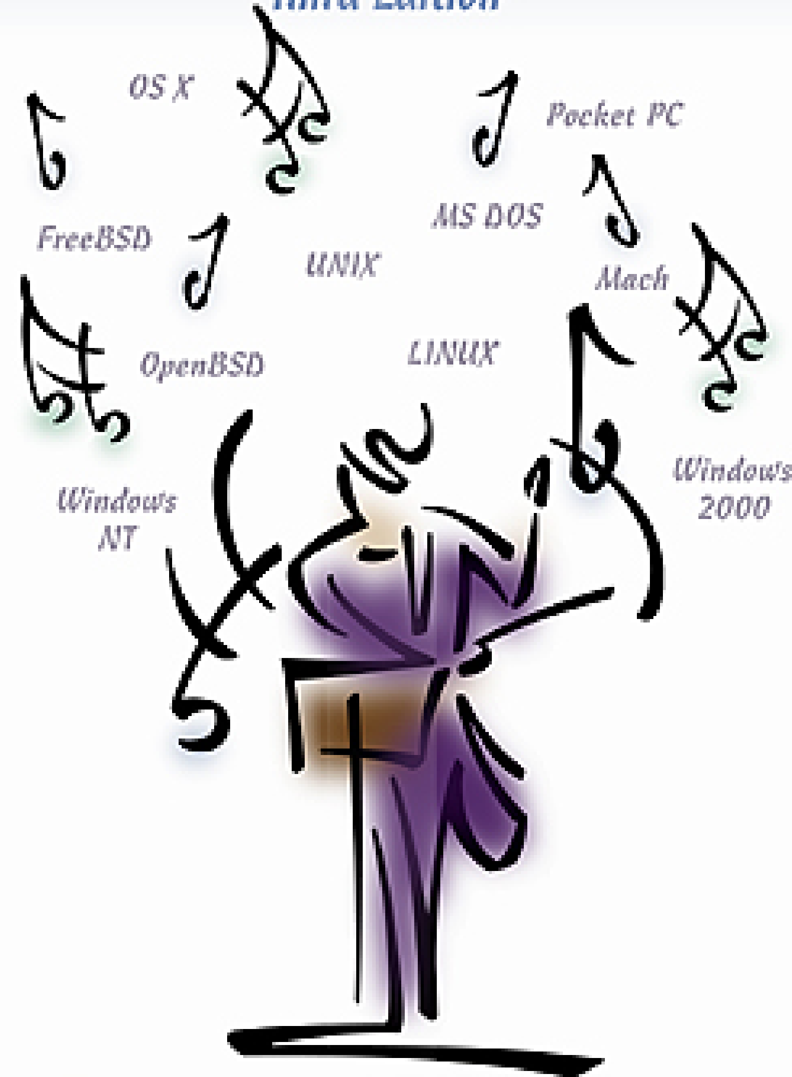
# OPERATING SYSTEMS

*Third Edition*





GARY NUTT

# File Management

Copyright © 2004 Pearson Education, Inc.

# Fig 13-2: The External View of the File Manager

Application Program

mount()
write()
close()
open()
read()
lseek()

CreateFile()
WriteFile()
CloseHandle()
ReadFile()
SetFilePointer()

File Mgr
Device Mgr
Memory Mgr
Process Mgr

UNIX

File Mgr
Device Mgr
Memory Mgr
Process Mgr

Windows

Hardware

Copyright © 2004 Pearson Education, Inc.

# Why Programmers Need Files

HTML Editor

```
<head>
…
</head>
<body>
…
</body>
```

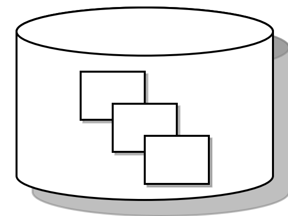
Web Browser

File Manager

foo.html

```
<head>
…
</head>
<body>
…
</body>
```

File Manager



- Persistent storage
- Shared device

- Structured information
- Can be read by any applic
  - Accessibility
  - Protocol

Copyright © 2004 Pearson Education, Inc.

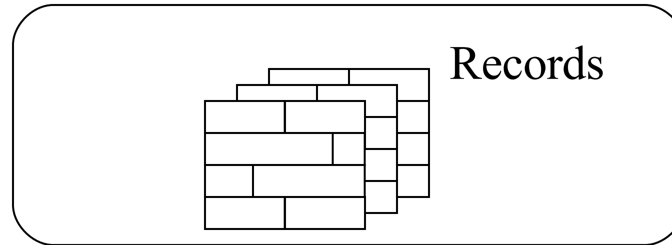
# File Management

- *File* is a named, ordered collection of information
- The file manager administers the collection by:
  - Storing the information on a device
  - Mapping the block storage to a logical view
  - Allocating/deallocating storage
  - Providing file directories
- What abstraction should be presented to programmer?

Copyright © 2004 Pearson Education, Inc.

# Information Structure

Applications



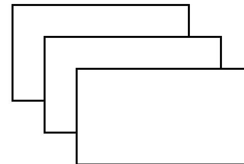

Structured Record Files

Record-Stream Translation

Byte Stream Files

Stream-Block Translation

Storage device



Copyright © 2004 Pearson Education, Inc.

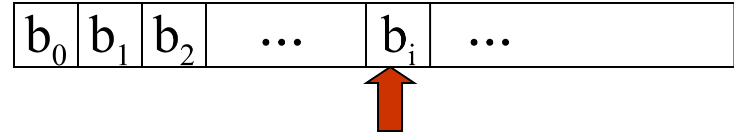
# Byte Stream File Interface

```
fileID = open(fileName)
close(fileID)
read(fileID, buffer, length)
write(fileID, buffer, length)
seek(fileID, filePosition)
```

Copyright © 2004 Pearson Education, Inc.

# Low Level Files

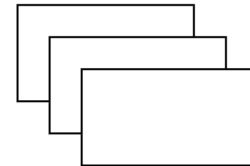
```
fid = open("fileName",…);
…
read(fid, buf, buflen);
…
close(fid);
```




```
int open(…) {…}
int close(…) {…}
int read(…) {…}
int write(…) {…}
int seek(…) {…}
```

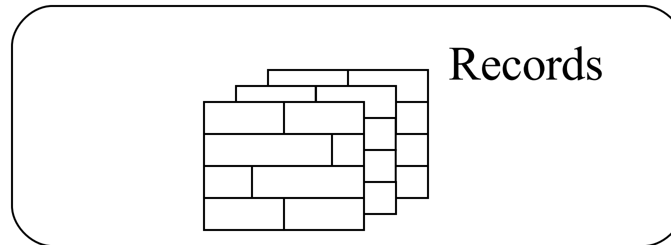
Stream-Block Translation

Storage device response to commands

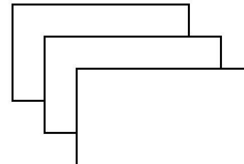


Copyright © 2004 Pearson Education, Inc.

# Structured Files

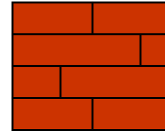



Record-Block Translation



Copyright © 2004 Pearson Education, Inc.

# Record-Oriented Sequential Files

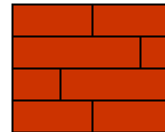


Logical Record

```
fileID = open(fileName)
close(fileID)
getRecord(fileID, record)
putRecord(fileID, record)
seek(fileID, position)
```

Copyright © 2004 Pearson Education, Inc.

# Record-Oriented Sequential Files

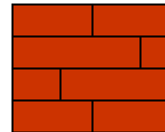


Logical Record

H byte header k byte logical record

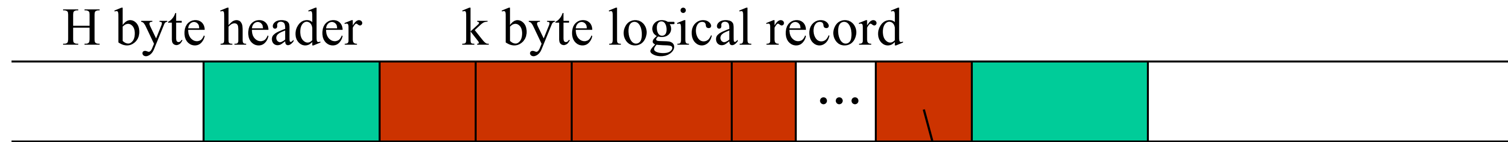
Copyright © 2004 Pearson Education, Inc.

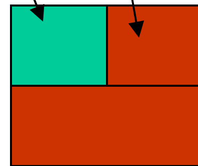
# Record-Oriented Sequential Files

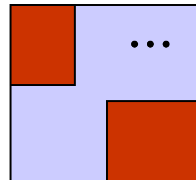


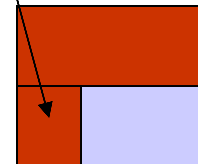
Logical Record

H byte header    k byte logical record





Physical Storage Blocks

Fragment

Copyright © 2004 Pearson Education, Inc.

# Electronic Mail Example

```
struct message {
/* The mail message */
   address to;
   address from;
   line subject;
   address cc;
   string body;
};
struct message *getRecord(void) {
    struct message *msg;
    msg = allocate(sizeof(message));
    msg->to = getAddress(...);
    msg->from = getAddress(...);
    msg->cc = getAddress(...);
    msg->subject = getLine();
    msg->body = getString();
    return(msg);
}
```

```
putRecord(struct message *msg) {
    putAddress(msg->to);
    putAddress(msg->from);
    putAddress(msg->cc);
    putLine(msg->subject);
    putString(msg->body);
}
```

Copyright © 2004 Pearson Education, Inc.

# Indexed Sequential File

- Suppose we want to directly access records
- Add an *index* to the file

```
fileID = open(fileName)
close(fileID)
getRecord(fileID, index)
index = putRecord(fileID, record)
deleteRecord(fileID, index)
```

Copyright © 2004 Pearson Education, Inc.

# Indexed Sequential File (cont)

Application structure

| Account # | Index |
|---|---|
| 012345 | |
| 123456 | i |
| 294376 | k |
| ... | |
| 529366 | j |
| ... | |
| 965987 | |

index = i

index = k

index = j

Copyright © 2004 Pearson Education, Inc.

# More Abstract Files

- Inverted files
  - *System index* for each datum in the file
- Databases
  - More elaborate indexing mechanism
  - DDL & DML
- Multimedia storage
  - Records contain radically different types
  - Access methods must be general

Copyright © 2004 Pearson Education, Inc.

# Implementing Low Level Files

- Secondary storage device contains:
  - Volume directory (sometimes a root directory for a file system)
  - External file descriptor for each file
  - The file contents
- Manages blocks
  - Assigns blocks to files (descriptor keeps track)
  - Keeps track of available blocks
- Maps to/from byte stream

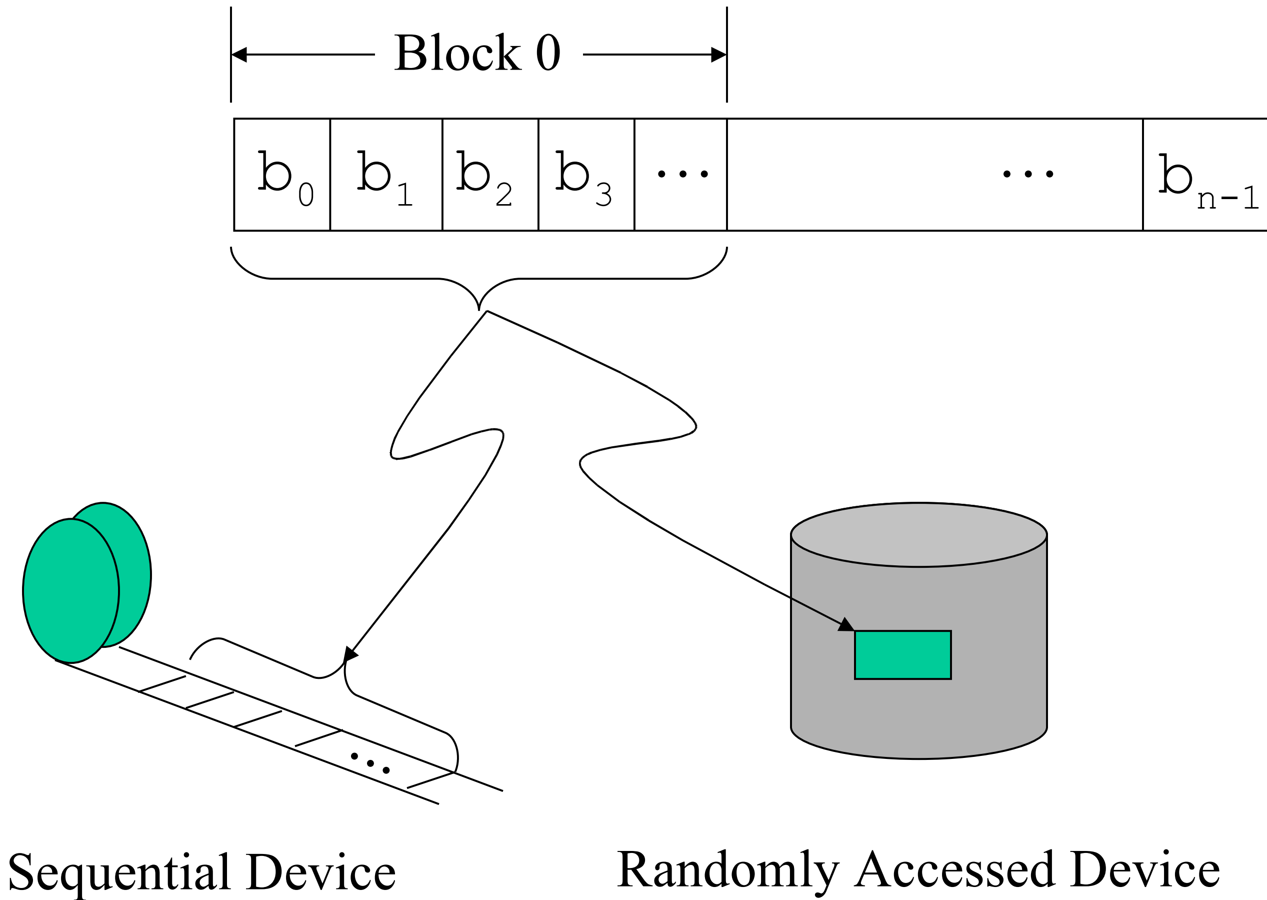
Copyright © 2004 Pearson Education, Inc.

# Disk Organization



Copyright © 2004 Pearson Education, Inc.

# Low-level File System Architecture




Copyright © 2004 Pearson Education, Inc.

# File Descriptors

- External name
- Current state
- Sharable
- Owner
- User
- Locks
- Protection settings
- Length
- Time of creation
- Time of last modification
- Time of last access
- Reference count
- Storage device details

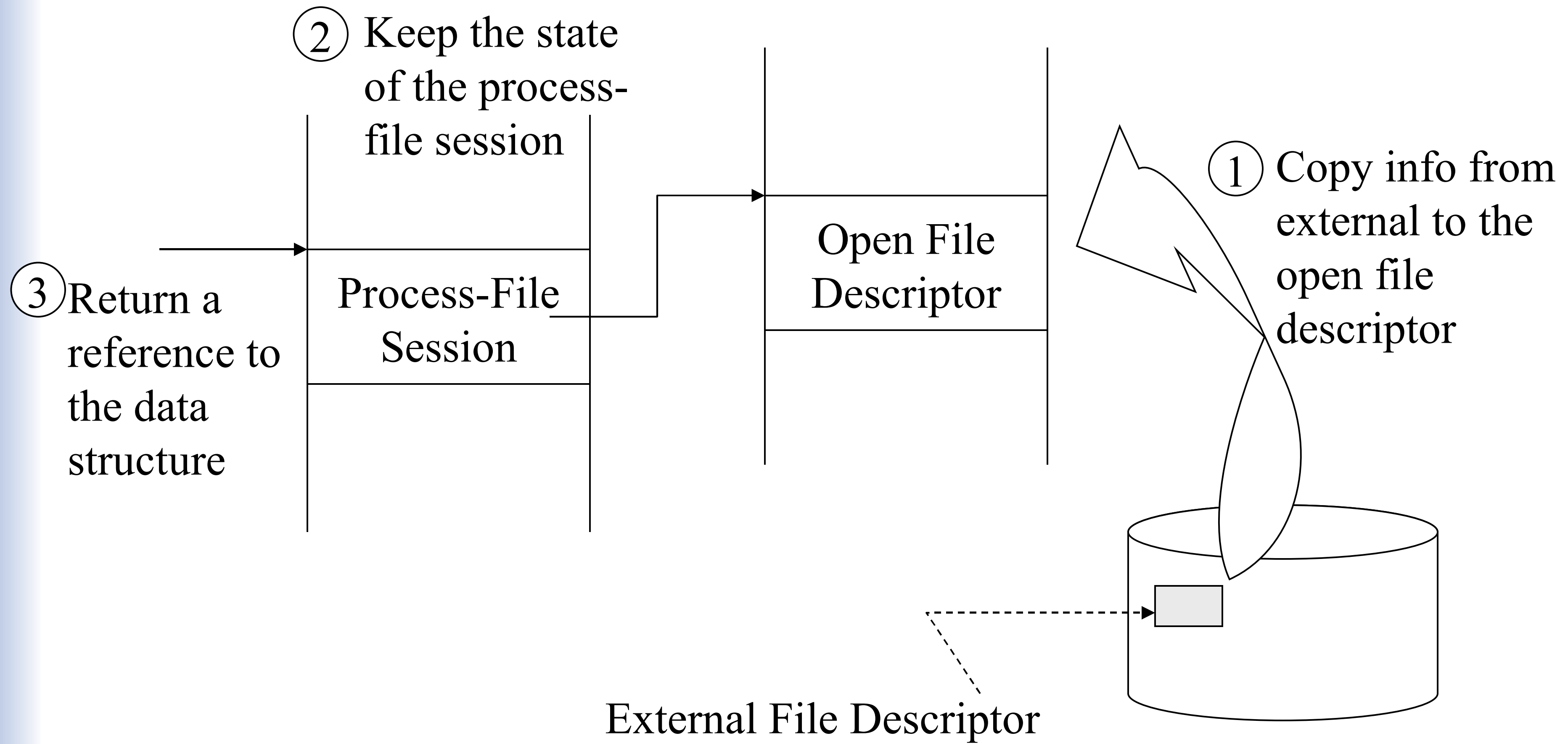
Copyright © 2004 Pearson Education, Inc.

# An `open()` Operation

- Locate the on-device (external) file descriptor
- Extract info needed to read/write file
- Authenticate that process can access the file
- Create an internal file descriptor in primary memory
- Create an entry in a “per process” open file status table
- Allocate resources, e.g., buffers, to support file usage

Copyright © 2004 Pearson Education, Inc.

# File Manager Data Structures




Copyright © 2004 Pearson Education, Inc.

# Opening a UNIX File

```
fid = open("fileA", flags);
…
read(fid, buffer, len);
```

| | |
|---|---|
| 0 | stdin |
| 1 | stdout |
| 2 | stderr |
| 3 | ... |
| | |

Open File Table

File structure

On-Device File Descriptor

inode

Internal File Descriptor

Copyright © 2004 Pearson Education, Inc.

# Block Management

- The job of selecting & assigning storage blocks to the file
- For a fixed sized file of k blocks
  - File of length m requires $N = \quad m/k \quad$ blocks
  - Byte $b_i$ is stored in block $\quad i/k$
- Three basic strategies:
  - Contiguous allocation
  - Linked lists
  - Indexed allocation

Copyright © 2004 Pearson Education, Inc.

# Contiguous Allocation

- Maps the N blocks into N contiguous blocks on the secondary storage device
- Difficult to support dynamic file sizes

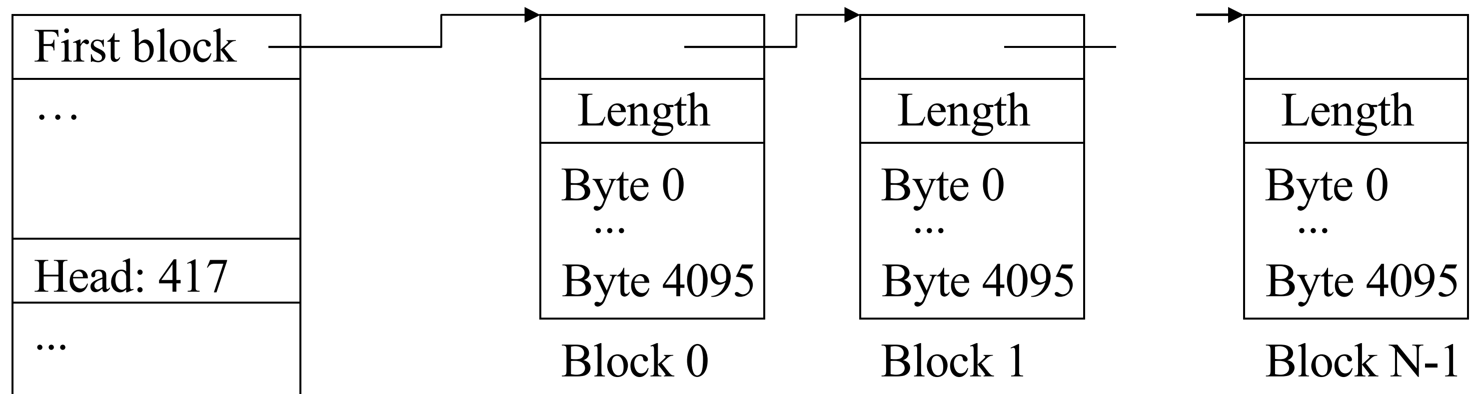
File descriptor

| | |
|---|---|
| Head position | 237 |
| … | |
| First block | 785 |
| Number of blocks | 25 |

Copyright © 2004 Pearson Education, Inc.

# Linked Lists

- Each block contains a header with
  - Number of bytes in the block
  - Pointer to next block
- Blocks need not be contiguous
- Files can expand and contract
- Seeks can be slow




Copyright © 2004 Pearson Education, Inc.

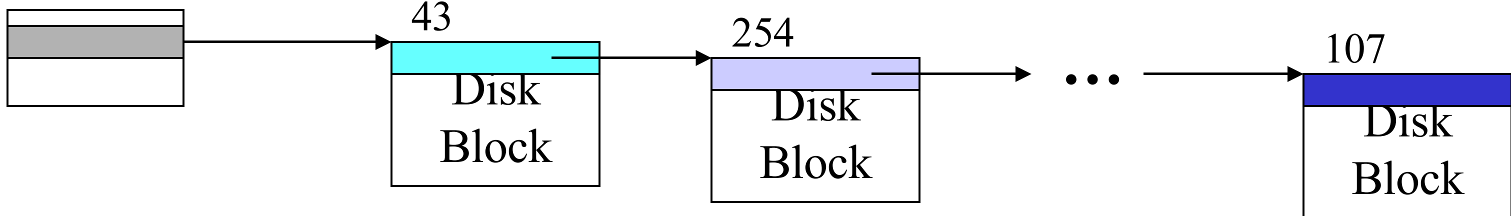
# Indexed Allocation

- Extract headers and put them in an index
- Simplify seeks
- May link indices together (for large files)



Copyright © 2004 Pearson Education, Inc.

# DOS FAT Files






Copyright © 2004 Pearson Education, Inc.

# UNIX Files

Copyright © 2004 Pearson Education, Inc.

# Unallocated Blocks

- How should unallocated blocks be managed?
- Need a data structure to keep track of them
  - Linked list
    - Very large
    - Hard to manage spatial locality
  - Block status map ("disk map")
    - Bit per block
    - Easy to identify nearby free blocks
    - Useful for disk recovery

Copyright © 2004 Pearson Education, Inc.

# Marshalling the Byte Stream

- Must read at least one buffer ahead on input
- Must write at least one buffer behind on output
- Seek flushing the current buffer and finding the correct one to load into memory
- Inserting/deleting bytes in the interior of the stream

Copyright © 2004 Pearson Education, Inc.

# Full Block Buffering

- Storage devices use block I/O
- Files place an explicit order on the bytes
- Therefore, it is possible to predict what is likely to be read after $byte_i$
- When file is opened, manager **reads** as many blocks **ahead** as feasible
- After a block is logically written, it is queued for **writing behind**, whenever the disk is available
- Buffer pool – usually variably sized, depending on virtual memory needs
  - Interaction with the device manager and memory manager

Copyright © 2004 Pearson Education, Inc.

# Directories

- A set of logically associated files and sub directories
- File manager provides set of controls:
  - `enumerate`
  - `copy`
  - `rename`
  - `delete`
  - `traverse`
  - etc.

# Directory Structures

- How should files be organized within directory?
  - Flat name space
    - All files appear in a single directory
  - Hierarchical name space
    - Directory contains files and subdirectories
    - Each file/directory appears as an entry in exactly one other directory -- a *tree*
    - Popular variant: All directories form a tree, but a file can have multiple parents.

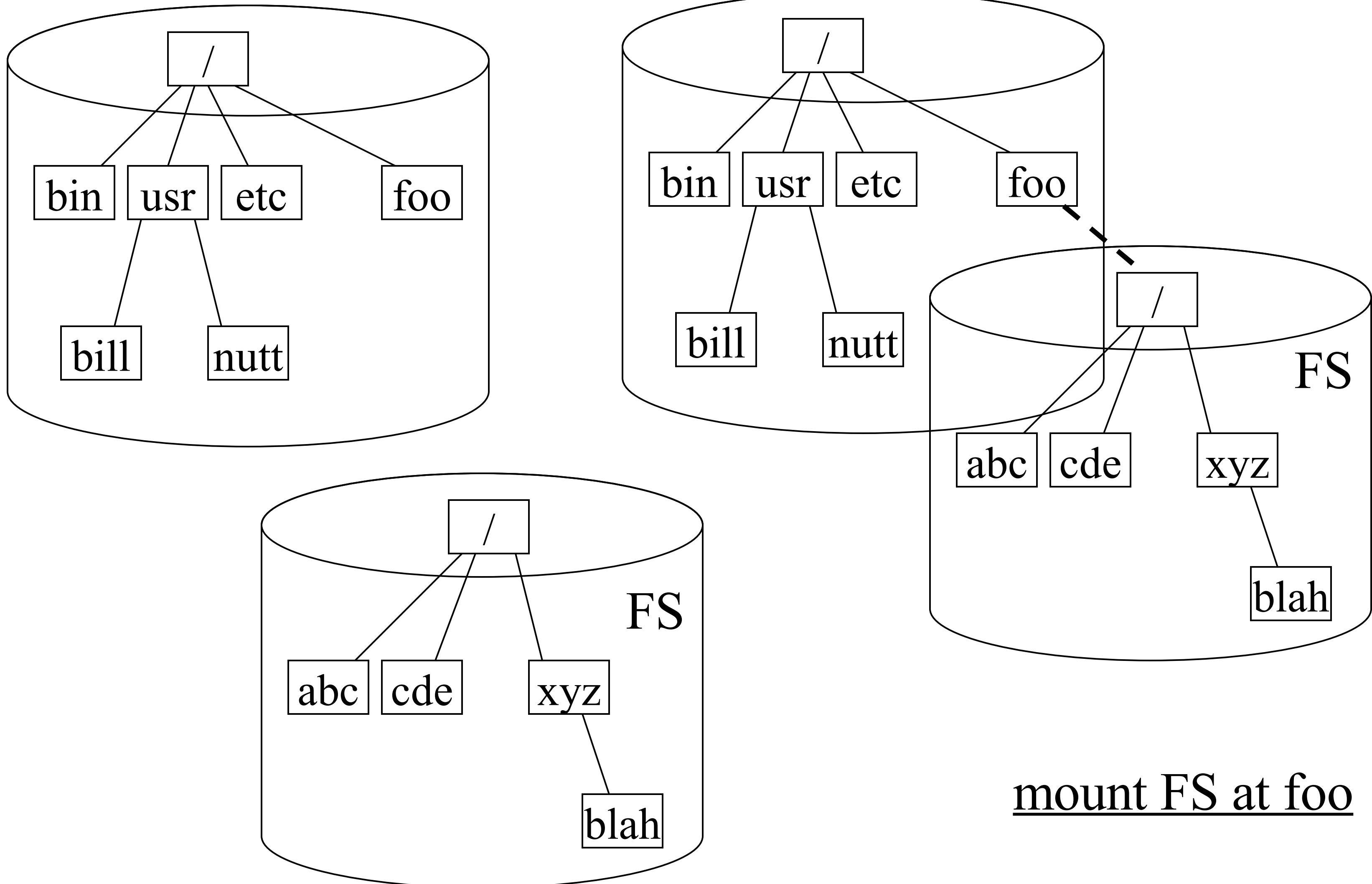
Copyright © 2004 Pearson Education, Inc.

# Directory Implementation

- Device Directory
  - A device can contain a collection of files
  - Easier to manage if there is a root for every file on the device -- the device root directory
- File Directory
  - Typical implementations have directories implemented as a file with a special format
  - Entries in a file directory are handles for other files (which can be files or subdirectories)

Copyright © 2004 Pearson Education, Inc.

# UNIX `mount` Command




Copyright © 2004 Pearson Education, Inc.

# VFS-based File Manager

Exports OS-specific API

File System Independent Part of File Manager

Virtual File System Switch

MS-DOS Part of File Manager

ISO 9660 Part of File Manager

…

ext2 Part of File Manager

Copyright © 2004 Pearson Education, Inc.