CS343 - Operating Systems

Module-6C

File Allocation and Free Space Management

Dr. John Jose

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

http://www.iitg.ac.in/johnjose/

File-System Implementation

- File Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

File-System Implementation

- **File Control Block** contains many details about the file
  - inode number, permissions, size, dates

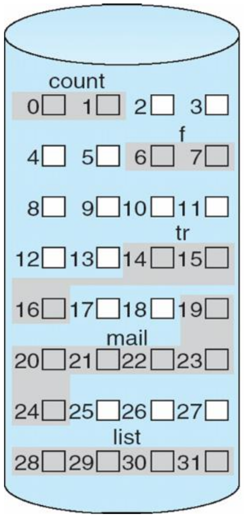
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Contiguous Allocation

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Contiguous Allocation

- Mapping from logical to physical
  - Block to be accessed = starting address
  - Displacement into block = length



directory

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

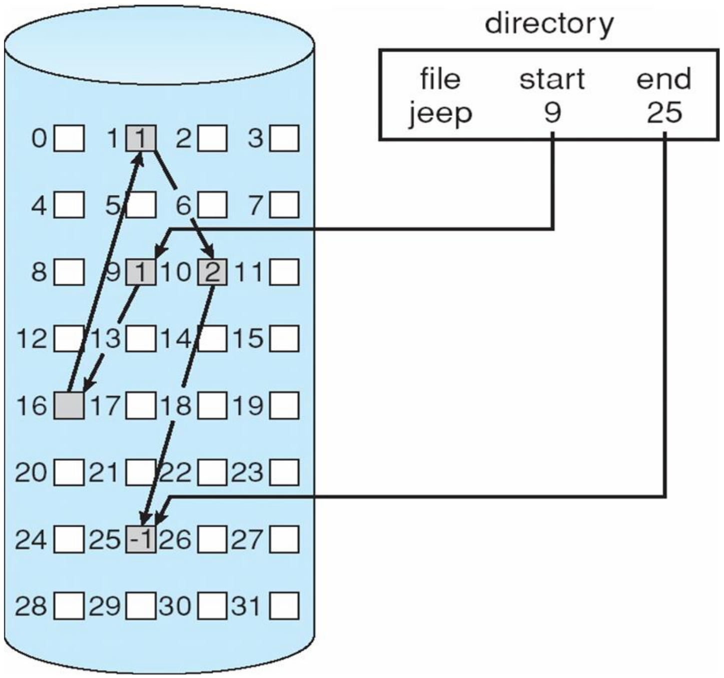
Extent-Based Systems

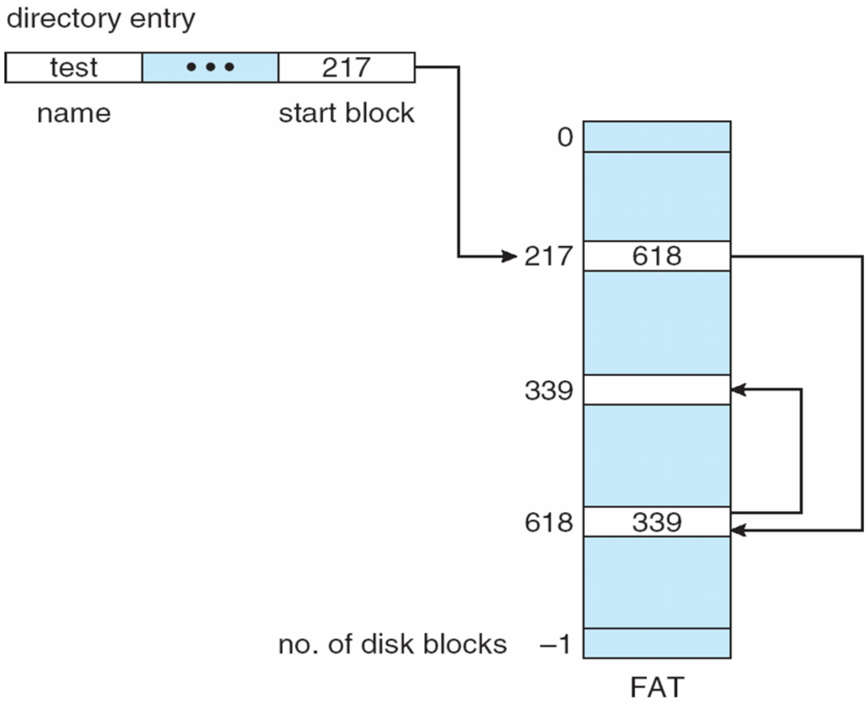
- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Linked Allocation

- **Linked allocation** – each file 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

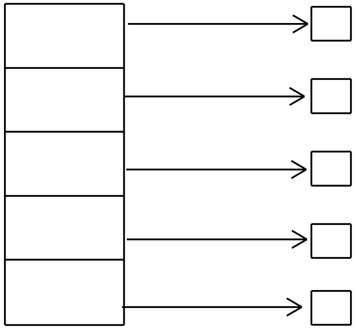
Linked Allocation



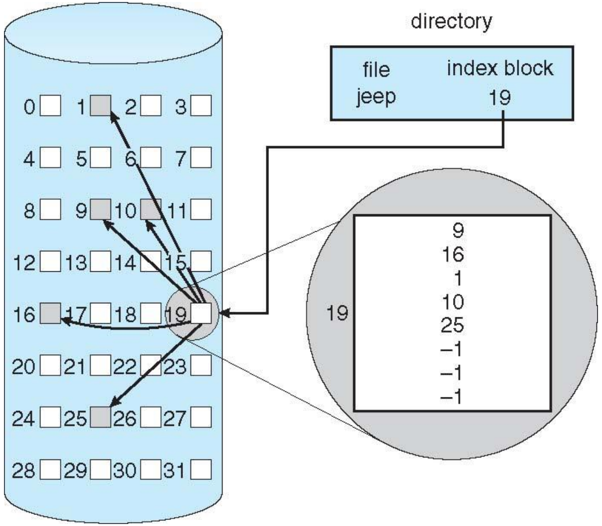


Indexed Allocation

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



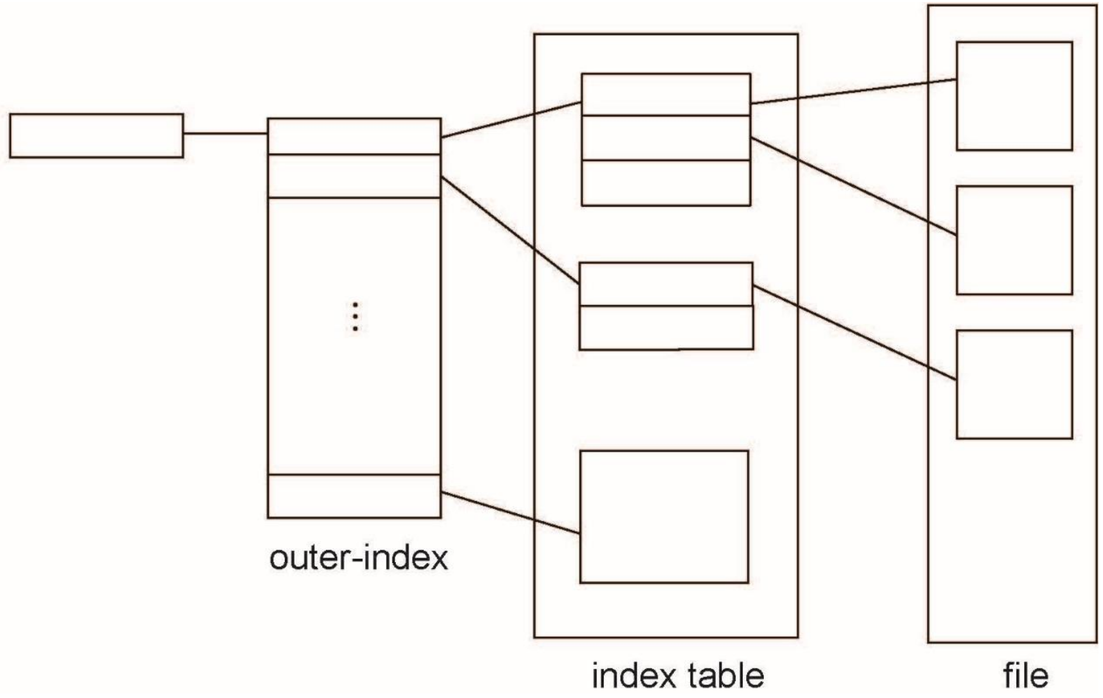
index table



Indexed Allocation

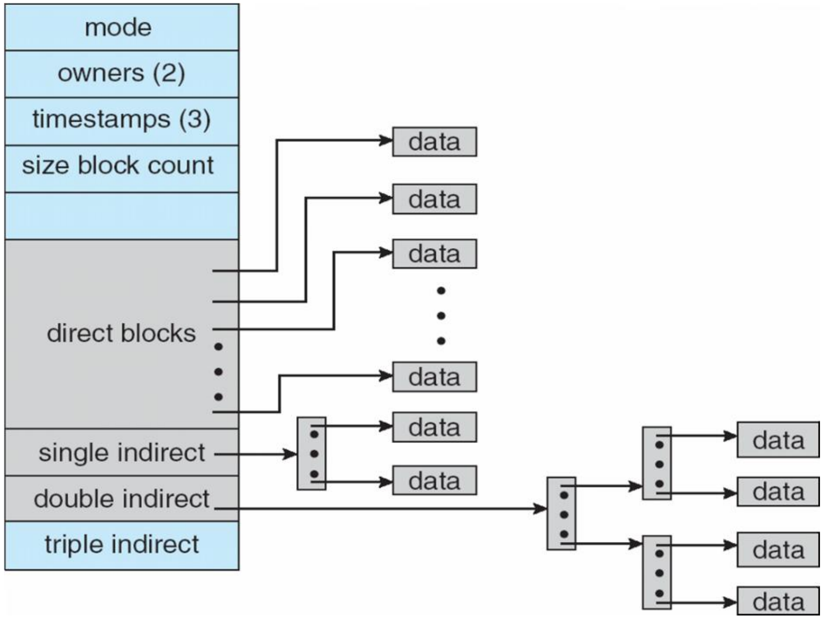
- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Single Level and Multilevel Index for small and large files

Indexed Allocation



UNIX UFS

4K bytes per block, 32-bit addresses



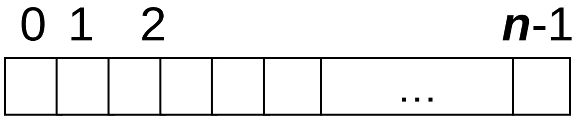
More index blocks than can be addressed with 32-bit file pointer

Performance

- Best method depends on file access type
  - Contiguous great for sequential and random
- Linked good for sequential, not random
- Declare access type at creation → select either contiguous or linked
- Indexed more complex – multiple index block reads.

Free-Space Management

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



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

Free-Space Management

- Bit map requires extra space
  - Example:

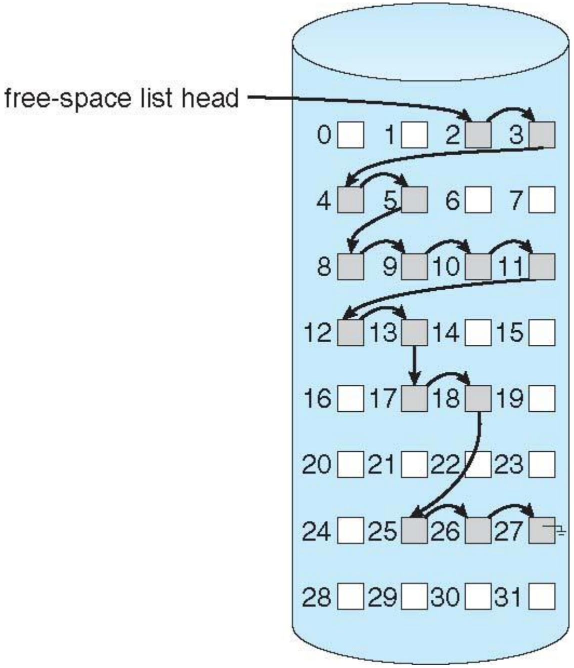
    block size = 4KB = 2^{12} bytes

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

    $\boldsymbol{n} = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)
- Easy to get contiguous files

Linked Free Space List on Disk

- **Linked list (free list)**
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



Free-Space Management

- **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

Free-Space Management

- **Space Maps**
  - 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

Efficiency and Performance

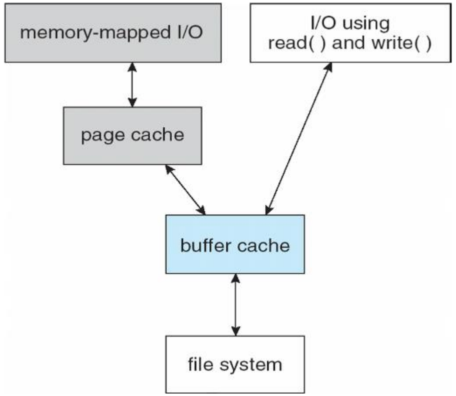
- Efficiency dependents on:
  - Disk allocation and directory algorithms
  - Types of data kept in file's directory entry
  - Pre-allocation or as-needed allocation of metadata structures
  - Fixed-size or varying-size data structures

Efficiency and Performance

- 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 done on disk directly
    - **Asynchronous** writes more common, buffered, faster

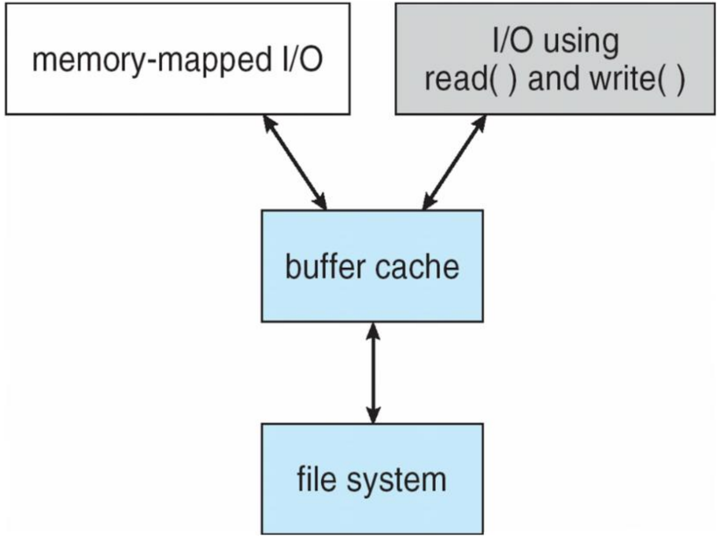
Page Cache and Buffer Cache

- 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



Unified Buffer Cache

- 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**



Recovery

- **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

Log Structured File Systems

- **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
  - The transactions in the log are asynchronously written to the file system structures
- 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

Thank you

johnjose@iitg.ac.in
http://www.iitg.ac.in/johnjose/