KOÇ UNIVERSITY

File System

Didem Unat
Lecture 22
COMP304 - Operating Systems (OS)

File

- A collection of related bytes having meaning only to the creator. The file can be "free formed", indexed, structured, etc.
- The file is an entry in a directory.
- The file may have structure (O.S. may or may not know about this.) It's a tradeoff of capabilities versus overhead. For example,

  a) An Operating System understands program image format in order to create a process.

  b) The UNIX **shell** understands how directory files look. (In general the UNIX **kernel** doesn't interpret files.)

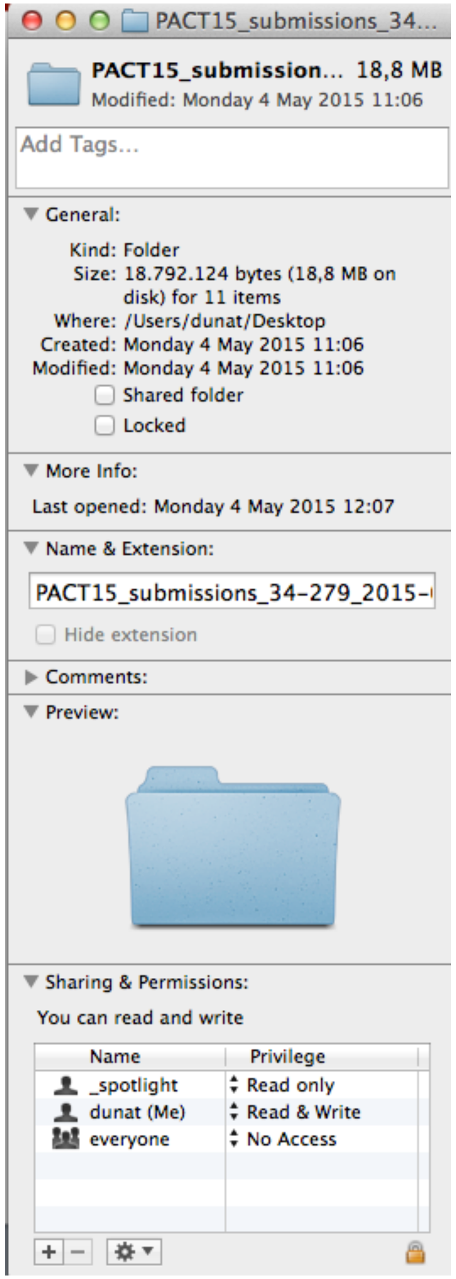
  c) Usually the Operating System understands and interprets file types.

File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring

- Information about files are kept in the directory structure, which is maintained on the disk

File info



PACT15_submissions_34...

PACT15_submission... 18,8 MB
Modified: Monday 4 May 2015 11:06

Add Tags...

General:

Kind: Folder
Size: 18.792.124 bytes (18,8 MB on disk) for 11 items
Where: /Users/dunat/Desktop
Created: Monday 4 May 2015 11:06
Modified: Monday 4 May 2015 11:06
☐ Shared folder
☐ Locked

More Info:

Last opened: Monday 4 May 2015 12:07

Name & Extension:

PACT15_submissions_34-279_2015-

☐ Hide extension

Comments:

Preview:

Sharing & Permissions:

You can read and write

Name	Privilege
_spotlight	Read only
dunat (Me)	Read & Write
everyone	No Access

What can we find out about a Linux File?

dunat$ stat configure.ac

```
 File: `configure.ac'
 Size: 7905          Blocks: 16       IO Block: 4096   regular file
Device: fd02h/64770d      Inode: 597296633   Links: 1
Access: (0644/-rw-r--r--)  Uid: (329464/   dunat)   Gid: (200513/domainusers)
Access: 2014-12-08 12:42:35.000000000 +0200
Modify: 2014-09-24 17:08:49.000000000 +0300
Change: 2014-12-08 12:42:36.041757537 +0200
```

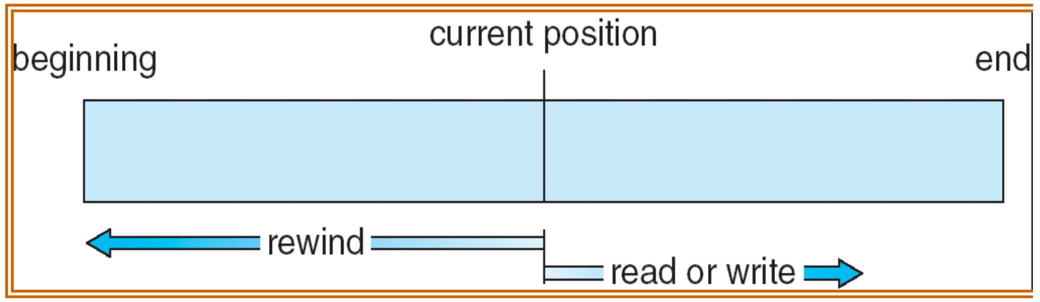
On Mac OS X

File Systems Interface

Access Methods

If files had only one "chunk" of data, life would be simple. But for large files, the files themselves may contain structure, making access faster.

SEQUENTIAL ACCESS



- Implemented by the filesystem.
- Data is accessed one record right after the last.
- Reads cause a pointer to be moved ahead by one.
- Writes allocate space for the record and move the pointer to the new End Of File.
- Such a method is reasonable for tape

File Systems Interface

Access Methods

DIRECT ACCESS

- Method useful for disks.
- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on which blocks are read/written in any order.
- User now says "read n" rather than "read next".
- "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

File System Interface

Access Methods

OTHER ACCESS METHODS

Built on top of direct access and often implemented by a user utility.

Indexed ID plus pointer.

An index block says what's in each remaining block or contains pointers to blocks containing particular items. Suppose a file contains many blocks of data arranged by name alphabetically.

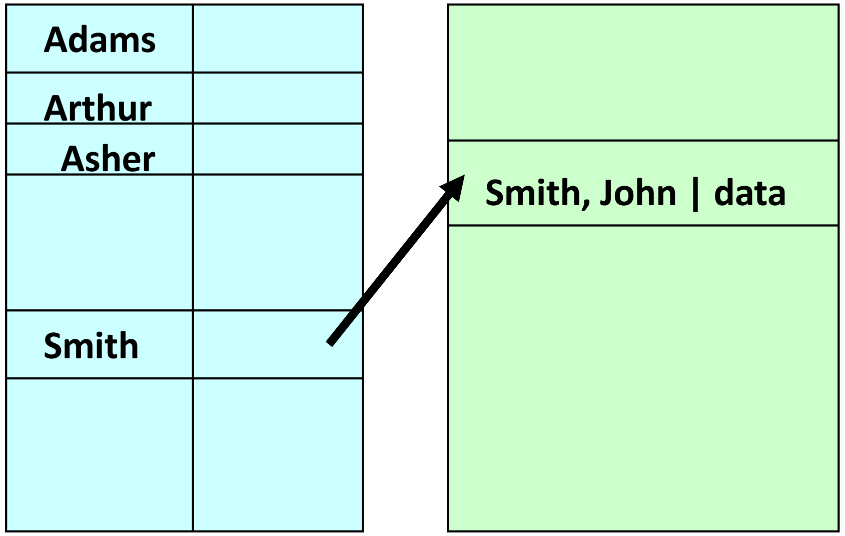
Example 1: Index contains the name appearing as the first record in each block. There are as many index entries as there are blocks.

Example 2: Index contains the block number where "A" begins, where "B" begins, etc. Here there are only 26 index entries.

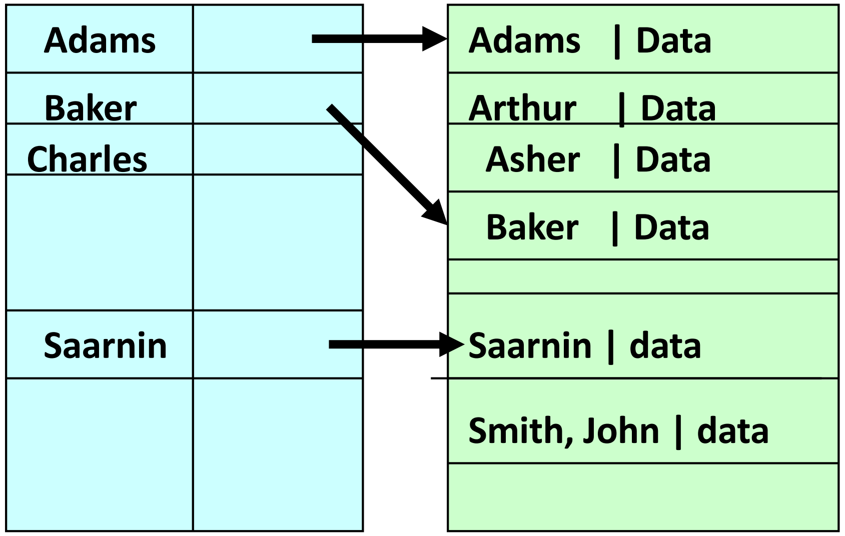
File System Interface

Access Methods

Example 1: Index contains the name appearing as the first record in each block. There are as many index entries as there are blocks.



Example 2: Index contains the block number where "A" begins, where "B" begins, etc. Here there are only 26 index entries.



Directory Structure

Directories maintain information about files, a directory itself is a file.

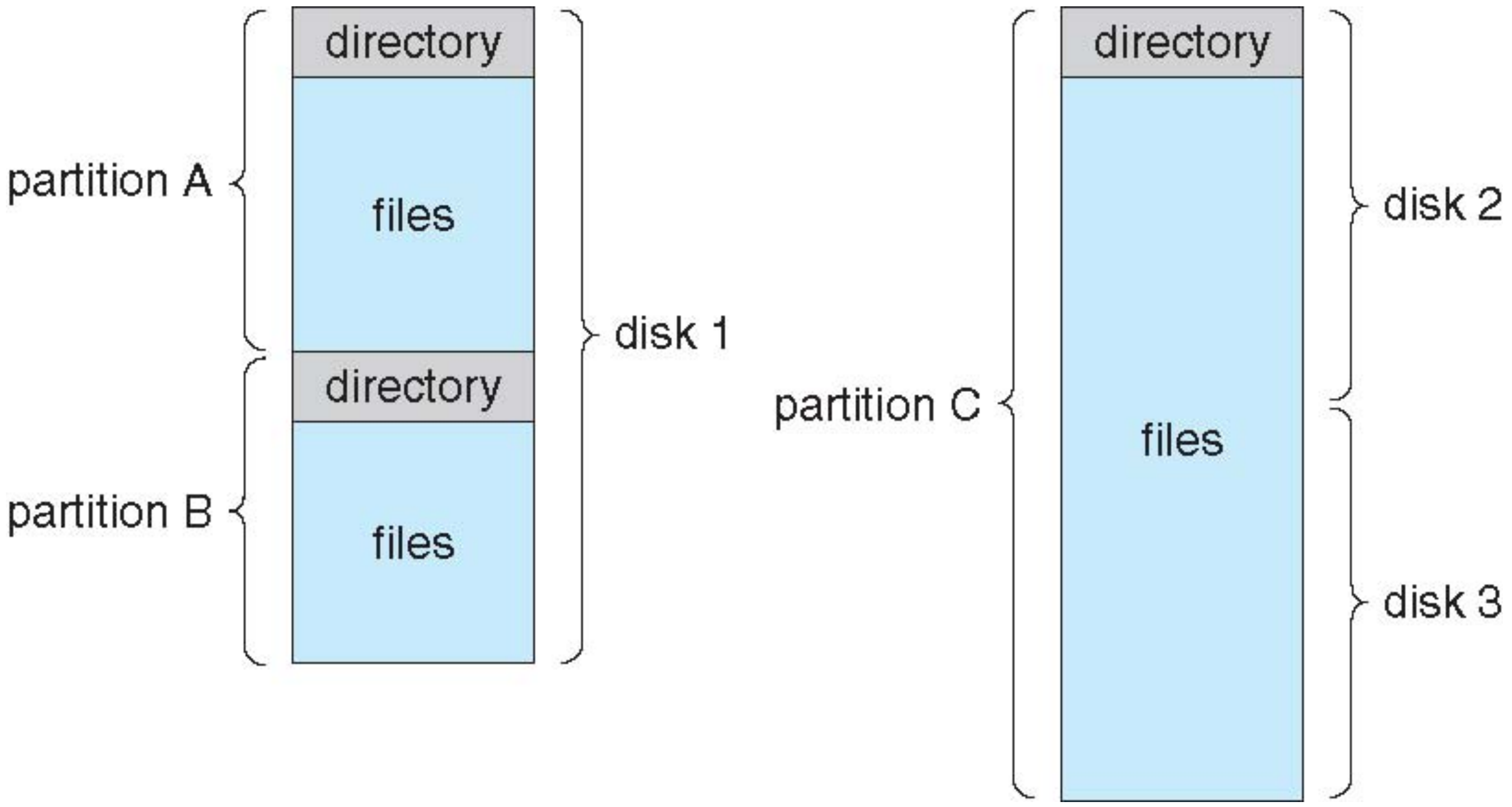
For a large number of files, may want a directory structure - directories under directories.

Information maintained in a directory:

Name	The user visible name.
Type	The file is a directory, a program image, a user file, a link, etc.
Location	Device and location on the device where the file header is located.
Size	Number of bytes/words/blocks in the file.
Position	Current next-read/next-write pointers.
Protection	Access control on read/write/ execute/delete.
Usage	Open count
Usage	time of creation/access, etc.
Mounting	a filesystem occurs when the root of one filesystem is "grafted" into the existing tree of another filesystem.

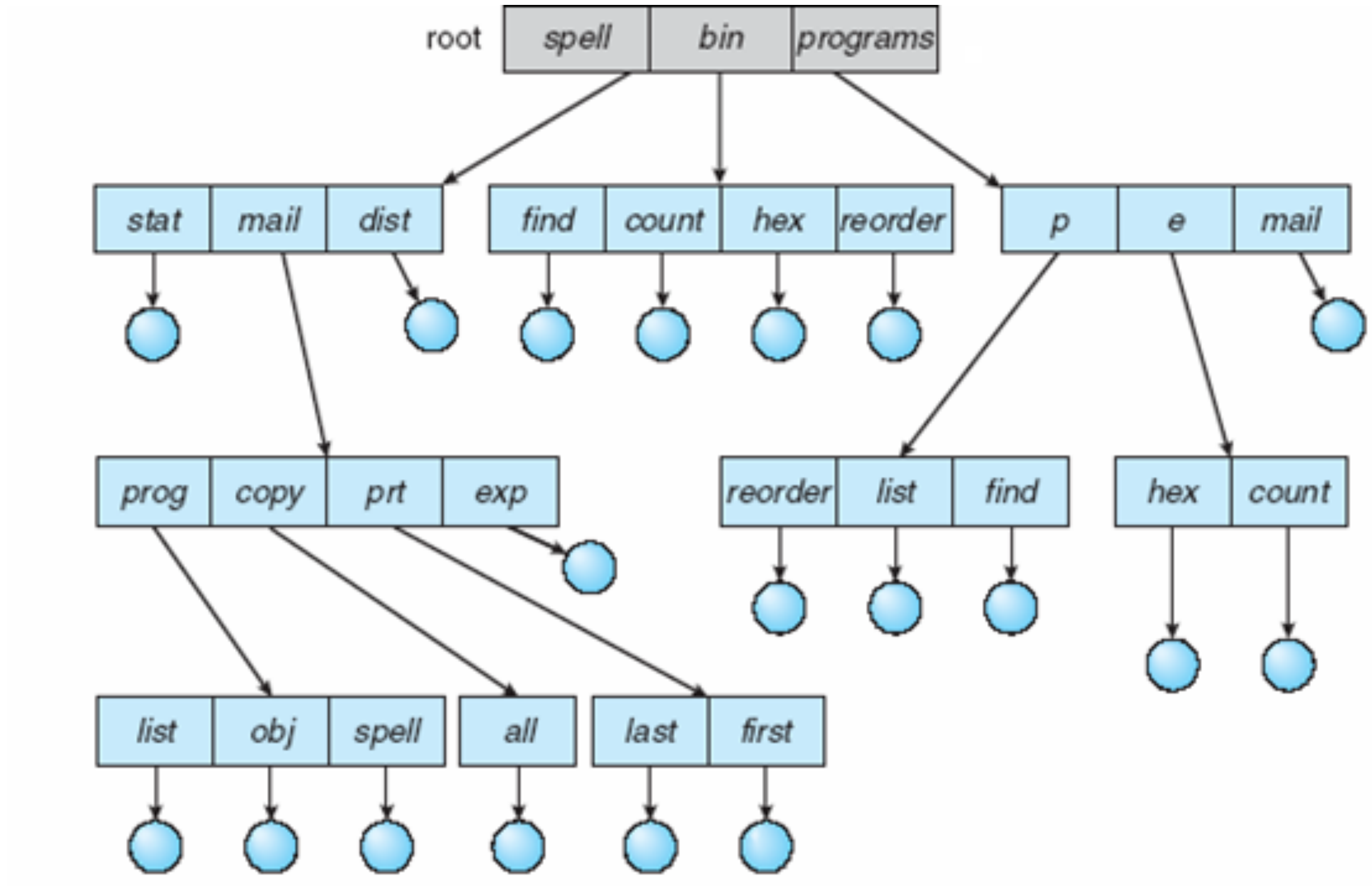
In Memory only!

A Typical File-system Organization



Both the directory structure and the files reside on disk

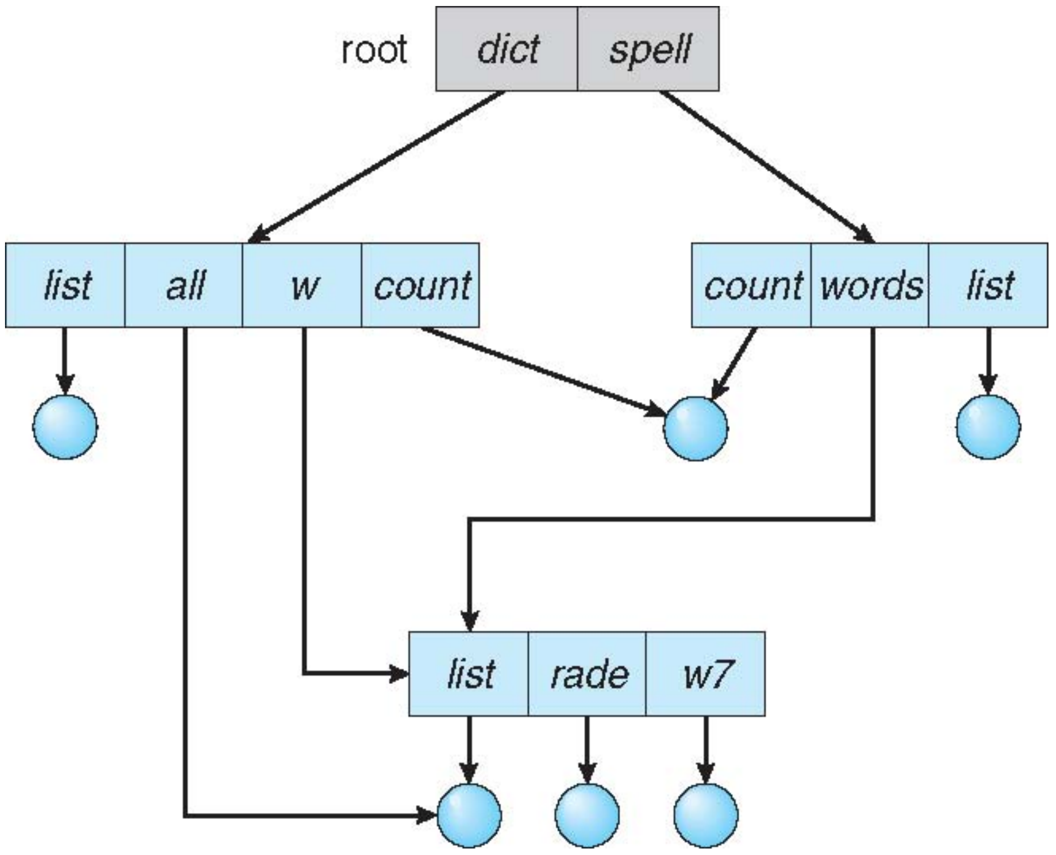
Tree-Structured Directories



Other non-tree structures possible?

Acyclic-Graph Directories

- Have shared subdirectories and files
- How can we implement this?
  - Links

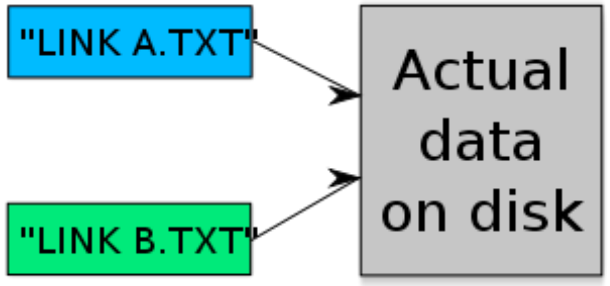


What happens when the link is deleted?

What happens when the original file is deleted?

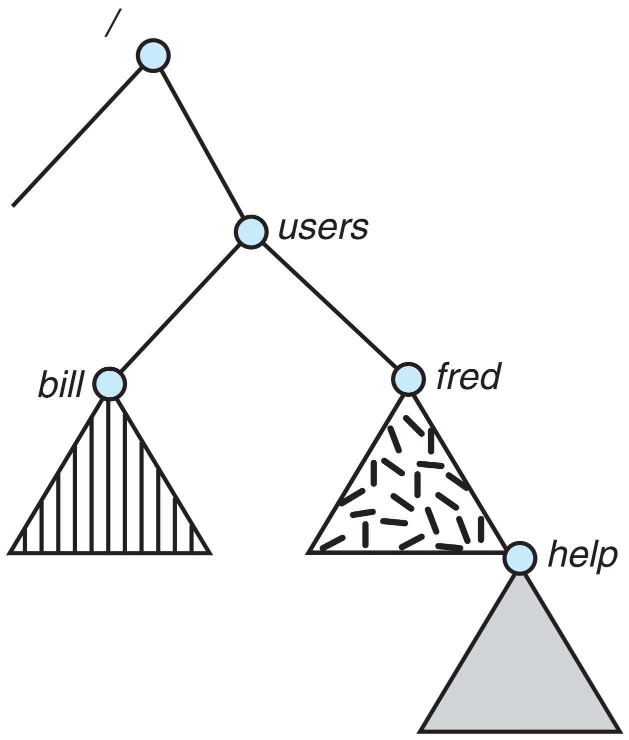
Symbolic vs Hard Links

- Both are used for aliasing files
  - Multiple names for the same file
- Symbolic Link
  - contains a reference to another file or directory in the form of an absolute or relative path
  - When the link is deleted, file remains
  - When the file is deleted, the link remains – user has to clean up
  - ln -s target_path link_path
- Hard Links
  - Keep a reference (link) count
  - When count is zero, delete the file as well
  - Not recommended to use
  - Only few OSs support with a root access

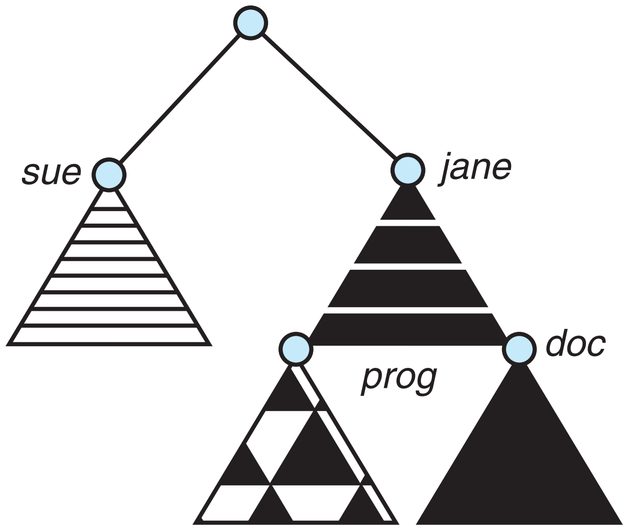


File System Mounting

- A file system must be **mounted** before it can be accessed
- An unmounted file system (i.e., Fig. b) is mounted at a **mount point**

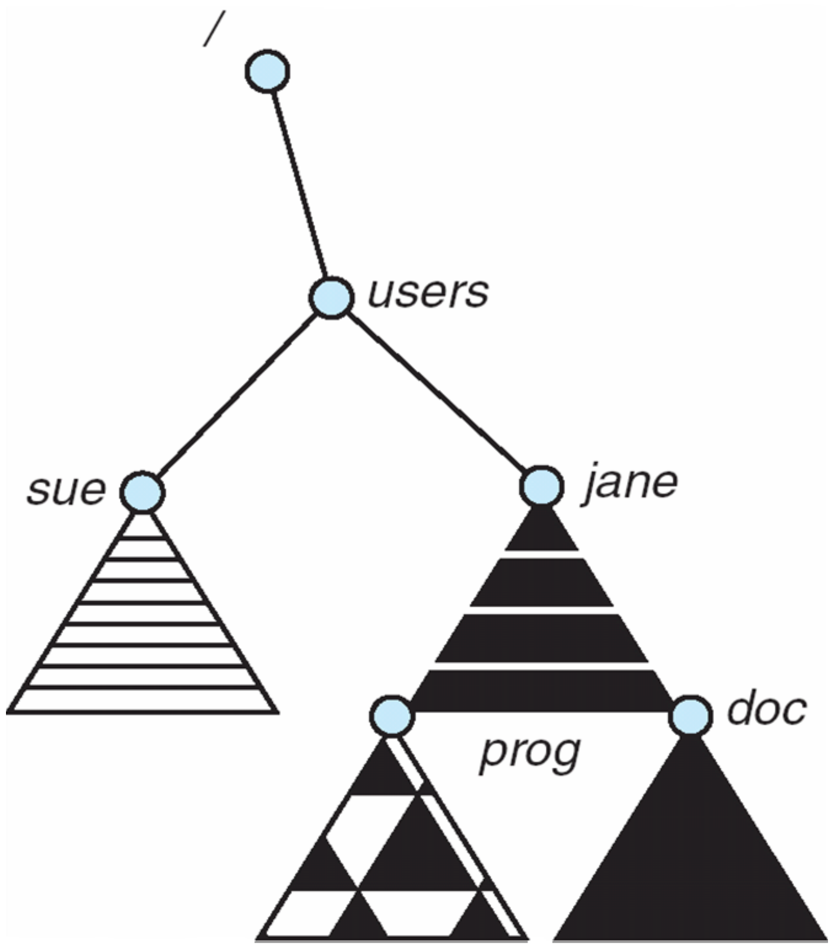


(a)



(b)

Mount Point



- Mac OS X searches for a file system on the device at boot time or while the system running
  - **It automatically mounts the file system under the /Volume directory**
- Unix
  - Requires explicit mount
    - Usually under /mnt
  - The ones listed in configuration file containing list of devices are automatically mounted

File Sharing

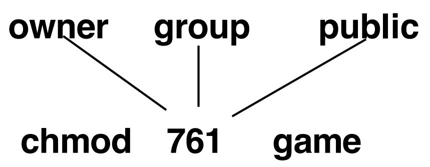
- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- In multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

Access Lists and Groups in Unix

- Mode of access: read (4), write (2), execute (1)
- Three classes of users on Unix / Linux

a) **owner access**	7	⇒	RWX 1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



```
chmod 761 game
```

Attach a group to a file

```
chgrp     G     game
```

Sample Unix Access Control List

- 4: Read Access
- 2: Write Access
- 1: Execute Right

- 4+2 = 6 means both read and write accesses but not execute right
- 4+ 2 + 1 = 7 means all rights

Example

- We have a file named ‘questions.txt’
  - Begum should be able to invoke all operations on the file
  - Ata, Cansu and Zafer should be able to only read and write the file, they should not be allowed to delete the file
  - All other users should not be allowed to do anything but read

- How do you set up the access control list for this file?

- More about access control lists:
  - https://www.redhat.com/sysadmin/linux-access-control-lists

Virtual File Systems

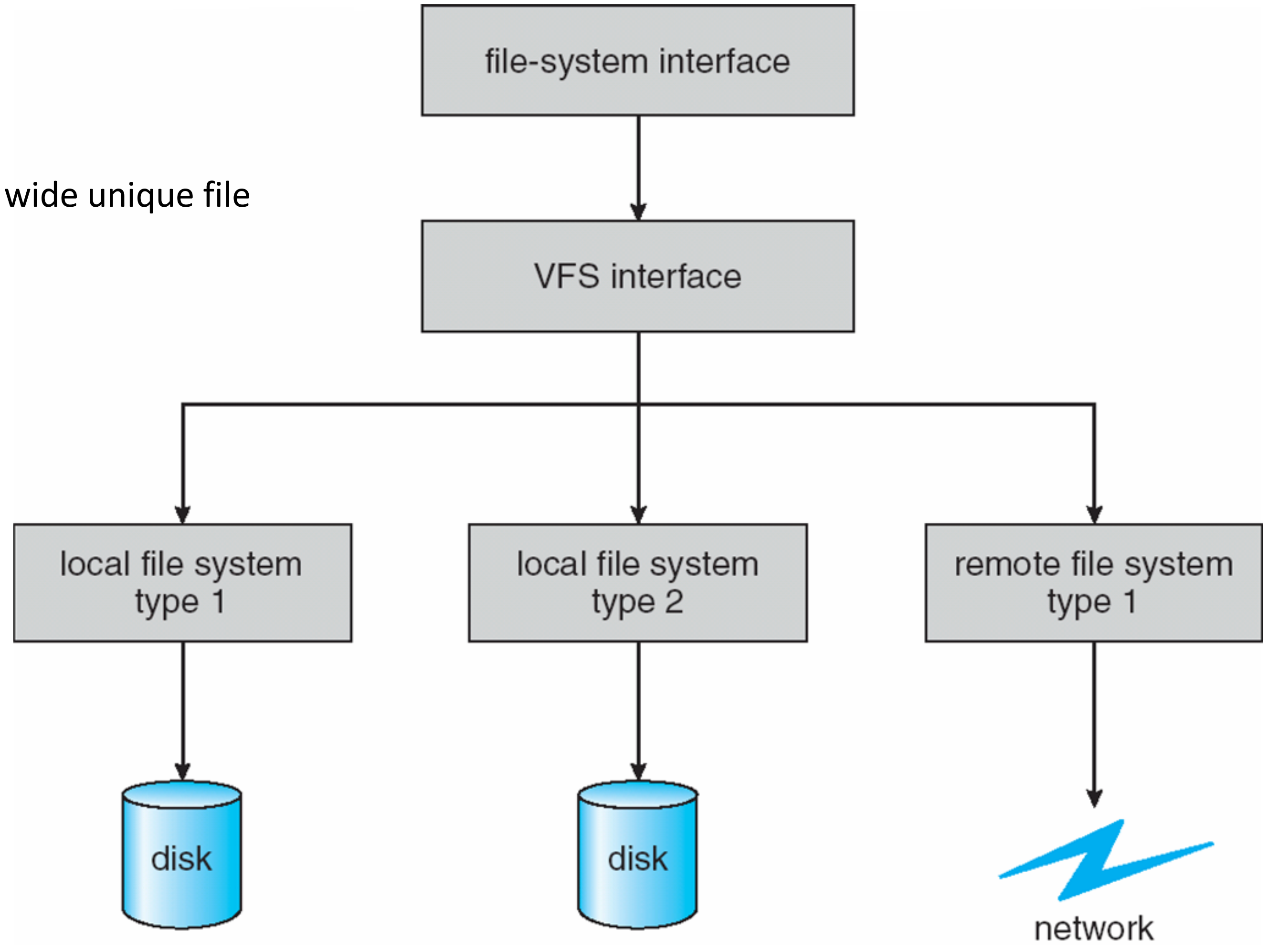
- Virtual File Systems (VFS) on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - Implementation can be one of many file systems types, or network file system
  - Then dispatches operation to appropriate file system implementation routines
- The API is to the VFS interface, rather than any specific type of file system

Schematic View of Virtual File System

`open(), read(), write(), close()`

`vnode:` network wide unique file

`inode:`
Unique within only
a single file system



Virtual File System Implementation

- For example, Linux VFS has four object types:
  - Inode object: represents an individual file
  - File object: represents an open file
  - Superblock object: represents entire file system
  - Dentry object: individual directory entry

- VFS defines set of operations on the objects that must be implemented
  - Such as open, close, read, write, ….

File Descriptors

- **Unix I/O: file descriptors**
  - One of the most important resources managed by the kernel is the file
  - A user process and the kernel must agree on names for open file connections. Analogous to the difference between a program and a process, there is a difference between a file and an open "connection" to a file: for example we may have two open connections to the same file, but be in different positions in the file with respect to our next read. So the file name itself isn't really appropriate for communicating which connection you want to read the next byte from. So the OS and the user process refer to each open connection by a number (type int) called a *file descriptor*.
  - Standard input, output and error default to file descriptors 0, 1 and 2 respectively.

file descriptor 0 is a processes' stdin
file descriptor 1 is a processes' stdout
file descriptor 2 is a processes' stderr

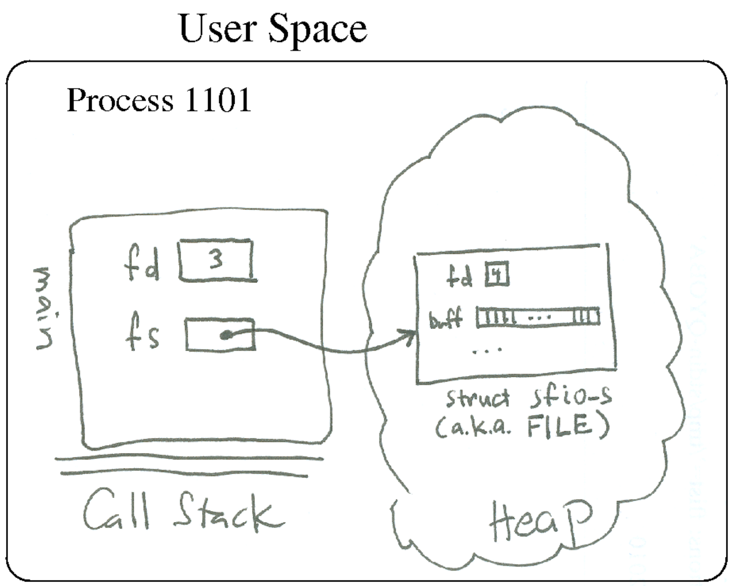
Kernel Space

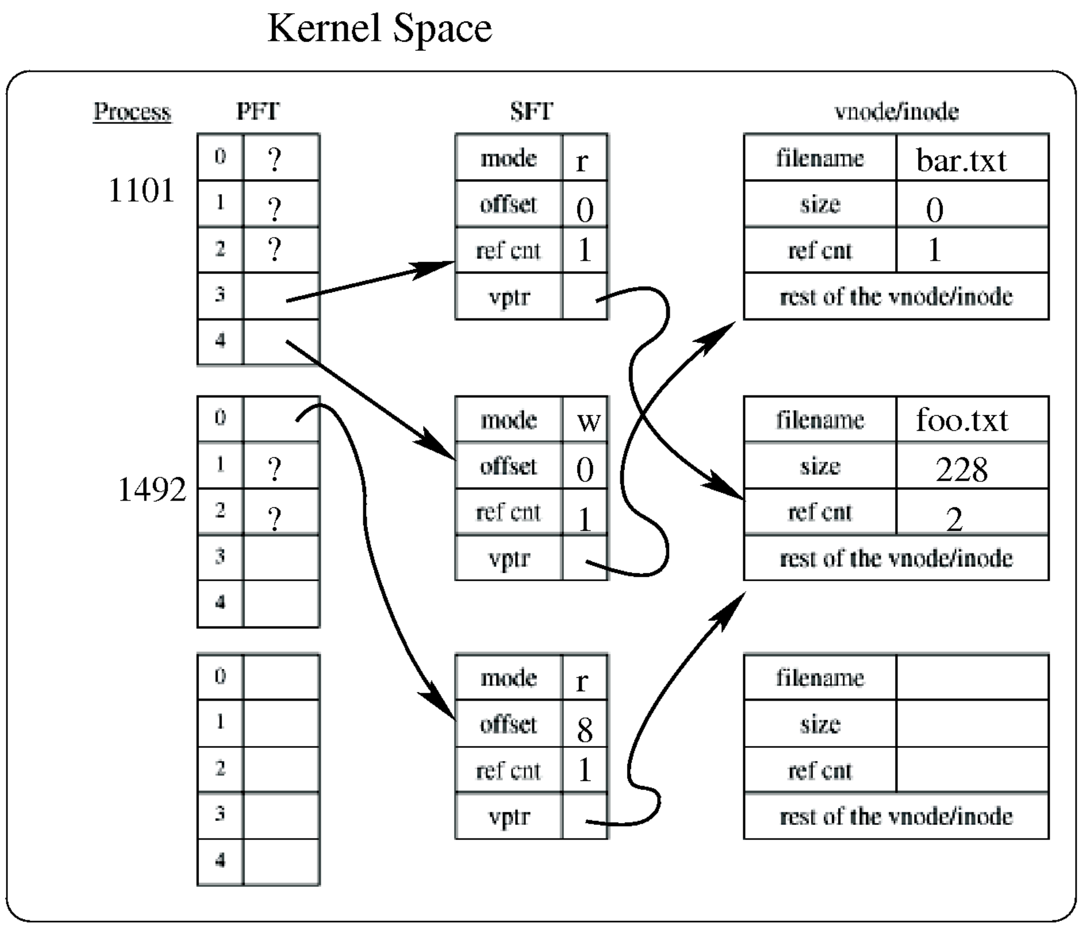
- **System Open File Table**
  - the kernel keeps a data structure called the *system open-file table* which has an entry for each connection (process-to-file). Each entry contains
    - the connection *status*, e.g. read or write,
    - the *current offset* in the file, and
    - a pointer to a *vnode*, which is the OS's structure representing the file, irrespective of where in the file you may currently be looking.
- **Vnode Table**
  - has an entry for each open file or device.
  - contains information about the type of file and pointers to functions that operate on the file.
  - Typically for files, the vnode also contains a copy of the *inode* for the file, which has "physical" information about the file, e.g. where exactly on the disk the file's data resides.

Physical Drive

- **The physical device: inodes, etc.**
  - a file may be broken up into many data blocks, which may be widely distributed across the physical drive.
  - The *inode* for a file contains the locations of each of the data blocks comprising the file.
  - Directories don't have data blocks, but in a similar fashion have directory blocks, which contain inode/filename pairs; i.e. the names of the files/directories in the directory, along with the inodes for each. Each directory contains entries for "." and ".." --- the current directory and its parent.

System Open File Table





```
int main()
{
int fd = open("foo.txt", O_RDONLY);
FILE* fs = fopen("bar.txt", "w");
⋮
```

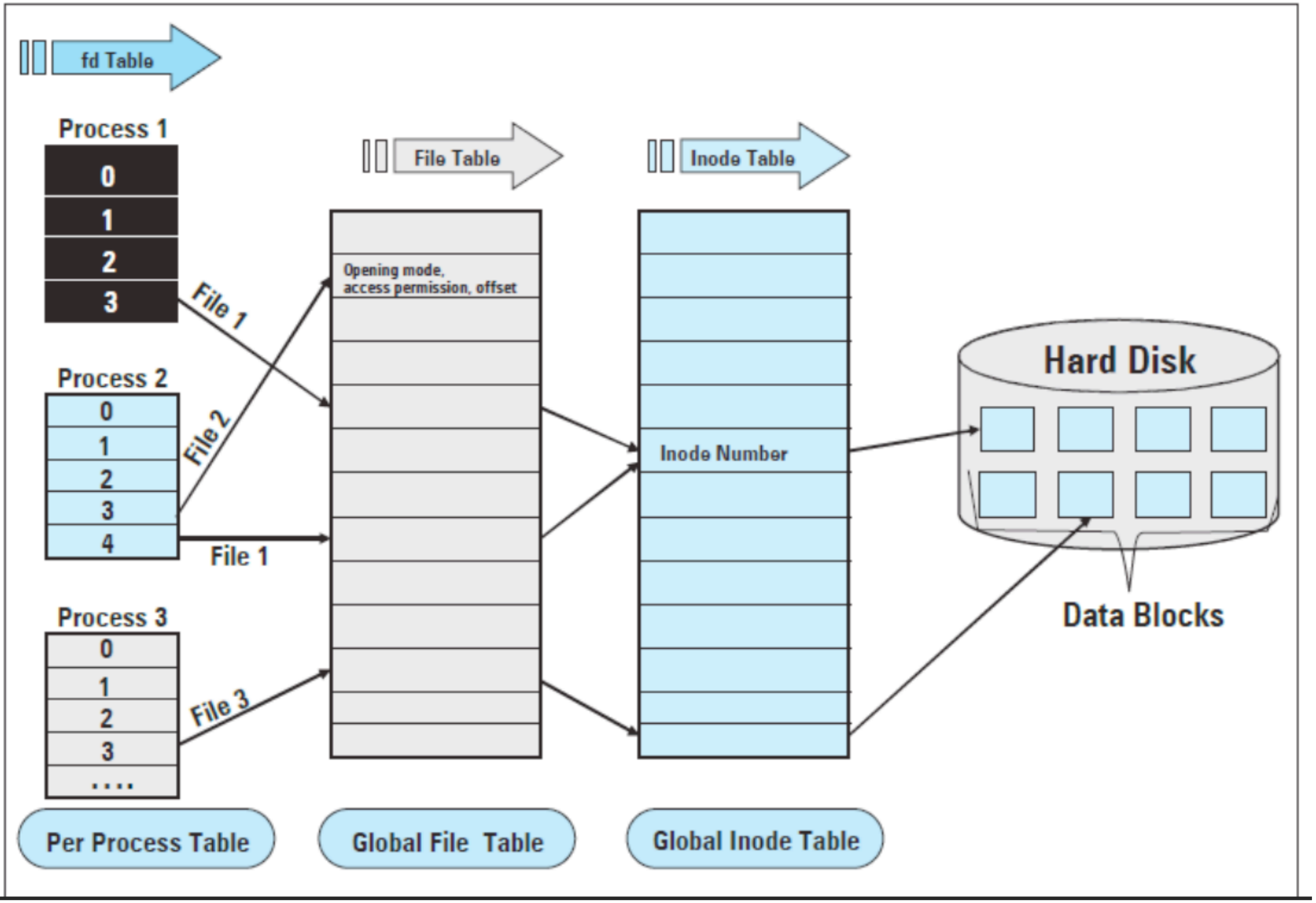
for I/O System Calls $\rightsquigarrow$

for C standard library I/O $\rightsquigarrow$

https://www.usna.edu/Users/cs/wcbrown/courses/IC221/classes/L09/Class.html

Reference Counts

- **Several file descriptors** may actually refer to the same system open-file table entries. That entry in the system open-file table can't be removed until *all* of those referencing file descriptors have been closed.
- **Several system open-file table entries** may actually refer to the same vnode table entry. That vnode table entry cannot be removed from the vnode table until *all* of those referencing system open-file table entries have been removed.
- A file may be referenced by several entries in the file system (this comes from "**hard links**", which you can create with the *ln* utility) and the file cannot be removed from the filesystem until *all* of those references to the file have been removed.


fd Table
Process 1
0
1
2
3
File 1
Process 2
0
1
2
3
4
File 2
File 1
Process 3
0
1
2
3
....
File 3
File Table
Opening mode, access permission, offset
Inode Table
Inode Number
Hard Disk
Data Blocks
Per Process Table
Global File Table
Global Inode Table

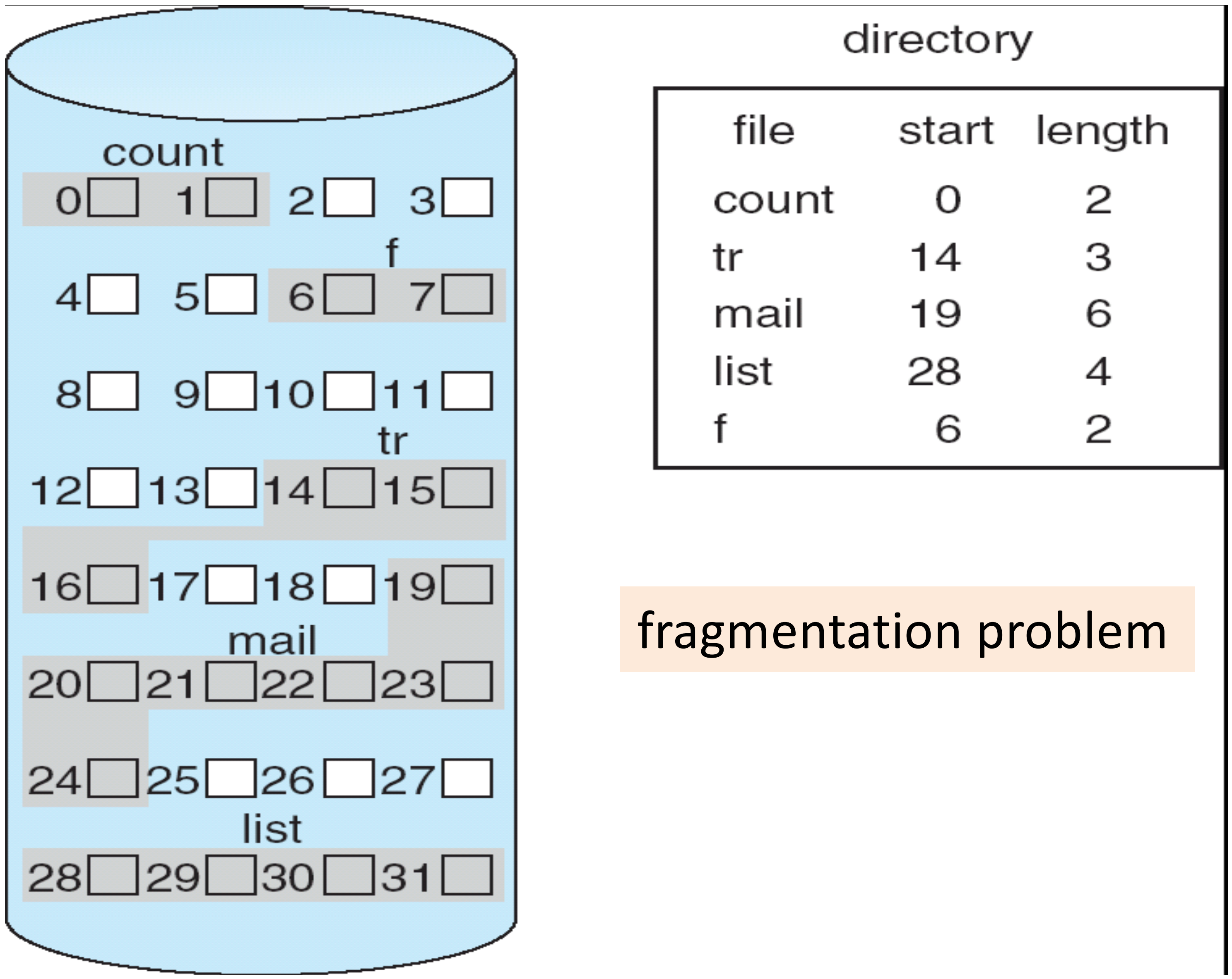
Allocation Methods

- Memory is divided into pages, similarly disk is divided in blocks
- An allocation method refers to how disk blocks are allocated for files so that disk space is utilized effectively and files can be accessed efficiently:
  - 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 may not grow.

Contiguous Allocation of Disk Space



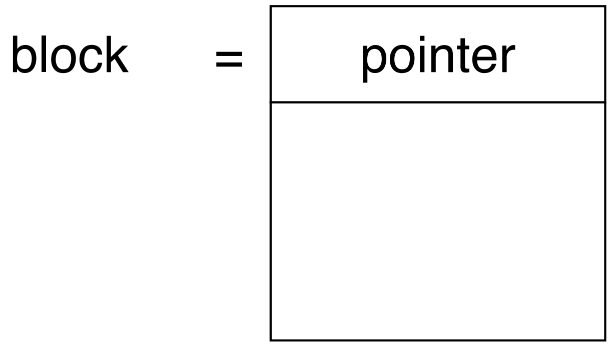
directory

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

fragmentation problem

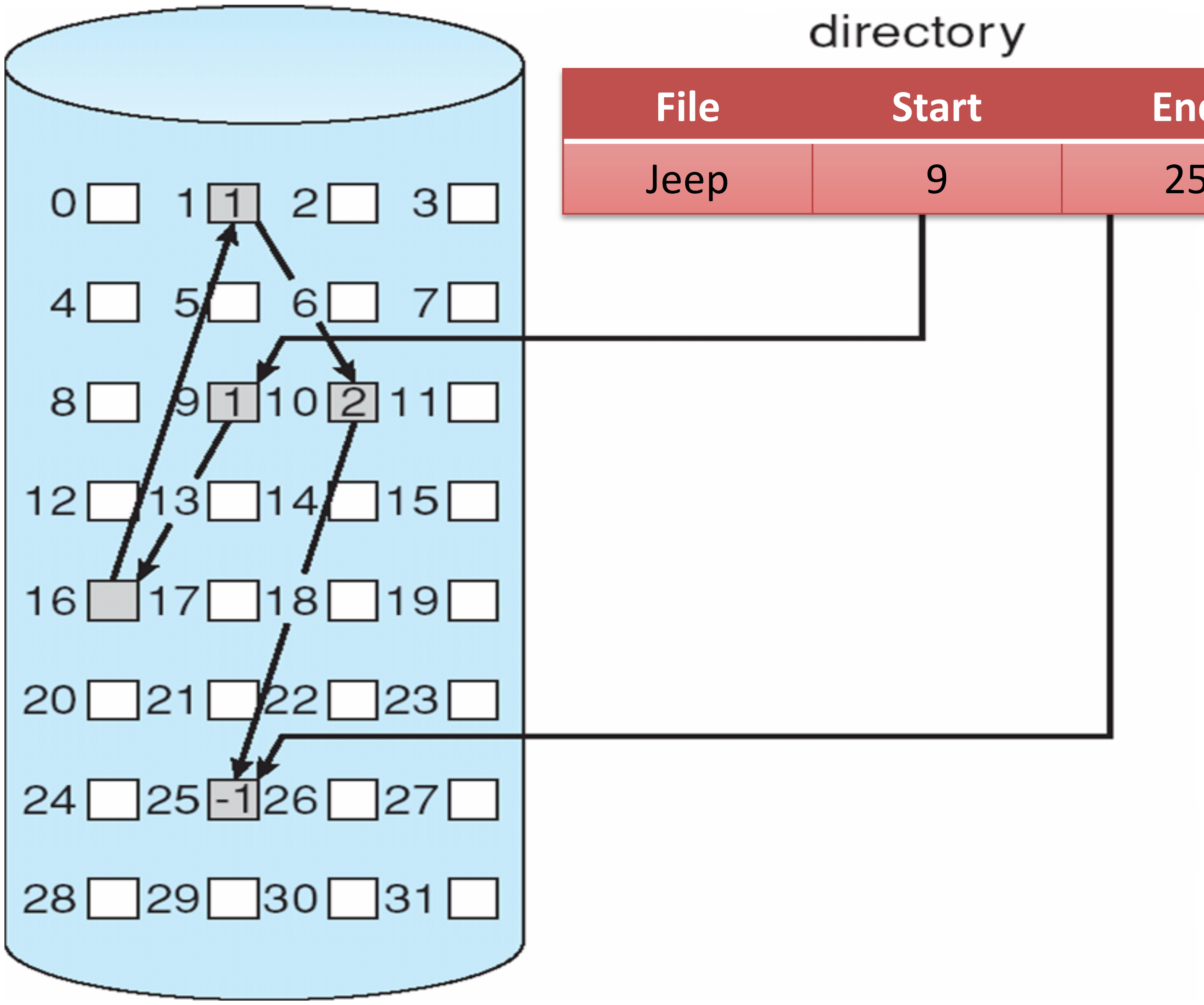
Linked Allocation

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



- Simple – need only starting address
- No random access
- Free-space management system:
  - no waste of space

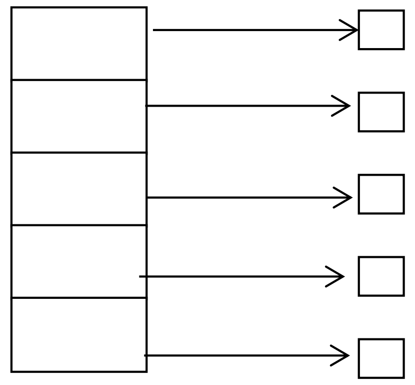
Linked Allocation



File	Start	End
Jeep	9	25

Indexed Allocation

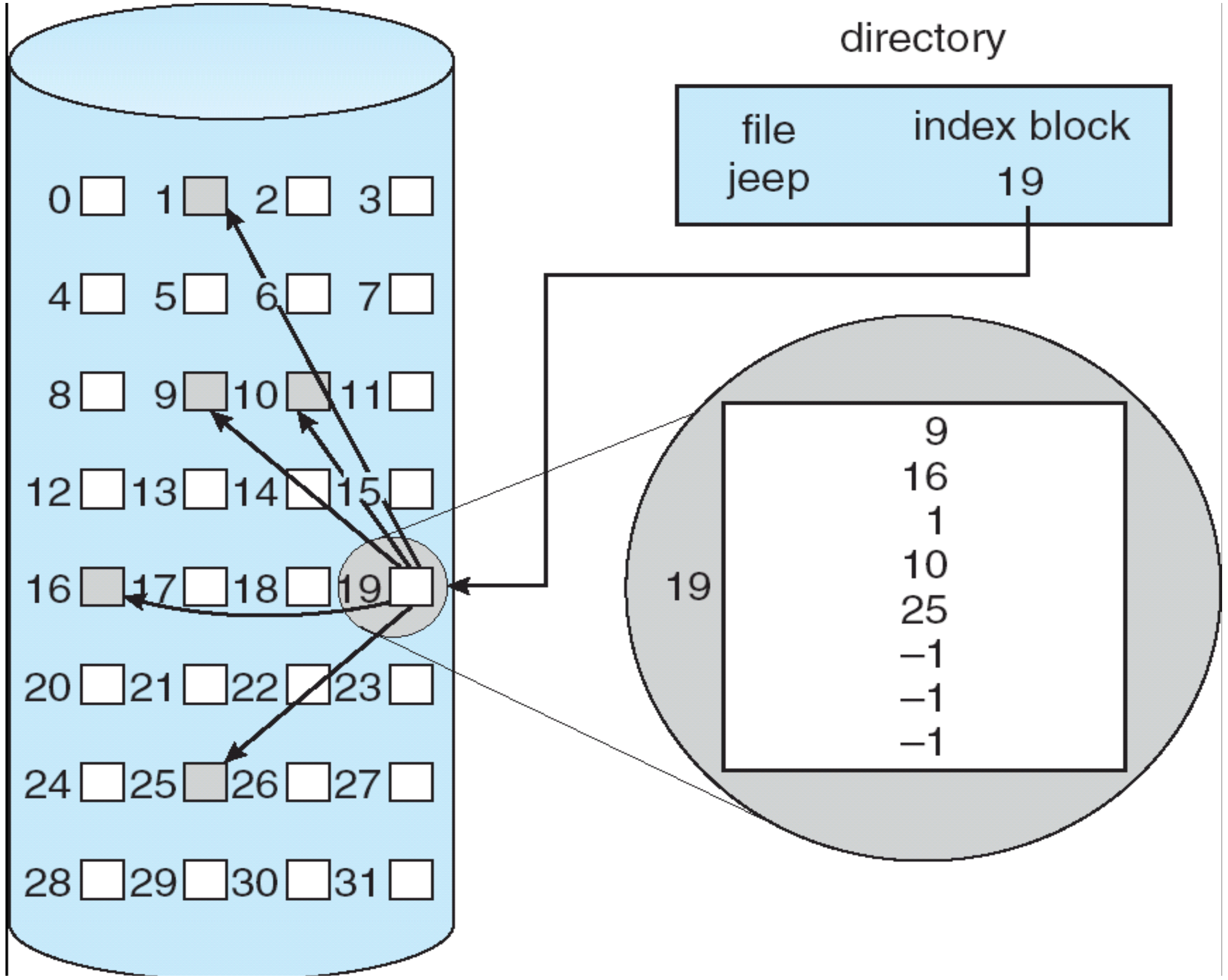
- Brings all pointers together into the index block.
- Logical view.



index table

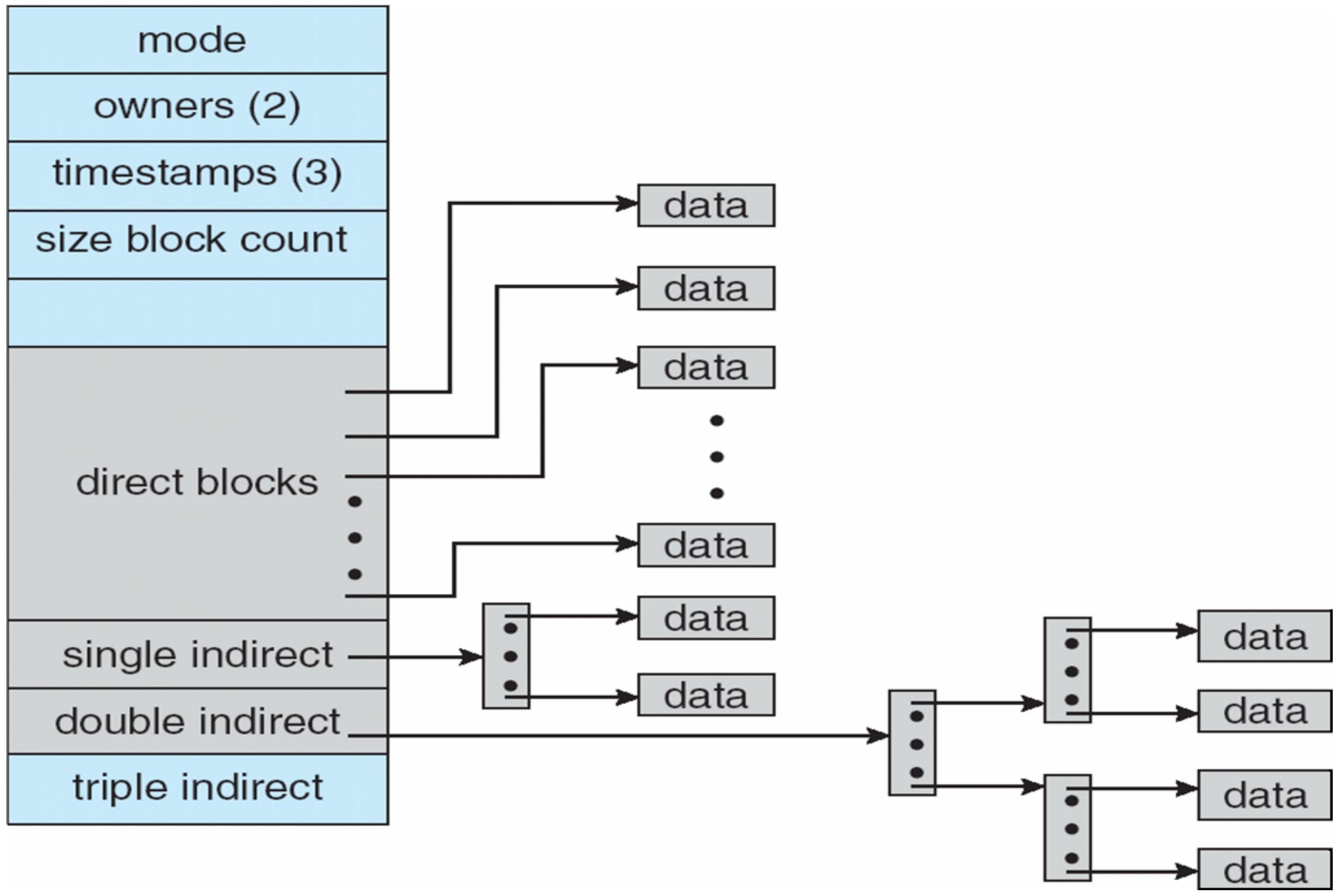
- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.

Example of Indexed Allocation

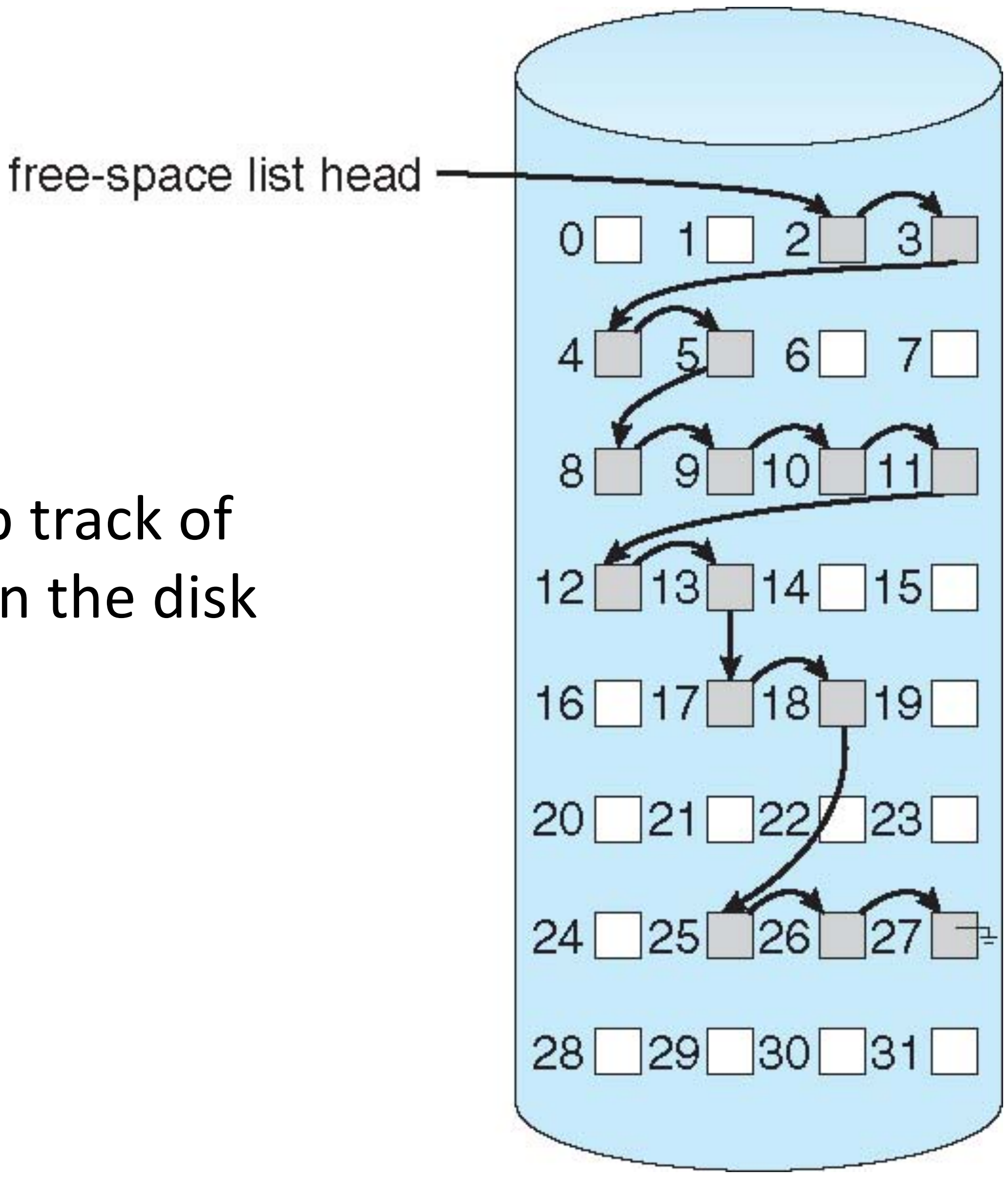


Combined Method:UNIX (4K bytes per block)

- Small files do not have an index block
- Large files can have double or triple indirection



Linked Free Space List on Disk



Need to keep track of free blocks on the disk

OS Course

- Process Management
  - Process Creation/Termination
  - Process State, PCB
  - Multithreading - Pthreads
  - Process Scheduling
  - Synchronization
  - Deadlocks
- Memory Management
  - Memory Management
  - Virtual Memory Management
  - File System
  - Distributed File Systems (Next Lecture)

Related Upper-Level Courses

- Network and Distributed Systems
- Computer Security
- Parallel Computing

Acknowledgments

- These slides are adapted from
  - Öznur Özkasap (Koç University)
  - Operating System and Concepts (9th edition) Wiley
  - Jerry Breeche

https://www.usna.edu/Users/cs/wcbrown/courses/IC221/classes/L09/Class.html