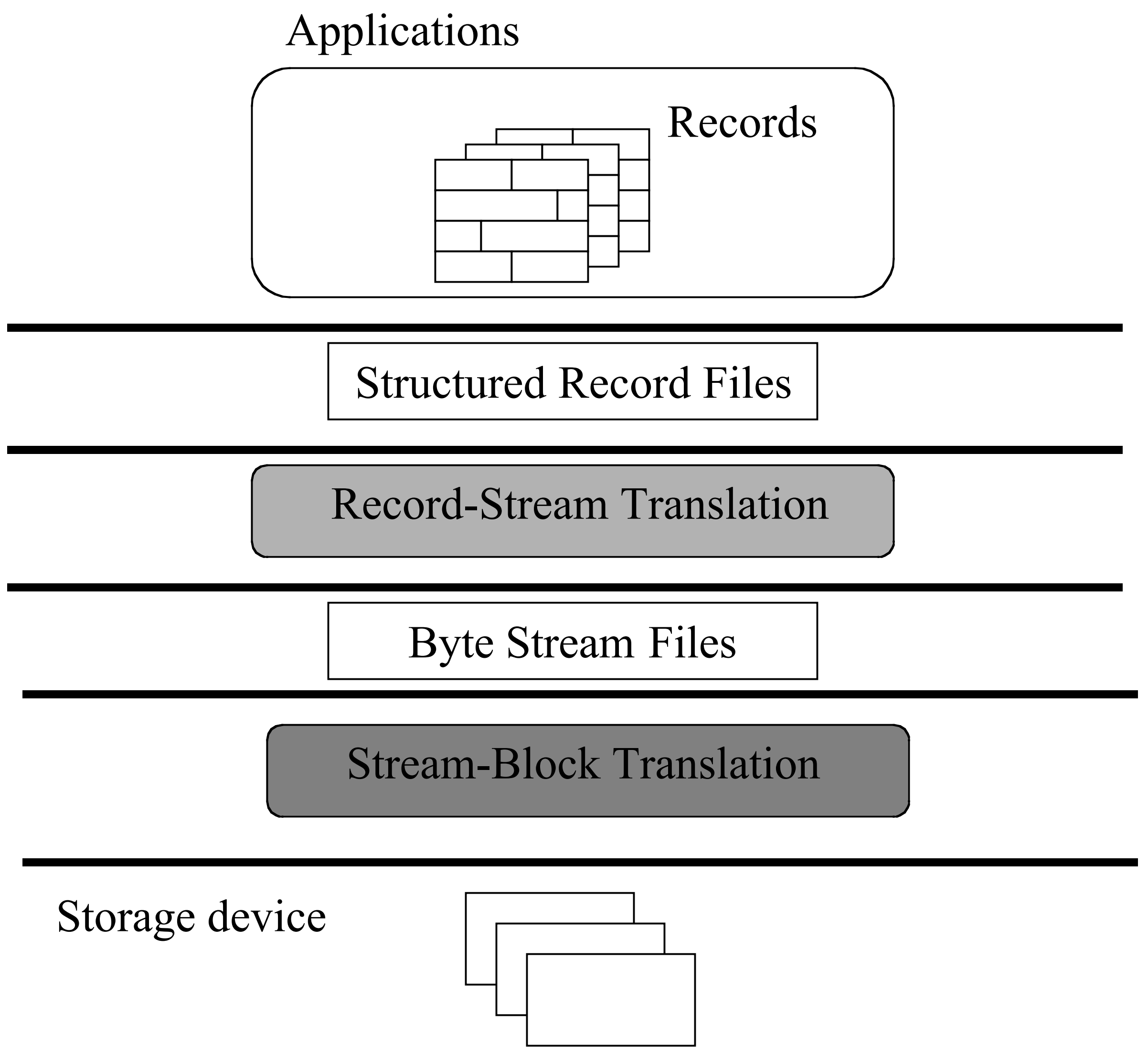
File Management

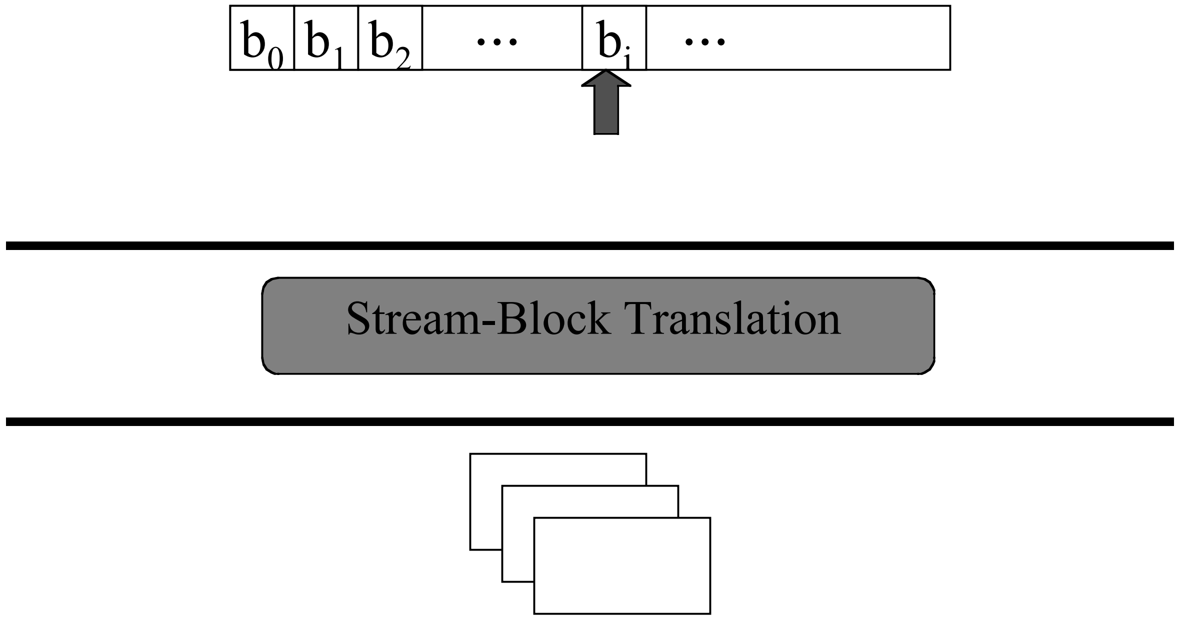
File Management

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

Information Structure



Low Level Files



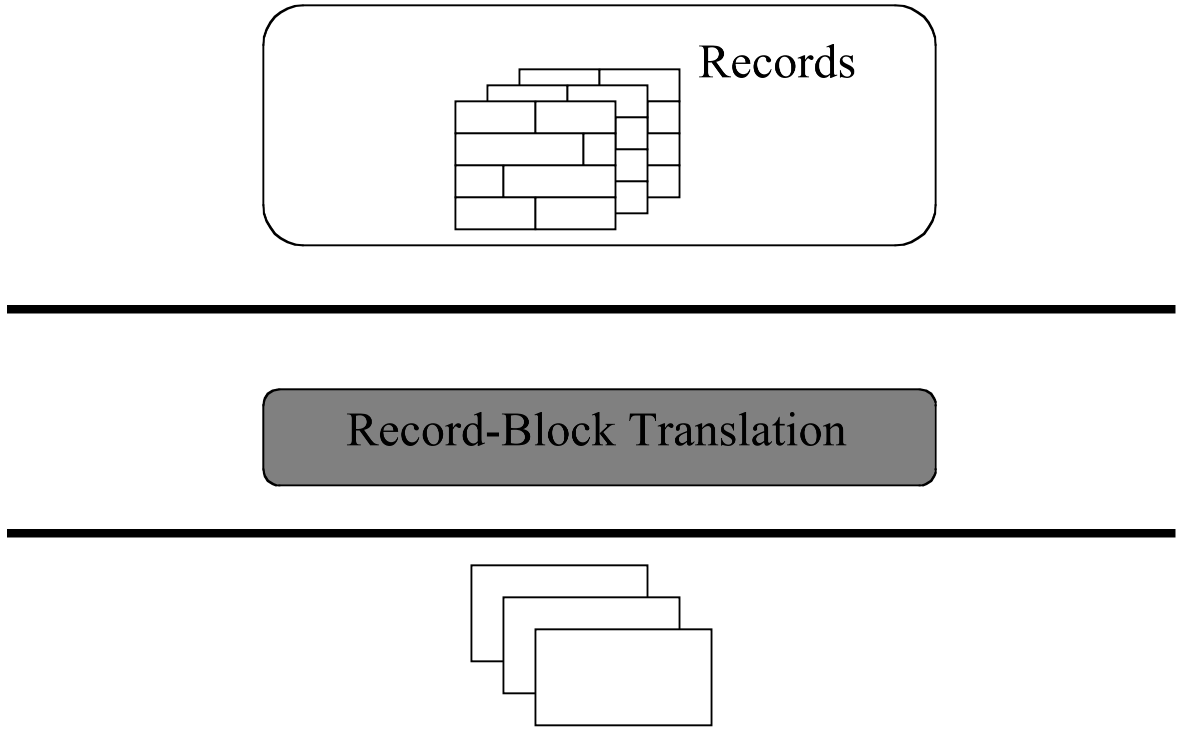
File Descriptors

- 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

Byte Stream File Interface

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

Structured Files

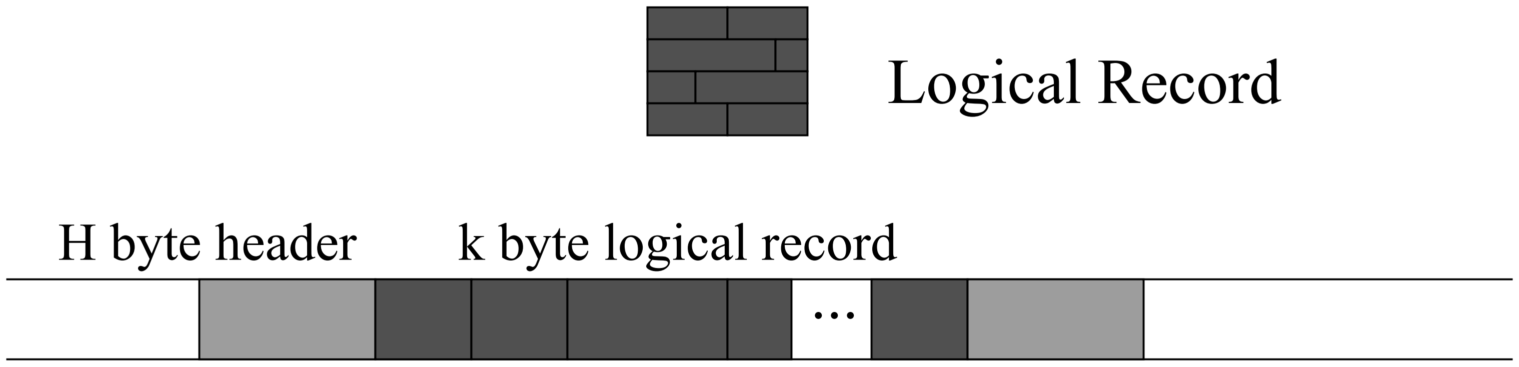


Record-Oriented Sequential Files

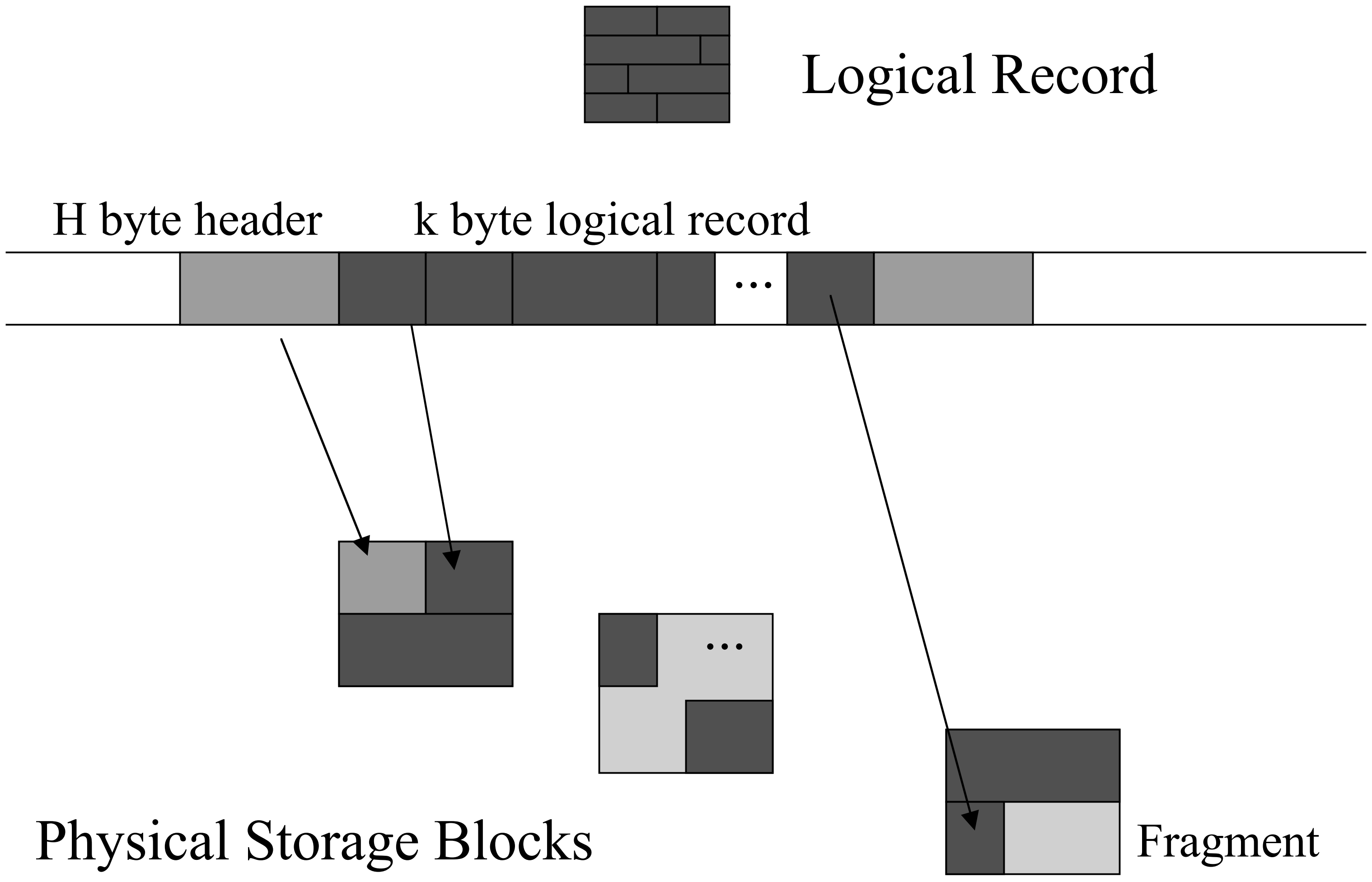
Logical Record

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

Record-Oriented Sequential Files



Record-Oriented Sequential Files

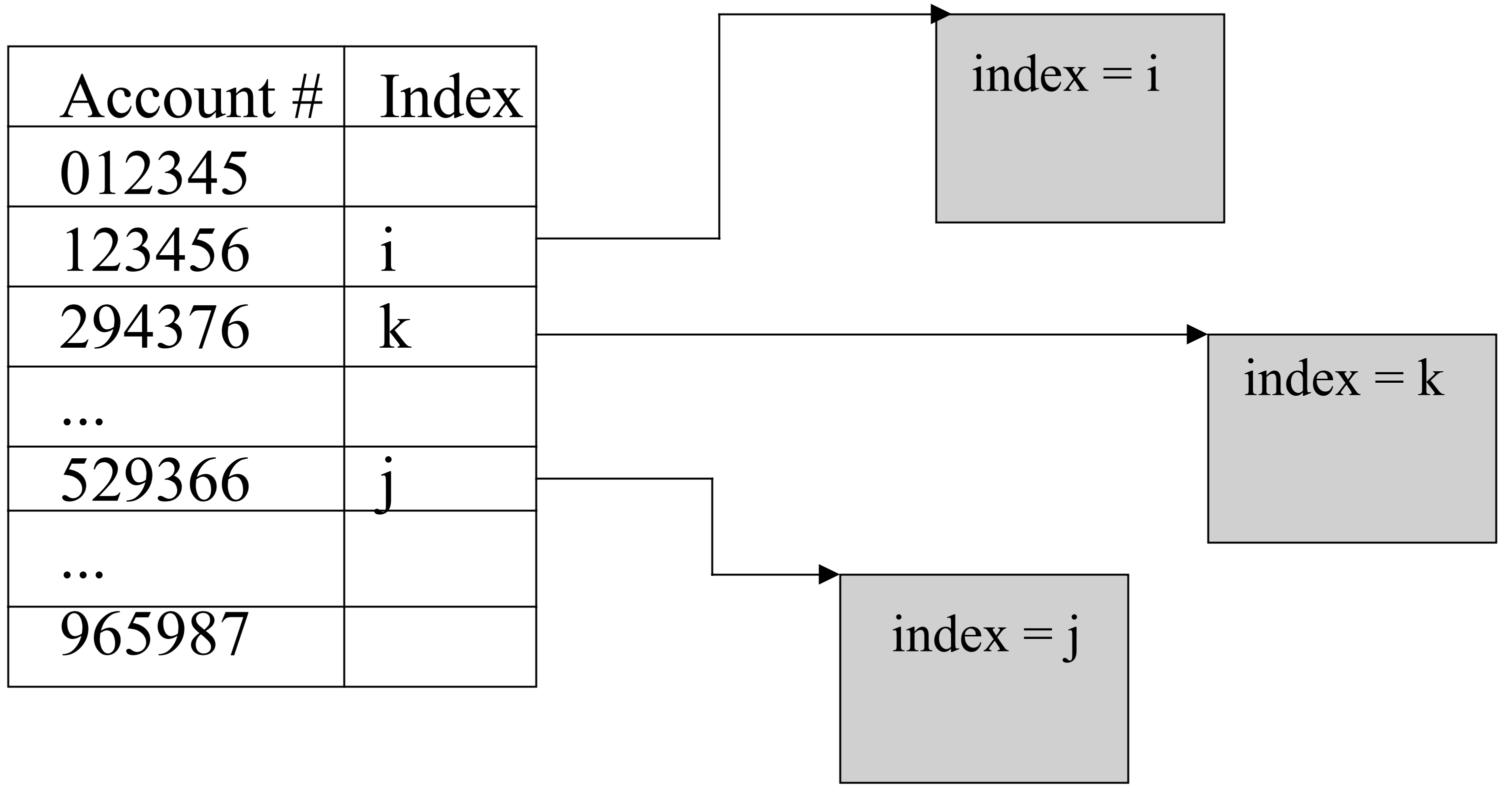


Indexed Sequential File

- 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)
```

Indexed Sequential File (cont)



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

More Abstract Files

- Inverted files
  - 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

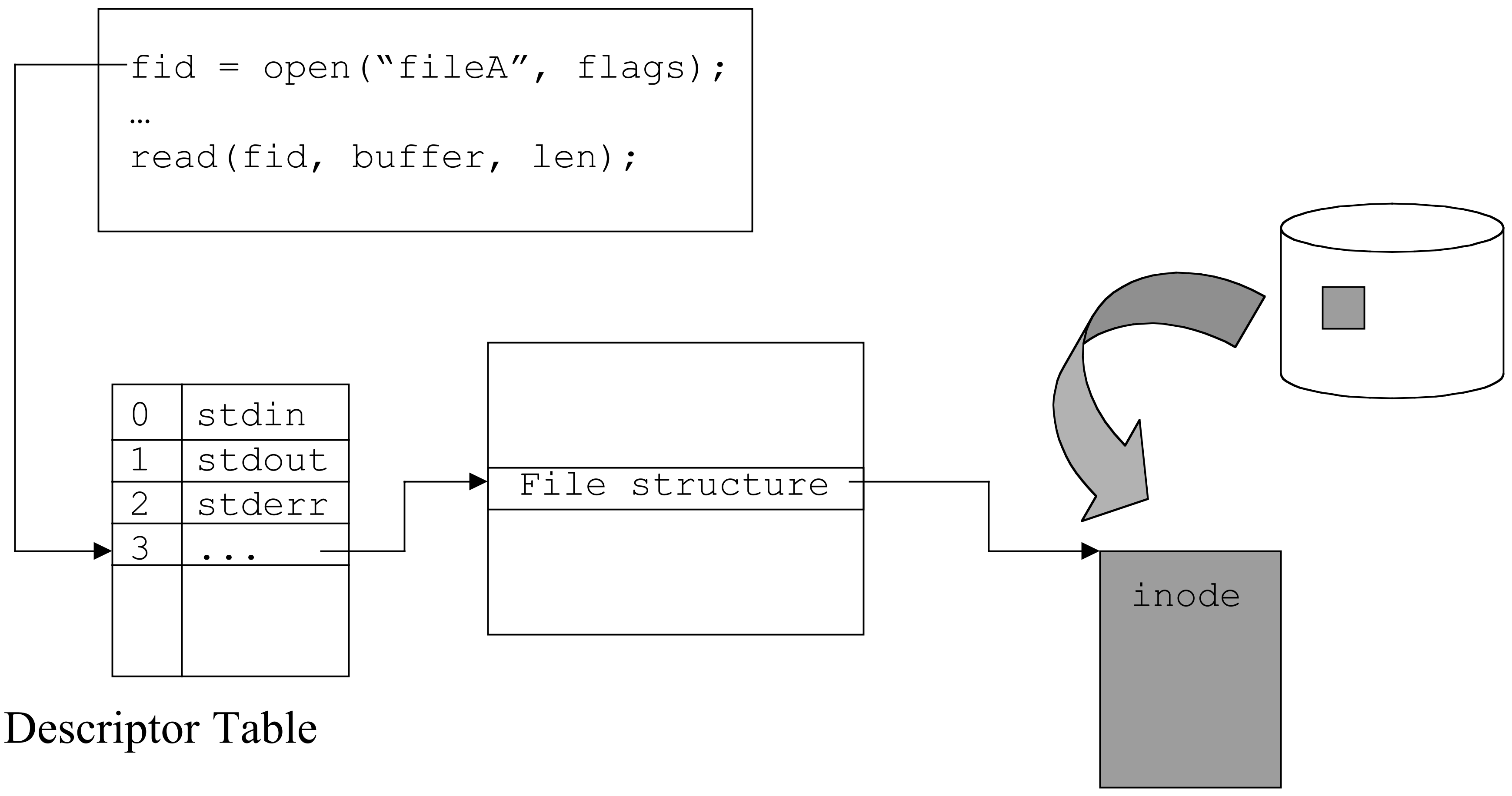
Implementing Low Level Files

- 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

An `open` Operation

- Locate the 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

Opening a UNIX File



Descriptor Table

Block Management

- The job of assigning storage blocks to the file
- Fixed sized, k, blocks
- File of length m requires $N = \lceil m/k \rceil$ blocks
- Byte b_i is stored in block $\lfloor i/k \rfloor$
- Three basic strategies:
  - Contiguous allocation
  - Linked lists
  - Indexed allocation

Contiguous Allocation

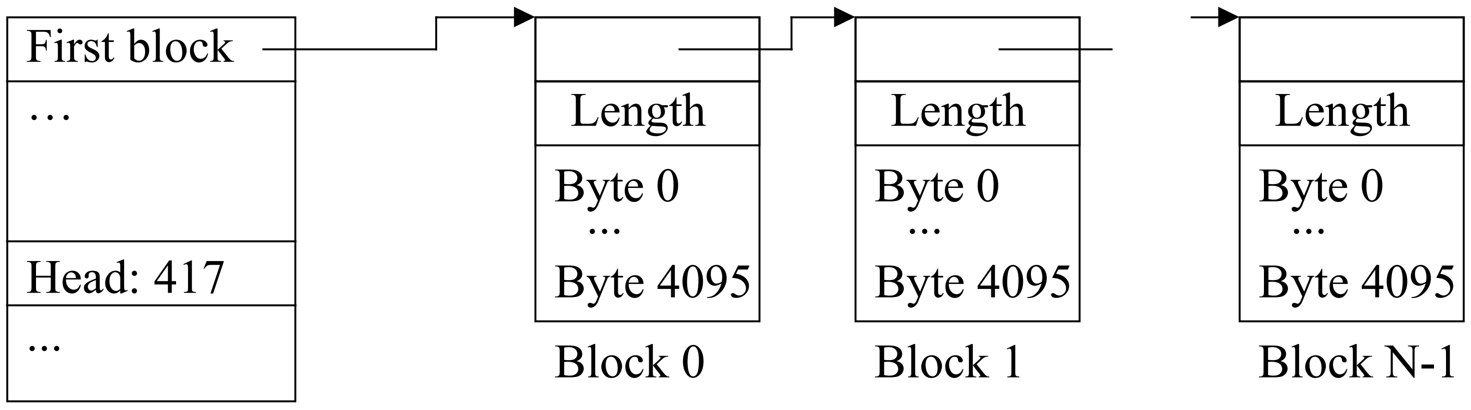
- Maps the N blocks into N contiguous blocks on the secondary storage device
- Difficult to support dynamic file sizes

File descriptor

Head position	237
…	
First block	785
Number of blocks	25

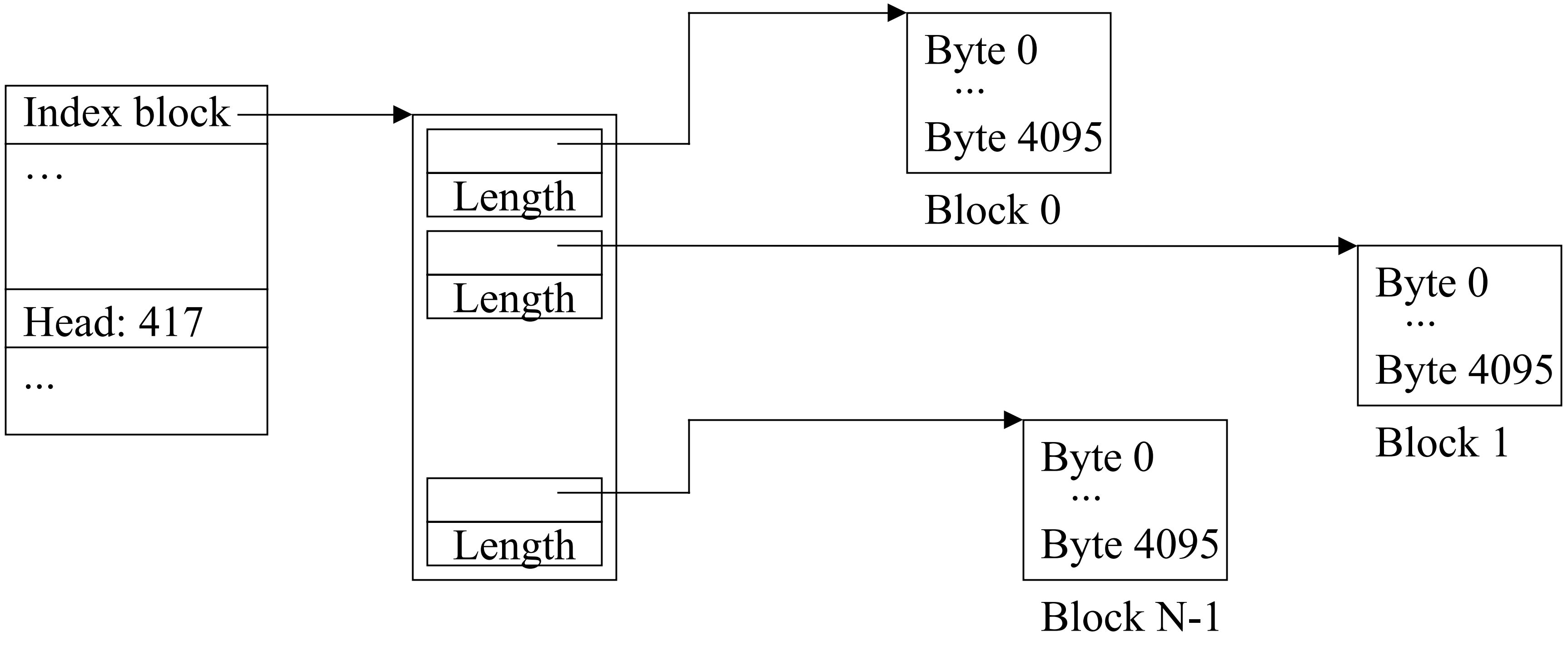
Linked Lists

- 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

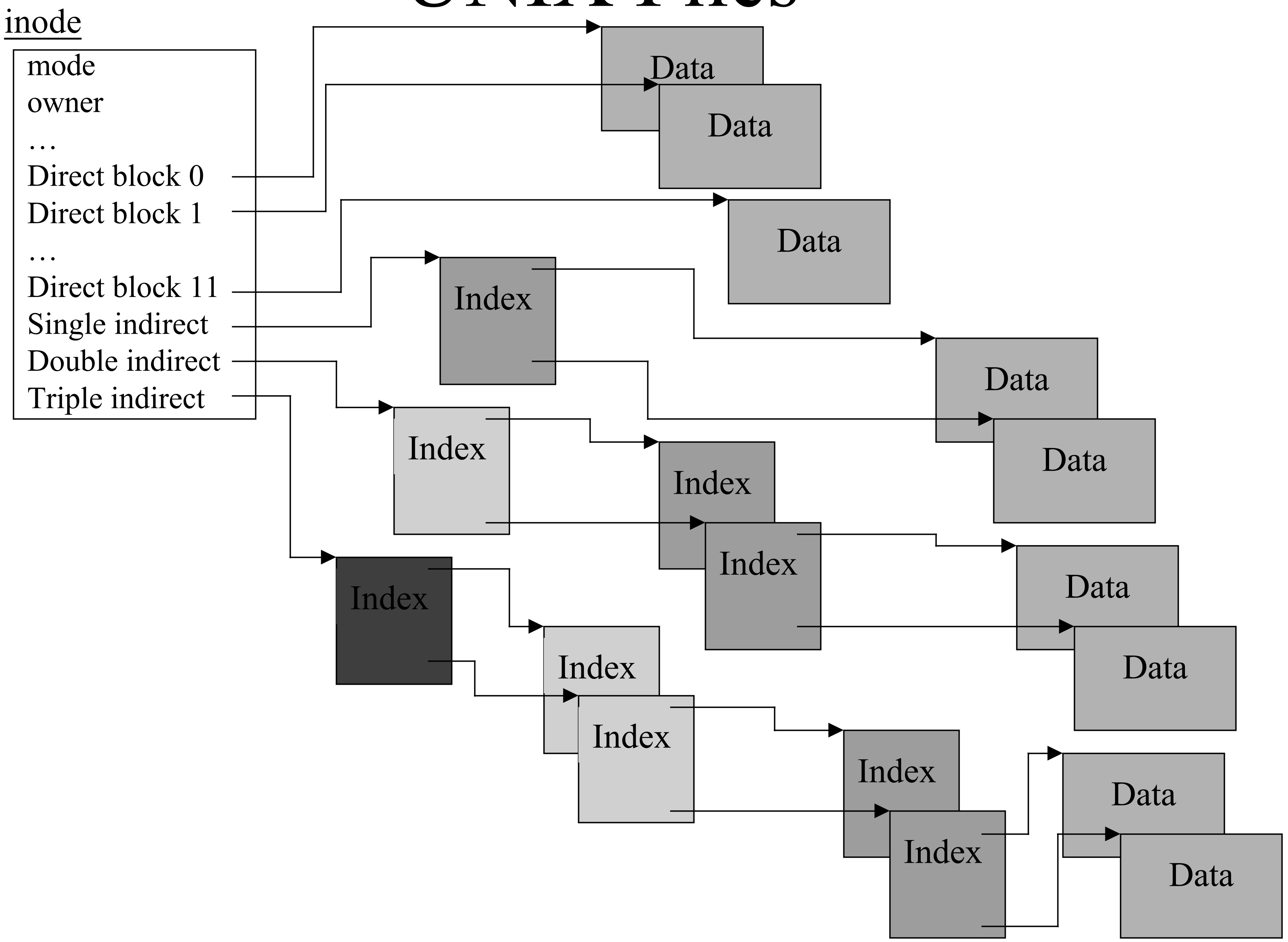


Indexed Allocation

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



UNIX Files


inode
mode
owner
…
Direct block 0
Direct block 1
…
Direct block 11
Single indirect
Double indirect
Triple indirect
Data
Data
Data
Index
Data
Data
Index
Index
Index
Data
Data
Index
Index
Index
Index
Index
Data
Data

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

Managing the Byte Stream

- Packing and unpacking blocks
  - Must read-ahead on input
  - Must write-behind on output
  - Seek
  - Inserting/deleting bytes in the interior of the stream
- Block I/O
  - Buffer several blocks
  - Memory mapped files

Directories

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

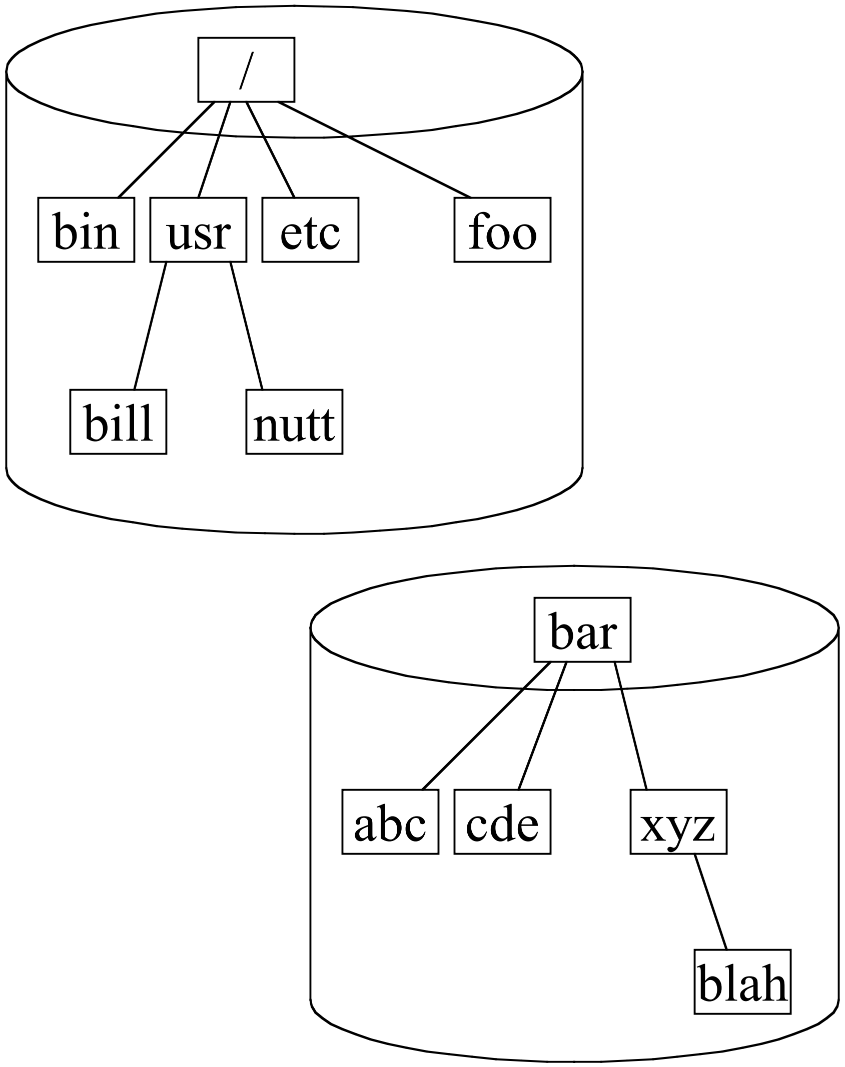
Directory Structures

- 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.

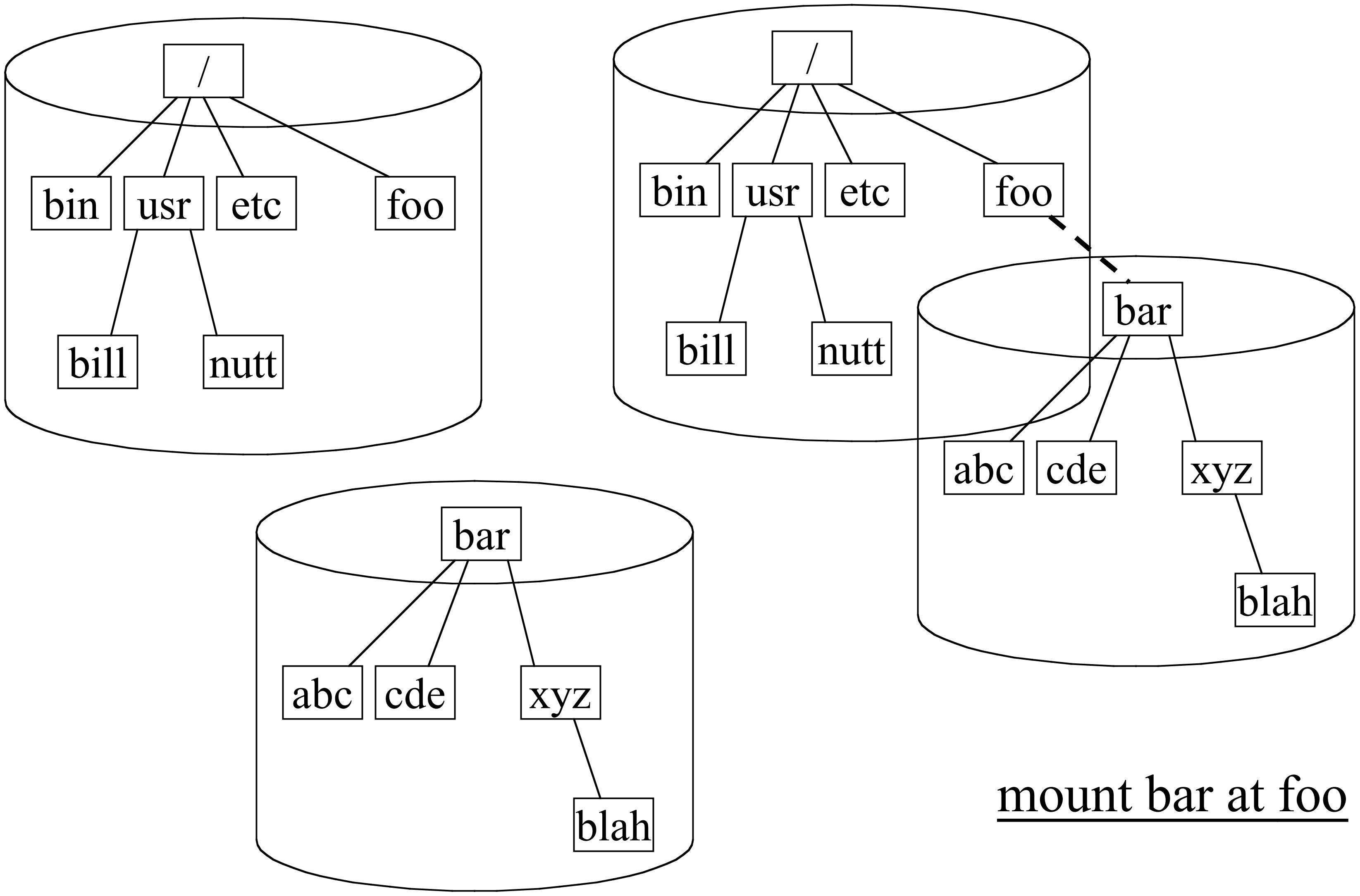
Directory Implementation

- 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)

UNIX `mount` Command



UNIX `mount` Command



<u>mount bar at foo</u>