12. File-System Implementation

Sungyoung Lee

College of Engineering
KyungHee University

Contents

Overview

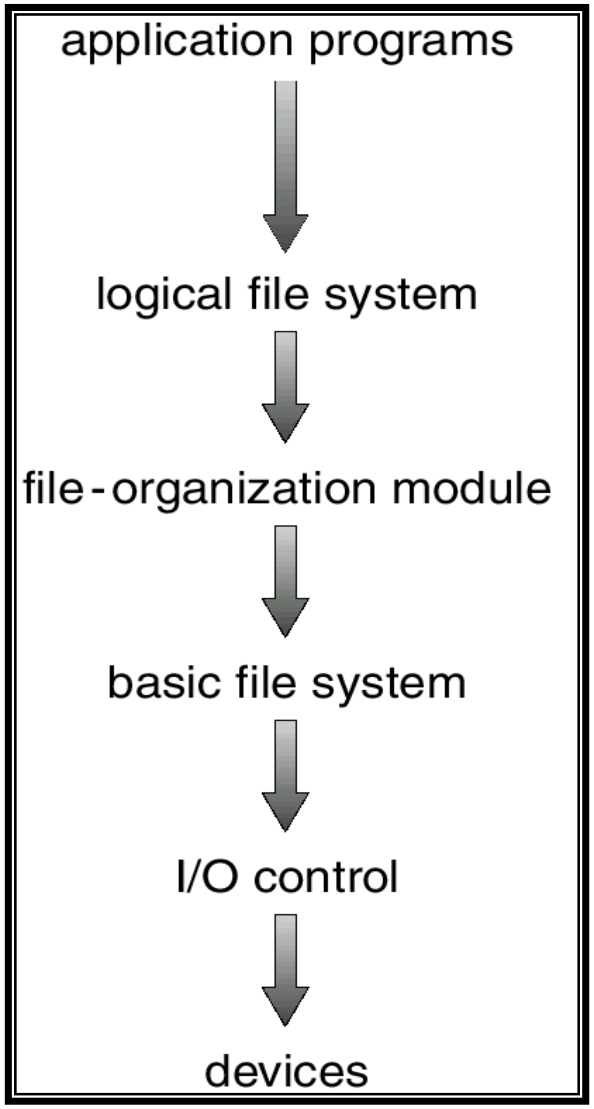
- User’s view on file systems:
  - How files are named?
  - What operations are allowed on them?
  - What the directory tree looks like?

- Implementor’s view on file systems:
  - How files and directories are stored?
  - How disk space is managed?
  - How to make everything work efficiently and reliably?

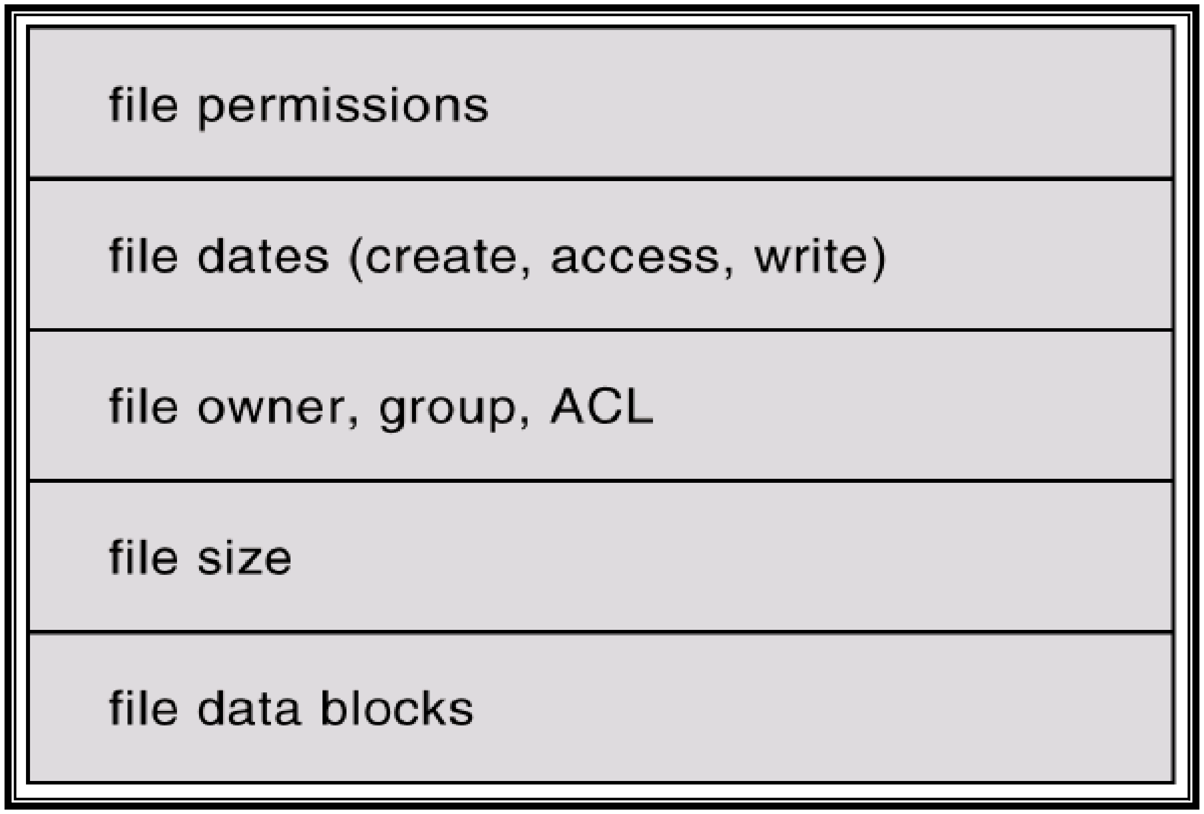
File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File system organized into layers
- *File control block*
  - storage structure consisting of information about a file

Layered File System



A Typical File Control Block



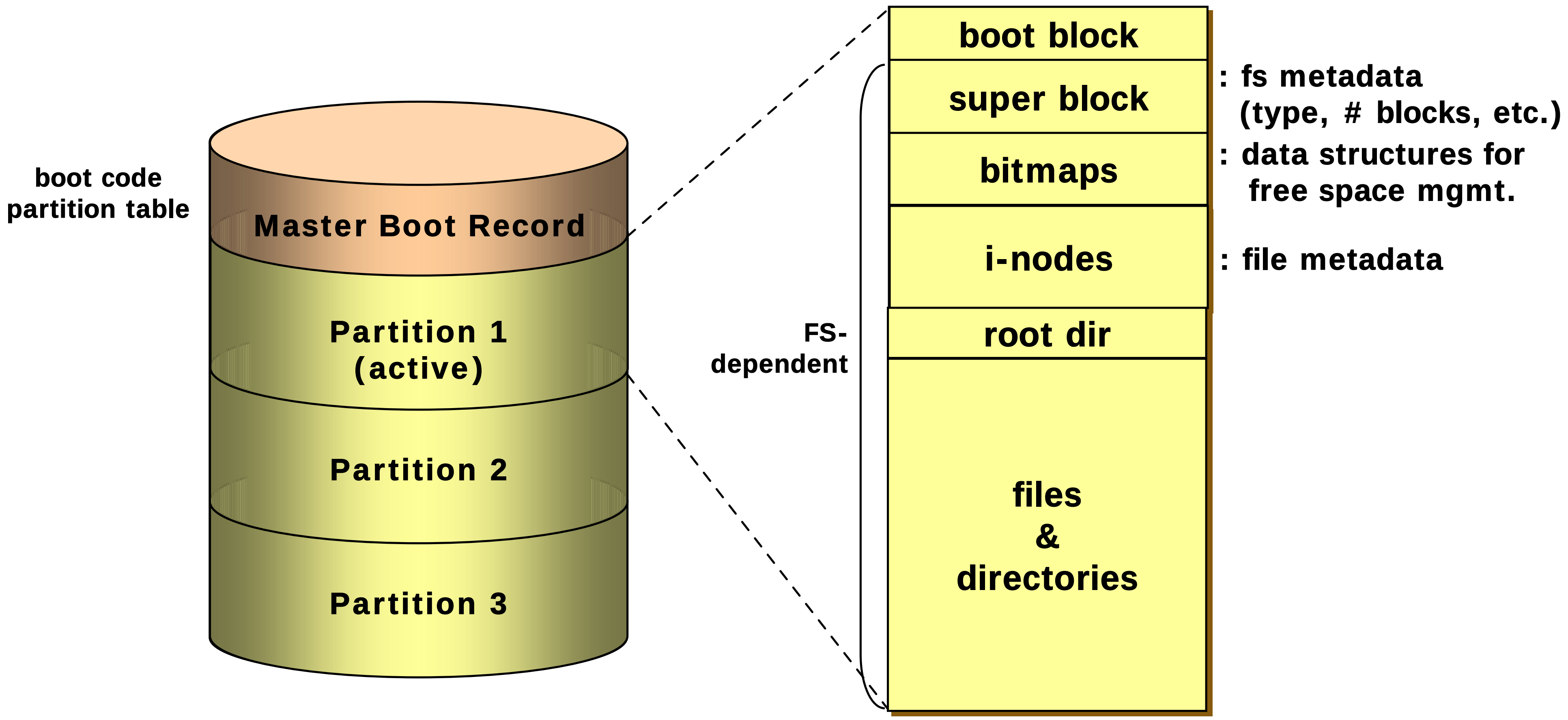
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks

File-System Implementation

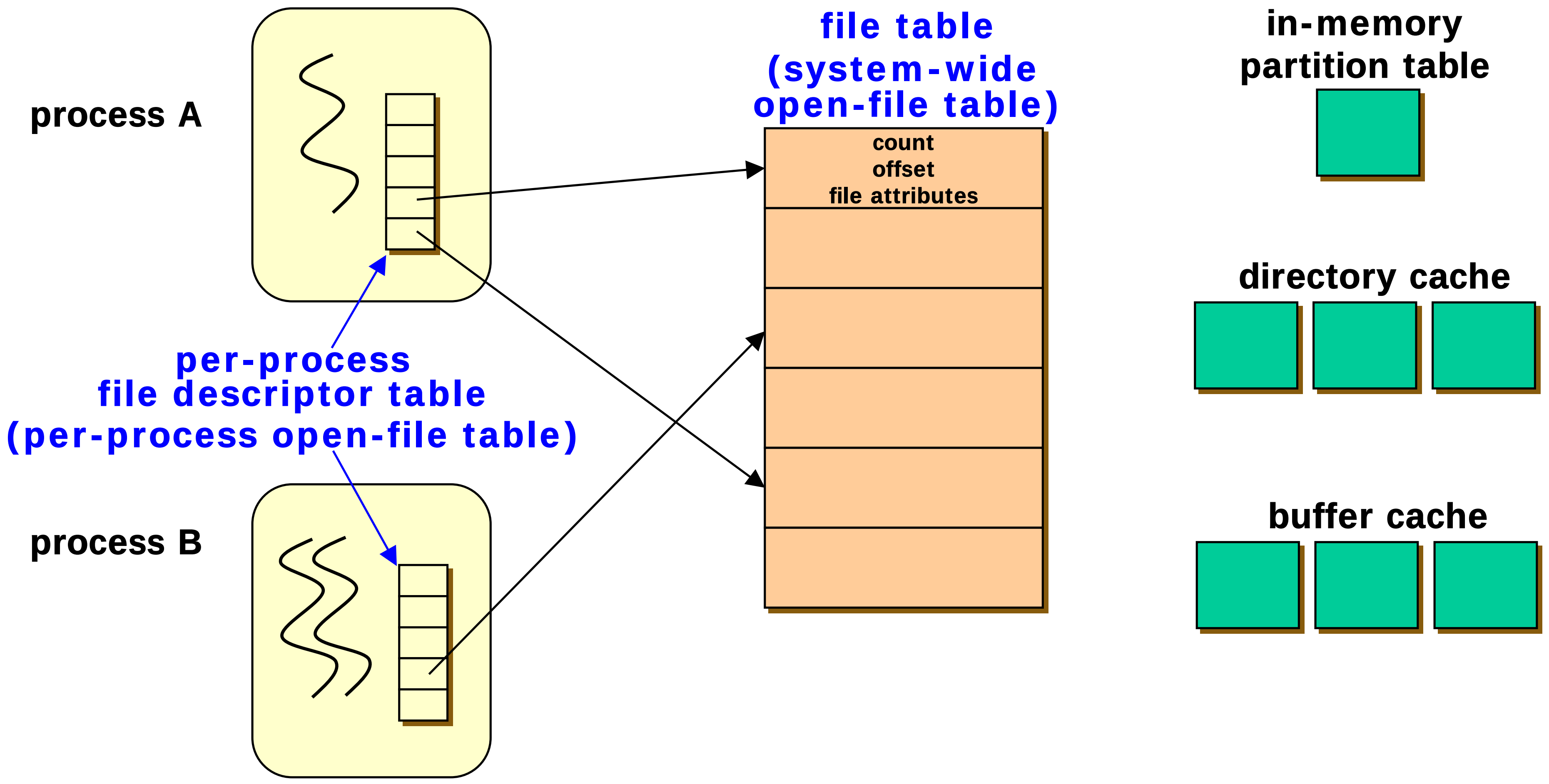
- On-disk structure
  - Boot control block
    - Boot block(UFS) or Boot sector(NTFS)
  - Partition control block
    - Super block(UFS) or Master file table(NTFS)
  - Directory structure
  - File control block (FCB)
    - I-node(UFS) or In master file table(NTFS)

- In-memory structure
  - In-memory partition table
  - In-memory directory structure
  - System-wide open file table
  - Per-process open file table

On-Disk Structure



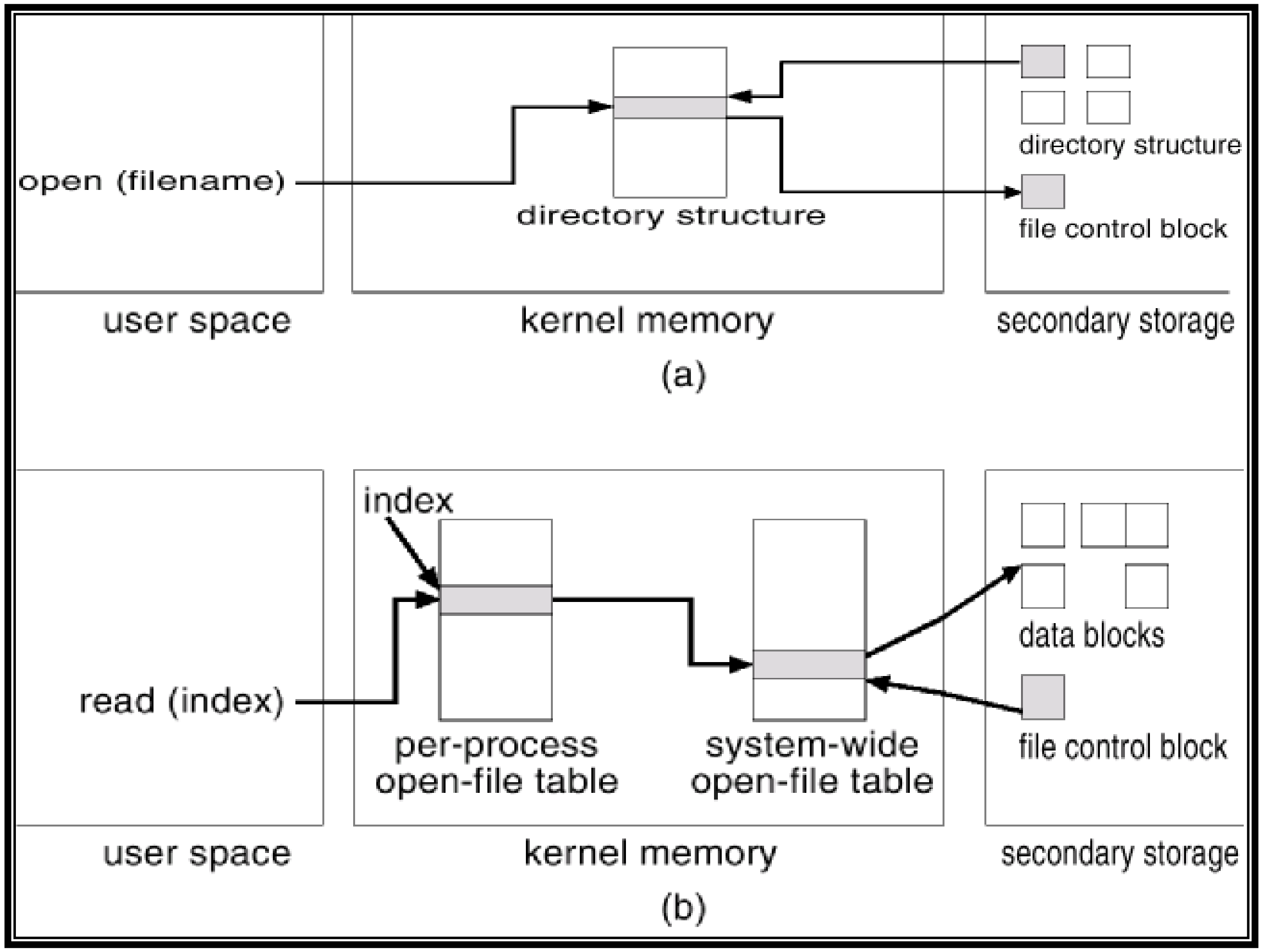
In-Memory Structure



In-Memory File System Structures

- The following figure illustrates the necessary file system structures provided by the operating systems
- Figure 12-3(a) refers to opening a file
- Figure 12-3(b) refers to reading a file

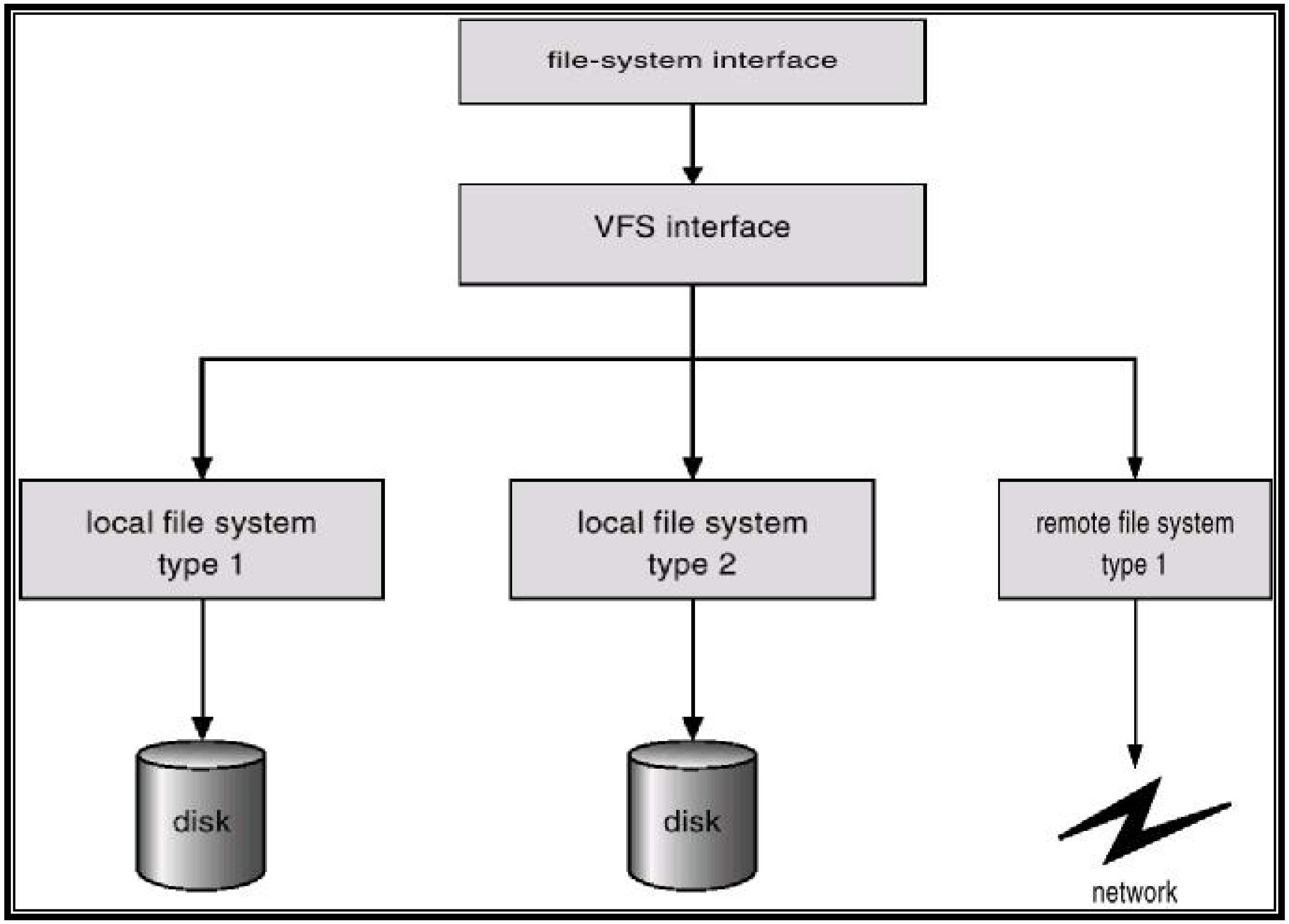
In-Memory File System Structures



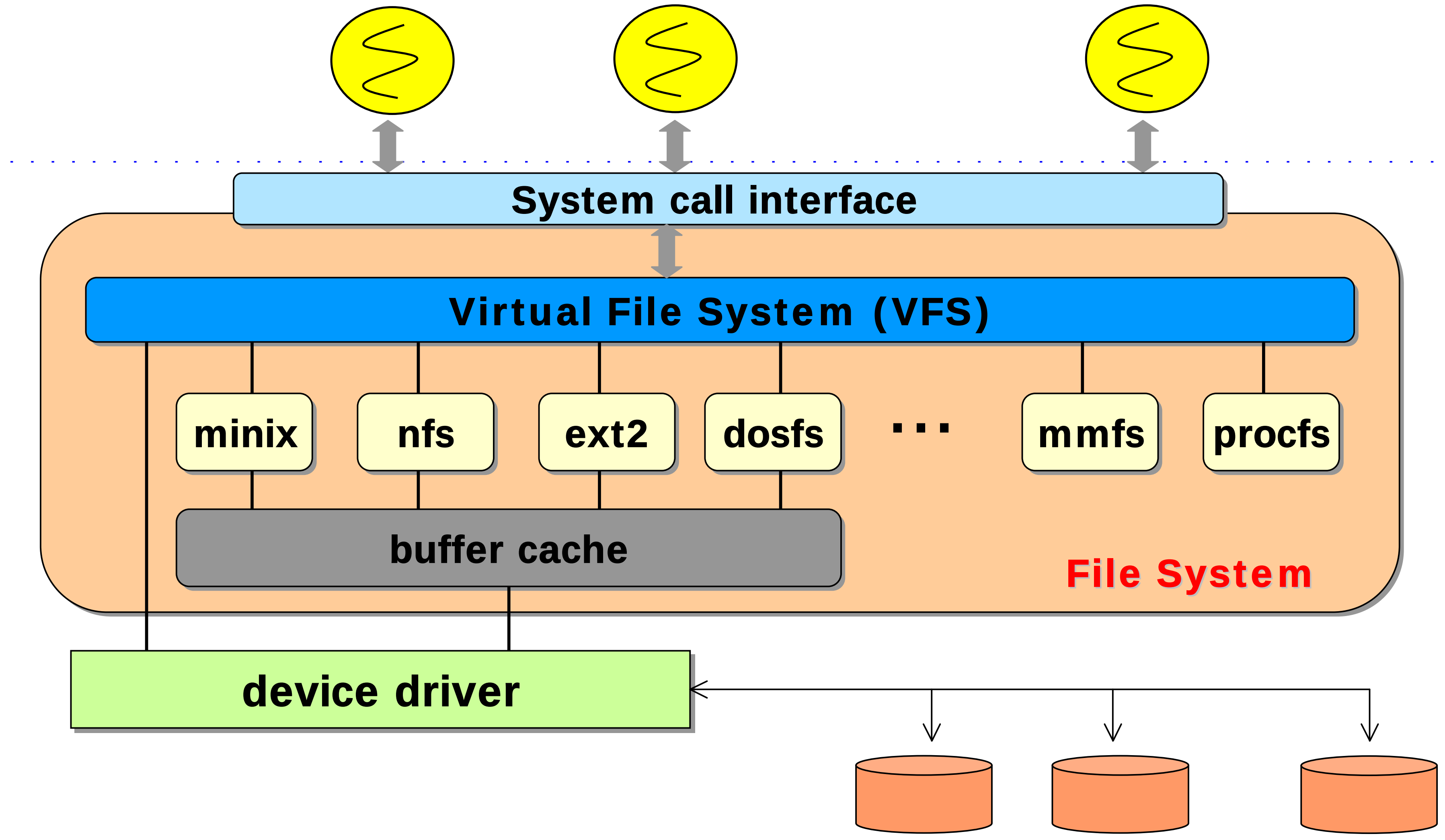
Virtual File Systems

- 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 to the VFS interface, rather than any specific type of file system

Schematic View of Virtual File System



File System Internals



VFS

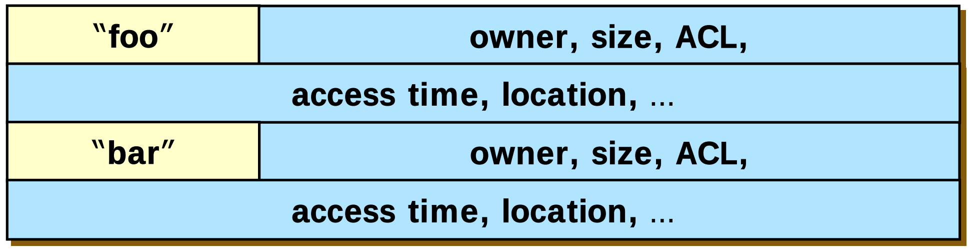
- Virtual File System
  - Manages kernel-level file abstractions in one format for all file systems
  - Receives system call requests from user-level (e.g., open, write, stat, etc.)
  - Interacts with a specific file system based on mount point traversal
  - Receives requests from other parts of the kernel, mostly from memory management
  - Translates file descriptors to VFS data structures (such as vnode)
- Linux: VFS common file model
  - The superblock object
    - stores information concerning a mounted file system
  - The inode object
    - stores general information about a specific file
  - The file object
    - stores information about the interaction between an open file and a process
  - The dentry object
    - stores information about the linking of a directory entry with the corresponding file

Directory Implementation

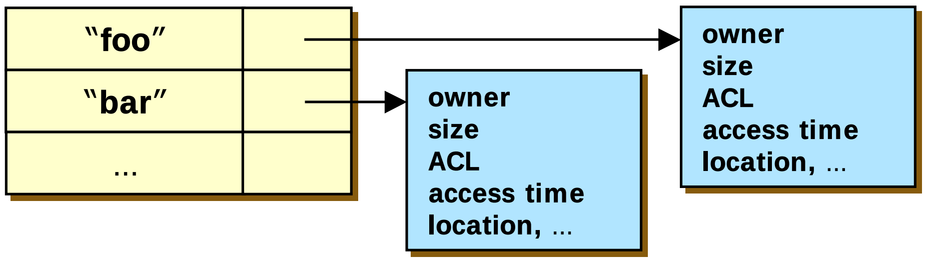
- Linear list of file names with pointer to the data blocks
  - simple to program
  - time-consuming to execute
- Hash Table
  - linear list with hash data structure
  - decreases directory search time
  - *collisions*
    - situations where two file names hash to the same location
  - fixed size

Directory Implementation

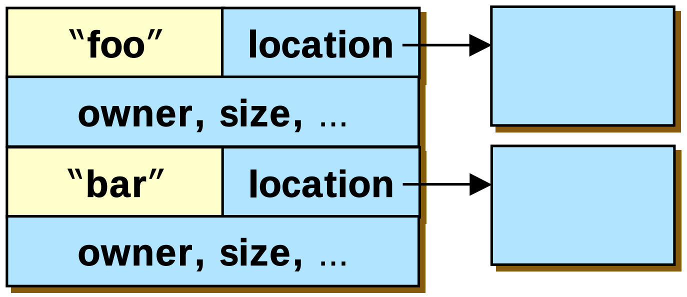
- The location of metadata
  - In the directory entry



  - In the separate data structure (e.g., i-node)



  - A hybrid approach



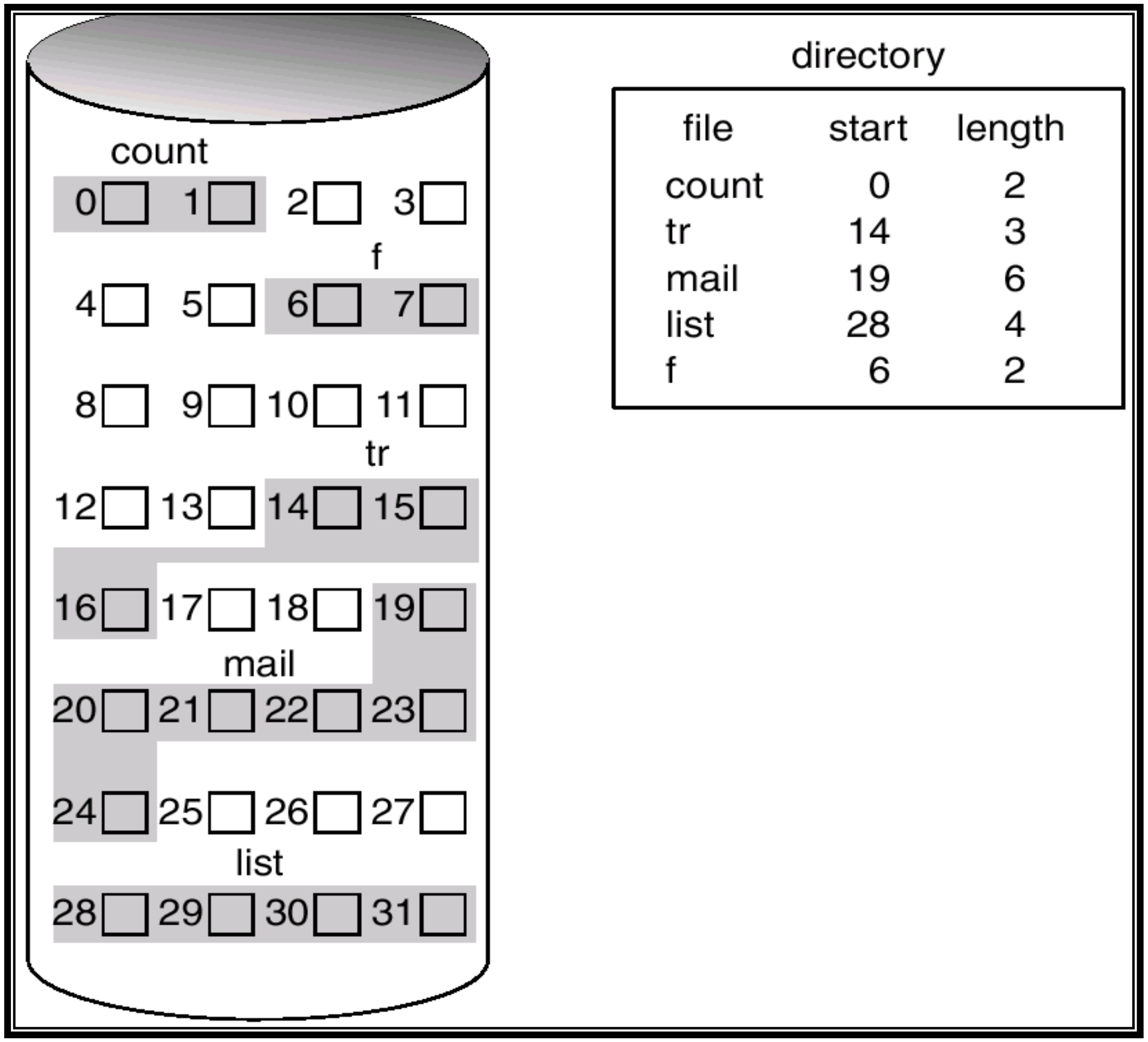
Allocation Methods

- An allocation method refers to how disk blocks are allocated for files
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Simple
  - only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow

Contiguous Allocation of Disk Space



directory

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

Extent-Based Systems

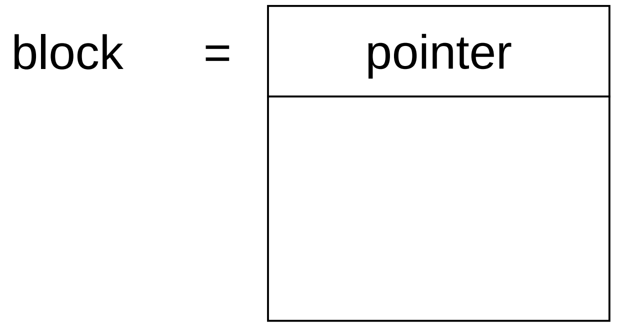
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in **extents**
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Contiguous Allocation

- Advantages
  - The number of disk seeks is minimal
  - Directory entries can be simple:
    <file name, starting disk block, length, etc.>
- Disadvantages
  - Requires a dynamic storage allocation: First / best fit
  - External fragmentation: may require a compaction
  - The file size is hard to predict and varying over time
- Feasible and widely used for CD-ROMS
  - All the file sizes are known in advance
  - Files will never change during subsequent use

Linked Allocation

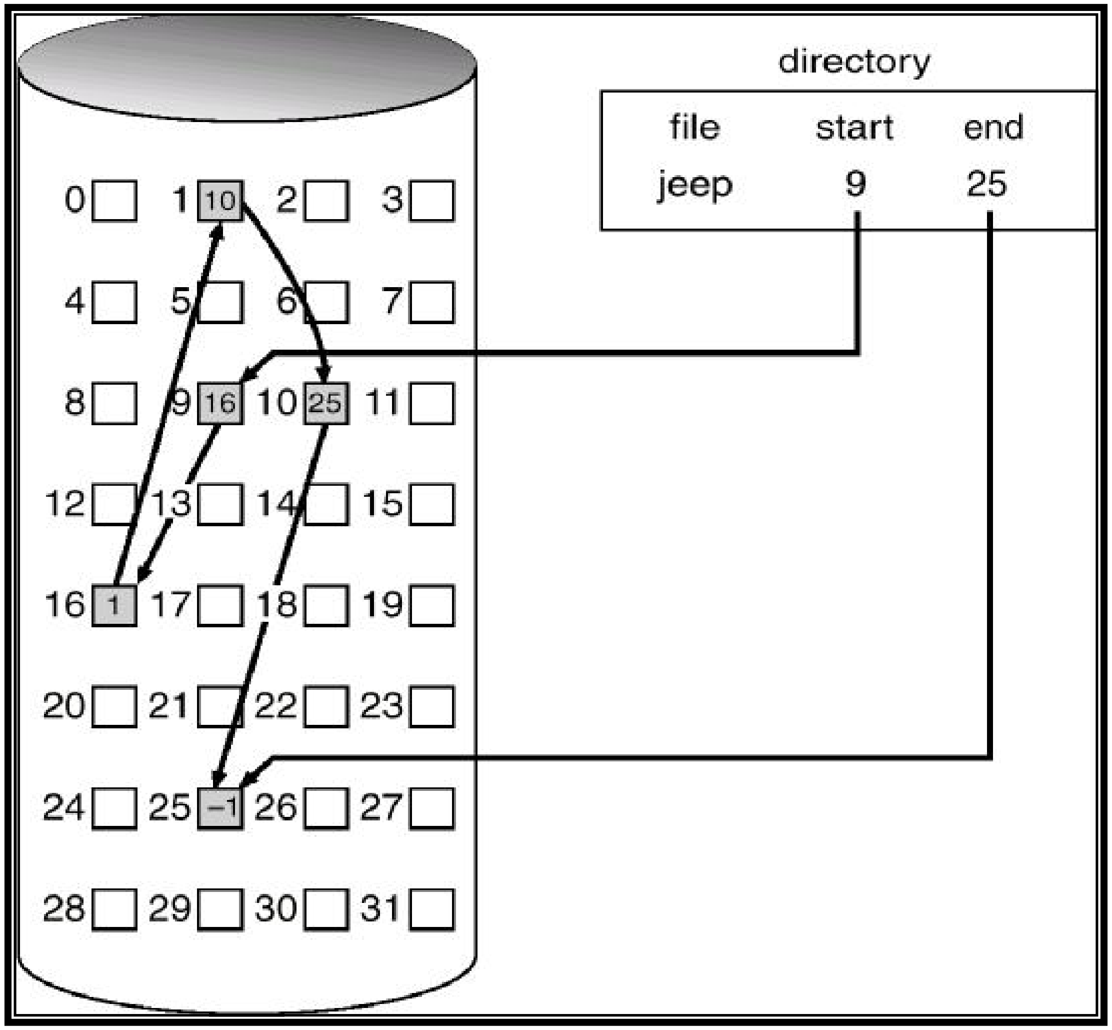
- Each file is a linked list of disk blocks
  - Blocks may be scattered anywhere on the disk



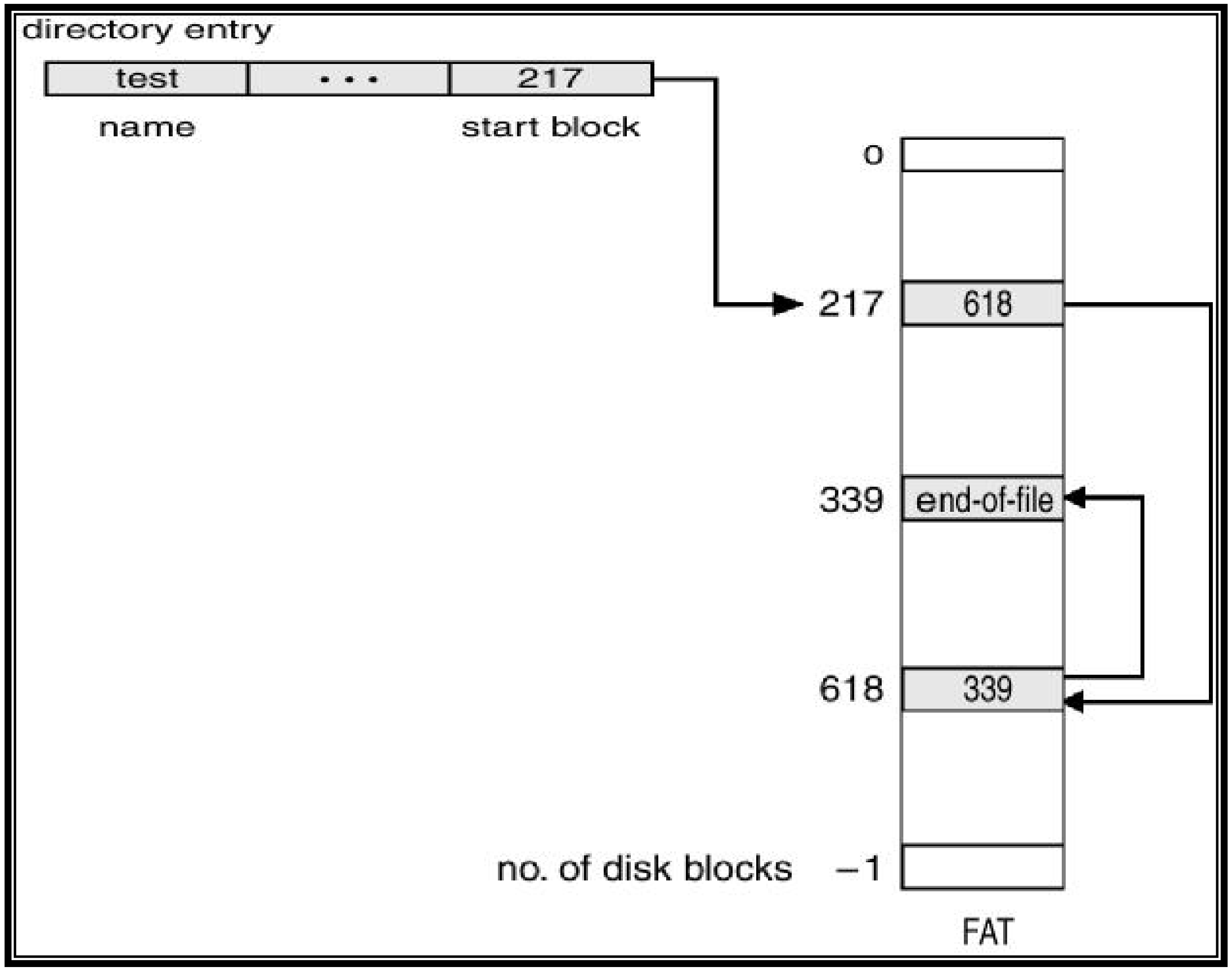
Linked Allocation (Cont'd)

- Simple
  - need only starting address
- Free-space management system
  - no waste of space
- No random access
- File-allocation table (FAT)
  - disk-space allocation used by MS-DOS and OS/2

Linked Allocation



File-Allocation Table



Linked Allocation

- Advantages
  - Directory entries are simple:

    <file name, starting block, ending block, etc.>
  - No external fragmentation
    - the disk blocks may be scattered anywhere on the disk
  - A file can continue to grow as long as free blocks are available

- Disadvantages
  - It can be used only for sequentially accessed files
  - Space overhead for maintaining pointers to the next disk block
  - The amount of data storage in a block is no longer a power of two because the pointer takes up a few bytes
  - Fragile: a pointer can be lost or damaged

Linked Allocation using Clusters

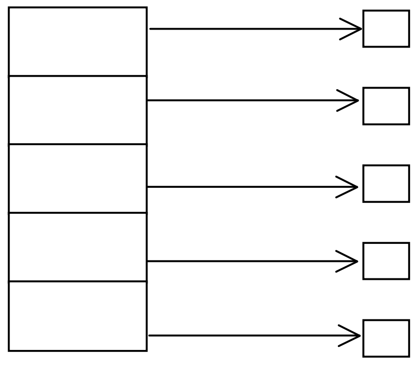
- Collect blocks into multiples (clusters) and allocate the clusters to files
  - e.g., 4 blocks / 1 cluster

- Advantages
  - The logical-to-physical block mapping remains simple
  - Improves disk throughput (fewer disk seeks)
  - Reduced space overhead for pointers

- Disadvantages
  - Internal fragmentation

Indexed Allocation

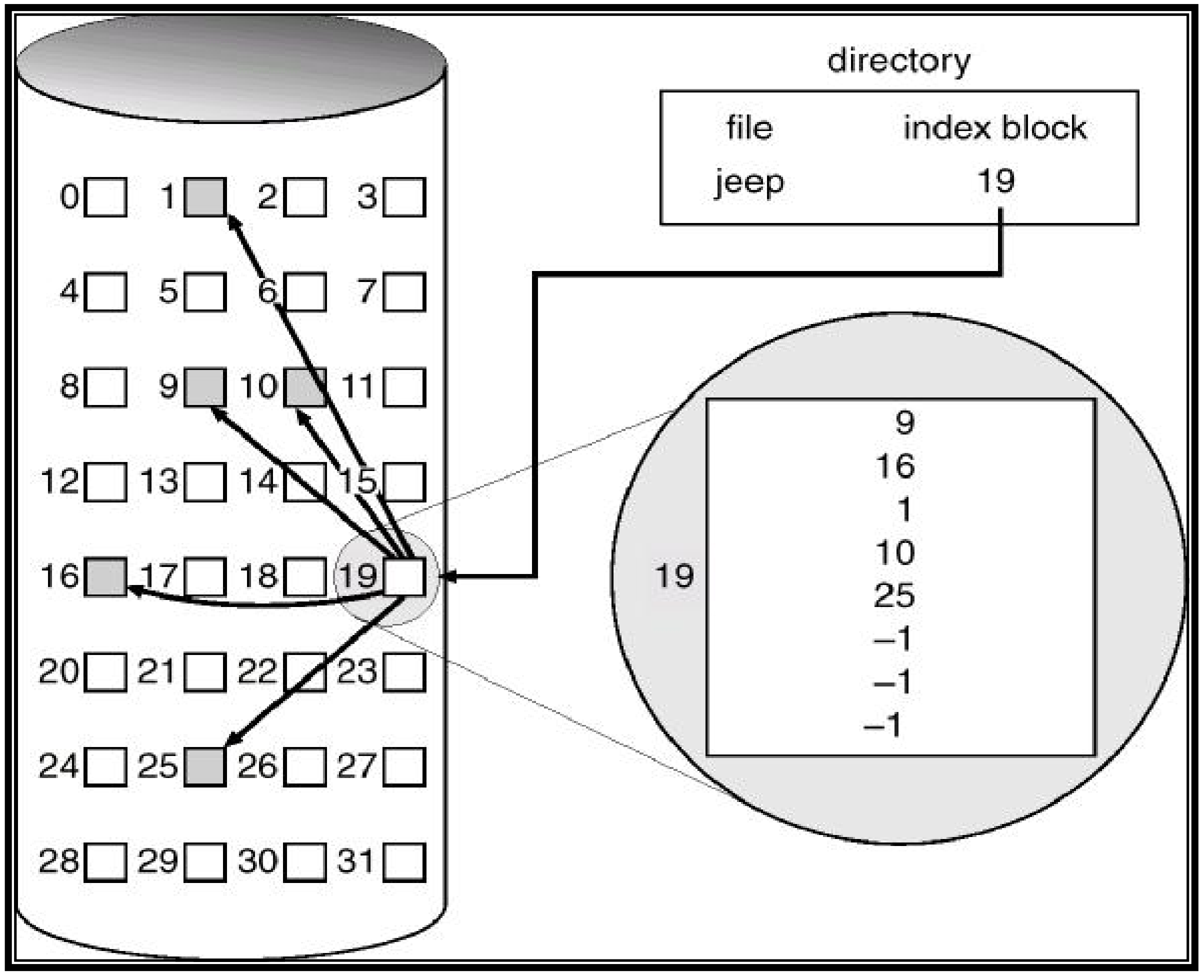
- Brings all pointers together into the *index block*
- Logical view



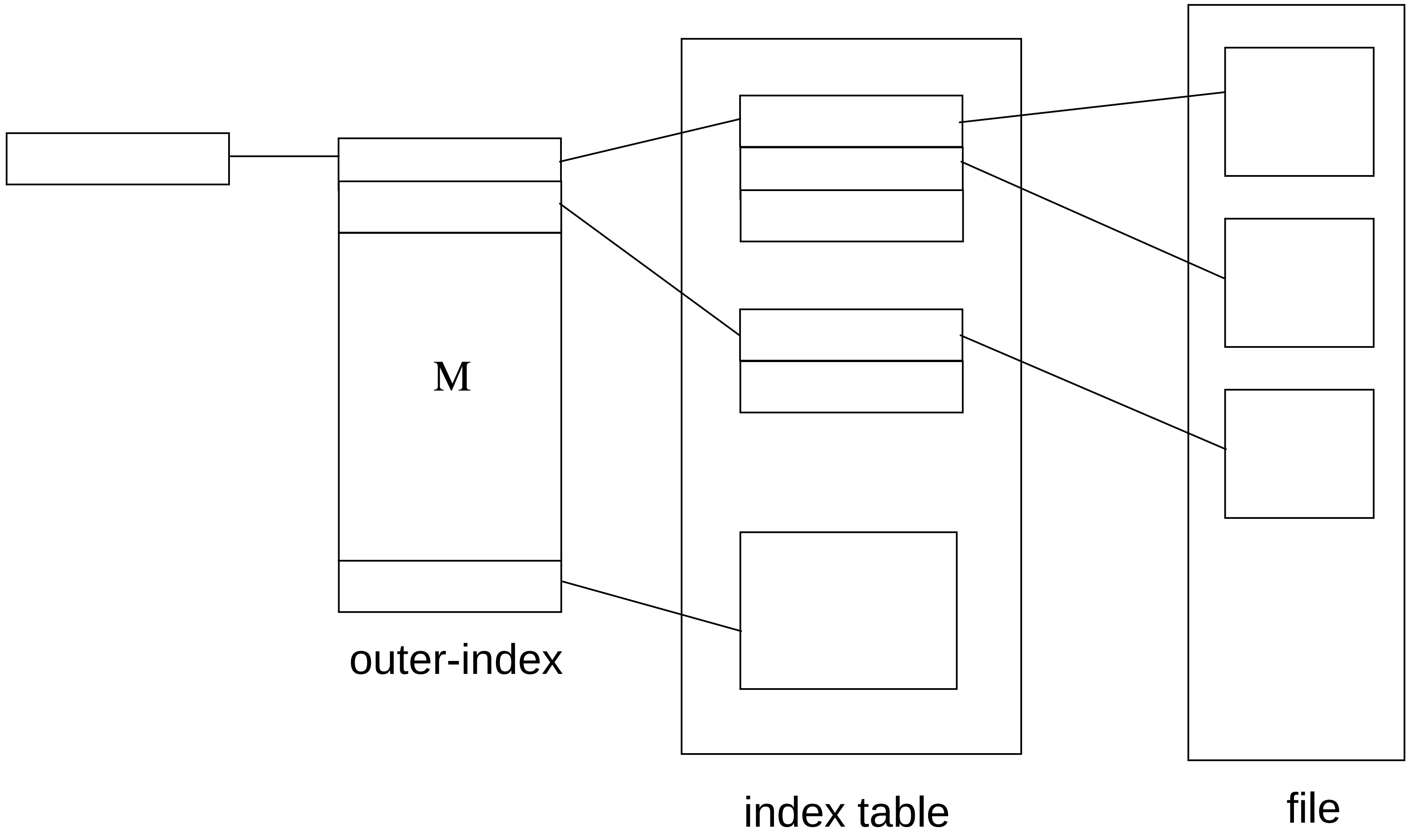
index table

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block

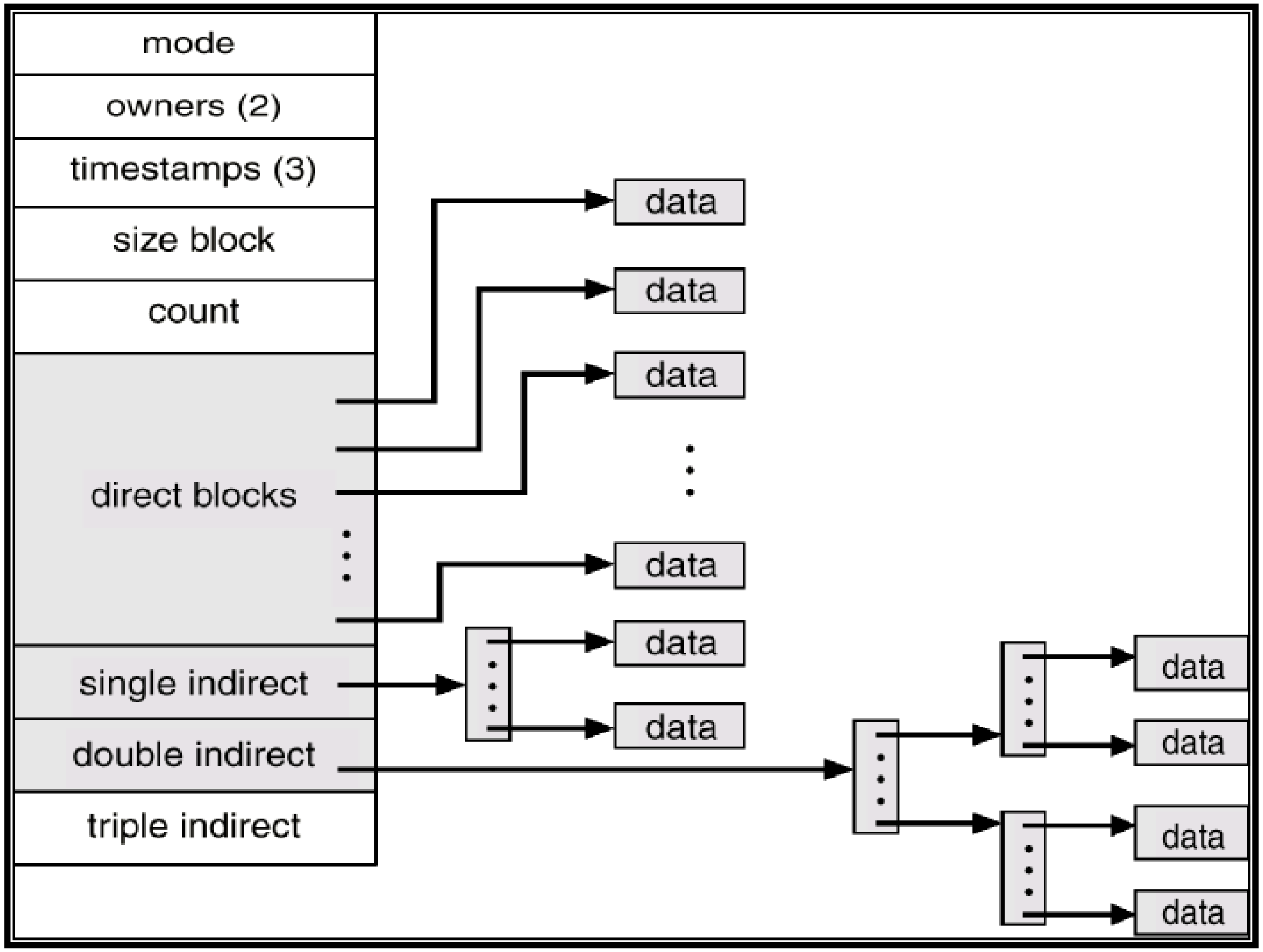
Example of Indexed Allocation



Indexed Allocation – Mapping (Cont’d)



Combined Scheme: UNIX (4K bytes per block)



Indexed Allocation

- Advantages
  - Supports direct access, without suffering from external fragmentation
  - I-node need only be in memory when the corresponding file is open

- Disadvantages
  - Space overhead for indexes:

    (1) Linked scheme: link several index blocks

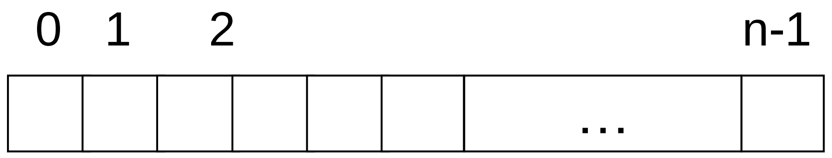
    (2) Multilevel index blocks

    (3) Combined scheme: UNIX

    - 12 direct blocks, single indirect block, double indirect block, triple indirect block

Free-Space Management

- Bit vector (*n* blocks)



bit[*i*] =

1 ⇒ block[*i*] free

0 ⇒ block[*i*] occupied

Block number calculation

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

Free-Space Management (Cont'd)

- Bit map requires extra space. Example:

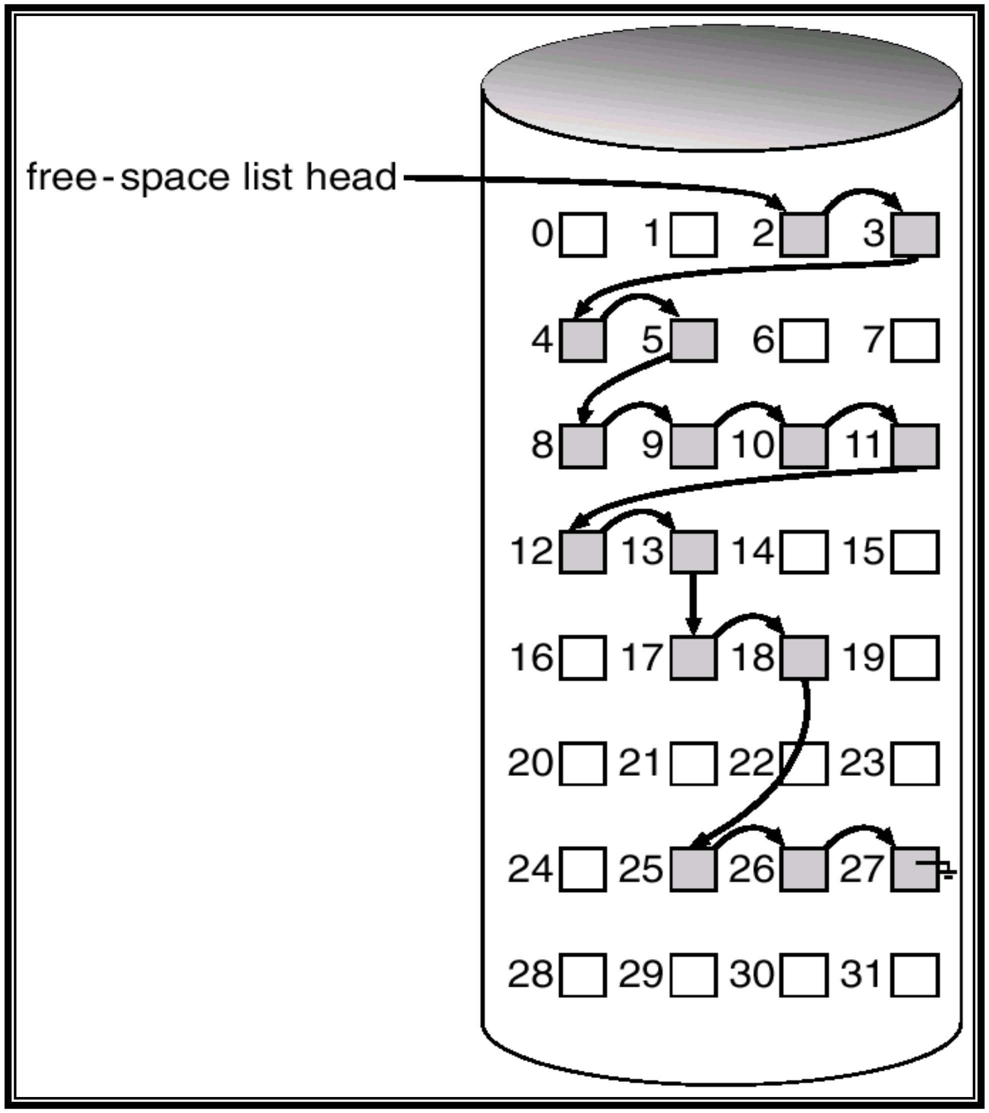
  block size = 2^{12} bytes

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

  $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
- Counting

Linked Free Space List on Disk



Free Space Management

- Grouping
  - Store the addresses of *n* free blocks in the first free block
  - The addresses of a large number of free blocks can be found quickly

- Counting
  - Keep the address of the free block and the number of free contiguous blocks
  - The length of the list becomes shorter and the count is generally greater than 1
    - Several contiguous blocks may be allocated or freed simultaneously

Efficiency and Performance

- Efficiency dependent on:
  - Disk allocation and directory algorithms
  - Types of data kept in file's directory entry
- Performance
  - Disk cache
    - separate section of main memory for frequently used blocks
  - Free-behind and Read-ahead
    - techniques to optimize sequential access
  - Improve PC performance by dedicating section of memory as virtual disk, or RAM disk

Buffer Cache

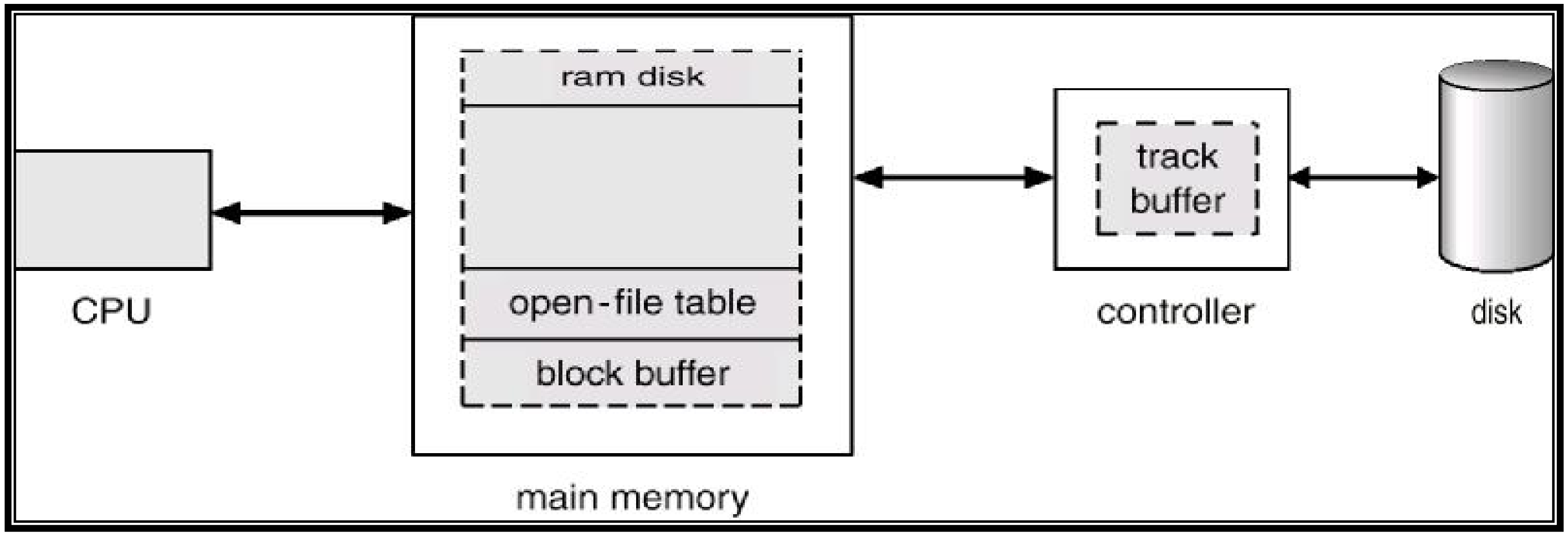
- Applications exhibit significant locality for reading and writing files

- Idea: cache file blocks in memory to capture locality in buffer cache (or disk cache)
  - Cache is system wide, used and shared by all processes
  - Reading from the cache makes a disk perform like memory
  - Even a 4MB cache can be very effective

- Issues
  - The buffer cache competes with VM
  - Like VM, it has limited size
  - Need replacement algorithms again

    (References are relatively infrequent, so it is feasible to keep all the blocks in exact LRU order)

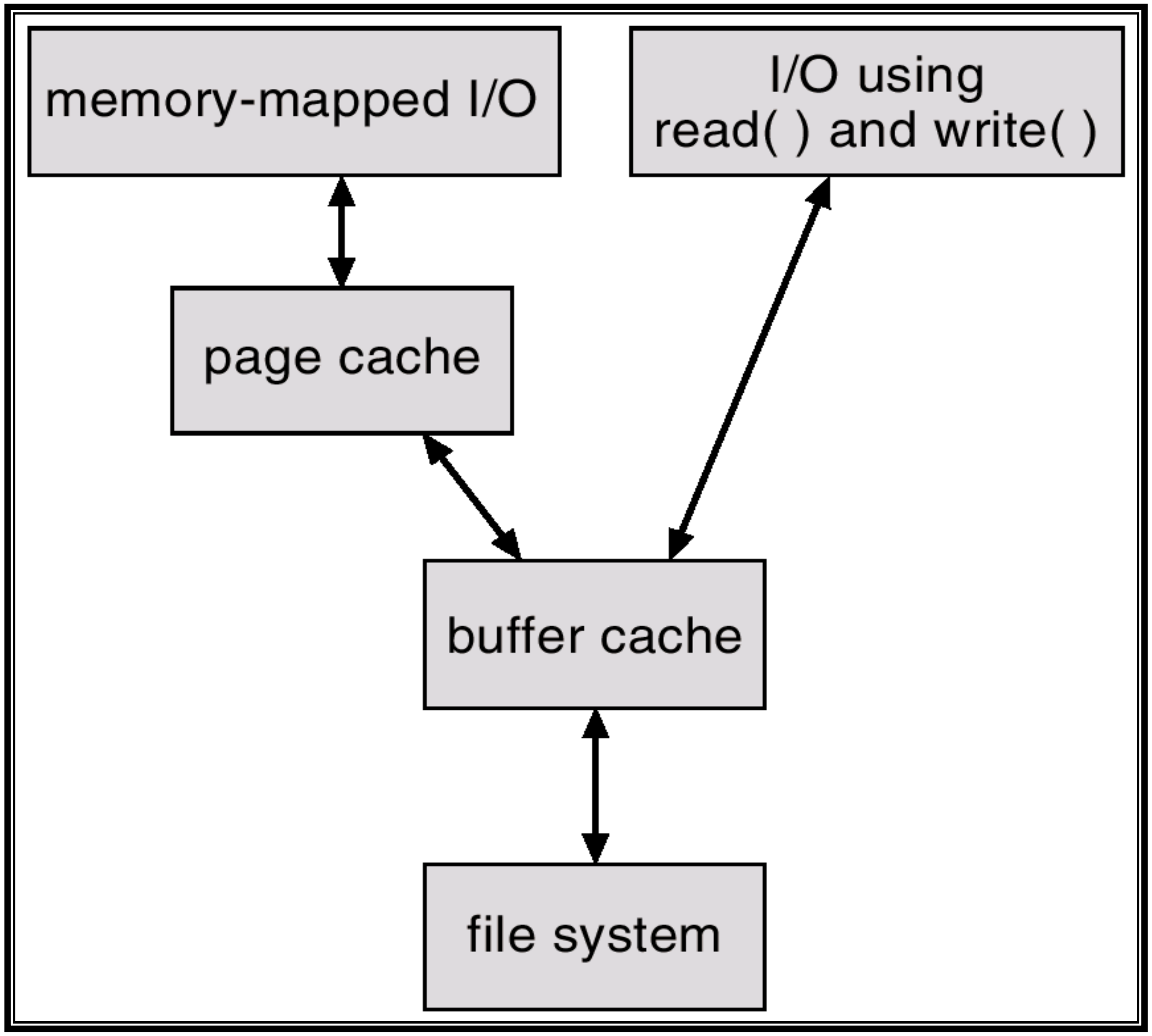
Various Disk-Caching Locations



Page Cache

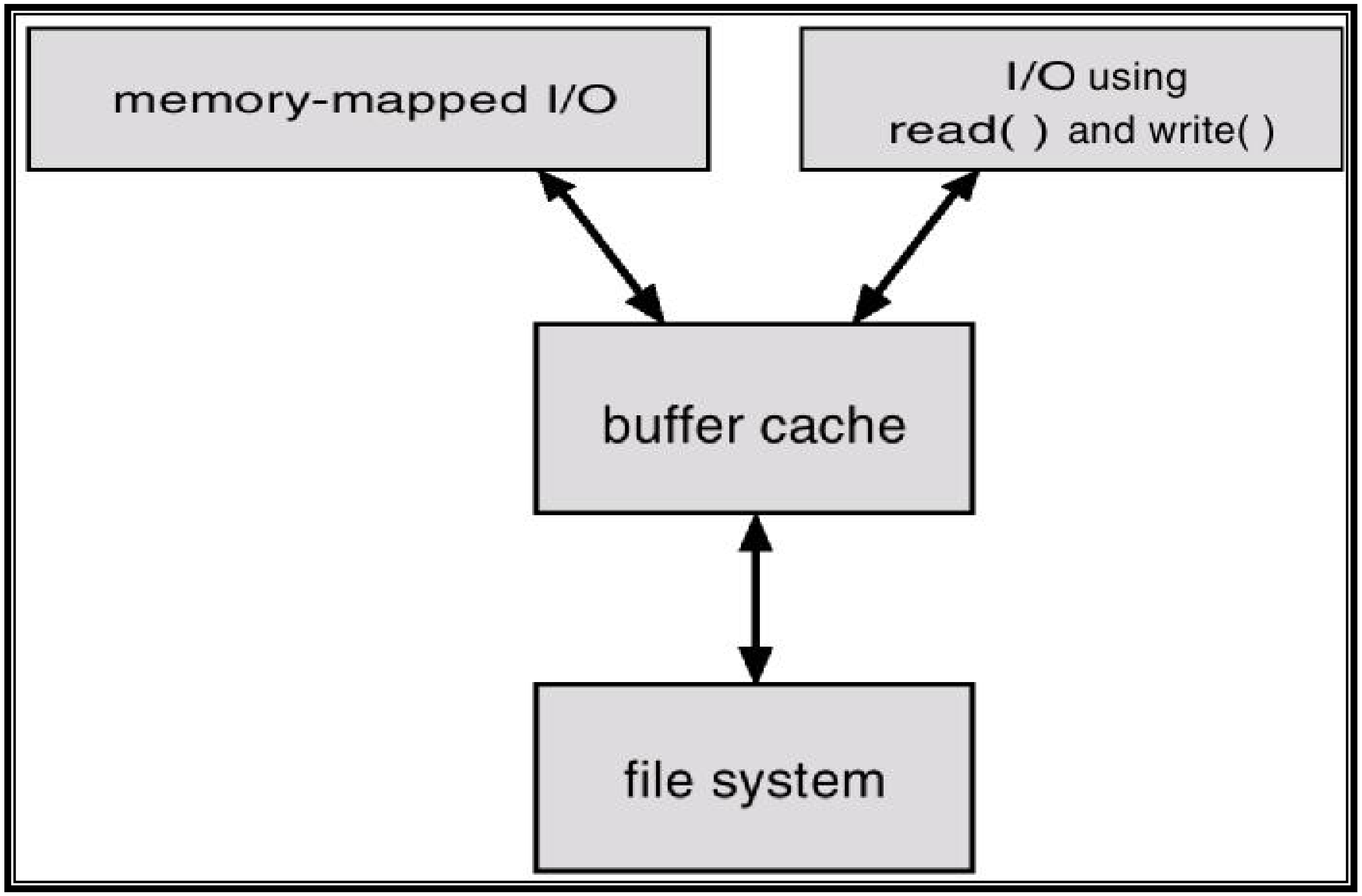
- A **page cache** caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure

I/O Without a Unified Buffer 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



Caching Writes

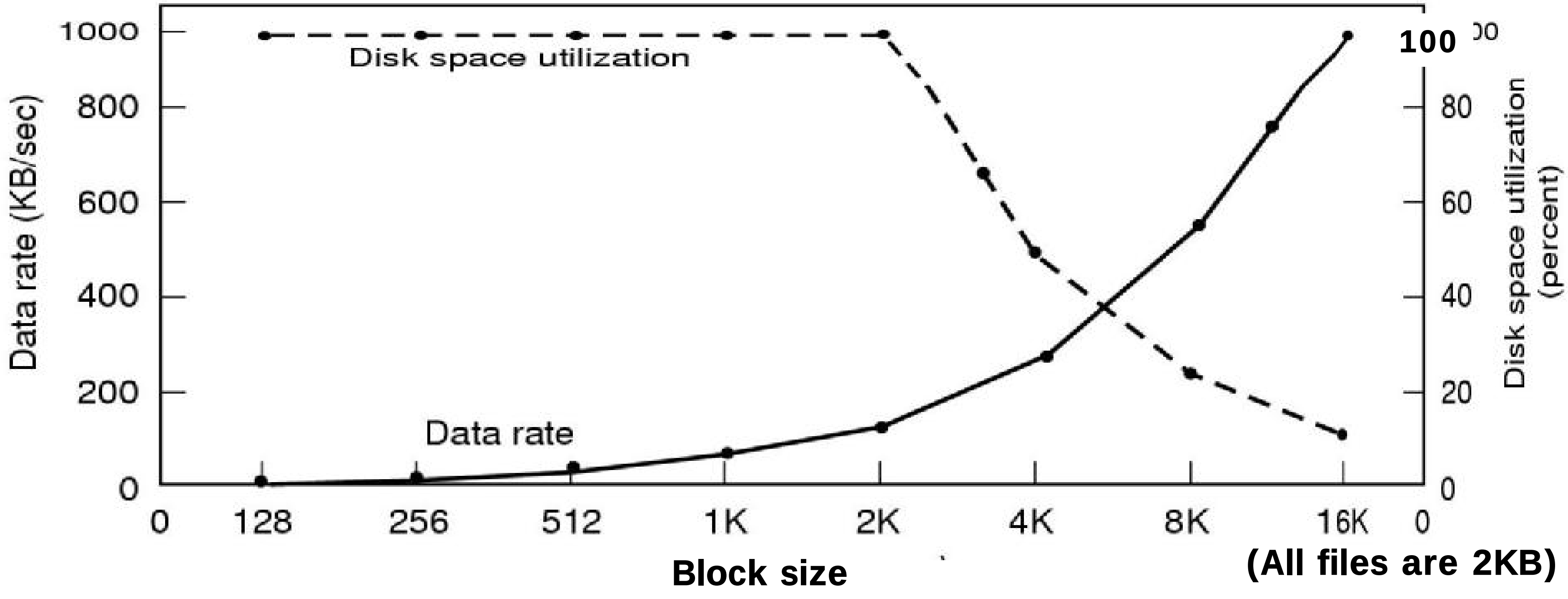
- Synchronous writes are very slow
- Asynchronous writes (or write-behind, write-back)
  - Maintain a queue of uncommitted blocks
  - Periodically flush the queue to disk
  - Unreliable: metadata requires synchronous writes (with small files, most writes are to metadata)

Read-Ahead

- File system predicts that the process will request next block
  - File system goes ahead and requests it from the disk
  - This can happen while the process is computing on previous block, overlapping I/O with execution
  - When the process requests block, it will be in cache
- Compliments the disk cache, which also is doing read ahead
- Very effective for sequentially accessed files
- File systems try to prevent blocks from being scattered across the disk during allocation or by restructuring periodically
- Cf) Free-behind

Block Size Performance vs. Efficiency

- Block size
  - Disk block size vs. file system block size
  - The median file size in UNIX is about 1KB



Recovery

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

Reliability

- File system consistency
  - File system can be left in an inconsistent state if cached blocks are not written out due to the system crash
  - It is especially critical if some of those blocks are i-node blocks, directory blocks, or blocks containing the free list
  - Most systems have a utility program that checks file system consistency
    - Windows: scandisk
    - UNIX: fsck

Log Structured File Systems

- **Log structured** (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

Log Structured File Systems

- Journaling file systems
  - Fsck'ing takes a long time, which makes the file system restart slow in the event of system crash
  - Record a log, or journal, of changes made to files and directories to a separate location (preferably a separate disk)
  - If a crash occurs, the journal can be used to undo any partially completed tasks that would leave the file system in an inconsistent state
  - IBM JFS for AIX, Linux

    Veritas VxFS for Solaris, HP-UX, Unixware, etc.

    SGI XFS for IRIX, Linux

    Reiserfs, ext3 for Linux

    NTFS for Windows

The Sun Network File System (NFS)

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol and Ethernet)

NFS (Cont'd)

- 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. The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local 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

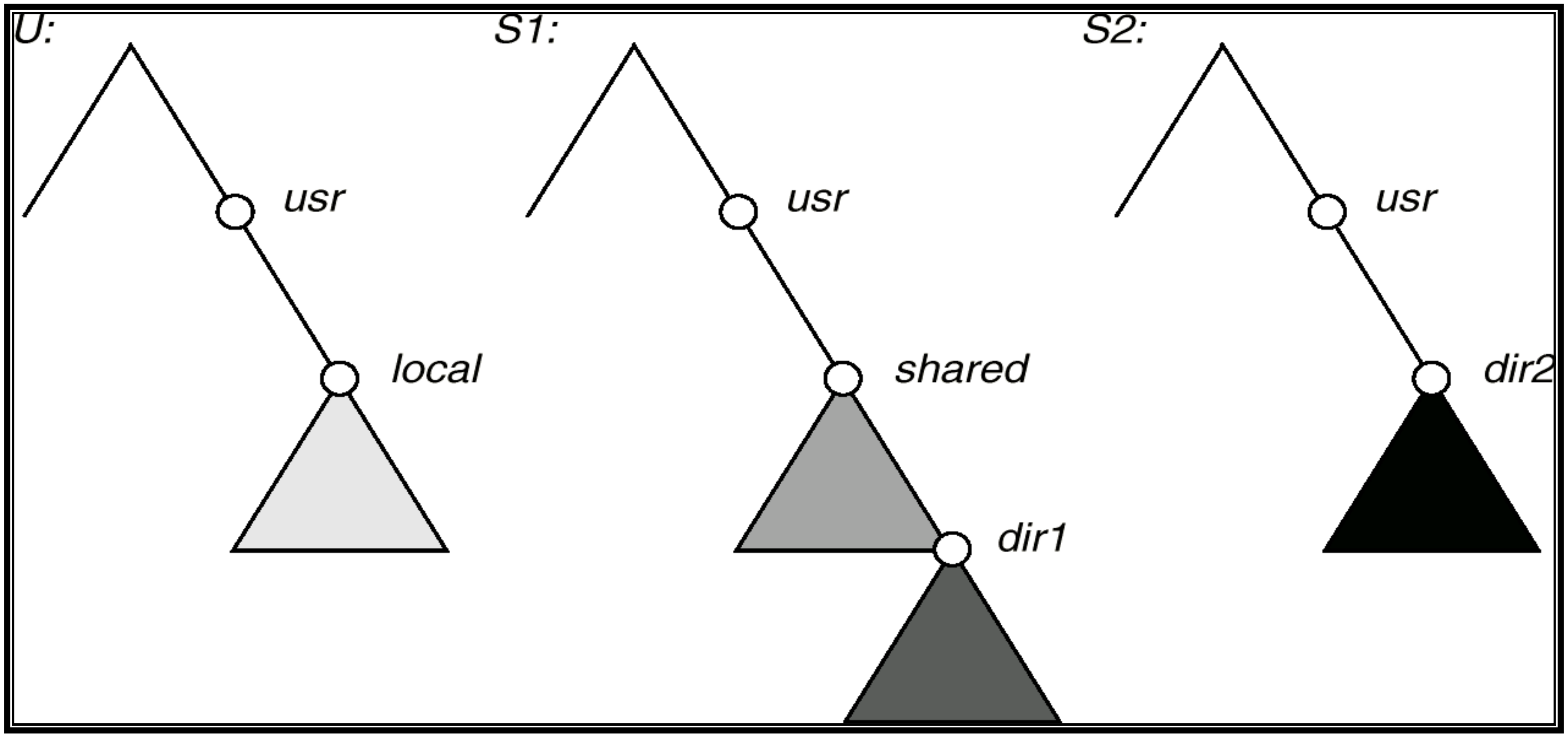
NFS (Cont'd)

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

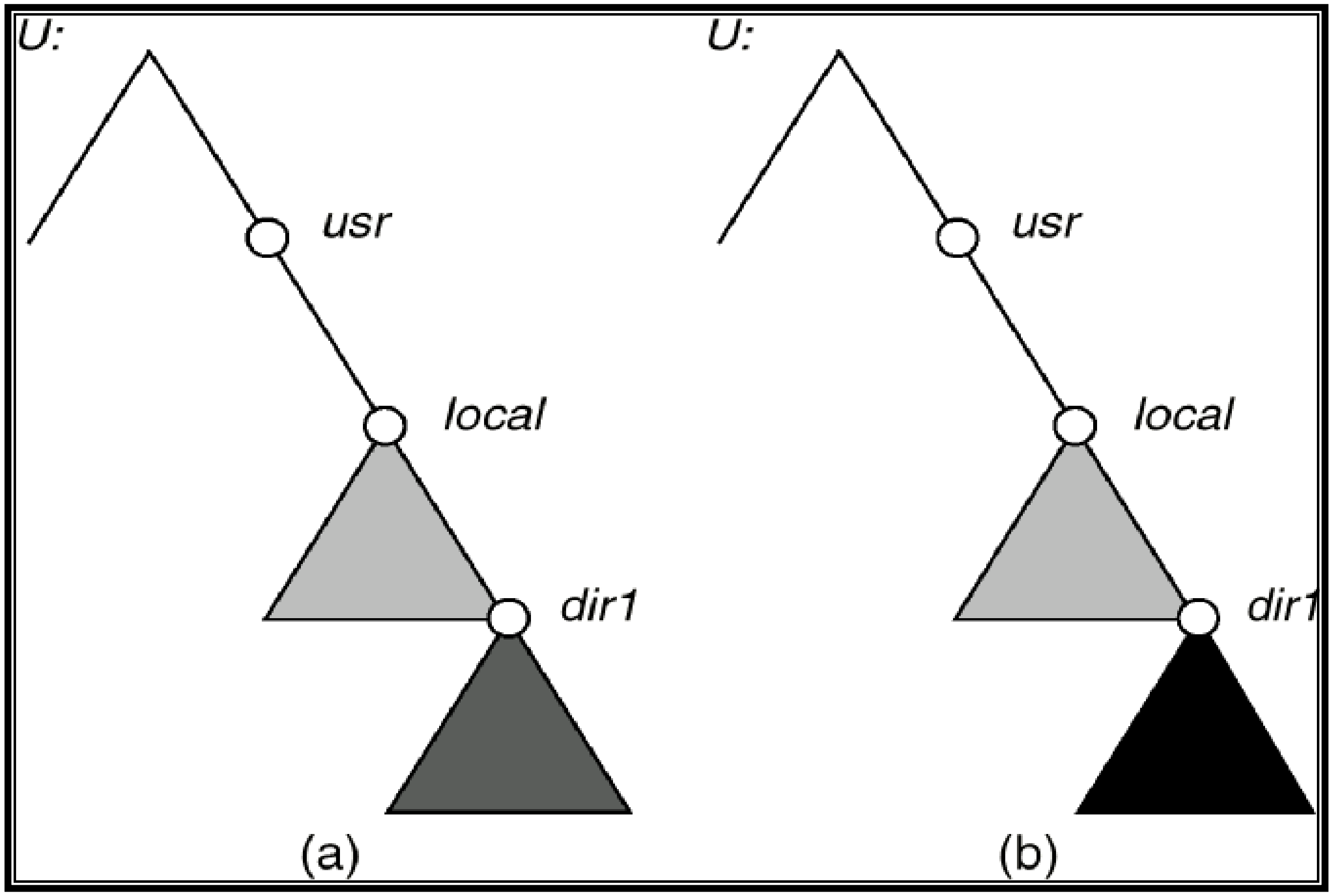
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces

- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services

Three Independent File Systems



Mounting in NFS



Mounts

Cascading mounts

NFS Mount Protocol

- Establishes initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
  - 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 for mounting, along with names of machines that are permitted to mount them
- Following a mount request that conforms to its export list, the server returns a *file handle* — a key for further accesses
- File handle
  - A file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user’s view and does not affect the server side

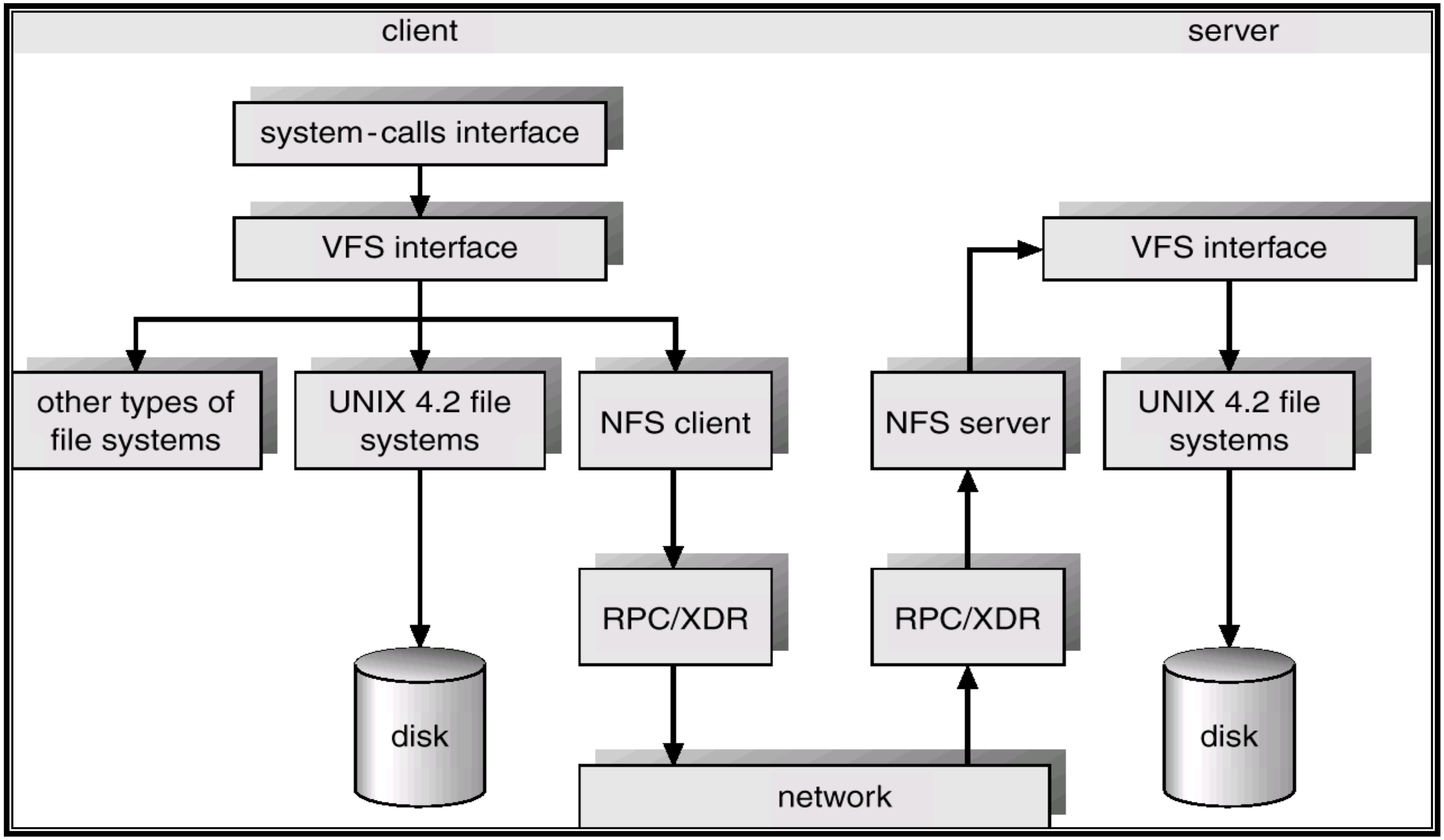
NFS Protocol

- Provides a set of remote procedure calls for remote file operations
- The procedures support the following operations:
  - searching for a file within a directory
  - 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
- 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 Architecture



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
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk

Appendix) Implementation Examples

Fast File System

- Fast file system (FFS)
  - The original Unix file system (70's) was very simple and straightforwardly implemented:
    - Easy to implement and understand
    - But very poor utilization of disk bandwidth (lots of seeking)
  - BSD Unix folks redesigned file system called FFS
    - McKusick, Joy, Fabry, and Leffler (mid 80's)
    - Now it is the file system from which all other UNIX file systems have been compared
  - The basic idea is aware of disk structure
    - Place related things on nearby cylinders to reduce seeks
    - Improved disk utilization, decreased response time

Fast File System (Cont'd)

- Data and i-node placement
  - Original Unix FS had two major problems:

  (1) Data blocks are allocated randomly in aging file systems
    - Blocks for the same file allocated sequentially when FS is new
    - As FS "ages" and fills, need to allocate blocks freed up when other files are deleted
    - Problem: Deleted files essentially randomly placed
    - So, blocks for new files become scattered across the disk

  (2) i-nodes are allocated far from blocks
    - All i-nodes at the beginning of disk, far from data
    - Traversing file name paths, manipulating files and directories require going back and forth from i-nodes to data blocks
  - Both of these problems generate many long seeks!

Fast File System (Cont'd)

- Cylinder groups
  - BSD FFS addressed these problems using the notion of a cylinder group
  - Disk partitioned into groups of cylinders
  - Data blocks from a file all placed in the same cylinder group
  - Files in same directory placed in the same cylinder group
  - I-nodes for files allocated in the same cylinder group as file's data blocks

Fast File System (Cont'd)

- Free space reserve
  - If the number of free blocks falls to zero, the file system throughput tends to be cut in half, because of the inability to localize blocks in a file
  - A parameter, called free space reserve, gives the minimum acceptable percentage of file system blocks that should be free
  - If the number of free blocks drops below this level, only the system administrator can continue to allocate blocks
  - Normally 10%; this is why df may report > 100%

Fast File System (Cont’d)

- Fragments
  - Small blocks (1KB) caused two problems:
    - Low bandwidth utilization
    - Small max file size (function of block size)
  - FFS fixes by using a larger block (4KB)
    - Allows for very large files (1MB only uses 2 level indirect)
    - But introduces internal fragmentation: there are many small files (i.e., < 4KB)
  - FFS introduces “fragments” to fix internal fragmentation
    - Allows the division of a block into one or more fragments (1K pieces of a block)

Fast File System (Cont’d)

- Media failures
  - Replicate master block (superblock)
- File system parameterization
  - Parameterize according to disk and CPU characteristics
    - Maximum blocks per file in a cylinder group
    - Minimum percentage of free space
    - Sectors per track
    - Rotational delay between contiguous blocks
    - Tracks per cylinder, etc.
  - Skip according to rotational rate and CPU latency

Linux Ext2 File System

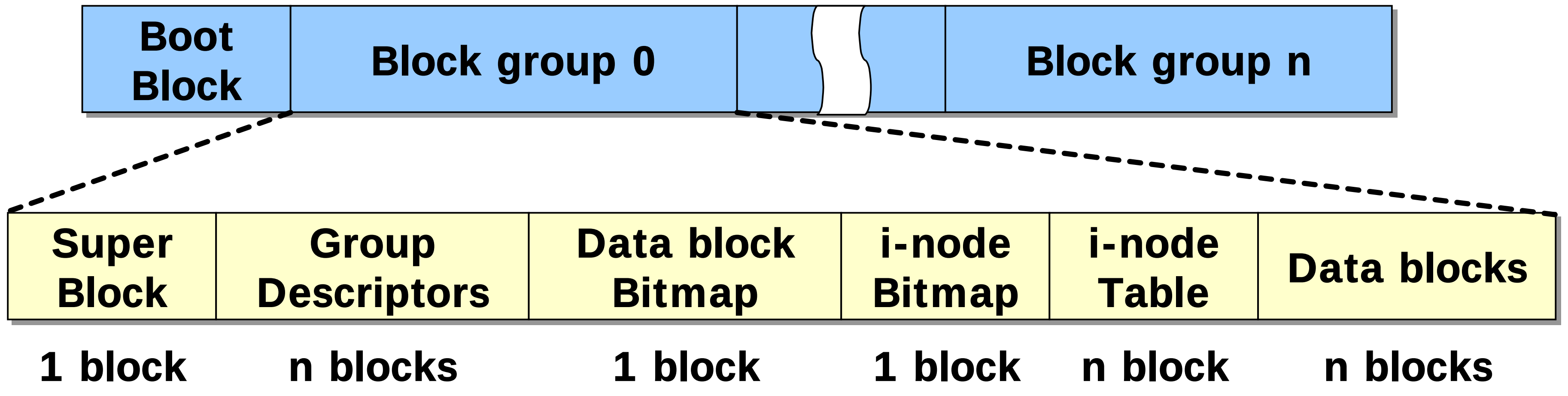
- History
  - Evolved from Minix file system
    - Block addresses are stored in 16bit integers – maximal file system size is restricted to 64MB
    - Directories contain fixed-size entries and the maximal file name was 14 characters
  - Virtual File System (VFS) is added
  - Extended Filesystem (Ext FS), 1992
    - Added to Linux 0.96c
    - Maximum file system size was 2GB, and the maximal file name size was 255 characters
  - Second Extended File-system (Ext2 FS), 1994

Linux Ext2 File System (Cont'd)

- Ext2 Features
  - Configurable block sizes (from 1KB to 4KB)
    - depending on the expected average file size
  - Configurable number of i-nodes
    - depending on the expected number of files
  - Partitions disk blocks into groups
    - lower average disk seek time
  - Preallocates disk data blocks to regular files
    - reduces file fragmentation
  - Fast symbolic links
    - If the pathname of the symbolic link has 60 bytes or less, it is stored in the i-node
  - Automatic consistency check at boot time

Linux Ext2 File System (Cont'd)

- Disk layout
  - Boot block
    - reserved for the partition boot sector
  - Block group
    - Similar to the cylinder group in FFS
    - All the block groups have the same size and are stored sequentially



Linux Ext2 File System (Cont’d)

- Block group
  - Superblock: stores file system metadata
    - Total number of i-nodes
    - File system size in blocks
    - Free blocks / i-nodes counter
    - Number of blocks / i-nodes per group
    - Block size, etc.
  - Group descriptor
    - Number of free blocks / i-nodes / directories in the group
    - Block number of block / i-node bitmap, etc.
  - Both the superblock and the group descriptors are duplicated in each block group
    - Only those in block group 0 are used by the kernel
    - fsck copies them into all other block groups
    - When data corruption occurs, fsck uses old copies to bring the file system back to a consistent state

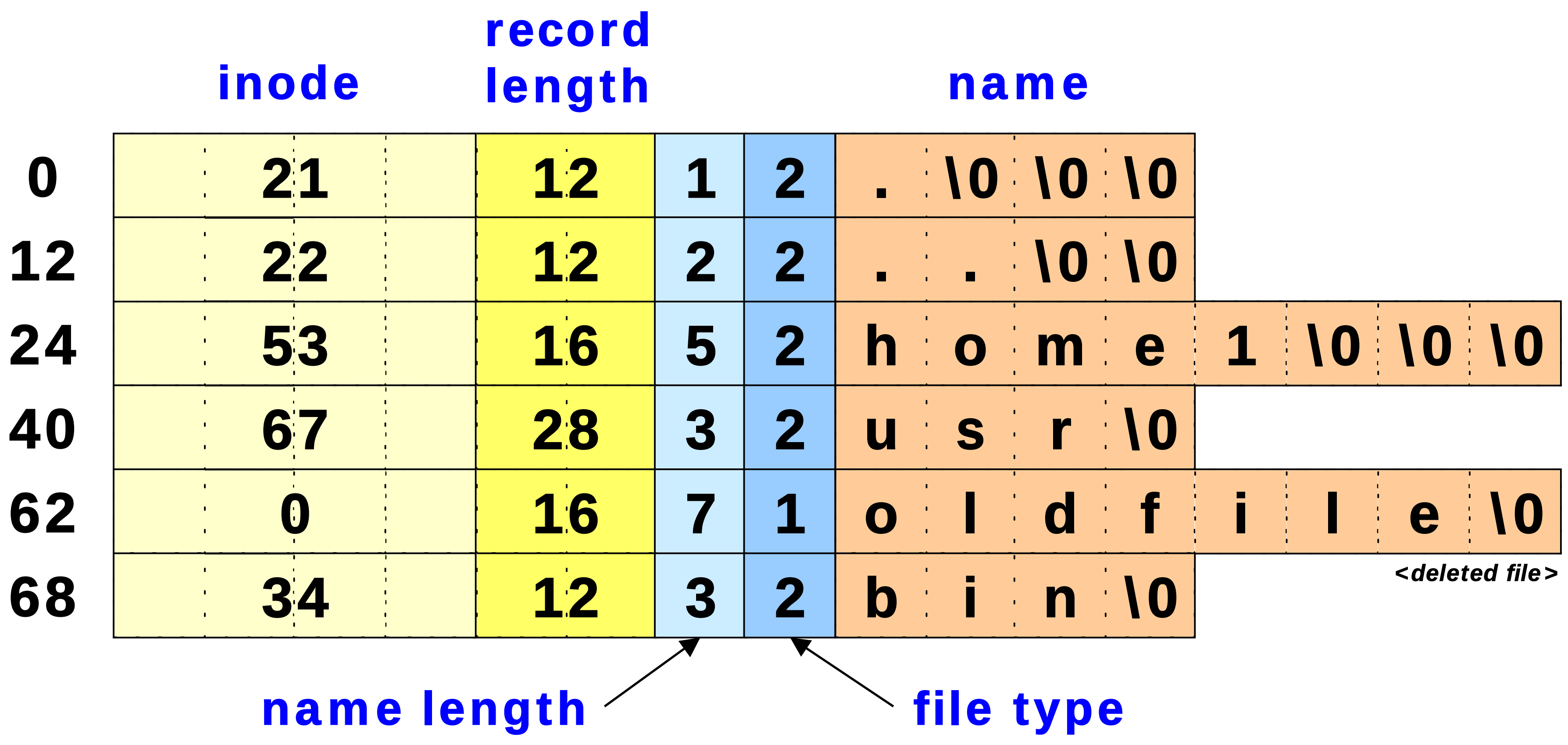
Linux Ext2 File System (Cont’d)

- Block group size
  - The block bitmap must be stored in a single block
    - In each block group, there can be at most 8xb blocks, where b is the block size in bytes
  - The smaller the block size, the larger the number of block groups
  - Example: 8GB Ext2 partition with 4KB block size
    - Each 4KB block bitmap describes 32K data blocks

      = 32K * 4KB = 128MB
    - At most 64 block groups are needed

Linux Ext2 File System (Cont'd)

- Directory structure



	inode	record length	name length	file type	name
0	21	12	1	2	. \0 \0 \0
12	22	12	2	2	. . \0 \0
24	53	16	5	2	h o m e 1 \0 \0 \0
40	67	28	3	2	u s r \0
62	0	16	7	1	o l d f i l e \0
68	34	12	3	2	b i n \0