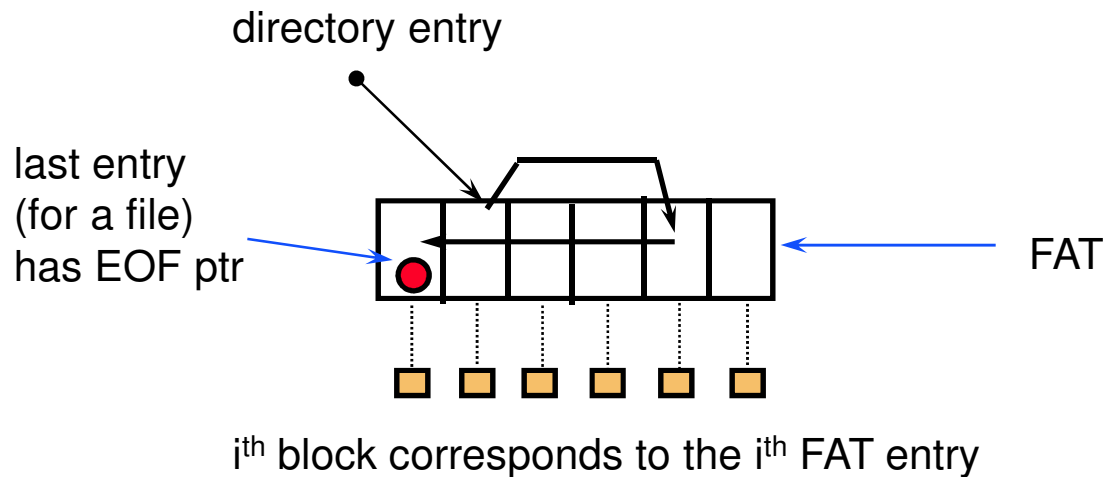


Announcements

- Midterm #2 is Thursday 4/14 in class
- Project #5 is out

Modified Linked Allocation (FAT)

- Section of disk contains a table
 - called the file allocate table (FAT)
 - used in MS-DOS
- Directory entry contains the block number of the first block in the file
- Table entry contains the number of the next block in the file
- Last block has a end-of-file value as a table entry



Performance Issues

- **FAT**
 - simple, easy to implement
 - faster to traverse than linked allocation
 - random access requires following links
 - files can't have holes in them
- **Hybrid indirect**
 - fast access to any part of the file
 - files can have holes in them
 - more complex

Free Space Management

- How do we find a disk block to allocate?
- Bit Vectors
 - array of bits (one per block) that indicates if a block is free
 - compact so can keep in memory
 - 100 GB disk, 4K blocks -> 6MB per disk (0.003%)
 - easy to find long runs of free blocks
- Linked lists
 - each disk block contains the pointer to the next free block
 - pointer to first free block is keep in a special location on disk
- Run length encoding (called counting in book)
 - pointer to first free block is keep in a special location on disk
 - each free block also includes a count of the number of consecutive blocks that are free

DOS Directories

- **Root directory**
 - immediately follows the FAT
- **Directory is a table of 32 byte entries**
 - 8 byte file name, 3 byte filename extension
 - size of file, data and time stamp, starting cluster number of the file, file attribute codes
 - Fixed size and capacity
- **Subdirectory**
 - This is just a file
 - Record of where the subdirectory is located is stored in the FAT

Implementing Directories

- **Linear List**
 - array of names for files
 - must search entire list to find or allocate a filename
 - sorting can improve search performance, but adds complexity
- **Hash table**
 - use hash function to find filenames in directory
 - needs a good hash function
 - need to resolve collisions
 - must keep table small and expand on demand since many directories are mostly empty

Unix Directories

- Space for directories are allocated in units called *chunks*
 - Size of a chunk is chosen so that each allocation can be transferred to disk in a single operation
 - Chunks are broken into variable-length directory entries to allow filenames of arbitrary length
 - No directory entry can span more than one chunk
 - Directory entry contains
 - pointer to inode (file data-structure)
 - size of entry
 - length of filename contained in entry (up to 255)
 - remainder of entry is variable length - contains file name

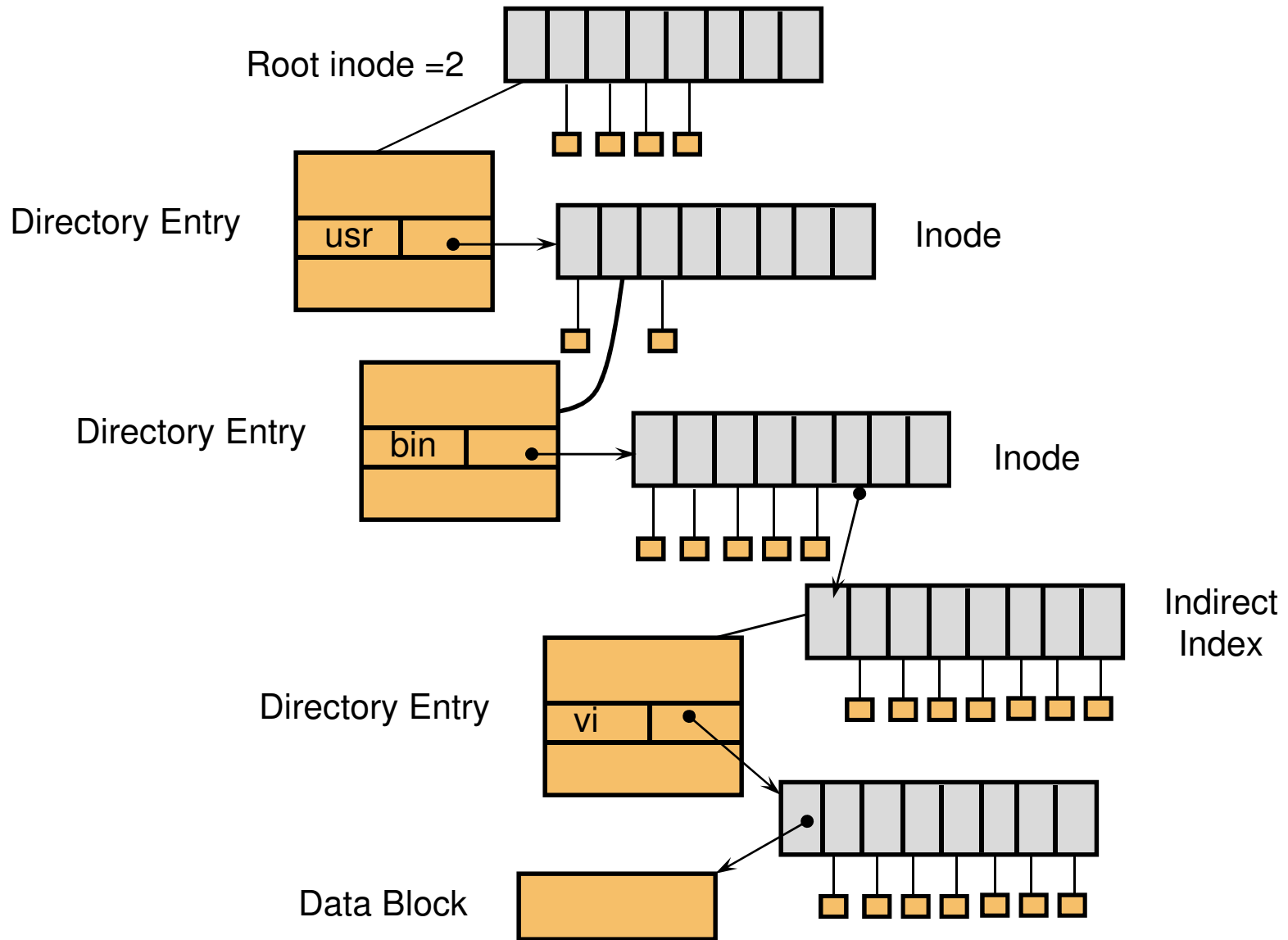
inodes

- File index node
- Contains:
 - Pointers to blocks in a file (direct, single indirect, double indirect, triple indirect)
 - Type and access mode
 - File's owner
 - Number of references to file
 - Size of file
 - Number of physical blocks

Unix directories - links

- Each file has unique inode but it may have multiple directory entries in the same filesystem to reference inode
- Each directory entry creates a hard link of a filename to the file's inode
 - Number of links to file are kept in reference count variable in inode
 - If links are removed, file is deleted when number of links becomes zero
- **Symbolic or soft link**
 - Implemented as a file that contains a pathname
 - Symbolic links do not have an effect on inode reference count

File Lookup (/usr/bin/vi)



Using UNIX filesystem data structures

- Example: `find /usr/bin/vi`
 - from Leffler, McKusick, Karels and Quarterman
 - Search root directory of filesystem to find `/usr`
 - root directory inode is, by convention, stored in inode #2
 - inode shows *where data blocks are* for root directory - *these blocks* (not the inode itself) *must* be retrieved and searched for entry `usr`
 - we discover that the directory `usr`'s inode is inode #4
 - Search `usr` for `bin`
 - access blocks pointed to by inode #4 and search contents of blocks for entry that gives us `bin`'s inode
 - we discover that `bin`'s inode is inode #7
 - Search `bin` for `vi`
 - access blocks pointed to by inode #7 and search contents of block for an entry that gives us `vi`'s inode
 - we discover that `vi`'s inode is inode #7
 - Access inode #7 - this is `vi`'s inode

How to Improve Speed?

- Use A Cache
- Name-to-Inode lookup
 - Hash on full path name
 - Find inode without and disk accesses on a hit

Mount System Call

- How to attach a file system into a name space?
- Simple Idea:
 - use letters C, D, E, etc.
 - use volume names (VMS) – fixed length string
- Better Idea:
 - Allow attachment at arbitrary points in namespace
 - Designate one tree as the “root” file system
 - Others are attached to the root
- Mount used in:
 - UNIX
 - Windows (NTFS mount points)
 - GeekOS

Log Structured File Systems

- Key Idea
 - Use transactions like model for filesystem updates
- Write data to a log (also called a journal)
 - Records meta data changes
 - Records data blocks written
 - File operation is committed once it is to the log
 - Partial updates to log are lost on failure
- Next Step
 - Eliminate the filesystem and just keep the log
 - Requires a process called a cleaner
 - Copies old data from log to head of log to allow compaction