# Operating Systems
# [ 14A. File-System Implementation ]

Chung-Wei Lin
cwlin@csie.ntu.edu.tw
CSIE Department
National Taiwan University

From Operating System Concepts (10th Edition)

# Objectives

- Describe the details of implementing local file systems and directory structures
- Discuss block allocation and free-block algorithms and trade-offs
- Explore file system efficiency and performance issues
- Look at recovery from file system failures
- Describe the WAFL file system as a concrete example

# Outline

- **File-System Structure**
- File-System Operations
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- Example: WAFL File System

# File-System Structure

- To improve I/O efficiency, I/O transfers between memory and mass storage are performed in units of **blocks**
  - Each block on a hard disk drive has one or more sectors
  - Depending on the disk drive, sector size is usually 512 or 4,096 bytes
- **File systems** provide access to the storage device by allowing data to be stored, located, and retrieved easily
  - Define a file and its attributes, the operations allowed on a file, and the directory structure for organizing files
    - How the file system should look to the user
  - Create algorithms and data structures to map the logical file system onto the physical secondary-storage devices

# Layered File Systems (1/3)

| |
|---|
| Application Programs |
| Logical File System |
| File-Organization Module |
| Basic File System |
| I/O control |
| Devices |

- Devices
- I/O control
  - Device drivers and interrupt handlers to transfer information between the main memory and the disk system
  - A device driver, as a translator, usually writes specific bit patterns to special locations in the I/O controller's memory
    - Its input consists of high-level commands, such as "retrieve block 123"
    - Its output consists of low-level, hardware-specific instructions that are used by the hardware controller

# Layered File Systems (2/3)

| |
|---|
| Application Programs |
| Logical File System |
| File-Organization Module |
| Basic File System |
| I/O control |
| Devices |

- **Basic file system (Linux block I/O subsystem)**
  - Need only to issue generic commands to the appropriate device driver to read and write blocks on the storage device
    - Issue commands to the drive based on logical block addresses
  - Also manage the memory buffers and caches that hold various filesystem, directory, and data blocks
    - A block in the buffer is allocated before the transfer of a mass storage block can occur
    - When the buffer is full, the buffer manager must find more buffer memory or free up buffer space to allow a requested I/O to complete
    - Caches are used to hold frequently used file-system metadata to improve performance

| |
|---|
| Application Programs |
| Logical File System |
| File-Organization Module |
| Basic File System |
| I/O control |
| Devices |

# Layered File Systems (3/3)

- File-organization module
  - Know about files and their logical blocks
    - Each file's logical blocks are numbered from 0 (or 1) through N
  - Also include the free-space manager
    - Track unallocated blocks
    - Provide these blocks to the file-organization module when requested
- Logical file system
  - Manage metadata which includes all of the file-system structure except the actual data (or contents of the files)
  - Manage the directory structure
  - Maintain file structure via **file-control blocks (FCBs)** (**inodes** in UNIX)
    - Information about the file, including ownership, permissions, and location of the file contents
- Application programs

# Typical File-Control Block

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

# Advantage and Disadvantage

- Advantage of layered file systems
  - Duplication of code is minimized
    - The I/O control and sometimes the basic file-system code can be used by multiple file systems
    - Each file system can then have its own logical file-system and file-organization modules
- Disadvantage of layered file systems
  - More operating-system overhead, which may result in decreased performance
- A major challenge in designing new systems
  - The use of layering, including the decision about how many layers to use and what each layer should do

# File Systems in Use

- Most CD-ROMs are written in the ISO 9660 format
- UNIX uses the UNIX file system (UFS), which is based on the Berkeley Fast File System (FFS)
- Windows supports disk file-system formats of FAT, FAT32, and NTFS, as well as CD-ROM and DVD file-system formats
- The standard Linux file system is known as the **extended file system**, with the most common versions being ext3 and ext4
  - Although Linux supports over 130 different file systems
- More are coming
  - ZFS, GoogleFS, Oracle ASM, FUSE

# Outline

- File-System Structure
- **File-System Operations**
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- Example: WAFL File System

# Structures on File Systems (1/2)

- A **boot control block** (per volume) contains information needed to boot an operating system from that volume
  - If the disk does not contain an operating system, this block can be empty
  - It is typically the first block of a volume
  - **Boot block** in UFS and **partition boot sector** in NTFS
- A **volume control block** (per volume) contains volume details
  - The number of blocks in the volume, the size of the blocks, a free-block count and free-block pointers, and a free-FCB count and FCB pointers
  - **Superblock** in UFS and stored **master file table** in NTFS

# Structures on File Systems (2/2)

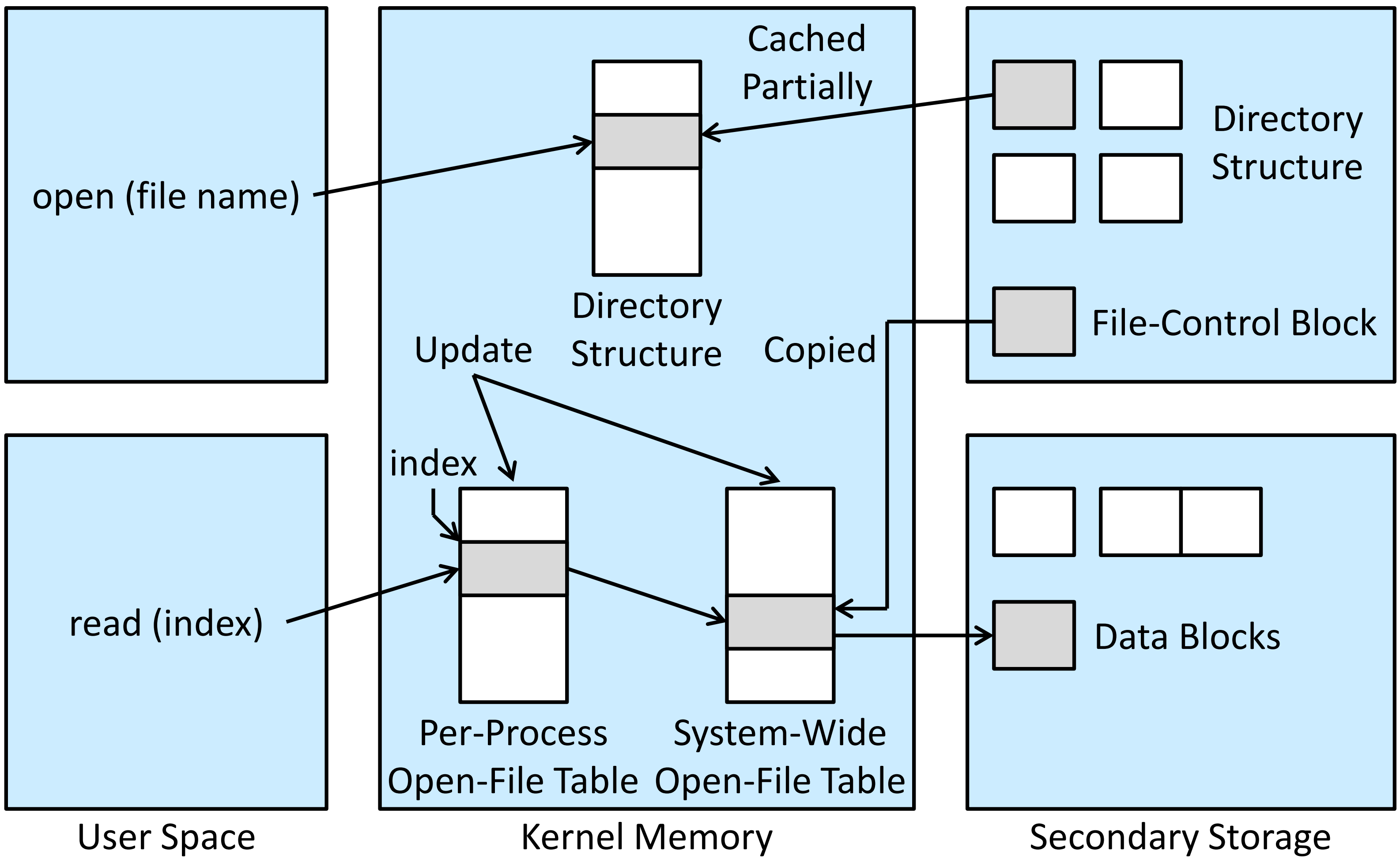
- A directory structure (per file system) organizes the files
  - In UFS, this includes file names and associated inode numbers
  - In NTFS, it is stored in the master file table
- A per-file FCB contains many details about the file
  - It has a unique identifier number to allow association with a directory entry
  - In NTFS, this information is actually stored within the master file table, which uses a relational database structure, with a row per file

# In-Memory File-System Structures

- An in-memory **mount table** contains information about each mounted volume
- An in-memory directory-structure cache holds the directory information of recently accessed directories
- The **system-wide open-file table** contains a copy of the FCB of each open file
  - As well as other information
- The **per-process open-file table** contains pointers to the appropriate entries in the system-wide open-file table
  - As well as other information
- Buffers hold file-system blocks when they are being read from or written to a file system

# File Open and File Read

- `open()` returns a **file handle** (**descriptor**) for subsequent use



# Outline

- ❑ File-System Structure
- ❑ File-System Operations
- ❑ **Directory Implementation**
- ❑ Allocation Methods
- ❑ Free-Space Management
- ❑ Efficiency and Performance
- ❑ Recovery
- ❑ Example: WAFL File System

# Directory Implementation

- **Linear list** of file names with pointers to the data blocks
  - Simple to program
  - Time-consuming (linear search) to execute
    - A sorted list allows a binary search and decreases the average search time
- **Hash table**: linear list with a hash data structure
  - Decrease the directory search time
  - Need some provision for **collisions**
    - Situations where two file names hash to the same location
  - Have difficulties with its generally fixed size and the dependence of the hash function on that size
    - Alternatively, we can use a chained-overflow hash table

# Outline

- File-System Structure
- File-System Operations
- Directory Implementation
- **Allocation Methods**
  - Contiguous Allocation
  - Linked Allocation
  - Indexed Allocation
  - Performance
- Free-Space Management
- Efficiency and Performance
- Recovery
- Example: WAFL File System

# Allocation Methods

- How to allocate space to these files so that storage space is utilized effectively and files can be accessed quickly

# Outline

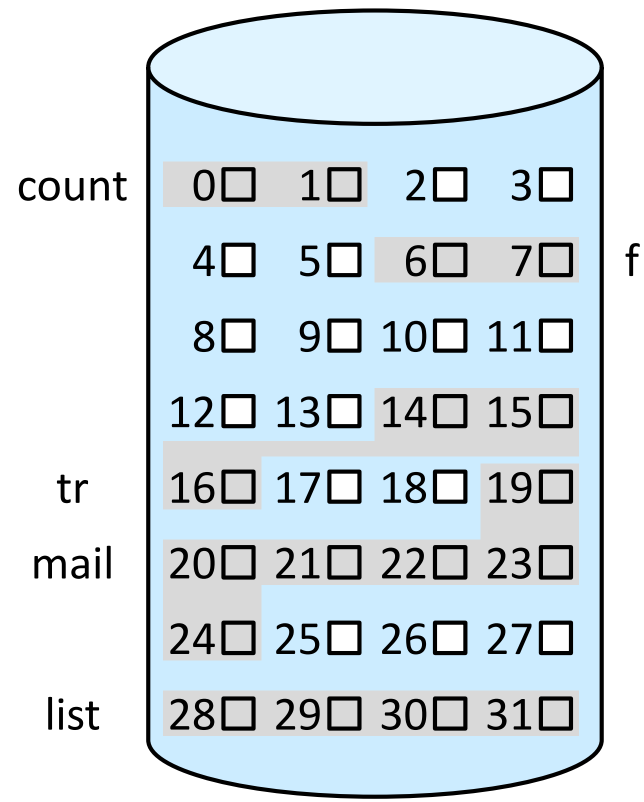
- File-System Structure
- File-System Operations
- Directory Implementation
- **Allocation Methods**
  - **Contiguous Allocation**
  - Linked Allocation
  - Indexed Allocation
  - Performance
- Free-Space Management
- Efficiency and Performance
- Recovery
- Example: WAFL File System

# Contiguous Allocation

- Each file occupy a set of contiguous blocks on the device
  - Easy implementation
  - External fragmentation
    - Dynamic storage-allocation problem (Section 9.2)
    - Free-space management (Section 14.5)
  - One strategy: copying an entire file system onto another device and then coping back
    - Compaction
    - Off-line (during down time)
    - On-line

Directory

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



# Extent-Based Systems

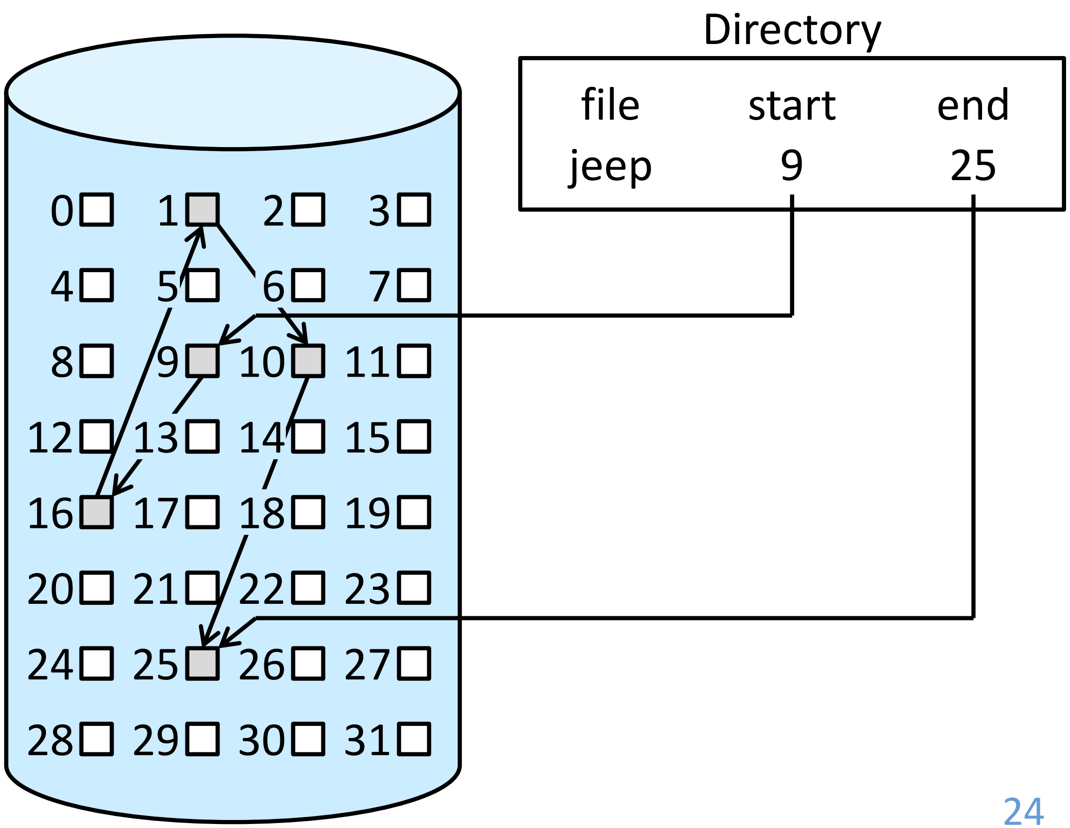
- How much space is needed for a file?
- How to make the file larger in place?
  - Terminate the program, allocate more space, and run the program again
    - These repeated runs may be costly
    - To prevent them, the user will normally overestimate the amount of space needed
  - Find a larger hole, copy the contents of the file to the new space, and release the previous space
- If that amount proves not to be large enough, another chunk of contiguous space, known as an **extent**, is added
  - The location of a file's blocks is then recorded as a location and a block count, plus a link to the first block of the next extent

# Outline

- File-System Structure
- File-System Operations
- Directory Implementation
- **Allocation Methods**
  - Contiguous Allocation
  - **Linked Allocation**
  - Indexed Allocation
  - Performance
- Free-Space Management
- Efficiency and Performance
- Recovery
- Example: WAFL File System

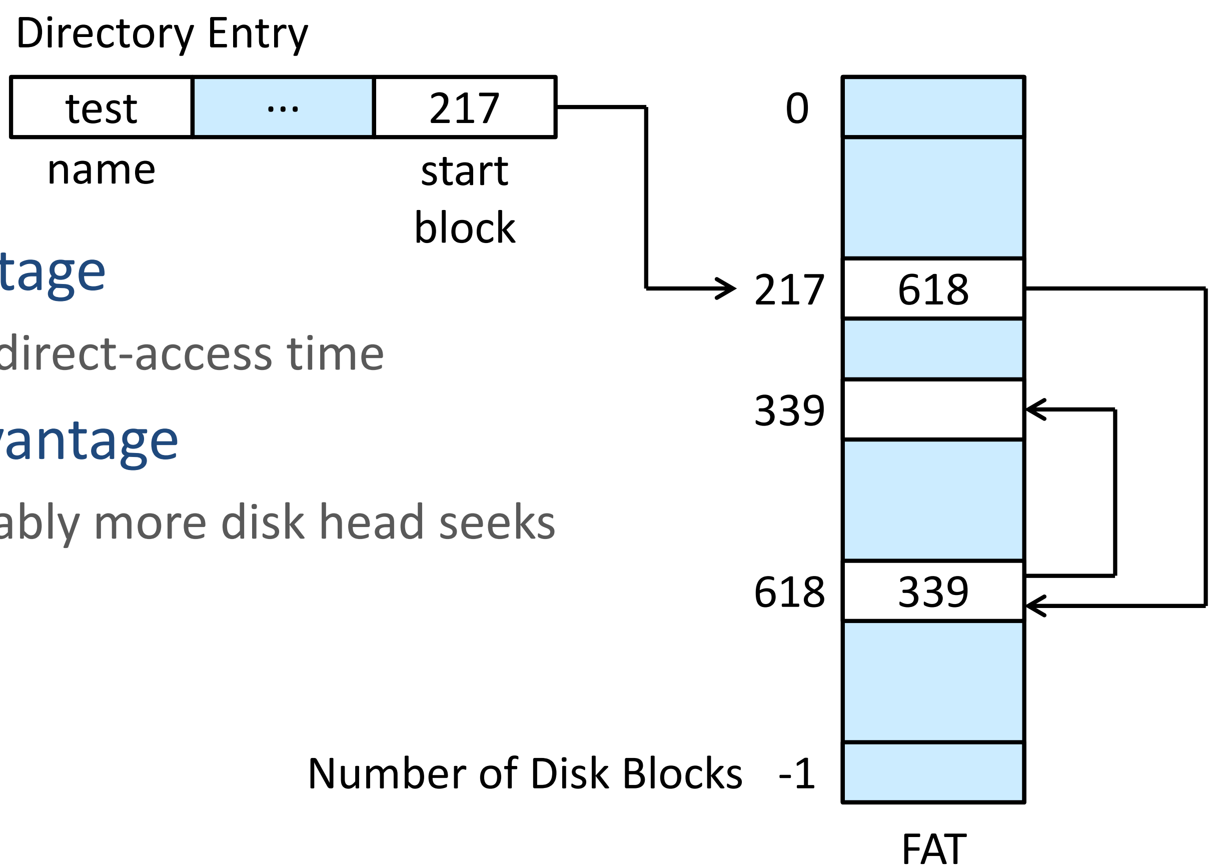
# Linked Allocation

- Each file is a linked list of storage blocks which may be scattered anywhere on the device
  - The directory contains a pointer to the first and last blocks of the file
  - Each block contains a pointer to next block
- Advantages
  - No external fragmentation
  - No compaction needed
- Disadvantages
  - Inefficiency of finding the i-th block of a file
  - More space for pointers
    - Solution: clusters
  - Reliability



# File-Allocation Table (FAT)

- A section of storage at the beginning of each volume is set aside to contain the table
  - The table has one entry for each block and is indexed by block number



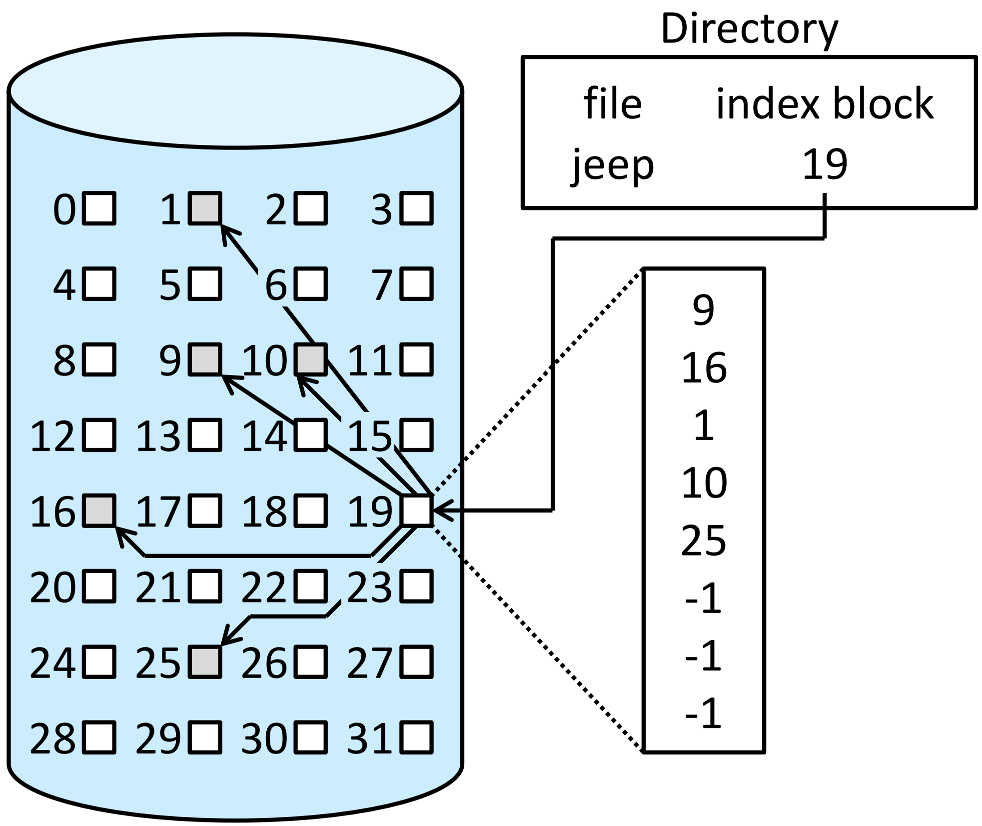

- Advantage
  - Less direct-access time
- Disadvantage
  - Probably more disk head seeks

# Outline

- File-System Structure
- File-System Operations
- Directory Implementation
- **Allocation Methods**
  - Contiguous Allocation
  - Linked Allocation
  - **Indexed Allocation**
  - Performance
- Free-Space Management
- Efficiency and Performance
- Recovery
- Example: WAFL File System

# Indexed Allocation

- Bring all the pointers together into the **index block**
  - In the absence of a FAT, linked allocation cannot support efficient direct access
- Advantages
  - No external fragmentation
  - No compaction needed
- Disadvantage
  - More space than linked allocation

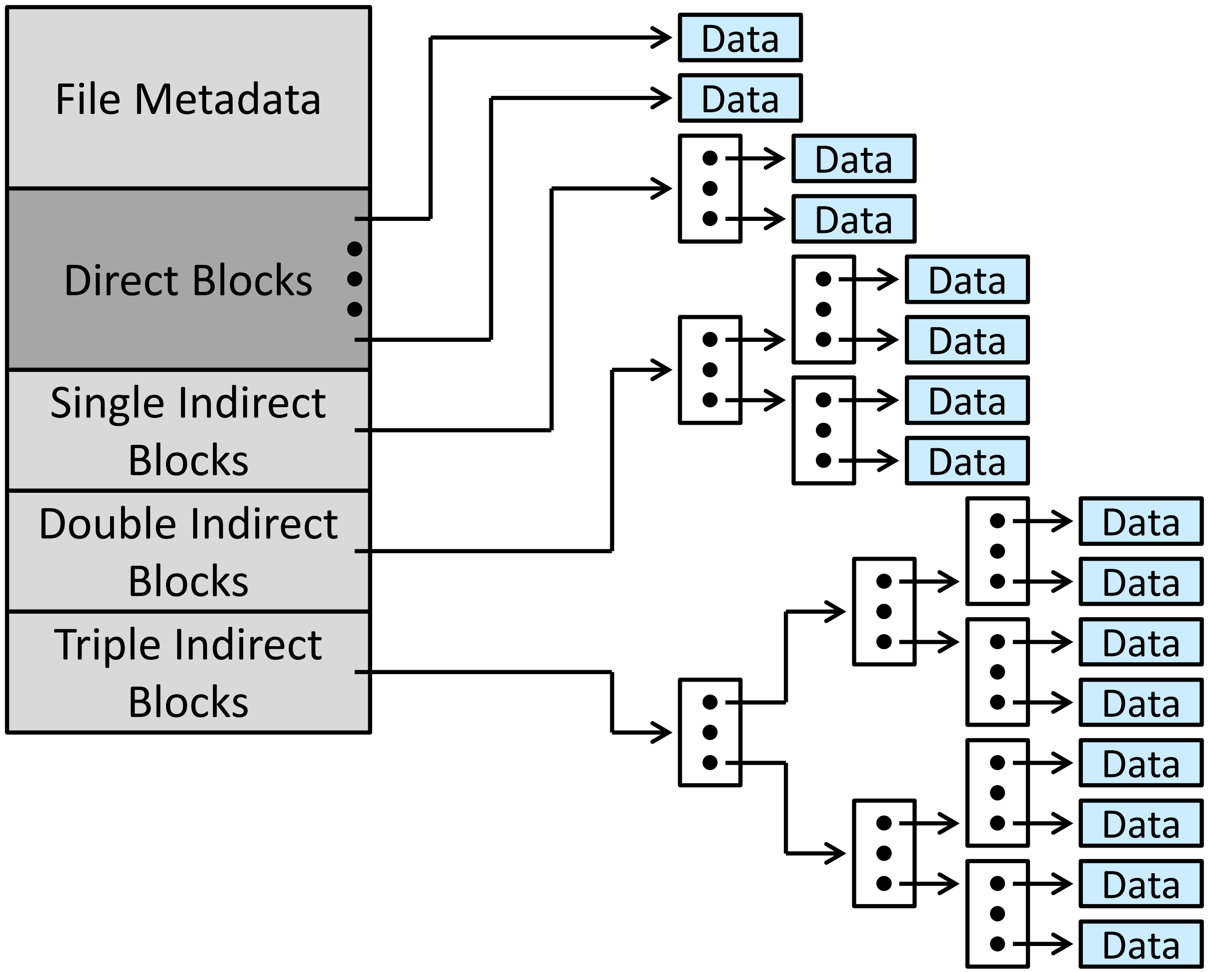


# Schemes of Index Blocks (1/2)

- **Linked scheme**
  - An index block is normally one storage block
  - To allow for large files, we can link together several index blocks
- **Multilevel index**
  - A first-level index block points to a set of second-level index blocks, which in turn point to the file blocks
  - This approach could be continued to a third or fourth level, depending on the desired maximum file size
  - Example
    - With 4,096-byte blocks, we could store 1,024 four-byte pointers in an index block
    - Two levels of indexes allow 1,048,576 data blocks and a file size of up to 4 GB

# Schemes of Index Blocks (2/2)

- Combined scheme (in UNIX-based file systems)



# Outline

- File-System Structure
- File-System Operations
- Directory Implementation
- **<u>Allocation Methods</u>**
  - Contiguous Allocation
  - Linked Allocation
  - Indexed Allocation
  - **<u>Performance</u>**
- Free-Space Management
- Efficiency and Performance
- Recovery
- Example: WAFL File System

# Performance

- Goals: storage efficiency and data-block access times
- A system with mostly sequential access should not use the same method as a system with mostly direct (random) access
  - Contiguous allocation requires only one access to get a block, for any type of access
  - Linked allocation should not be used for direct access
- Indexed allocation is more complex
  - Depend on the index structure, the file size, and the block position
- Mix of contiguous, linked, and indexed allocations
- For NVM, different algorithms and optimizations are needed
  - Reduce the instruction count and overall path between the storage device and application access to the data

# Outline

- ❑ File-System Structure
- ❑ File-System Operations
- ❑ Directory Implementation
- ❑ Allocation Methods
- ❑ **Free-Space Management**
- ❑ Efficiency and Performance
- ❑ Recovery
- ❑ Example: WAFL File System

# Q&A