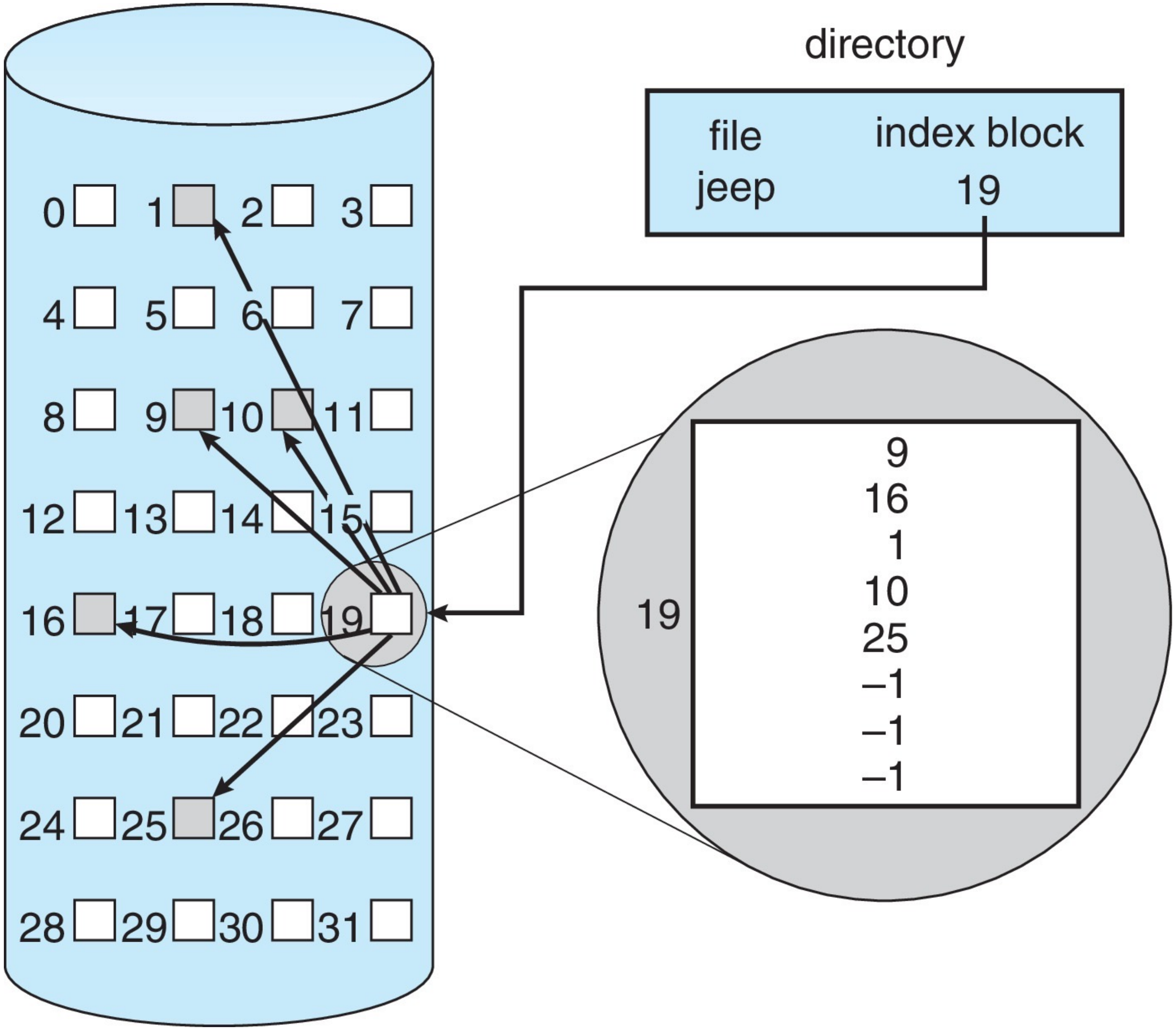
索引分配

- Each file has its own index block(s) of pointers to its data blocks
- Logical view



index table

索引分配


directory
file
index block
jeep
19
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
19
9
16
1
10
25
–1
–1
–1

索引分配——小文件

- 需要索引表
- 随机存取
- 无外部碎片的动态访问，但有索引块的开销；
- 在最大大小为256K字节、块大小为512字节的文件中从逻辑映射到物理。索引表只需要1个块

Q

LA/512

R

- 计算：
  - Q=进入索引表的位移
  - R=块体中的位移

索引分配——大文件

- 在长度无限的文件中从逻辑到物理地址的映射（块大小为512个字）
  - 链接方案 - 索引表的链接块（大小不限）
  - 多级索引

索引分配——大文件

- 两级索引（4K块可以在外部索引->1048576数据块中存储1024个四字节指针，文件大小高达4GB）

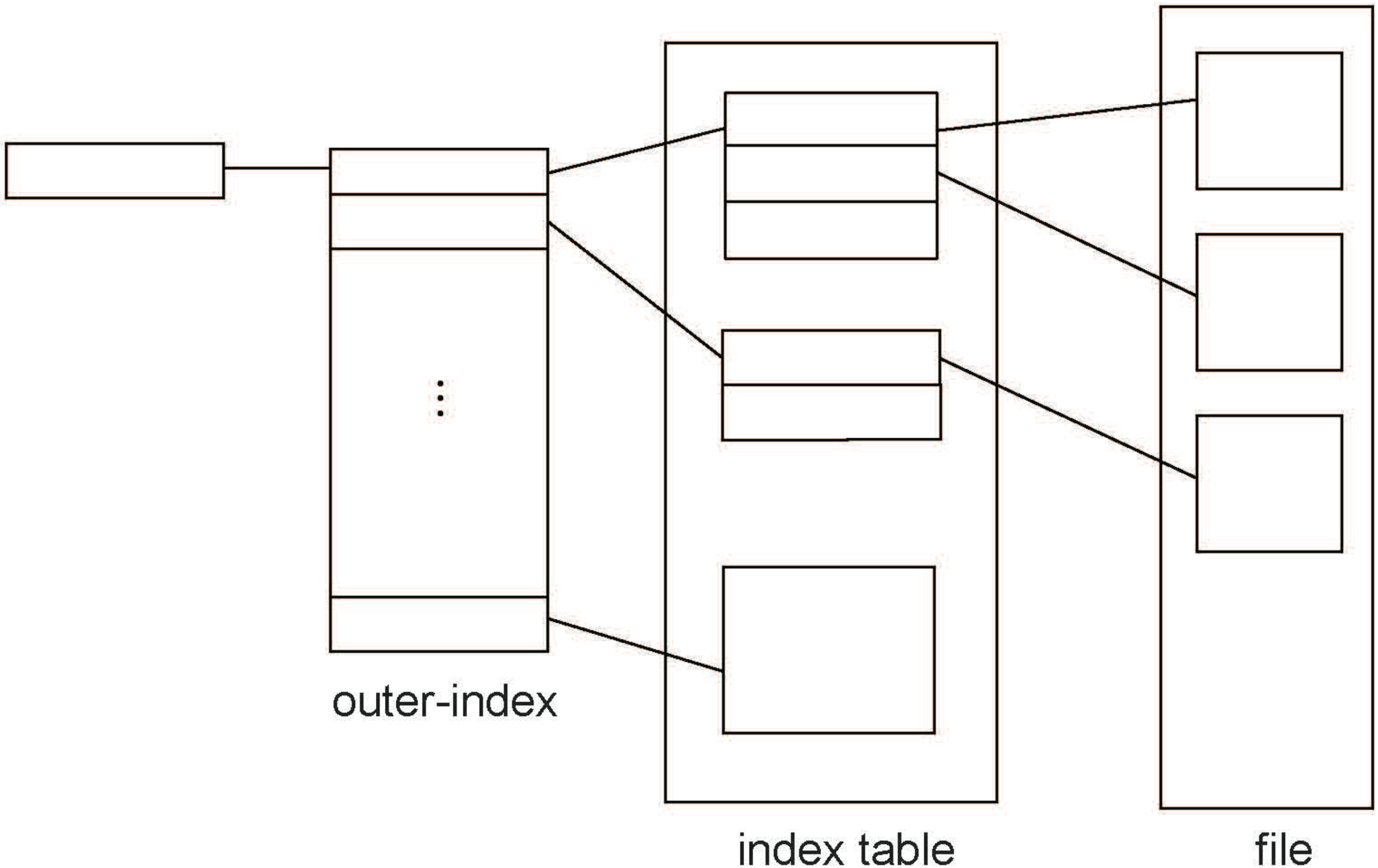
$$LA\ /\ (512\ \mathrm{x}\ 512) \begin{cases} Q_1 \\ R_1 \end{cases}$$

- 外部索引的映射方案：
  - Q1=进入外部索引的位移
  - R1的使用方式如下：

$$R_1\ /\ 512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

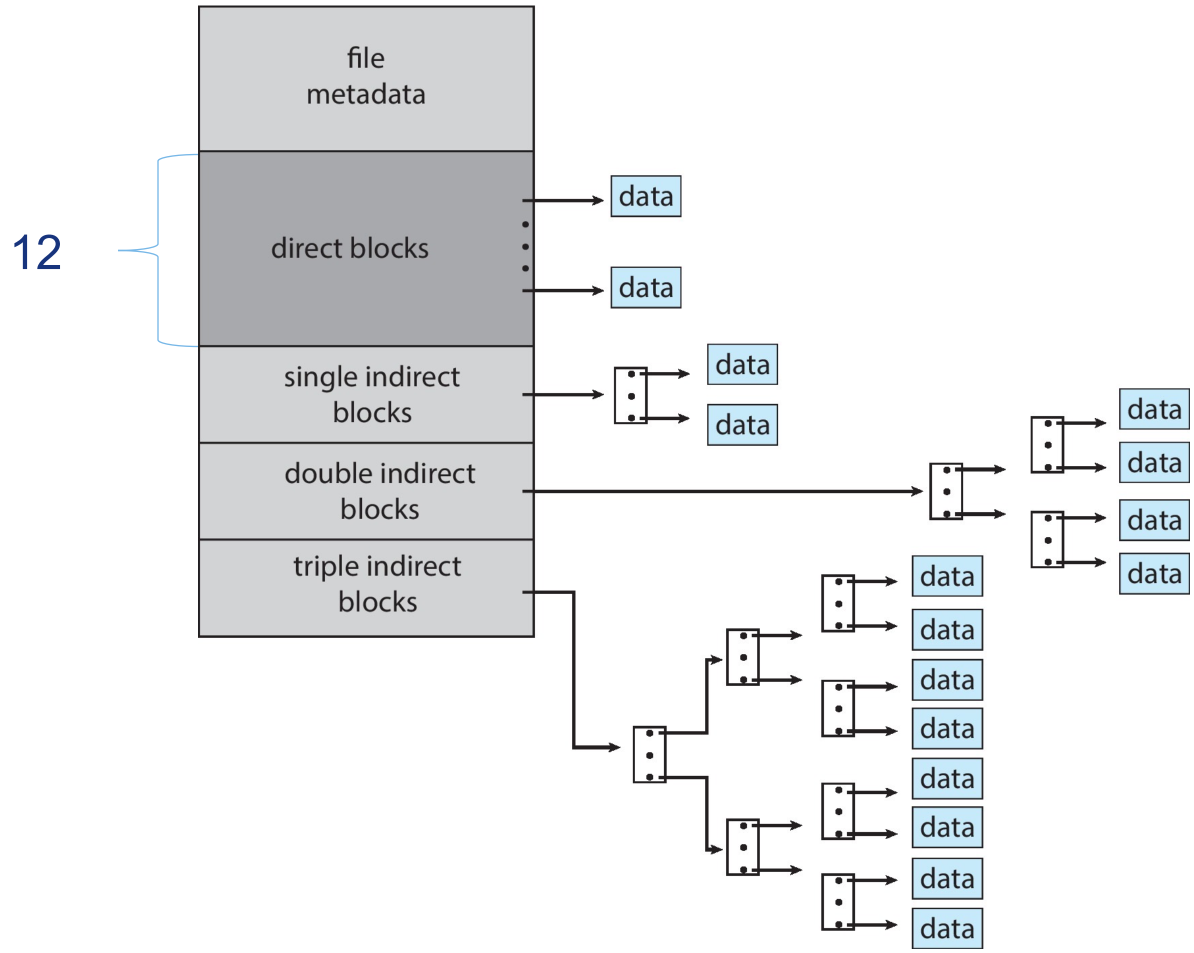
  - 索引级别的映射方案：
    - Q2=进入索引表块的位移
    - R2置换到文件块中

索引分配——两级方案



索引分配——组合方案

- 每个块4K字节，32位地址



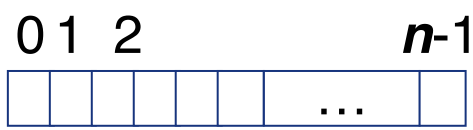
- 超过32位文件指针可寻址的索引块

性能

- **最佳方法取决于文件访问类型**
  - 连续序列和随机序列
- **链接适用于顺序，而不是随机**
- **在创建时声明访问类型**
  - 选择连续分配或链接分配；
- **索引分配更复杂**
  - 单块访问可能需要读取2个索引块（二级索引），然后读取数据块；
  - 性能取决于：索引的结构、文件的大小以及所需块的位置；
- **对于NVM，没有磁盘头，因此需要不同的算法和优化**
  - 使用旧算法会占用大量CPU周期，试图避免不存在的头部移动；
  - 目标是减少CPU周期和I/O所需的总体路径；

空闲空间管理

- **File system maintains free-space list to track available blocks/clusters**
  - (Using term “block” for simplicity)
- **Bit vector or bit map (*n* blocks)**



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

Block number calculation

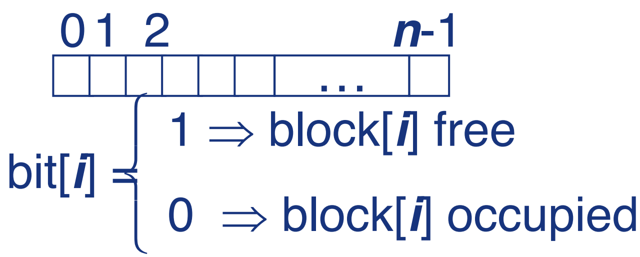
(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

查找第一个空闲块

CPUs have instructions to return offset within word of first “1” bit

空闲空间管理

- **File system maintains free-space list to track available blocks**
- **Bit vector or bit map (*n* blocks)**



- **Bit map requires extra space**
  - Example:

    block size = 4KB = 2^{12} bytes

    disk size = 2^{40} bytes (1 terabyte)

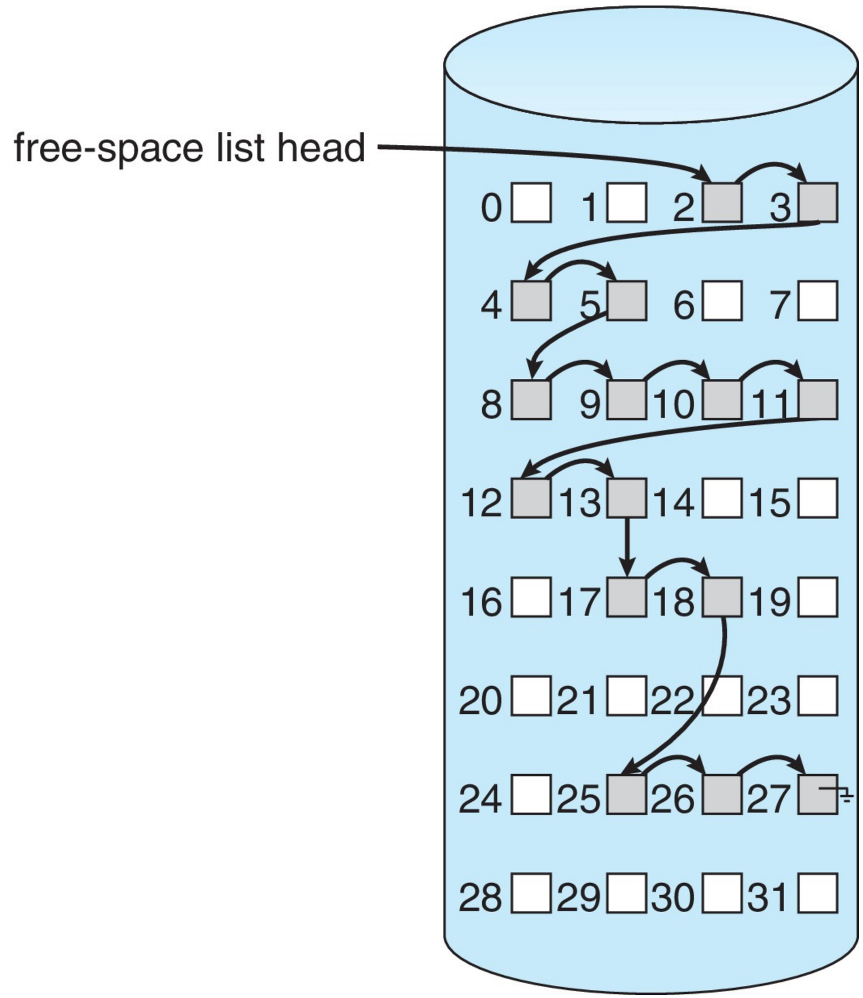
    $n = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

    if clusters of 4 blocks -> 8MB of memory

- **Easy to get contiguous files**

空闲空间管理——链表

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste. Linked Free Space List on Disk of space
  - No need to traverse the entire list (if # free blocks recorded)



空闲空间管理

- **Grouping**
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)

- **Counting**
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free blocks
    - Free space list then has entries containing addresses and counts

空闲空间管理

- **Space Maps**
  - Used in **ZFS**
  - Consider meta-data I/O on very large file systems
    - Full data structures like bit maps cannot fit in memory ➔ thousands of I/Os
  - Divides device space into **metaslab** units and manages metaslabs
    - Given volume can contain hundreds of metaslabs
  - Each metaslab has associated space map
    - Uses counting algorithm
  - But records to log file rather than file system
    - Log of all block activity, in time order, in counting format
  - Metaslab activity ➔ load space map into memory in balanced-tree structure, indexed by offset
    - Replay log into that structure
    - Combine contiguous free blocks into single entry

效率和性能

- **Efficiency （效率） dependent on:**
  - Disk allocation and directory algorithms
  - Types of data kept in file’s directory entry
  - Pre-allocation or as-needed allocation of metadata structures
  - Fixed-size or varying-size data structures

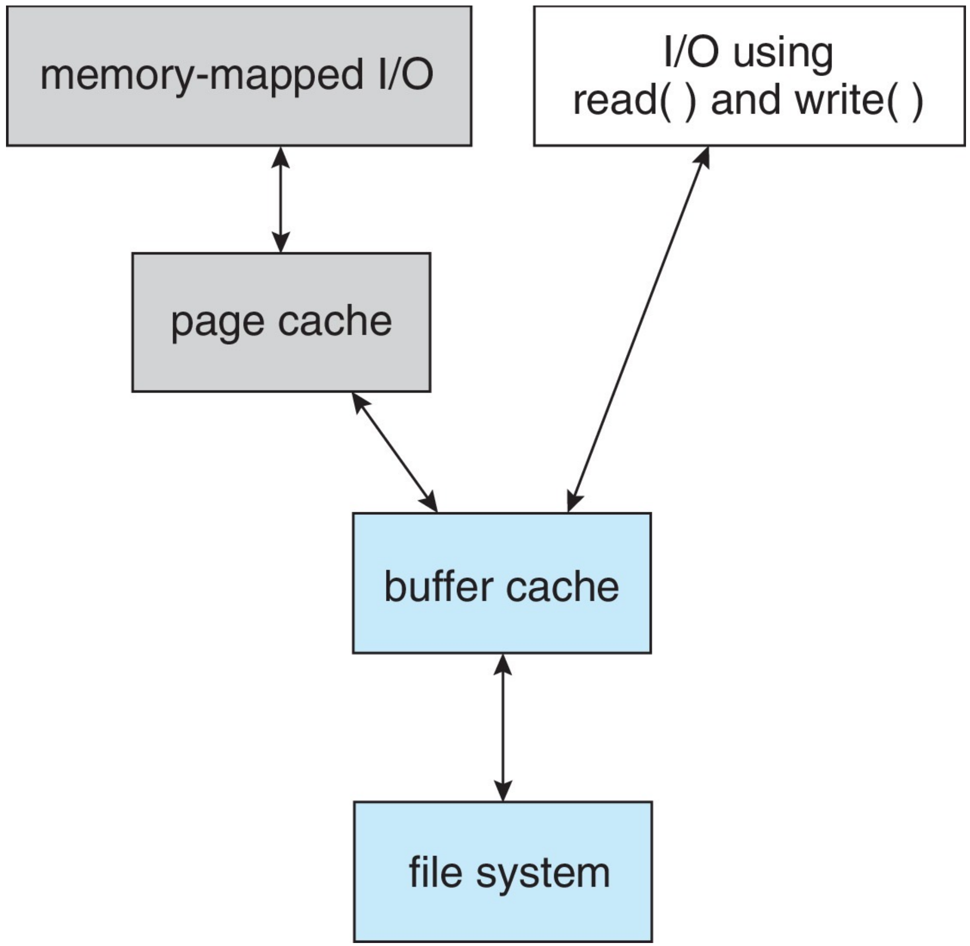
效率和性能

- **Performance**
  - Keeping data and metadata close together
  - **Buffer cache** – separate section of main memory for frequently used blocks
  - **Synchronous** writes sometimes requested by apps or needed by OS
    - No buffering / caching – writes must hit disk before acknowledgement
    - **Asynchronous** writes more common, buffer-able, faster
  - **Free-behind** and **read-ahead** – techniques to optimize sequential access
  - Reads frequently slower than writes

效率和性能

- A page cache caches pages rather than disk blocks using virtual memory techniques and addresses
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure

缺少统一缓冲区缓存的I/O

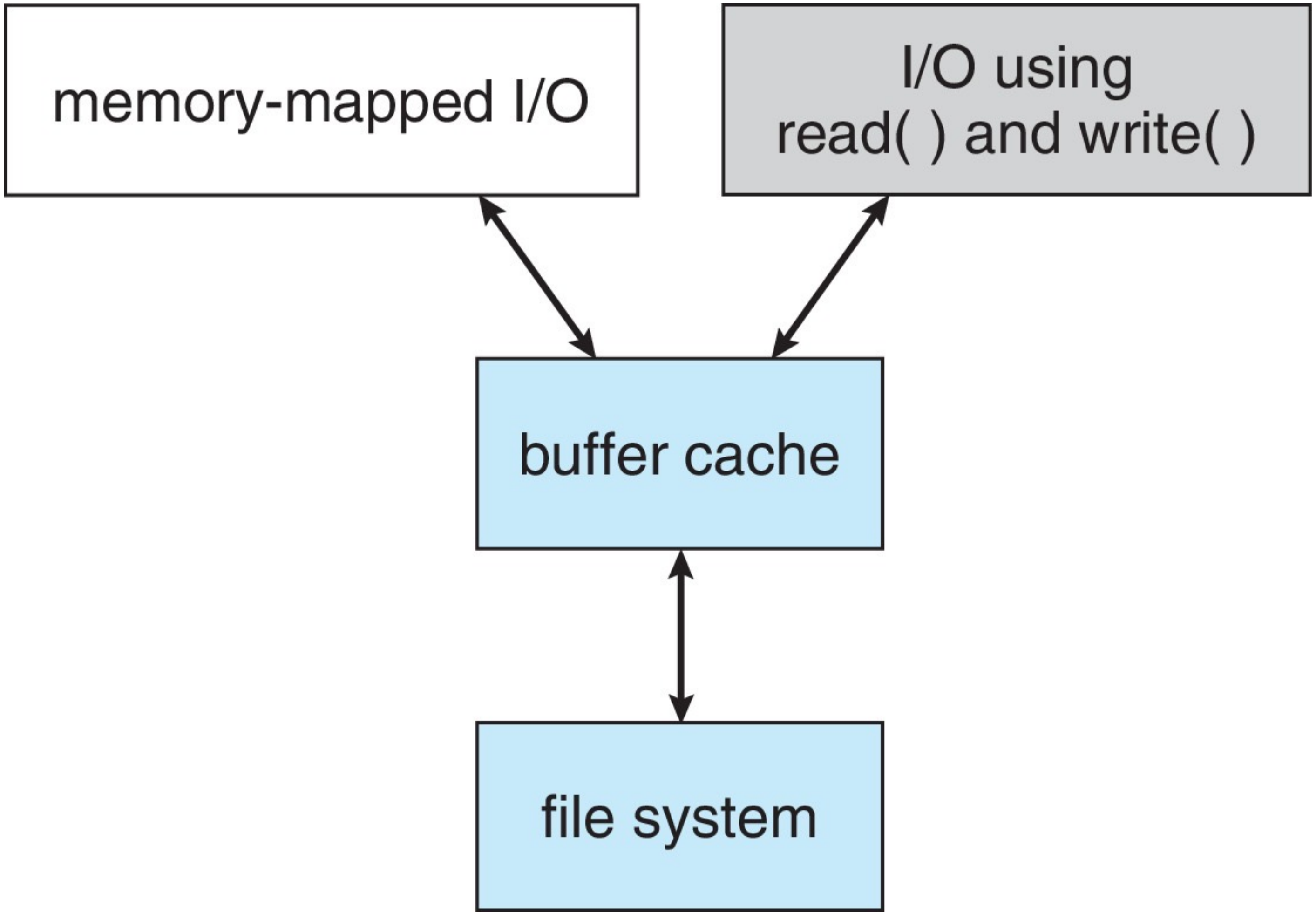


统一缓冲区缓存

- **A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O to avoid double caching**

- But which caches get priority, and what replacement algorithms to use?

统一缓冲区缓存


memory-mapped I/O
I/O using read( ) and write( )
buffer cache
file system

恢复

- **Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies**
  - Can be slow and sometimes fails
- **Use system programs to back up data from disk to another storage device (magnetic tape, other magnetic disk, optical)**
- **Recover lost file or disk by restoring data from backup**

基于日志的文件系统

- **Log structured (or journaling) file systems record each metadata update to the file system as a transaction**
- **All transactions are written to a log**
  - A transaction is considered committed once it is written to the log (sequentially)
  - Sometimes to a separate device or section of disk
  - However, the file system may not yet be updated
- **The transactions in the log are asynchronously written to the file system structures**
  - When the file system structures are modified, the transaction is removed from the log
- **If the file system crashes, all remaining transactions in the log must still be performed**
- **Faster recovery from crash, removes chance of inconsistency of metadata**

谢谢