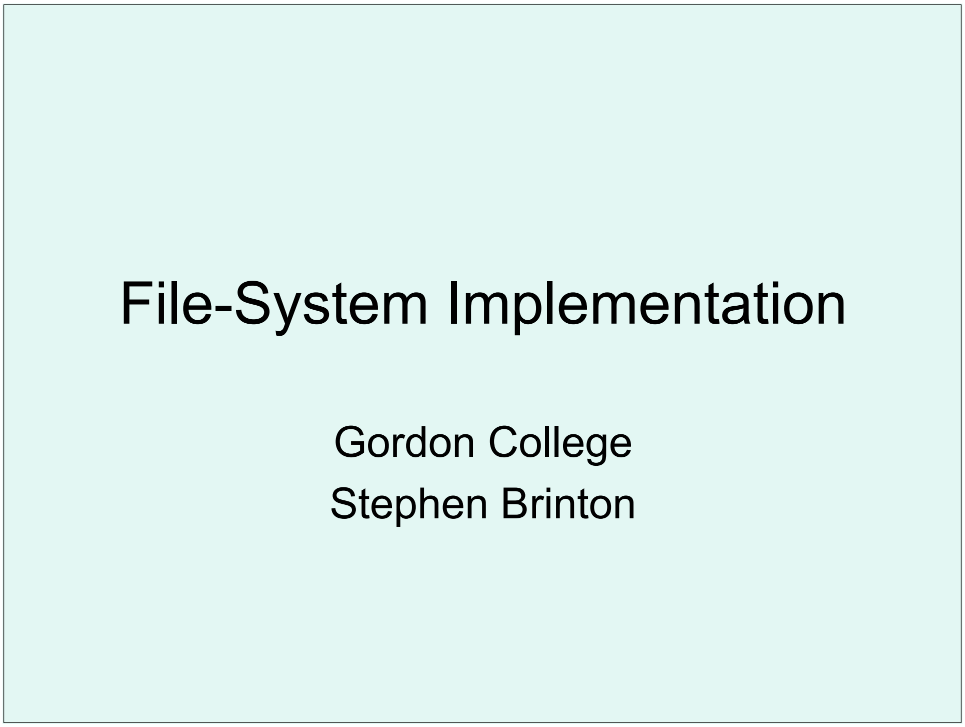
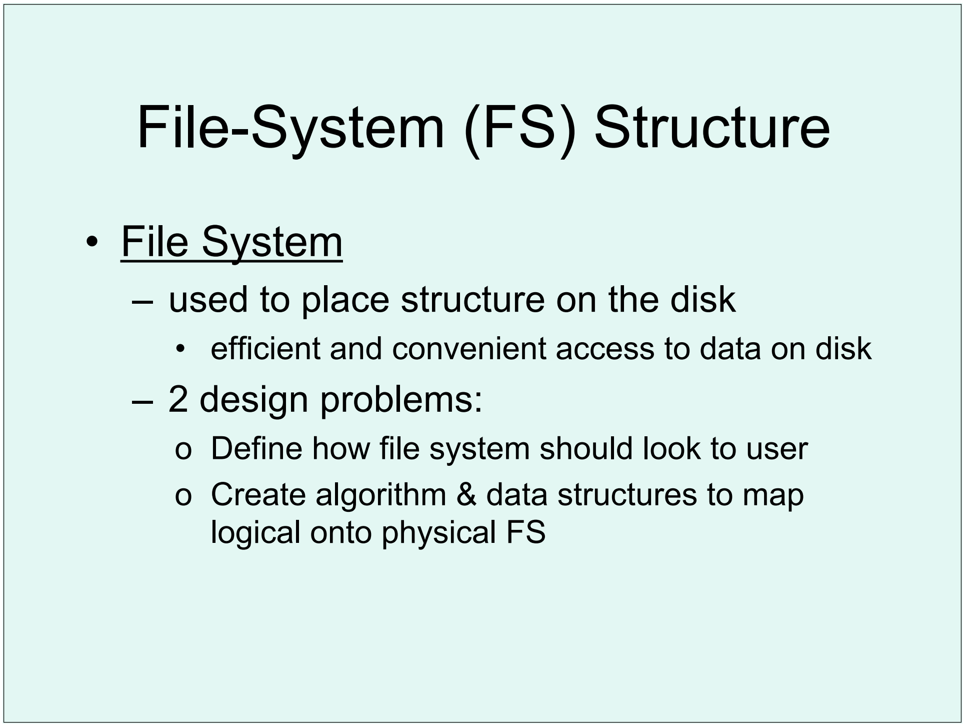


File-System Implementation

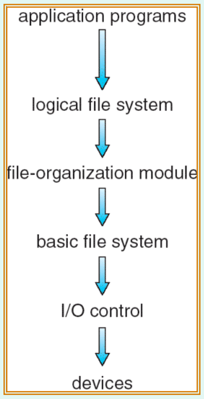
Gordon College
Stephen Brinton



File-System (FS) Structure

- File System
  - – used to place structure on the disk
    - • efficient and convenient access to data on disk
  - – 2 design problems:
    - o Define how file system should look to user
    - o Create algorithm & data structures to map logical onto physical FS

Layered File System



File system composed of many different layers.

I/O control – device drivers and interrupt handlers. Input is "high-level" commands like "retrieve block 123" and the output is low-level commands to the devices. (places values in registers and device controller's memory)

File-organization module – knows about the file allocation scheme. Therefore it translates a logical block into the addressing for a physical block. (free-space manager)

Logical file system – manages the metadata information. (file-control block)

Many different file systems used today – UFS, EXT, FAT, NTFS, etc.

What structures are needed on disk?

What info? How to boot the OS on disk, block total, # and location of free blocks, directory structure, and the files.

- Boot control block (per volume) – called boot block (UFS) and partition boot sector (NTFS). Typically first block of volume.
- Volume control block (per volume) – contains the volume or partition details. called superblock (UFS) and master file table (NTFS).
- Directory structure (per file system) – organize the files.
- File Control block (per file) – details about the file (including permissions, size, location of data blocks)

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

Typical file-control block

How about the in-memory structures?

1. mount table – info about each mounted volume

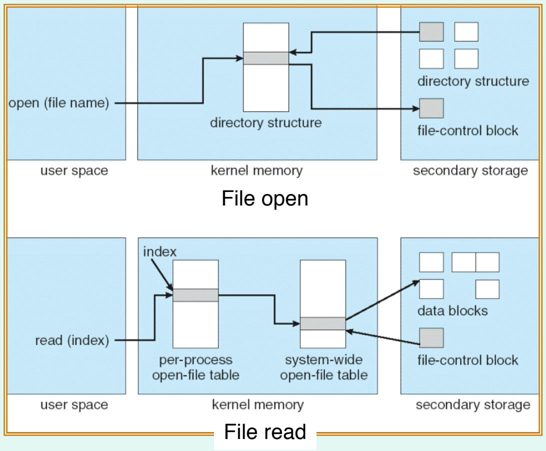
2. directory-structure cache – info about recently accessed directories

3. system-wide open-file table – copy of FCB of each open file

4. per-process open-file table – a pointer to the appropriate entry in the system-wide open-file table

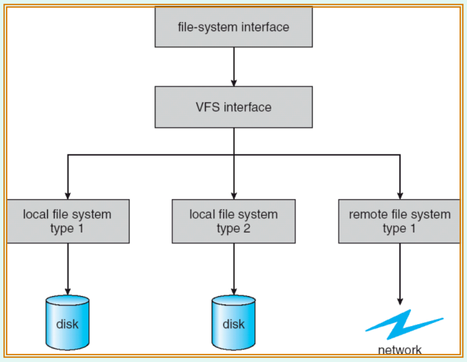
How about the in-memory structures?

- What happens when a file is opened or read?



How do you put multiple file systems on a system?

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is sent to the VFS interface, rather than any specific type of file system.



What is the best directory implementation?

- **Linear list** of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute (linear search)
  - better to use a 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
  - fixed size (depends on the hash function for a particular size)
    - Solution – use chaining

How are disk blocks allocated for files?

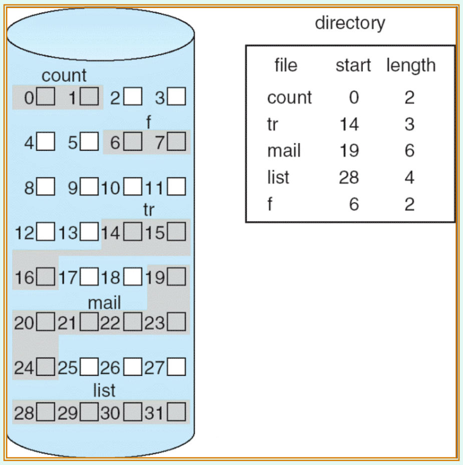
Consideration: speed of file access and effective use of disk space

Possible methods:

- o **Contiguous allocation**
- o **Linked allocation**
- o **Indexed allocation**

How does contiguous allocation work?

- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Minimal disk head movement
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow
  - – Allocate enough space when file is first created
  - – Program quits and file is allocated more space
  - – Find larger hole and copy file over to it.



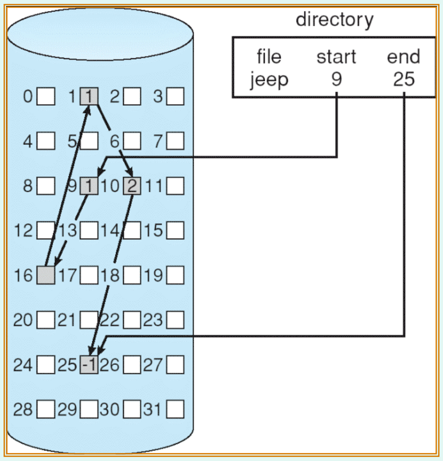
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

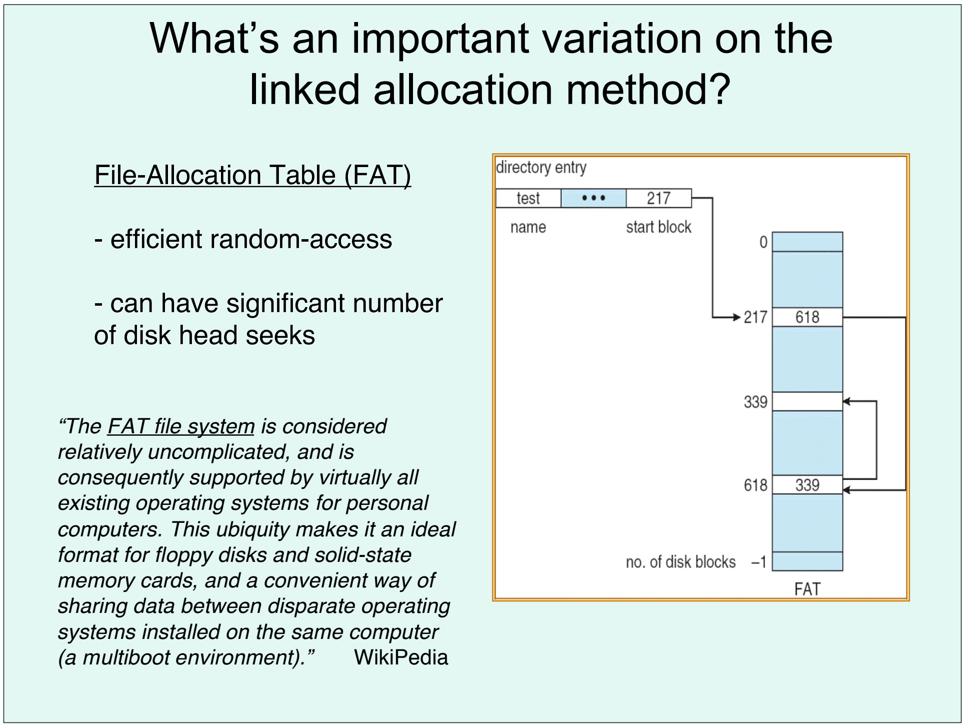
Extent-Based Systems

- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme
  - – An extent is a contiguous area of storage in a computer file system
  - – A file consists of one or more extents.
  - – Extent-based file systems allocate disk blocks in **extents**
- How do you use extents?
  - – Initial disk chunk is set aside for program
  - – If more disk space needed – allocate an extent

How does linked allocation work?

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Create File: new entry in directory with NULL pointer
- Disadvantages:
  - – only effectively handles sequential access files.
  - – Space required for pointers.
    - Solution: collect blocks into clusters
    - More internal frag.
  - – Reliability – link is bad





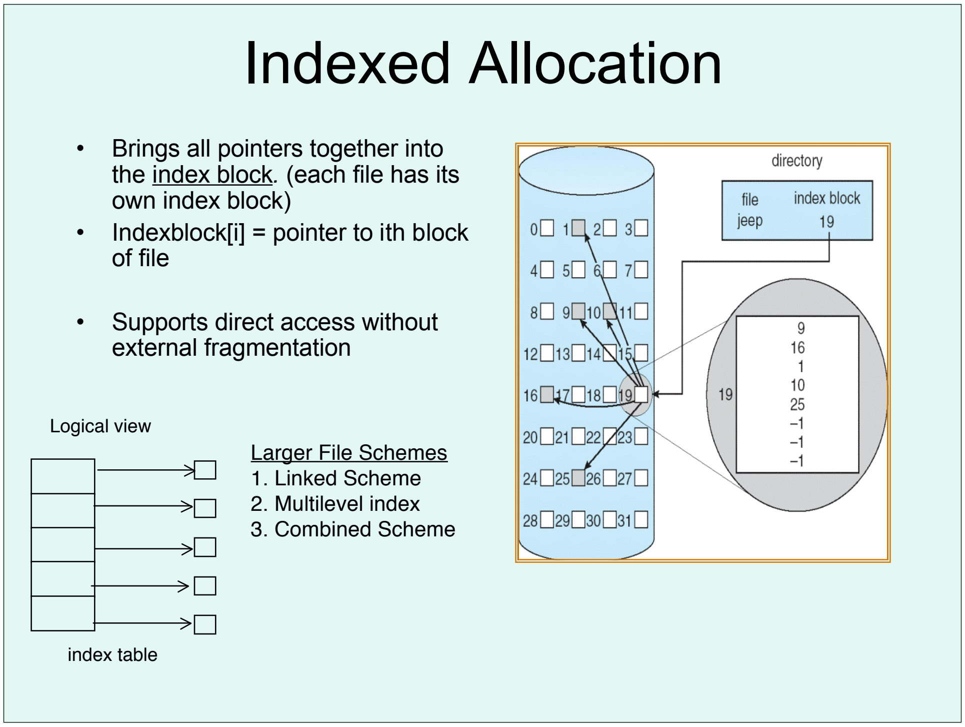
What's an important variation on the linked allocation method?

File-Allocation Table (FAT)

- efficient random-access

- can have significant number of disk head seeks

"The FAT file system is considered relatively uncomplicated, and is consequently supported by virtually all existing operating systems for personal computers. This ubiquity makes it an ideal format for floppy disks and solid-state memory cards, and a convenient way of sharing data between disparate operating systems installed on the same computer (a multiboot environment)." WikiPedia



Indexed Allocation

- Brings all pointers together into the index block. (each file has its own index block)
- Indexblock[i] = pointer to ith block of file
- Supports direct access without external fragmentation

Larger File Schemes
1. Linked Scheme
2. Multilevel index
3. Combined Scheme

Indexed Allocation

- Solves the problem of linked allocation by bringing all the pointers together into one location: index block
- New file – index block has all NULL pointers
- Write – obtain block from free-space manager and place address into index block

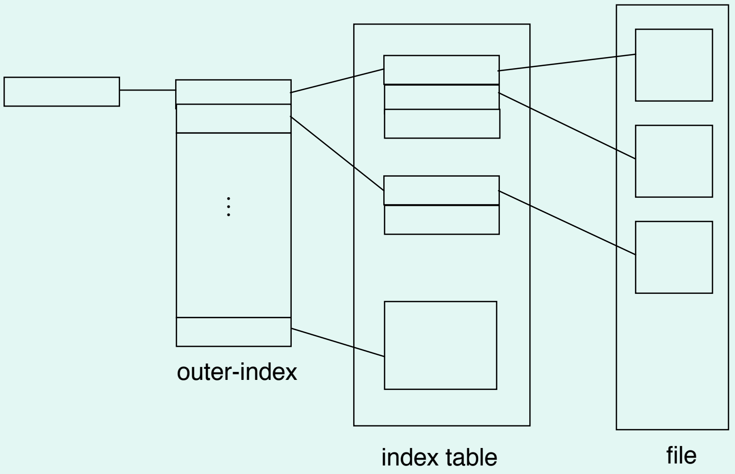
- Wasted space – index block – How large should the index block be?
- SIZE – normally one block long – with a possible link to another block if needed.

Indexed Allocation

<u>Larger File Schemes</u>

1. Linked Scheme – last address in index block points to another index block
2. Multilevel index – first level index block points to a set of second level index blocks which point to files (4BG) (see next page)
3. Combined Scheme – both of the above (see page after the next page)

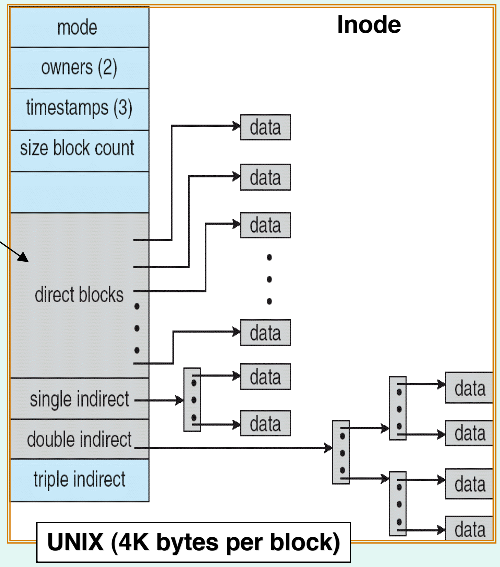
What is a multilevel index?



What is a combined scheme?

Keep some pointers that point directly to block

Indirect pointers

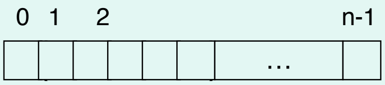


The term i-node perhaps came from the word index?

How is free-space managed?

Some processors have an instruction that return the offset in a word for the first bit with value 1

- Bit vector



Simple and can be a fairly efficient search

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

Block number calculation (which block is the bit found in?)

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

000000000000000000000000000000000000000000000
1

8 * 5 + 7 = 47   (block 47 is free)

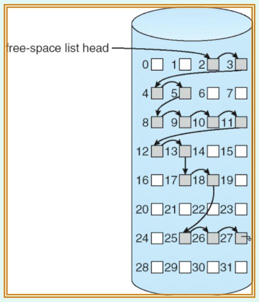
How is free-space managed?

- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

    disk size = 2^{30} bytes (1 gigabyte)

    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files



How else is free-space managed?

- Linked list (free list)   see right →
  - Traversing is costly (must read each block)
  - No waste of space
- Grouping (store addresses of n free blocks in 1st block)
- Counting (keep address and count of blocks in free list)

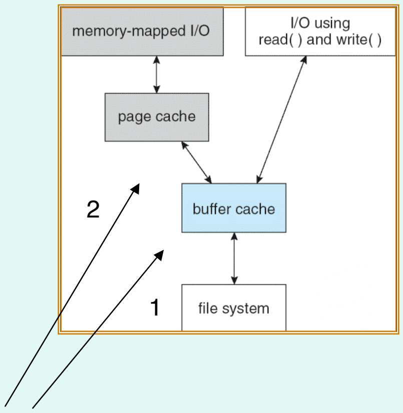
What about efficiency and performance?

- Efficiency dependents on:
  - disk allocation and directory algorithms
    - preallocate inode and strategically spread across disk
  - types of data kept in file's directory entry (ie. modify date)
- Performance
  - **disk cache** – for frequently used blocks
    - Disk controller cache (store tracks), memory cache (store blocks)
    - Main memory – buffer cache or page cache
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

Page Cache

Without a unified buffer cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache

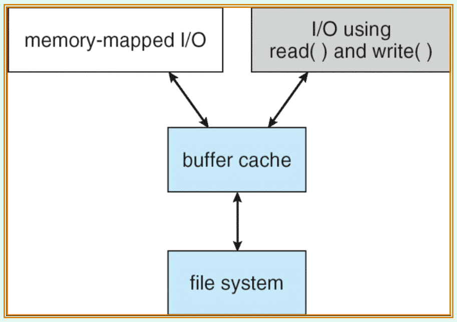


Double Caching
1. read() brings block into buffer cache
2. block is copied to page cache

Waste memory & CPU/IO cycles
Inconsistencies - corrupt files

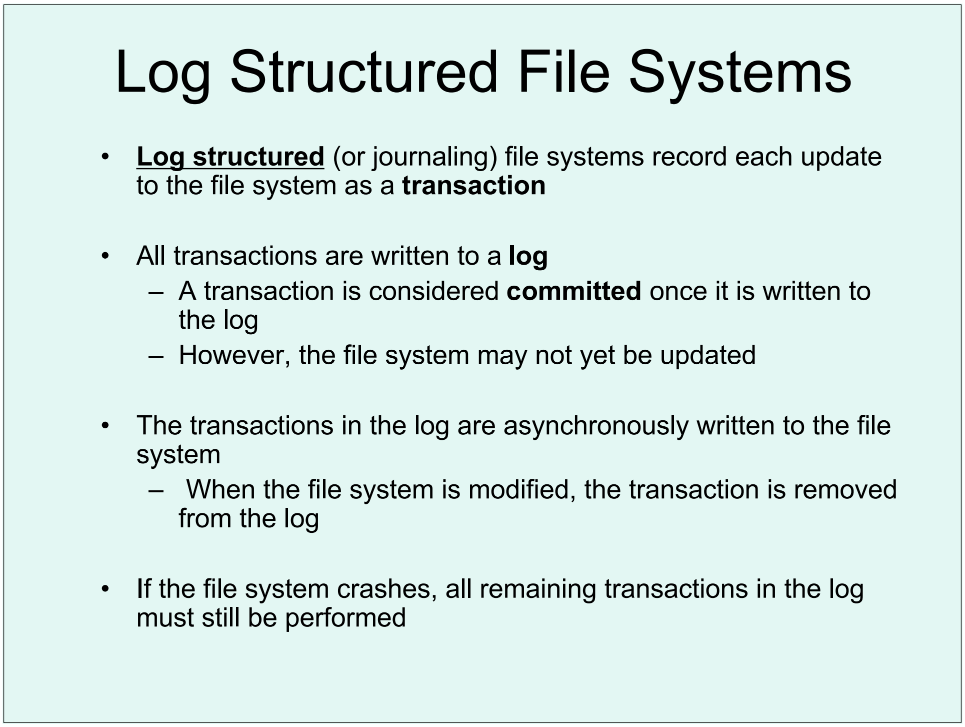
Unified Buffer Cache

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



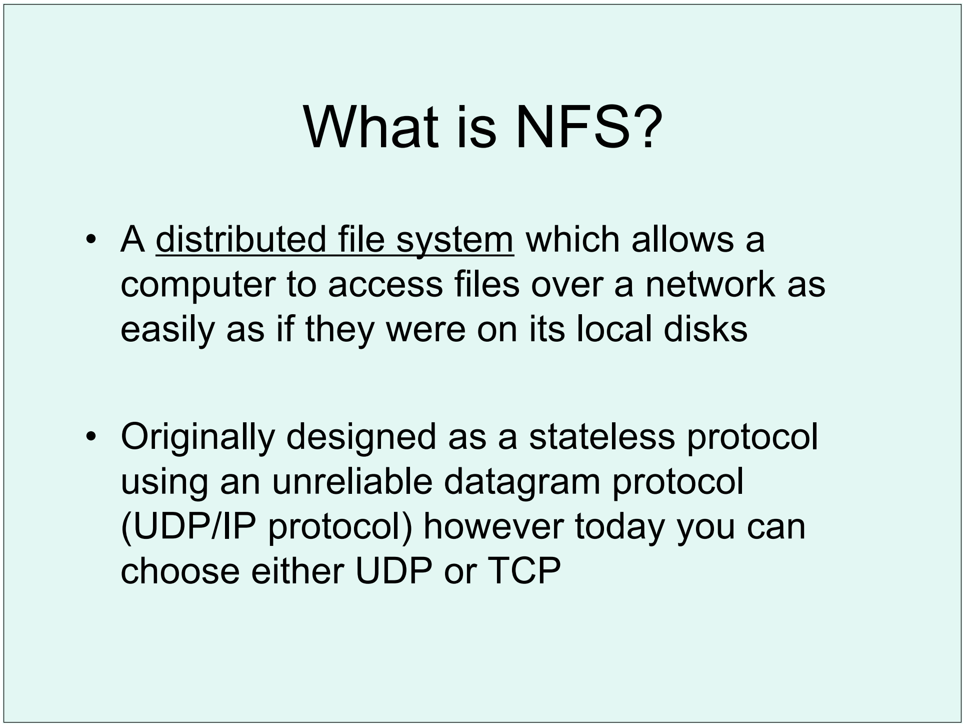
How can we recover critical directory data?

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



Log Structured File Systems

- **<u>Log structured</u>** (or journaling) file systems record each 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
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system
  - When the file system is modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed



What is NFS?

- A <u>distributed file system</u> which allows a computer to access files over a network as easily as if they were on its local disks
- Originally designed as a stateless protocol using an unreliable datagram protocol (UDP/IP protocol) however today you can choose either UDP or TCP

What is NFS?

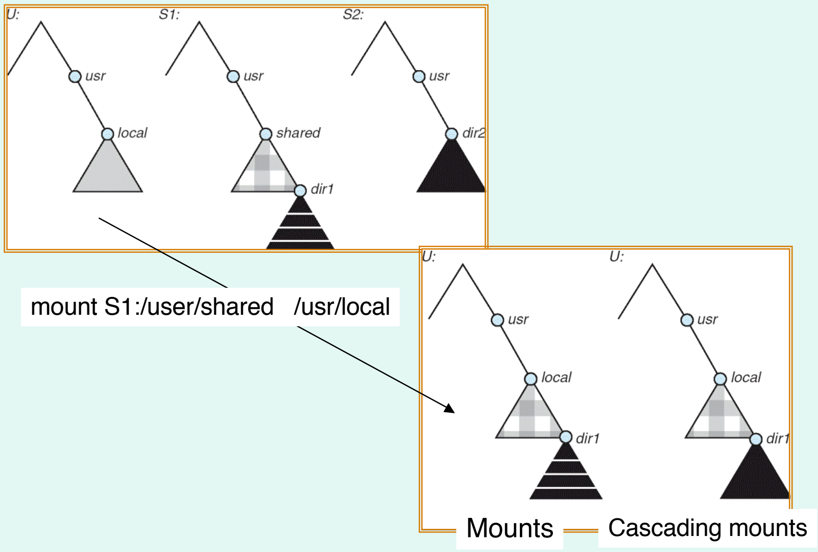
- *Interconnected workstations* viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
  - A remote directory is mounted over a local file system directory
  - Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
    - Files in the remote directory can then be accessed in a transparent manner
  - Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory

What is NFS?

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media

- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces

Mounting in NFS



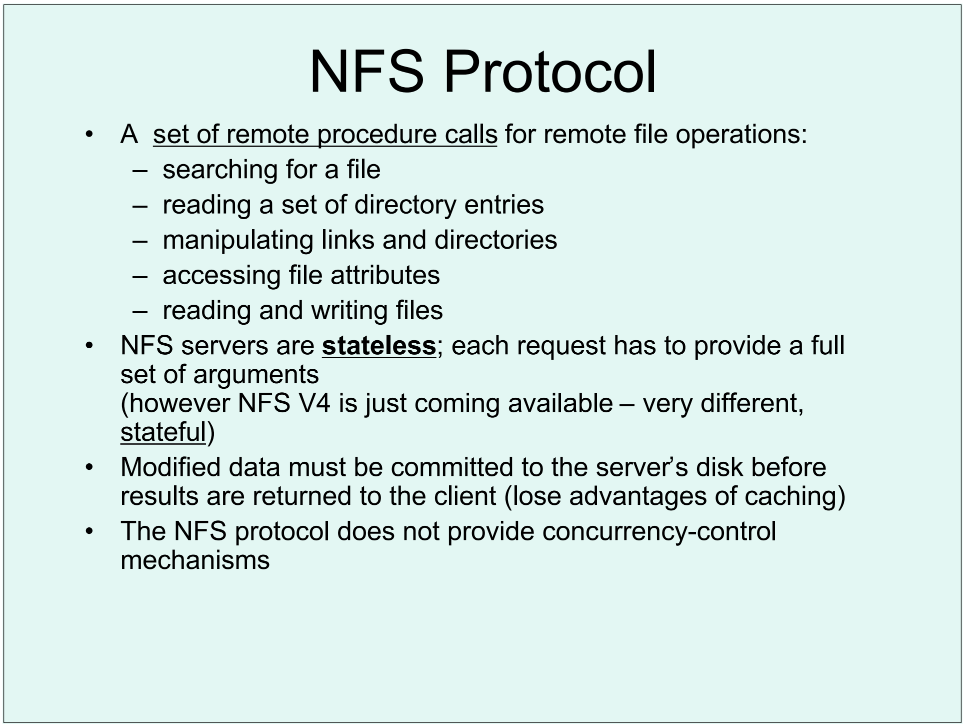
mount S1:/user/shared /usr/local

Mounts

Cascading mounts

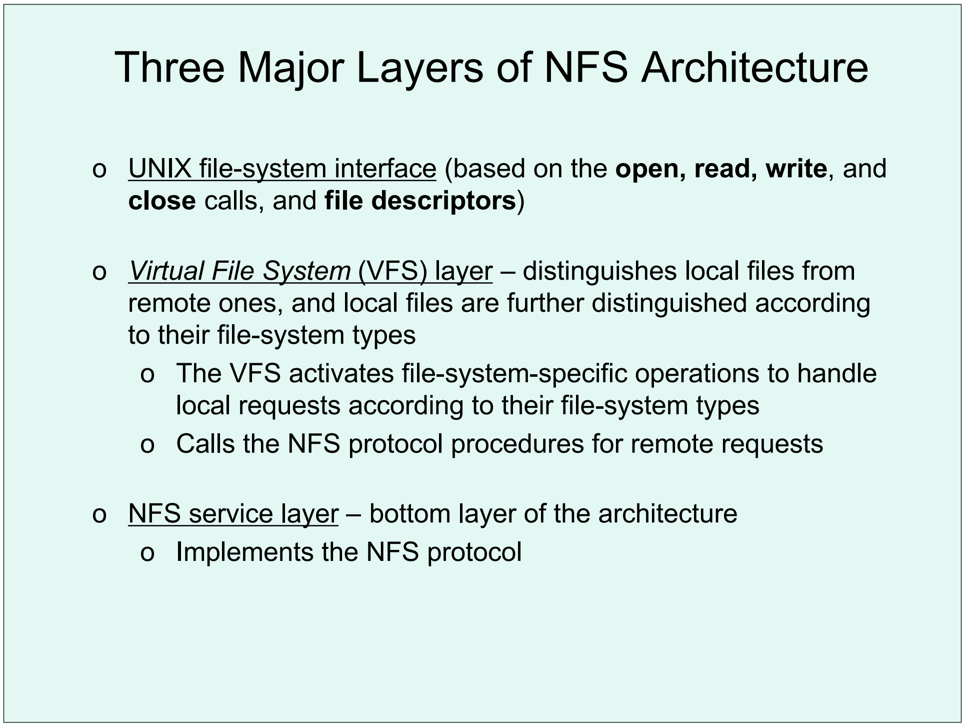
NFS Mount Protocol

- Establishes initial logical connection between server and client
- Mount operation
  - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
  - Export list – specifies local file systems that server exports and names of machines that are permitted to mount them
- the server returns a file handle
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- changes only the user’s view and does not affect the server side



NFS Protocol

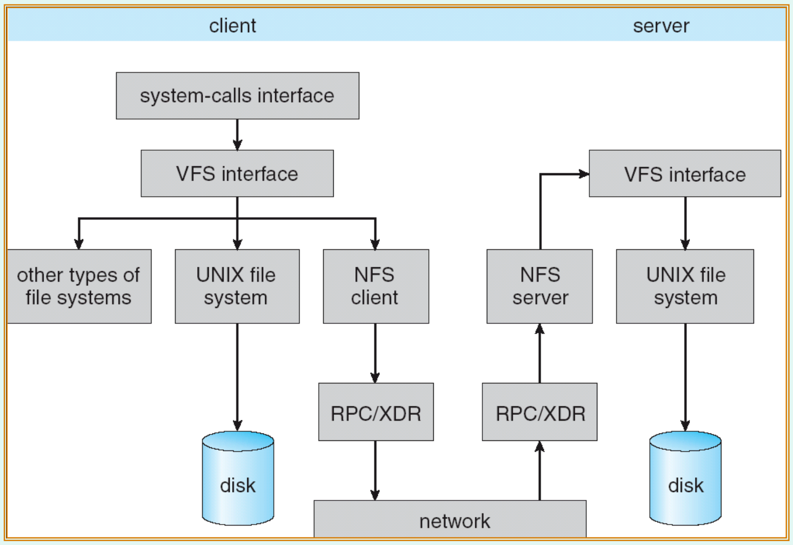
- A set of remote procedure calls for remote file operations:
  - searching for a file
  - reading a set of directory entries
  - manipulating links and directories
  - accessing file attributes
  - reading and writing files
- NFS servers are **stateless**; each request has to provide a full set of arguments
  (however NFS V4 is just coming available – very different, stateful)
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- The NFS protocol does not provide concurrency-control mechanisms



Three Major Layers of NFS Architecture

- UNIX file-system interface (based on the **open, read, write**, and **close** calls, and **file descriptors**)
- *Virtual File System* (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
  - The VFS activates file-system-specific operations to handle local requests according to their file-system types
  - Calls the NFS protocol procedures for remote requests
- NFS service layer – bottom layer of the architecture
  - Implements the NFS protocol

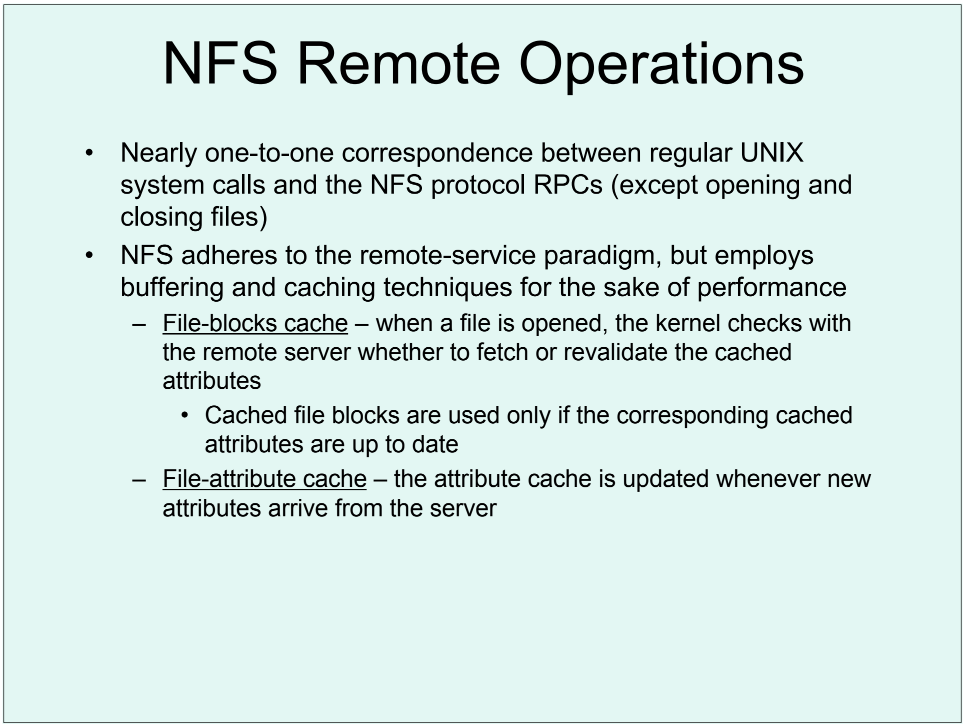
Schematic View of NFS



NFS Path-Name Translation

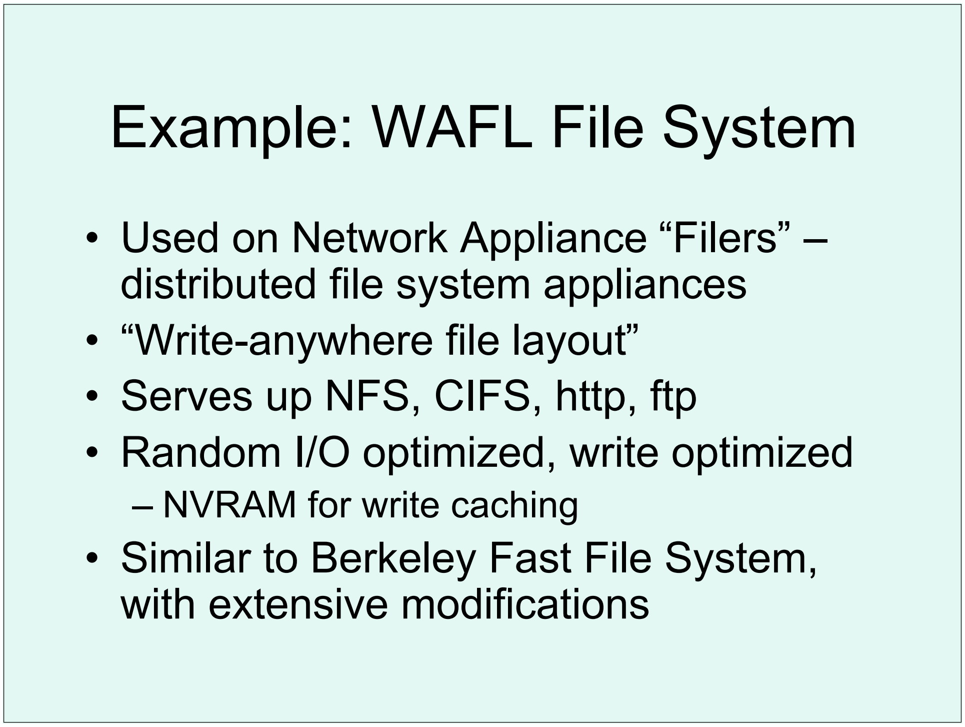
- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode

- To make lookup faster, a directory name lookup cache on the client’s side holds the vnodes for remote directory names



NFS Remote Operations

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance
  - File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
    - Cached file blocks are used only if the corresponding cached attributes are up to date
  - File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server



Example: WAFL File System

- Used on Network Appliance “Filers” – distributed file system appliances
- “Write-anywhere file layout”
- Serves up NFS, CIFS, http, ftp
- Random I/O optimized, write optimized
  - NVRAM for write caching
- Similar to Berkeley Fast File System, with extensive modifications