Distributed File Systems

CS432: Distributed Systems

Spring 2015

Reading

- Chapter 12.1, 21 [Coulouris ’11]
- Chapter 11 [Tanenbaum ‘06]

- Section 4.3, “Modern Operating Systems, Fourth Ed.”, Andrew S. Tanenbaum
- Section 11.4, “Operating Systems Concept, Ninth Ed.”, Abraham Silberschatz, et al.

Objectives

- Learn about the following:
  - Review file systems and the main requirements for designing a distributed file system.
  - Famous architecture models of distributed file systems.
- Study the design of three file systems NFS, AFS, and GFS.

Outline

- Introduction
  - Non-Distributed File System (Review)
  - File System Mounting
- Distributed File System Requirements
- File Service Architecture
- Case Studies:
  - Sun Network File System (NFS)
  - Andrew File System (AFS)
  - Google File System (GFS)

File Systems

- File systems, in centralized computer systems, provide a convenient programming interface to disk storage.
  - blocks of disks ➔ files, directories, ..
  - storage allocation and layout.
- Components:
  - Disk management: gathering disk blocks into files.
  - Naming: help users find files by their name instead of block identifiers.
  - Security: layers of permissions to access and modify files.
  - Durability: data written to files should not be tampered with in case of failures.

File Components

- A file contains:
  - data: sequence of data items that are accessible through read and write operations.
  - attributes: a single record containing information about the file.

Managed by the file system.
Users do not typically update them.

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list

Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

Reading and Writing

- Reading from the file system (e.g. getc()):
  - Fetch a block containing the required character.
  - Return the requested character from the block.
- Writing to the file system (e.g. putc()):
  - Modify existing data: fetch block, modify, and write.
  - Append data: buffer data until a block size is completed, then write.

Non-distributed File System Modules

- Each module depends only on the layers below it.
- Note: the implementation of a distributed file service also requires additional components to deal with: client-server communication, distributed naming, and location of files.

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

Unix File System Operations

filedes = open(name, mode) *filedes = creat(name, mode)*	Opens an existing file with the given *name*. Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The *mode* is *read*, *write* or both.
status = close(filedes)	Closes the open file *filedes*.
count = read(filedes, buffer, n) *count = write(filedes, buffer, n)*	Transfers *n* bytes from the file referenced by *filedes* to *buffer*. Transfers *n* bytes to the file referenced by *filedes* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
pos = lseek(filedes, offset, whence)	Moves the read-write pointer to offset (relative or absolute, depending on *whence*).
status = unlink(name)	Removes the file *name* from the directory structure. If the file has no other names, it is deleted.
status = link(name1, name2)	Adds a new name (*name2*) for a file (*name1*).
status = stat(name, buffer)	Gets the file attributes for file *name* into *buffer*.

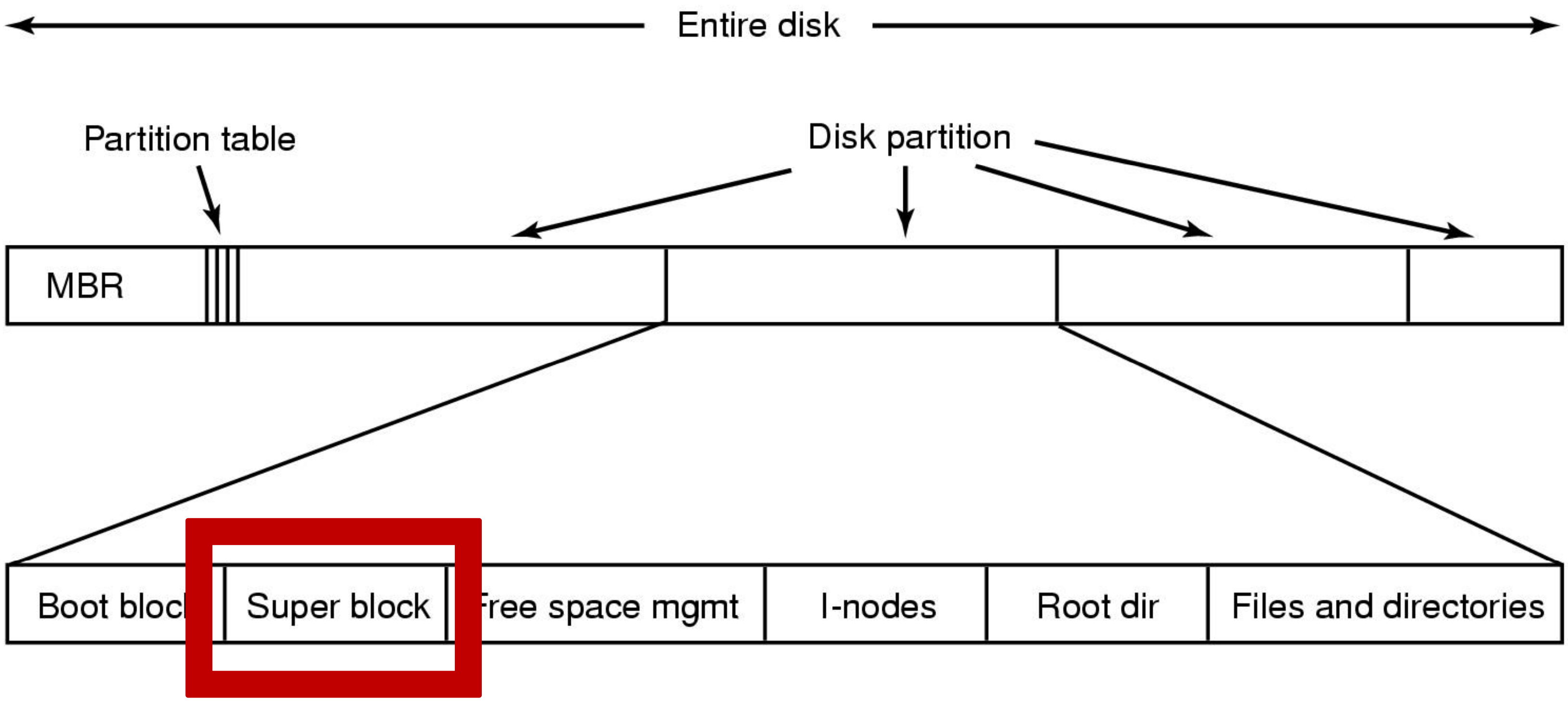
File System Layout

- File systems are stored on disk.
- Disks are divided into one or more partitions, independent file system on each partition.
- Master Boot Record (MBR)
  - Sector 0 of the disk.
  - Used to boot the computer.
  - The end of the MBR contains the partition table.

System Booting

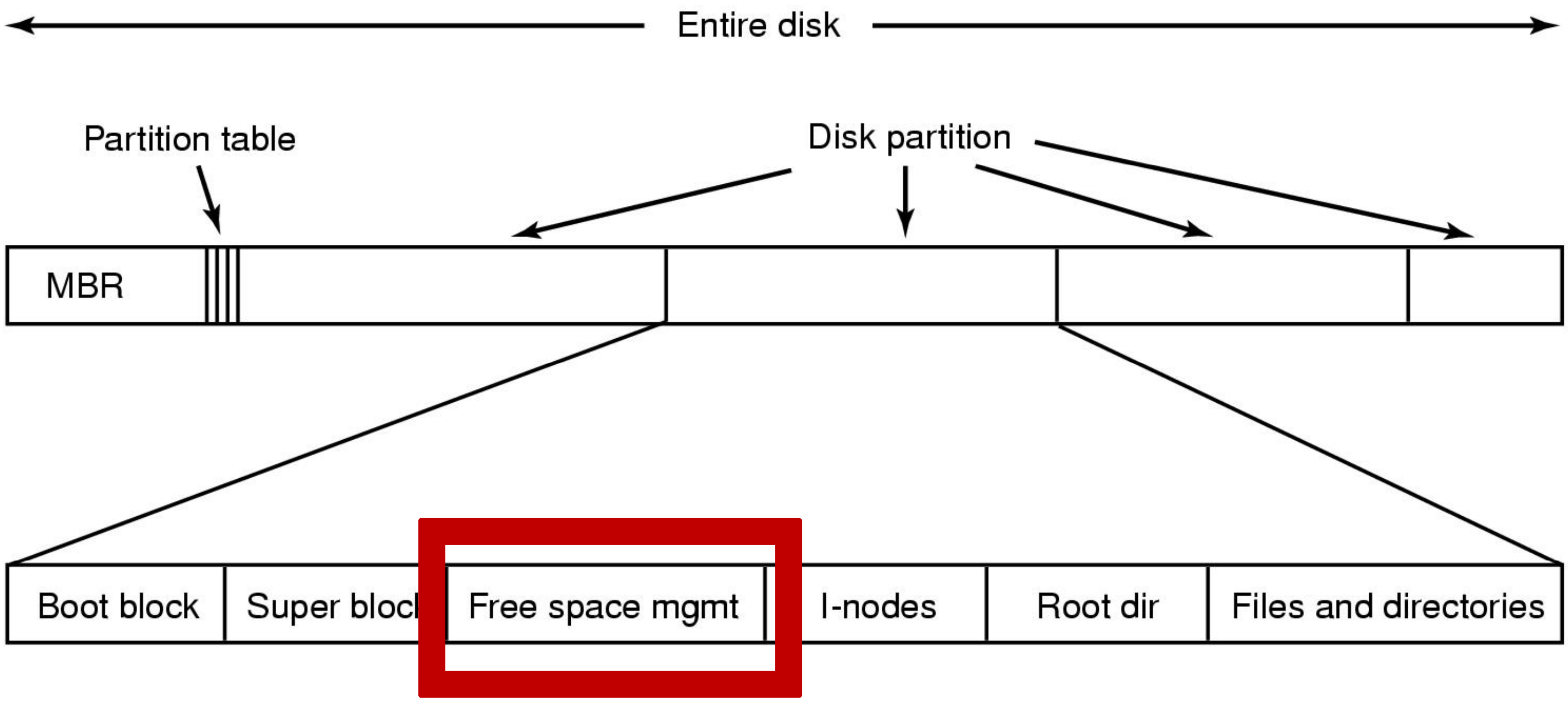
- The partition table gives the starting and ending addresses of each partition.
- One of the partitions in the table is marked as active.
- When the computer is booted, the BIOS reads it and executes the MBR.
- The MBR program locates the active partition, read in its first block (*boot block*) and executes it.
- The program in the boot block loads the operating system contained in that partition.
- Note: each partition starts with a boot block, even if it does not contain a bootable operating system.

Example: File System Layout



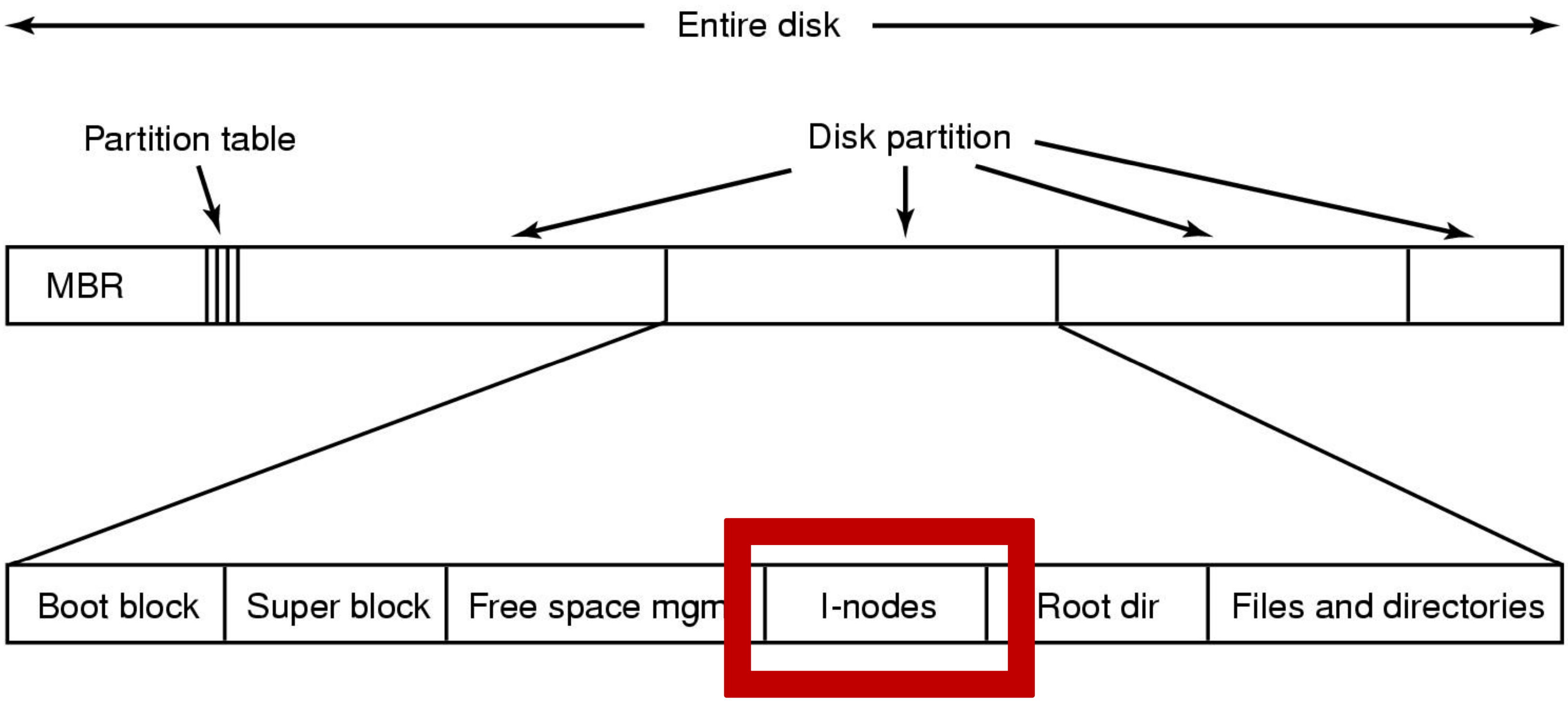
- Super block contains all the key parameters about the file system
- Read into memory when the computer is booted or the file system is first touched.
- Typical information: magic number to identify the file system type, the number of blocks in the file system, and other key administrative information.

Example: File System Layout



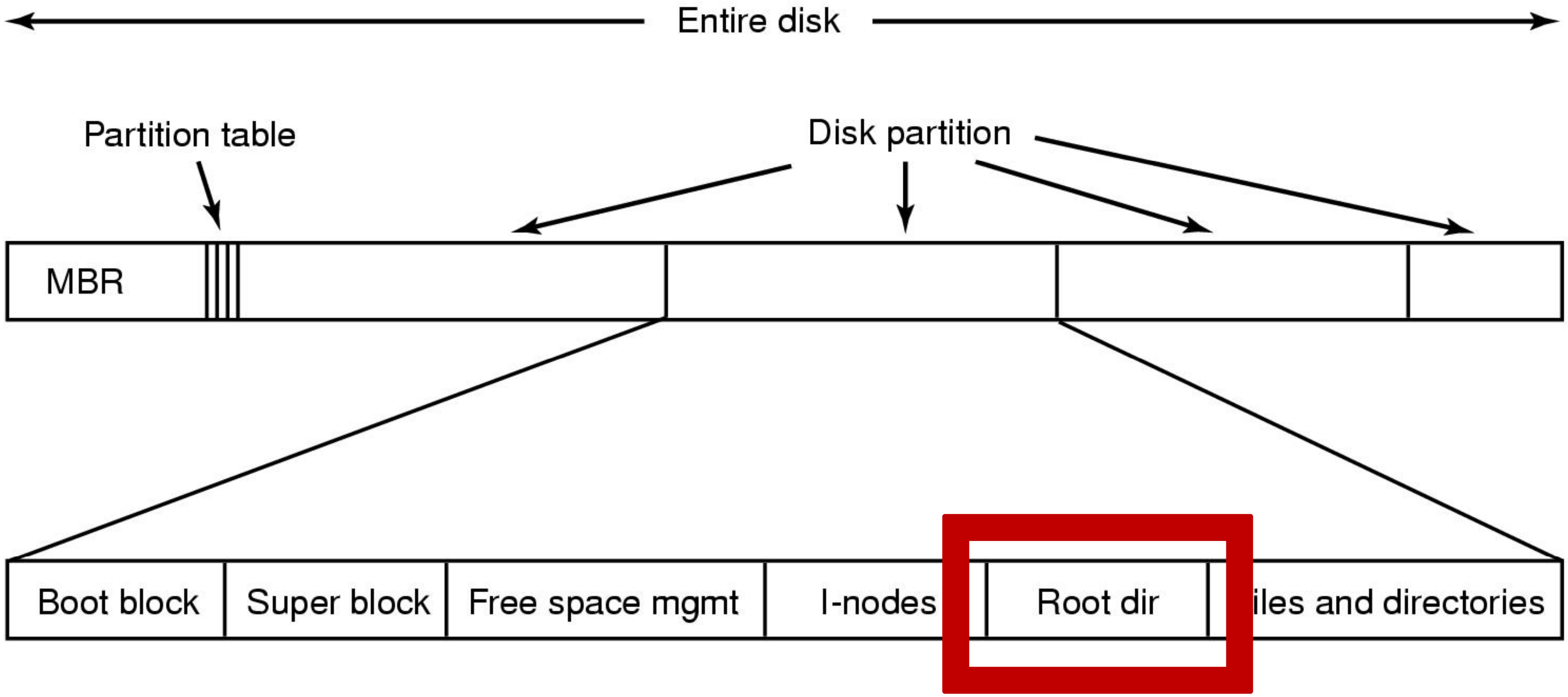
- Information about the free blocks in the file system.
- Example: bitmap, file pointers.

Example: File System Layout



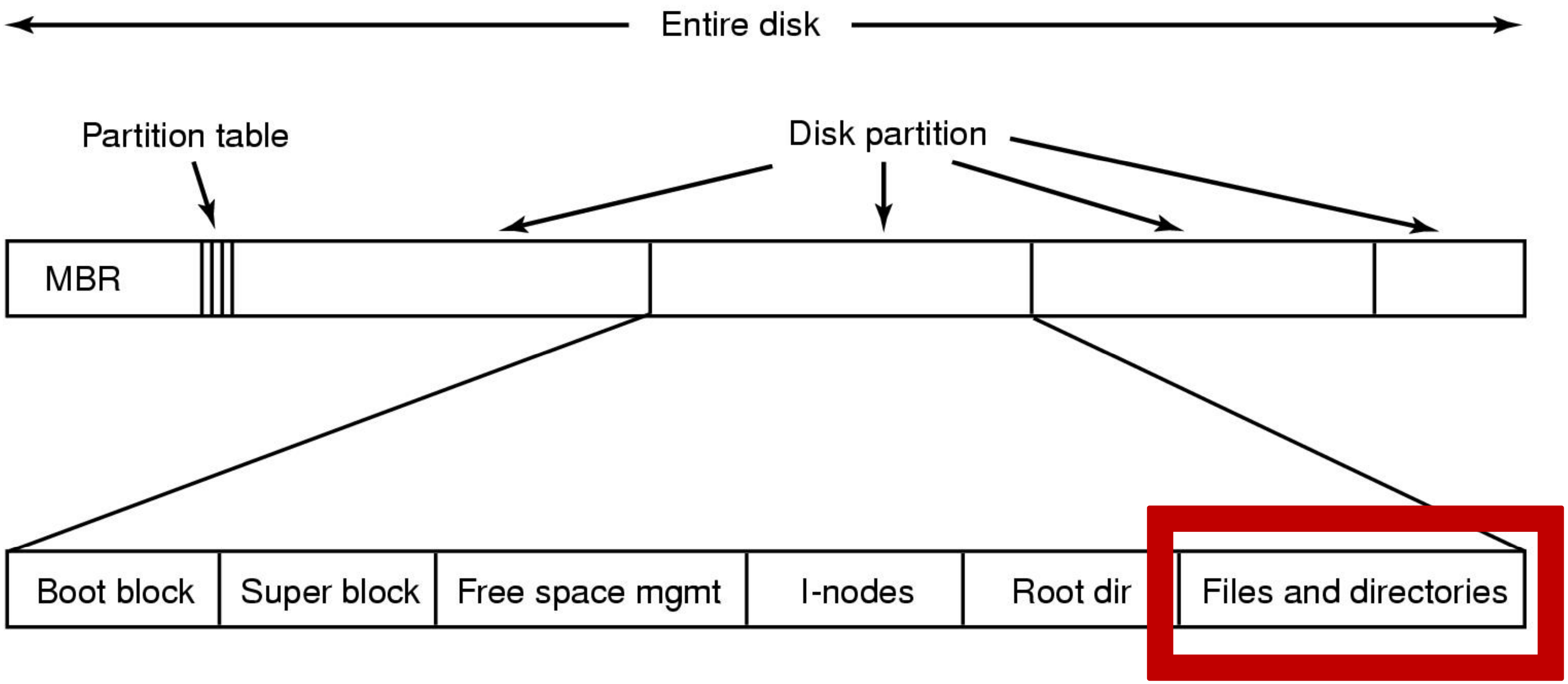
- I-node: a data structure used to represent information about a file system object (file, directory).

Example: File System Layout



- The top of the file-system tree.

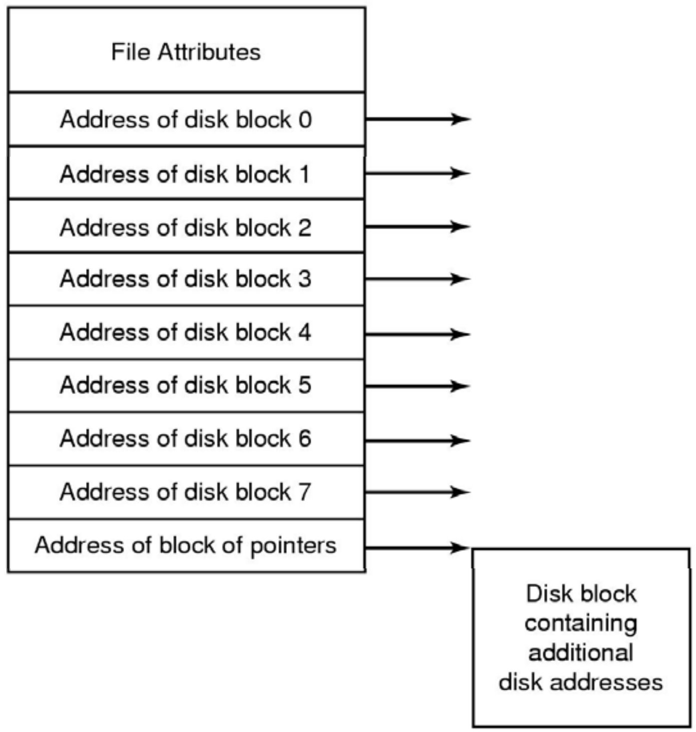
Example: File System Layout



- Directories and files contained in this partition.

Implementing Files
I-nodes

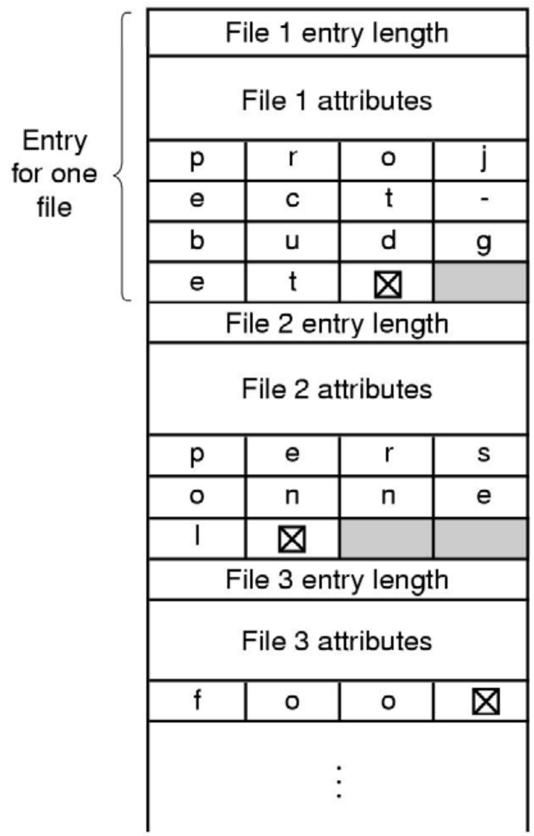
- Associate with each file a data structure called an *i-node (index-node)*, which lists the attributes and disk addresses of the file's blocks.
- The i-node need to be in memory only when the corresponding file is open.



Implementing Directories

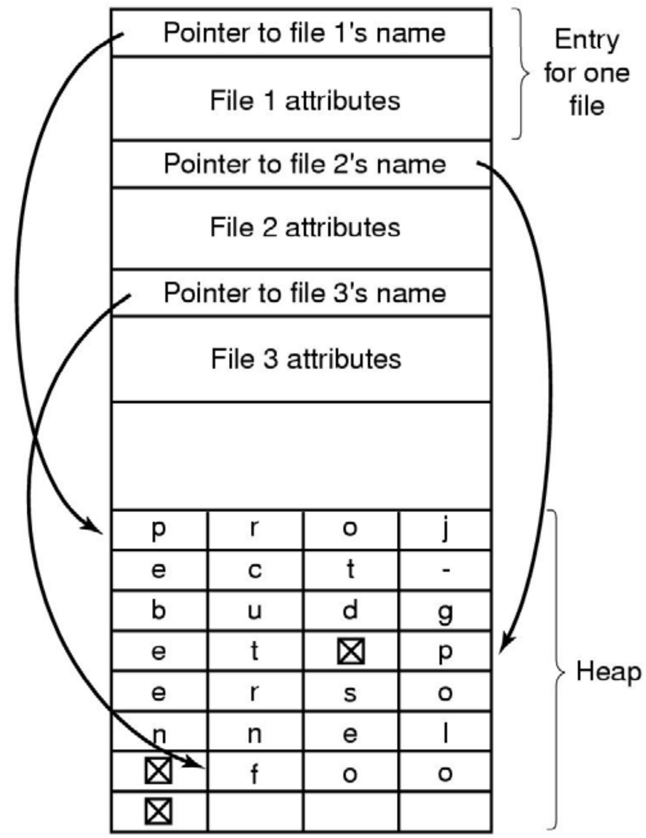
- The main function of the directory system is to map the ASCII name of the file onto the information needed to locate the data.
- The directory entry provides the information needed to find the disk blocks.
  - Number of the I-node.
- Storing files attributes:
  - Directly in the directory entry.
  - Store the attributes in the i-node. The directory entry can be a file name and i-node number

Implementing Directories: File Names



(a)

In-line



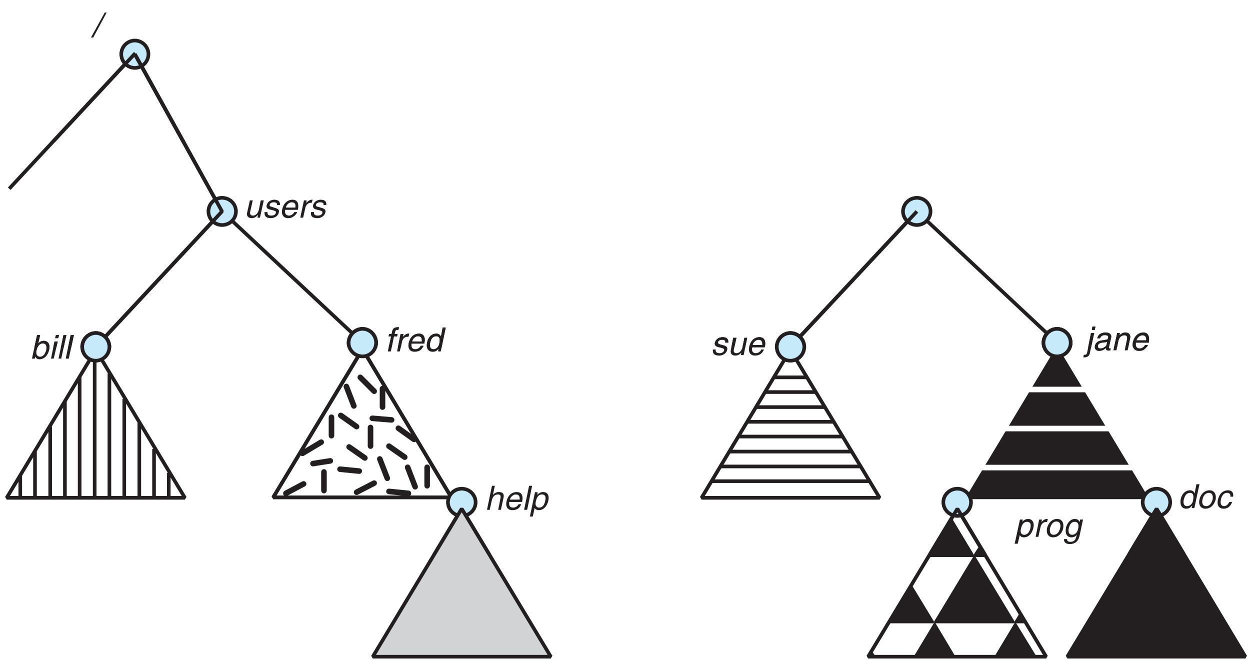
(b)

In a heap

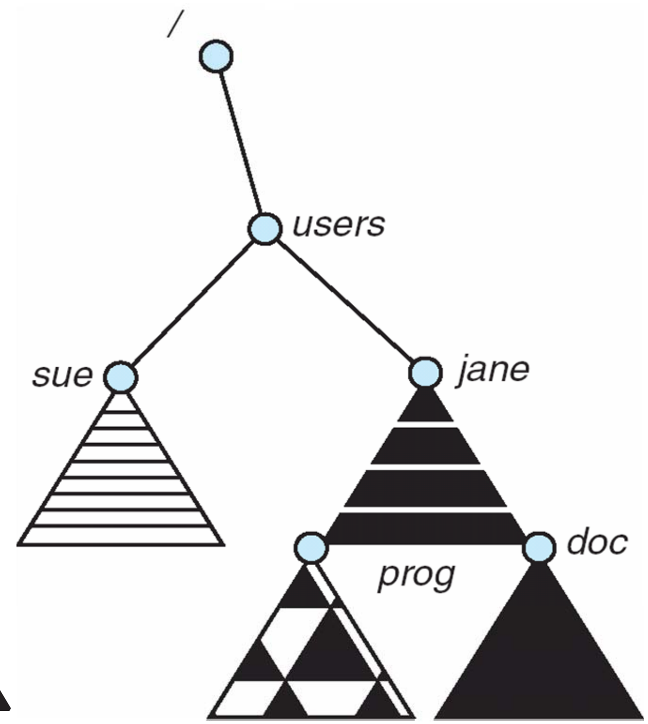
File System Mounting

- Just as a file must be opened before it is used, a file system must be mounted before it can be available to processes on the system.
- Mount point: the location within the file structure where the file system is to be attached.
  - Usually and empty directory.
- Mounting procedure: OS is given the name of the device and the mount point, once mounted, it will be able to traverse its directory structure
- Example, mounting home directories in unix.

Mount Point Example



(a) (b)



Existing file system Unmounted volume After mounting

Outline

- Introduction.
- **Distributed File System Requirements.**
  - Transparency.
  - Concurrent File Updates.
  - File Replication.
  - Hardware and Operating System Heterogeneity.
  - Fault Tolerance.
  - Consistency.
  - Security.
  - Efficiency.
- File Service Architecture.
- Case Studies.

Distributed File Systems

- Distributed file systems allow multiple processes to share data over long periods of time in a secure and reliable way.
- A well designed file service provides access to files stored at a server with performance and reliability similar to, and in some cases better than, files stored on local disks.
- A file service enables programs to store and access remote files exactly as they do local ones, allowing users to access their files from any computer in an intranet.
- A client-server architecture is typically used.

File System vs. Distributed File System

	File System	Distributed File System
Sharing	✗	✓
Persistence	✓	✓
Distributed cache/ replicas	✗	✓
Consistency	Strict-one-copy	Weak guarantees

Transparency

- **Access transparency**: Client programs should be unaware of the distribution of files.
- **Location transparency**: Client programs should see a uniform file name space.
- **Mobility transparency**: Neither client programs nor system administration tables in client nodes need to be changed when files are moved.
- **Performance transparency**: Client programs should continue to perform satisfactorily while the load on the service varies within a specified range.
- **Scaling transparency**: The service can be expanded by incremental growth to deal with a wide range of loads and network sizes.

Concurrent File Updates

- AKA **Concurrency Control**.
- Changes to a file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file.
- Levels of locking are required.
- Techniques that provide concurrency control have high costs.

File Replication

- Several copies of the same file at different locations.
- Advantages:
  - Scalability of a service: multiple servers share the load of providing a service to clients accessing the same set of files.
  - Fault tolerance: clients are able to locate another server that holds a copy of the file when one has failed.
- Caching files (fully or partially) at clients can be considered as a limited form of replication.

Hardware and Operating System Heterogeneity

- Services allowing file access are accessible from different operating systems and computers.
- File system server can be deployed on any operating systems or hardware.

Fault Tolerance

- The file service continues to operate in the face of communication and server failures.
- Coping with communication failures:
  - At-most-once invocation semantics.
  - At-least-once invocation semantics with a server protocol designed in terms of *idempotent* operations. This semantic ensures that duplicated requests do not result in invalid updates to files.
- Stateless servers: servers can be restarted and the service restored after a failure without needing to recover previous state.
- File replication is required.

Consistency

- one-copy update semantics (e.g. Unix):
  - all of the processes accessing or updating a given file see identical contents as if only a single copy of the file existed.
  - when files are replicated or cached at different sites, modifications are propagated to all of the other sites that hold copies.

Security

- Access control mechanism:
  – uses access control lists.
- Authentication:
  – access control at the server is based on correct user identities.
- Encryption can be used to protect the contents of request and reply messages.

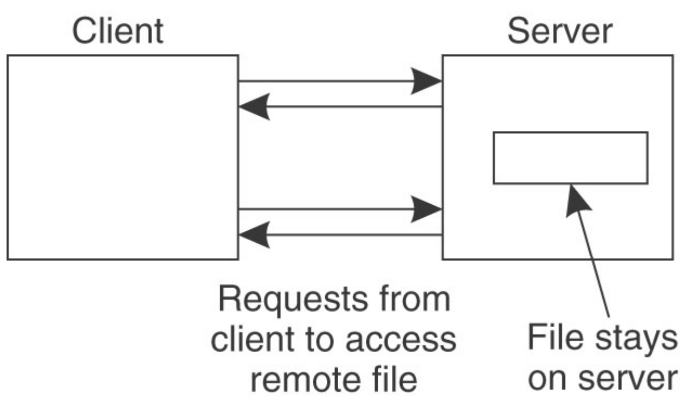
Efficiency

- A distributed file service:
  - offer facilities that are of at least the same power and generality as those found in conventional file systems
  - achieve a comparable level of performance.
- Trade-off:
  - Scalability, reliability, availability, …
  - Latency because of accessing remote files.

Outline

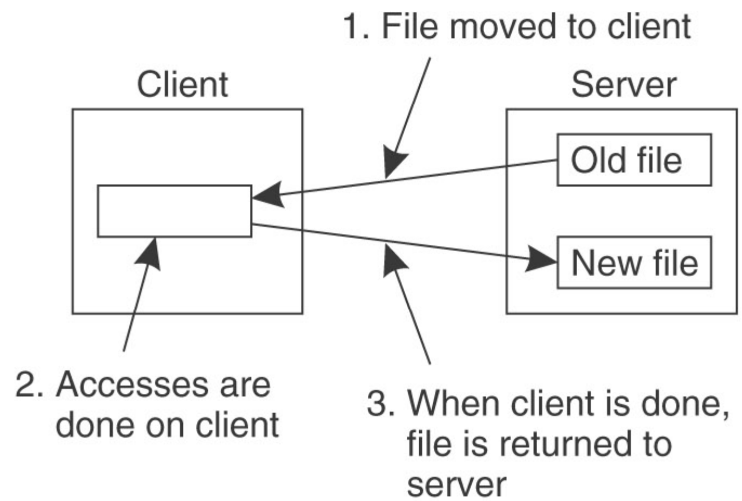
- Introduction.
- Distributed File System Requirements.
- **File Service Architecture.**
- Case Studies:
  - Sun Network File System (NFS).
  - Andrew File System (AFS).
  - Google File System (GFS).

Distributed File System Access Models



(a)

The remote access model.

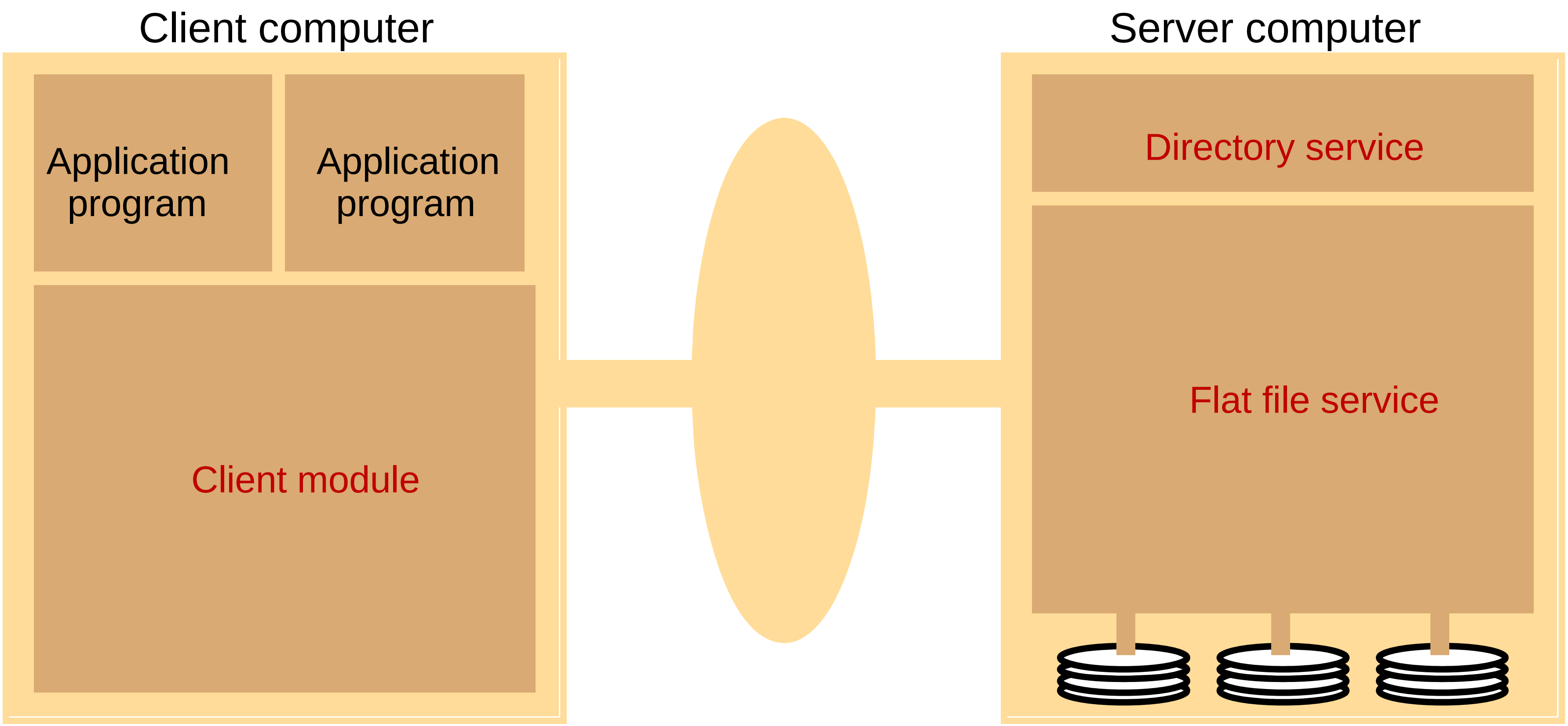


(b)

The upload/download model.

Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

File Service Architecture



Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

Flat File Service

- Implements operations on files.
- *Unique file identifiers (UFIDs)* are used to refer to files. A UFID uniquely identifies a file in a distributed file system.
- RPC interface provides a comprehensive set of operations for access to files.

Directory Service

- Provides a mapping between *text names* for files and their UFIDs.
- Provide the following services:
  - Generate directories.
  - Add new file names to directories.
  - Obtain UFIDs from directories.
- Can be considered as a client to the flat file service.

Client Module

- A client module runs at each client.
- Extends the operations of the flat file service and the directory service under a single application programming interface that is available to user-level programs in client computers.
- Holds information about the network locations of the flat file server and directory server processes.
- Can manage a cache of recently used file blocks at the client.

Thank You