

NJIT

New Jersey's Science &
Technology University

THE EDGE IN KNOWLEDGE

Alex. Gerbessiotis

CS Department



The yellowish/brownish notes are copyright the textbook author.

The bluish notes are copyright the instructor and are add-ons and provide more explanation or additional details



Module 1

*Operating
Systems:
Internals
and Design
Principles*

Chapter 12

File Management

Ninth Edition
By William Stallings

Sub 9 vs Sub 10



Sub 9:	Low level references to	
	Sectors	
	Clusters	(consecutive sectors)
	Tracks	
Sub 10:	Cylinders	(tracks over all surfaces with same track number)
	Higher level references to	
	Blocks	(multiple Sectors or Tracks)
	Portions	(multiple blocks)
FileSystems		
Files (one type is a Directory)		
Databases		



Physical Drive: A disk device also known as disk drive (HDD, SSD, FDD, etc) is a block device drive; It is divided into several partitions

Partition: A subdivision of a physical (disk) drive. Each partition can contain ONE filesystem

Partition: A 'logical' disk (e.g. Cylinders 0-50 for a partition)

Volume: A collection of sectors or other volumes

Filesystem: It is a structure that defines a format for a partition of a disk drive.

File: A filesystem contains files (data grouped together)

Database(s): File(s) that have some organization

Filesystem Operations: Create (i.e. format a partition of a physical drive),
Mount (a filesystem i.e. activate it)
Unmount (deactivate a filesystem)

Metadata: Data on Data

Volumes

- A collection of addressable sectors in secondary memory that an OS or application can use for data storage
- The sectors in a volume need not be consecutive on a physical storage device
 - They need only appear that way to the OS or application
- A volume may be the result of assembling and merging smaller volumes



A file system contains (BootBlock, SuperBlock, InodeT, DataB)

BootBlock bootstrap instructions used in boot time

SuperBlock contains
sizeofFSystem, ownership info,
#Inodes
#blocks (in files),
#freeblocks,
#startofFreeBlkList,
#startofFreeInodeList

InodeTable (List) is an array of Inodes (index nodes on files)

Data Blocks are data related blocks containing data of files.

In *NIX the **name of a file** is its **inode number** (index to InodeTable)

In *NIX the **filename of a file** is an alias to its **inode number**



Metadata

is information about the data

InodeTable/List or InodeT

The “array” referred to hereafter as **InodeT** in the file system that contains meta-data about files is known in Unix as inode table
An entry of InodeT (in Unix filesystems 128-256B typical size)

Inode table entry.

Inode number (index)

inode # : an index (offset) into InodeT that identifies a unique entry (inode# is a non-negative integer number)

InodeT[inode#]

The inode table entry for inode number inode#

InodeT.length

maximum number of files in filesystem!

Range of inode#

0 through InodeT.length -1

Name of a file (by filesystem)

The inode# (what is supposed to be a file name)

filename

An alias created by the UserOwner of the file to identify the file instead of using its inode# (aka name file): a file can have >1 aliases
It is the alias created and stored in directory into which file was created
It is a file of type directory that contains alias pairs for every file of any type created into it plus one alias for itself (. : current directory) and one alias for its parent (..: parent directory).

Pair (filename, inode#)

Directory

Root of the filesystem

The master (primary) directory of the filesystem; an ordinary directory that is only identified by the fact that its parent is itself!
That is parent of the root of the filesystem is the root itself!

Volume Structure

- A UNIX file system resides on a single logical disk or disk partition and is laid out with the following elements:

Boot block

Contains code required to boot the operating system

Superblock

Contains attributes and information about the file system

Inode table

Collection of inodes for each file

Data blocks

Storage space available for data files and subdirectories



In Linux, a 'file' can be ... a

1. **a file**, or

2. a **directory**

special

- 3. a character device file
- 4. a block device file
- 5. a socket
- 6. a named pipe

7. a symbolic link (soft link)

(hard) link

TEXTBOOK (next page) combines 3-4 into a type "special"; ignores 5.

TEXTBOOK calls a (hard) link

(aka everything is a file)

(a regular or ordinary file)

(a file of some specific content)

(a file used for character-based I/O)

(a file used for block-based I/O eg HDD)

(a file used for networked comm.)

(a FIFO I/O queue;

unnamed pipe |: `ps -ef | egrep userID`)

(a file of some specific content)

is not a file type; is another
alias for same file.

a file type: it is not!

UNIX File Management

- In the UNIX file system, six types of files are distinguished:

Regular, or ordinary

- Contains arbitrary data in zero or more data blocks

Directory

- Contains a list of file names plus pointers to associated inodes (index nodes)

Special

- Contains no data but provides a mechanism to map physical devices to file names

Named pipes

- An interprocess communications facility

Links

- An alternative file name for an existing file

Symbolic links

- A data file that contains the name of the file to which it is linked

File Systems

- Provide a means to store data organized as files as well as a collection of functions that can be performed on files
- Maintain a set of attributes associated with the file
- Typical **file** operations (**NOT FILE SYSTEM ops**) include:
 - Create
 - Delete
 - Open
 - Close
 - Read
 - Write

Files

- **FILES:** Data collections created by users **and maintained on secondary storage**
- The File System is one of the most important parts of the OS to a user
- Desirable properties of files:

Long-term existence

- Files are stored on disk or other secondary storage and do not disappear when a user logs off

Sharable between processes

- Files have names and can have associated access permissions that permit controlled sharing

Structure

- Files can be organized into hierarchical or more complex structure to reflect the relationships among files

Minimal User Requirements

■ Each user:

1

- Should be able to create, delete, read, write and modify files

2

- May have controlled access to other users' files

3

- May control what type of accesses are allowed to the users' files

4

- Should be able to move data between files

5

- Should be able to back up and recover files in case of damage

6

- Should be able to access his or her files by name rather than by numeric identifier

File Structure

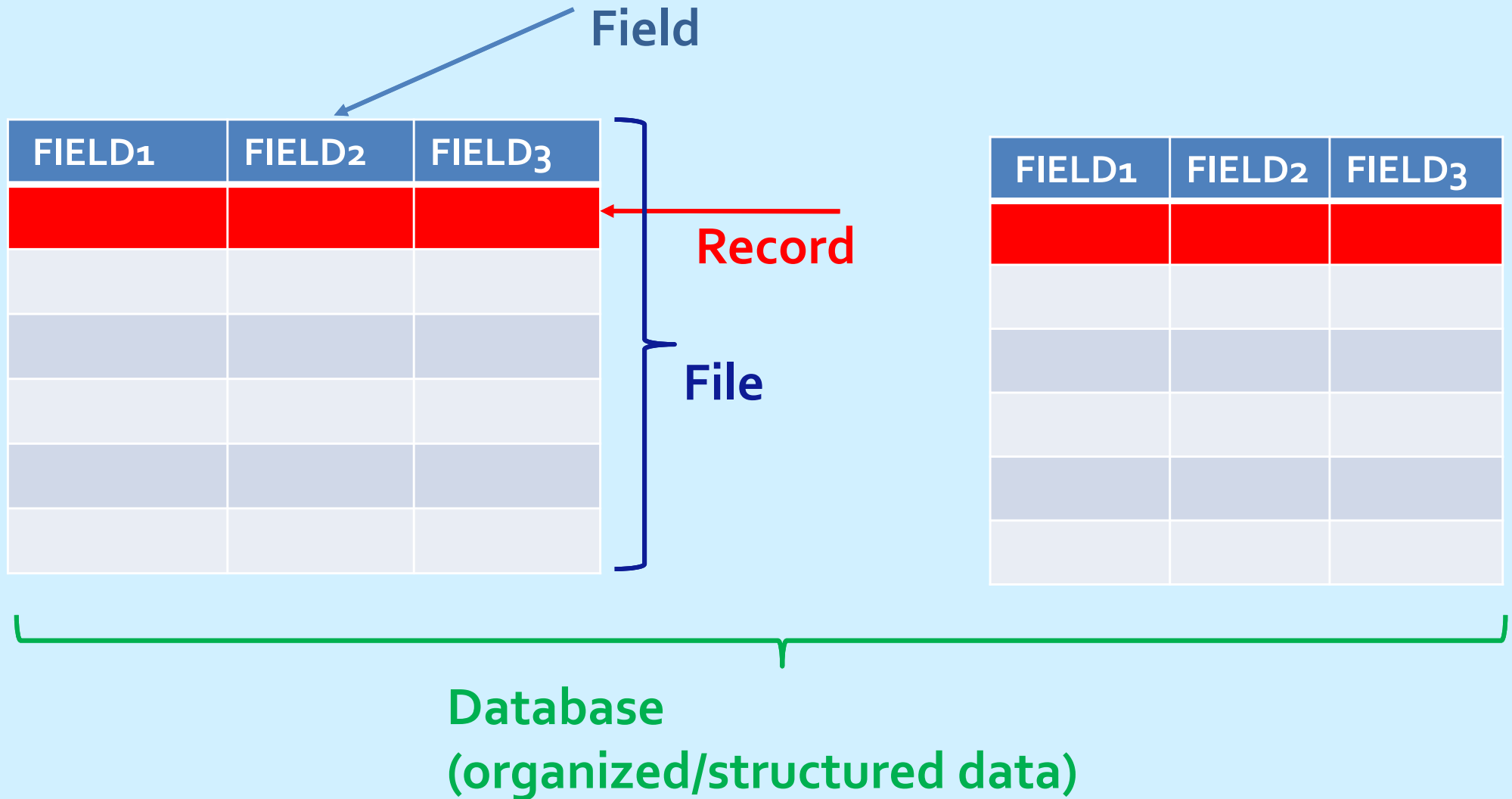
Four terms are commonly used when discussing files:

Field

Record

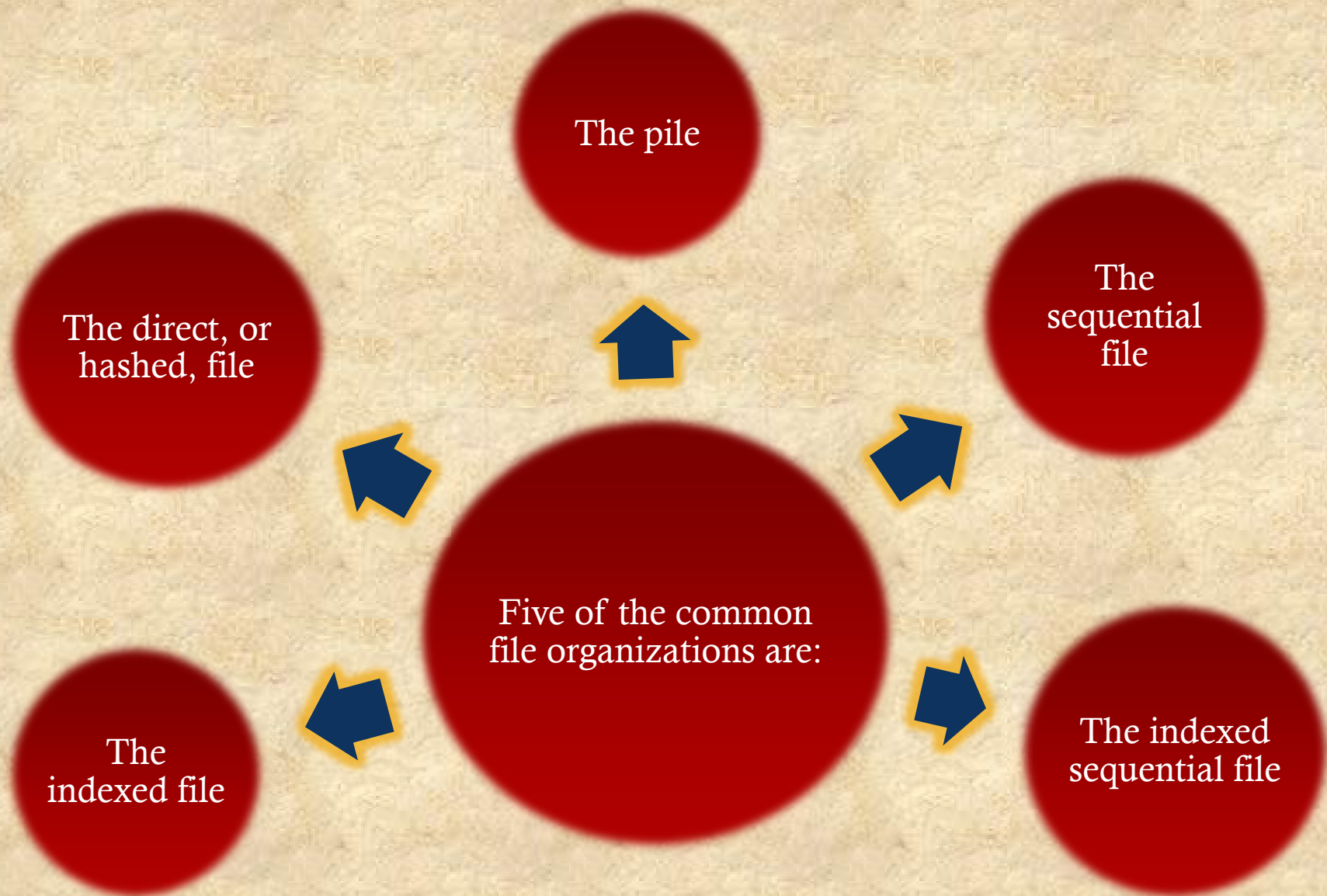
File

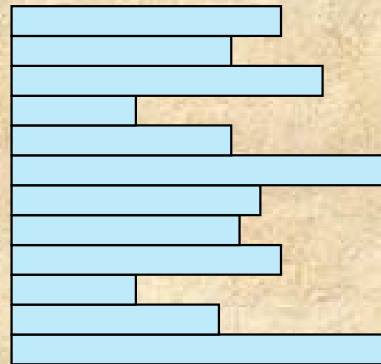
Database



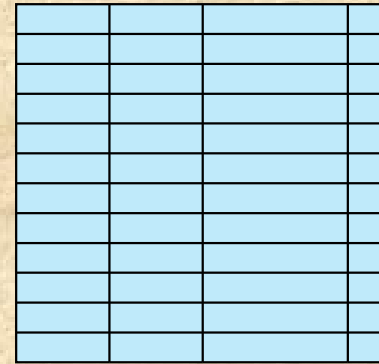
'Some view' does not mean 'exclusive view'

File Organization Types





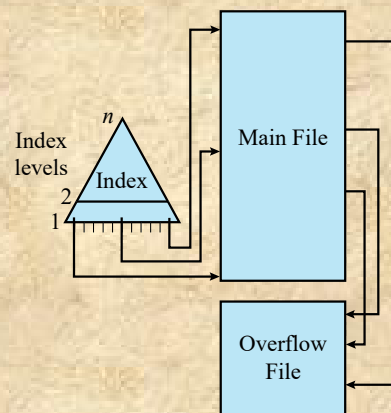
Variable-length records
Variable set of fields
Chronological order



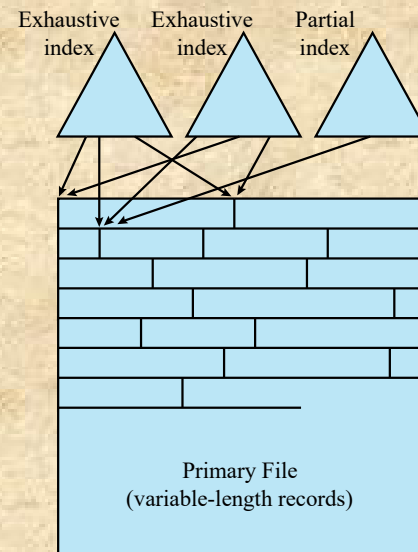
Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(a) Pile File

(b) Sequential File



(c) Indexed Sequential File



(d) Indexed File

Figure 12.3 Common File Organizations

B-Trees

Discard ... To be discussed in a DB course (CS 331)

- A balanced tree structure with all branches of equal length
- Standard method of organizing indexes for databases
- Commonly used in OS file systems
- Provides for efficient searching, adding, and deleting of items

Operations Performed on a Directory

- To understand the requirements for a file structure, it is helpful to consider the types of operations that may be performed on the directory:



Can be root directory of file system or arbitrary directory

Master Directory

An internal node that contains children i.e. it is a non-empty directory and its children are the files it contains

an empty subdirectory is a leaf here

Subirectory

Subirectory

Subirectory

Subirectory

Subirectory

File

File

File

File

A master directory or subdirectory (the indicated subirectory is a Typo☺) is not a file type; just a characterization for a specified directory

A leaf that is not an empty directory is a file of some type (other than directory)

Caution: This is not the structure of a Unix or Linux file system. In Unix / Linux a filesystem is a directed acyclic graph.

Figure 12.6 Tree-Structured Directory

File Sharing

Two issues arise
when allowing files
to be shared among
a number of users:

```
graph TD; A[Two issues arise when allowing files to be shared among a number of users:] --- B[Access rights]; A --- C[Management of simultaneous access]
```

Access rights

Management of
simultaneous
access

Access Rights

- *None*

- The user would not be allowed to read the user directory that includes the file

- *Knowledge*

- The user can determine that the file exists and who its owner is and can then petition the owner for additional access rights

- *Execution*

- The user can load and execute a program but cannot copy it

- *Reading*

- The user can read the file for any purpose, including copying and execution

- *Appending*

- The user can add data to the file but cannot modify or delete any of the file's contents

- *Updating*

- The user can modify, delete, and add to the file's data

- *Changing protection*

- The user can change the access rights granted to other users

- *Deletion*

- The user can delete the file from the file system

User Access Rights

Owner

Usually the initial creator of the file

Has full rights

May grant rights to others

Specific Users

Individual users who are designated by user ID

User Groups

A set of users who are not individually defined

All

All users who have access to this system

These are public files



Basic (Unix) Model,
Basic (Linux) Model:

Three entities (triads) are assigned access rights

u : **U**ser owner (creator) of file

g : **G**roup of user owner

o : Everybody else (**O**ther)

Posix Model:
(previous slide)

More than the three entities of the basic model

u, g, o (basic model entities) plus
arbitrary individual users (not just o) plus
arbitrarily defined groups (not just g)

(Access) Rights,
or Privileges,
or mode(s):

r: read, w:write, x:execute

For dirs: r : means list directory,

w:create/delete files

x: use cd command; traverse dir

Andrew File System(AFS):

Non-directory files: Basic Unix Model

Directory files: AFS supported access rights



Get the access rights for a file named filename

```
ls -l filename
```

```
ls -la filename #includes hidden files
```

```
stat filename OR ls -lai filename #includes inode#
```

Change the access right for a file

```
chmod newrights filename
```

where newrights is a triplet of 0-7 specifying the new permissions of entities u,g,o.

Example 1.

```
% ls -l file1
```

```
-rwxr-xr-x 1 alexg users 22 Dec 4 11:40 file1
```

```
% chmod 640 file1
```

```
% ls -li file1
```

```
315258964 -rw-r----- 1 alexg users 22 Dec 4 11:40 file1
```

```
% ls -l file1
```

```
-rw-r----- 1 alexg users 22 Dec 4 11:40 file1
```



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
-	r	w	x	r	-	x	r	-	x	1	alexg	users	22	Dec	4 11:40 file1

- a** is file type : - (regular), **d**(directory), **b** (block), **c**(character),
s (socket), **p** (named pipe), **l**(softlink/symbolic link)
- bcd** : User owner's permissions (r/w/x/ or no: r,w,x,-)
- efg** : Group permissions (r/w/x/ or no: r,w,x,-)
- hij** : Others' permissions (r/w/x/ or no: r,w,x,-)
- k** : Hard link count
- l** : UserID of user owner of file1
- m** : GroupID of UserID
- n** : Filesize in B
- o** : Timestamp (Creation date or time)
- p** : Name of file (alias used by user to identify file)



inode#	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
315258964	-	r	w	-	r	-	-	-	-	-	1	alexg	users	22	Dec	4 11:40 file1

inode#

IS the filename known to filesystem

file1

IS an alias 'filename' used by the creator

AND is not known by the filesystem

file1

IS available in directory storing file1

setUID bit

Affects d bit (x of u). An x becomes an s; a - becomes an S

setGID bit

Affects g bit (x of g). An x becomes an s; a - becomes an S

S setUID/setGID

File will be executed with permissions of owner of file
not the user executing it

How to:

chmod u+s filename

or

chmod g+s filename

chmod u-s filename

or

chmod g-s filename

Sticky bit

Affects j bit (x of o)

t sticky bit

For shared files only owner can remove (delete) it but no one else in group can do so

If you see a + after the j character (on the left of the one) then file1 must be a directory and Posix permissions are available (only for directories including files in them).

-rwxr-x-wx 1 alexg users 22 Dec 6 11:40 file


 4,2,1 4,2,1 4 2 1
 r w x r - x - w x
 4+2+1 4 + 1 2 + 1
 7 5 3

View 1

A dash is a 0;
 An r is 4, w is a 2, x is an 1
 Add them up
 Sum is in 0-7 range

View 2

A dash is a 0;
 An r is 1, w is a 1, x is an 1
 Access rights is a three-bit binary number
 Convert to denary or octal
 Result is in 0-7 range

r w x r - x - w x
 1 1 1 1 0 1 0 1 1
 4+2+1 4 + 1 2 + 1
 7 5 3

0	No Permission	---
1	eXecute	--X
2	Write	-W-
3	W+X	-WX
4	Read	r--
5	R+X	r-X
6	R+W	rw-
7	R+W+X	rwX

Examples:

% ls -l file1

-rw-r----- 1 alexg users 22 Dec 4 11:40 file1

% chmod 777 file1

% ls -l file1

-rwxrwxrwx 1 alexg users 22 Dec 4 11:40 file1

% chmod u-x file1

% ls -l file1

-rw-rwxrwx 1 alexg users 22 Dec 4 11:40 file1

% chmod permissions filename

**Access rights ==
 permissions ==
 privileges**

Practice 1



- 1.1. Name 5 (or 6) operations on files.
- 1.2. Does an indexed file allow for sequential access?
- 1.3. Who can read somefile?

```
-rwx--x-wx 1 alexg tape 0 Mar 27 15:41 somefile
```

- 1.4 Who can execute somefile?



Module 2

File Allocation

- On secondary storage, a file consists of a collection of blocks
- The operating system or file management system is responsible for allocating blocks to files
- The approach taken for file allocation may influence the approach taken for free space management
- Space is allocated to a file as one or more *portions* (contiguous set of allocated blocks)
eg. MS-DOS
- *File allocation table (FAT)*
 - Data structure used to keep track of the portions assigned to a file

Preallocation vs Dynamic Allocation

- A preallocation policy requires that the maximum size of a file be declared at the time of the file creation request
- For many applications it is difficult to estimate reliably the maximum potential size of the file
 - Tends to be wasteful because users and application programmers tend to overestimate size
- Dynamic allocation allocates space to a file in portions as needed

Portion Size

- In choosing a portion size there is a trade-off between efficiency from the point of view of a single file versus overall system efficiency
- Items to be considered:
 - 1) Contiguity of space increases performance, especially for `Retrieve_Next` operations, and greatly for transactions running in a transaction-oriented operating system
 - 2) Having a large number of small portions increases the size of tables needed to manage the allocation information
 - 3) Having fixed-size portions simplifies the reallocation of space
 - 4) Having variable-size or small fixed-size portions minimizes waste of unused storage due to overallocation

Alternatives

Two major alternatives:

Variable, large contiguous portions

- Provides better performance
- The variable size avoids waste
- The file allocation tables are small

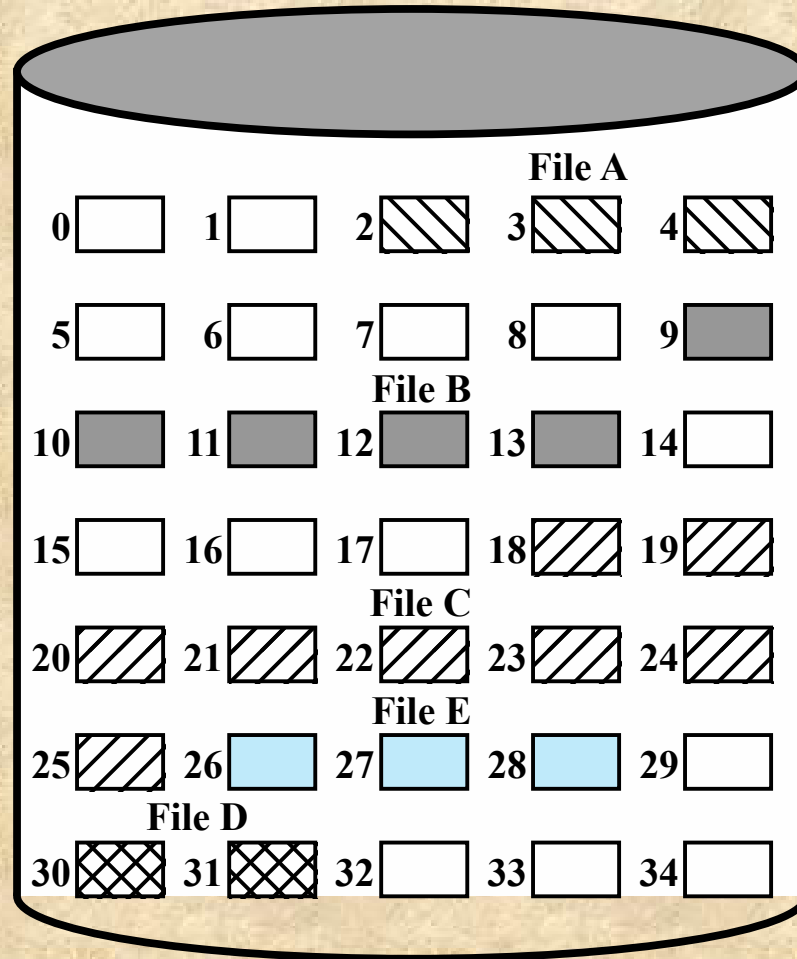
Blocks

- Small fixed portions provide greater flexibility
- They may require large tables or complex structures for their allocation
- Contiguity has been abandoned as a primary goal
- Blocks are allocated as needed

Table 12.2

File Allocation Methods

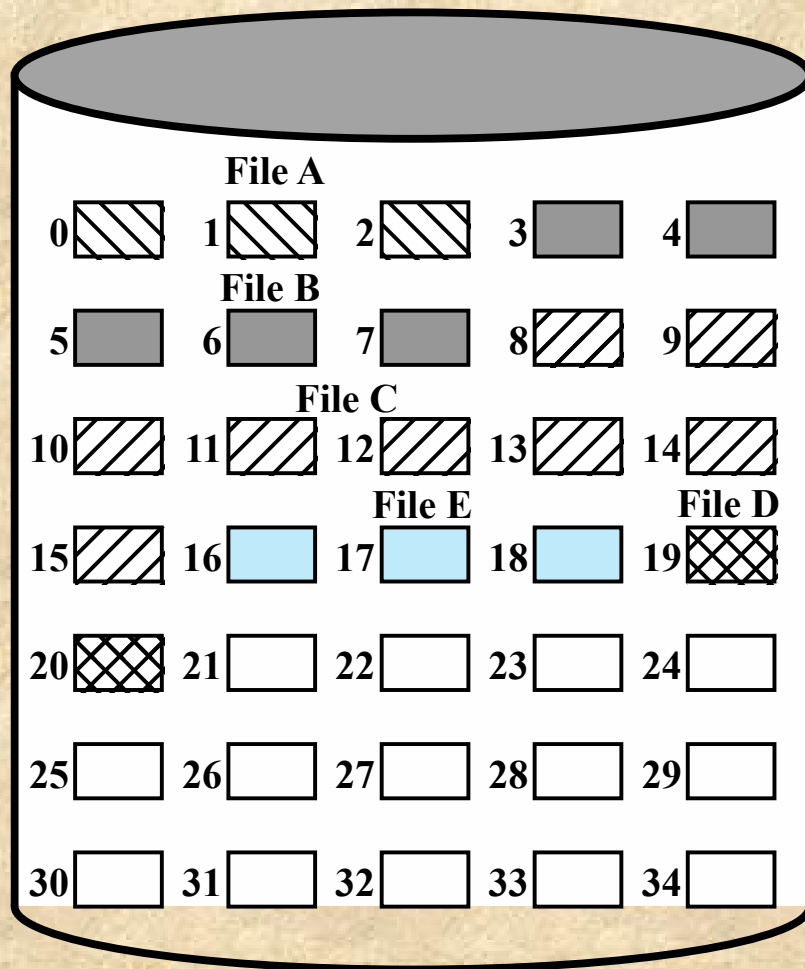
	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

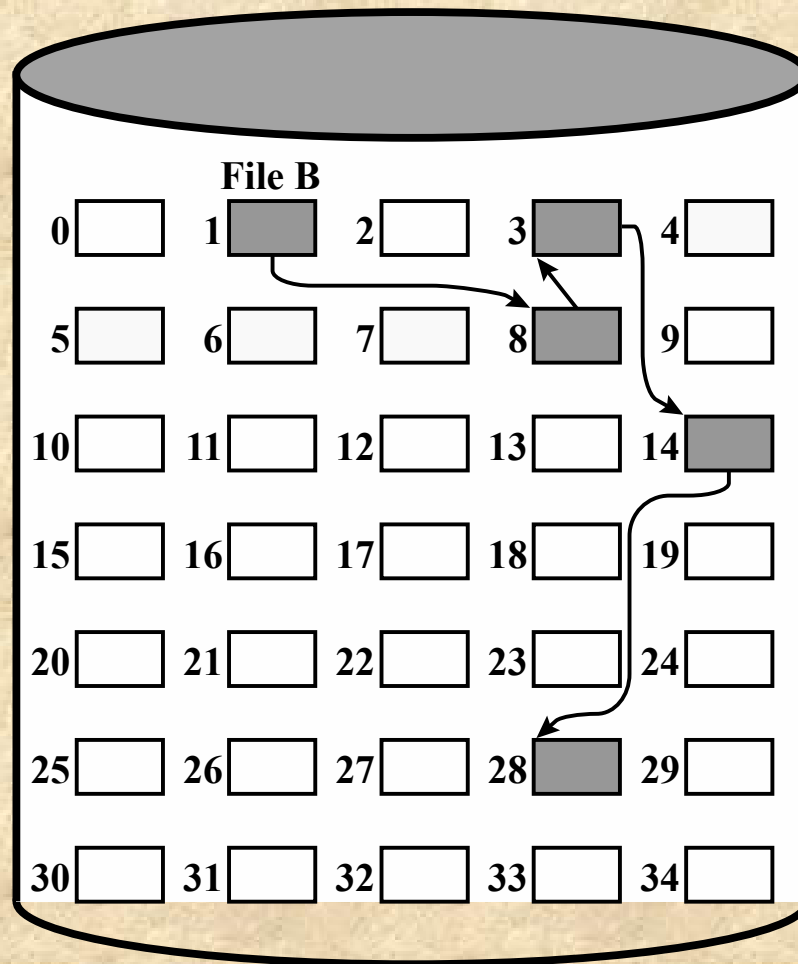
Figure 12.9 Contiguous File Allocation



File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

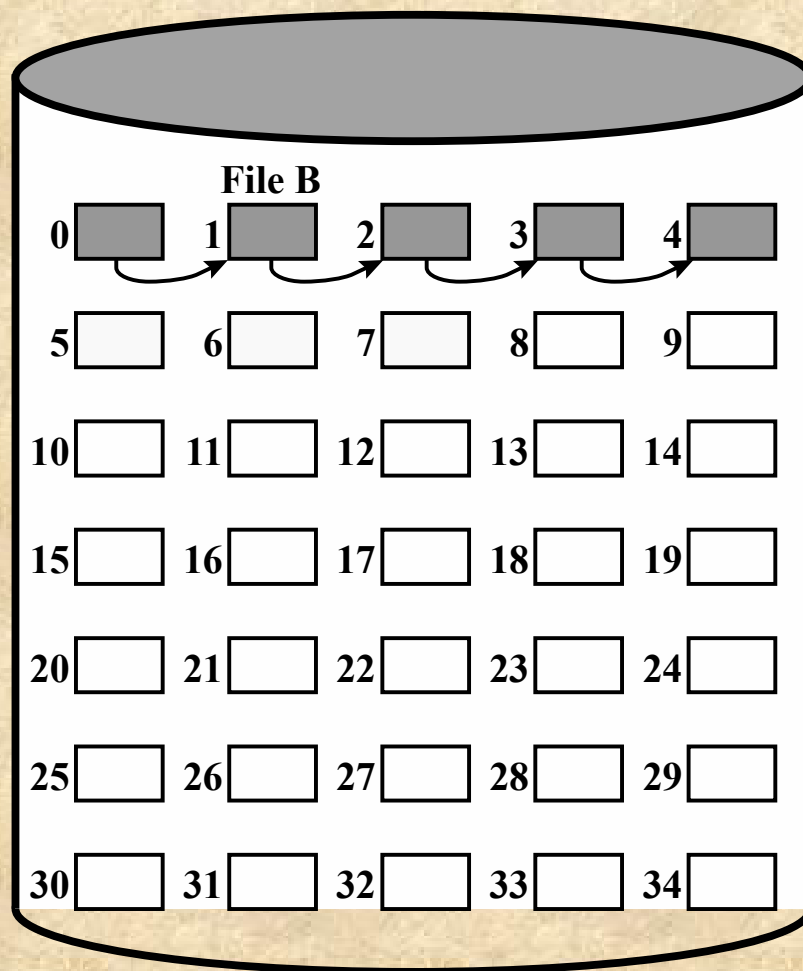
Figure 12.10 Contiguous File Allocation (After Compaction)



File Allocation Table

File Name	Start Block	Length
...	...	...
File B	1	5
...	...	...

Figure 12.11 Chained Allocation



File Allocation Table

File Name	Start Block	Length
...	...	...
File B	0	5
...	...	...

Figure 12.12 Chained Allocation (After Consolidation)

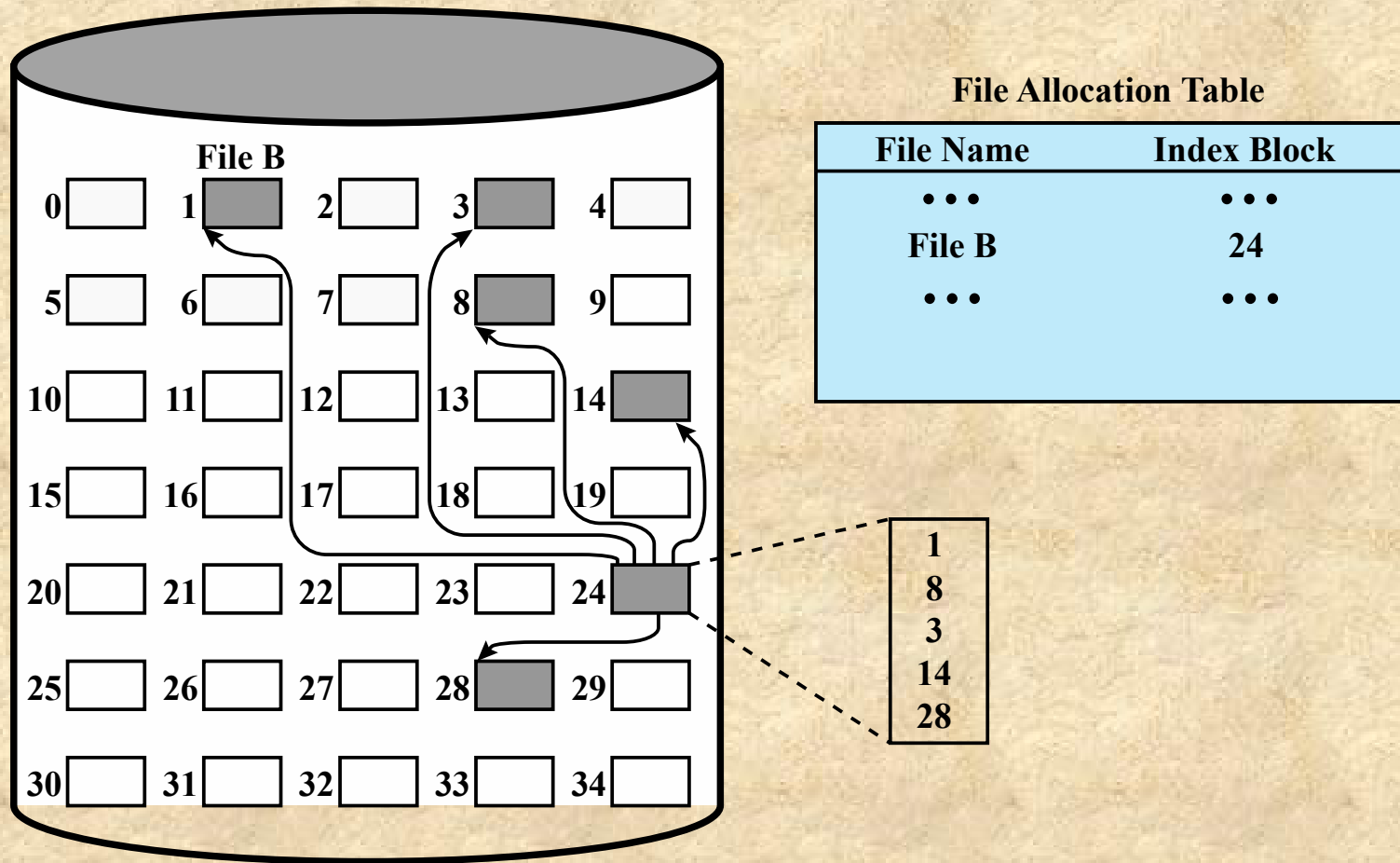
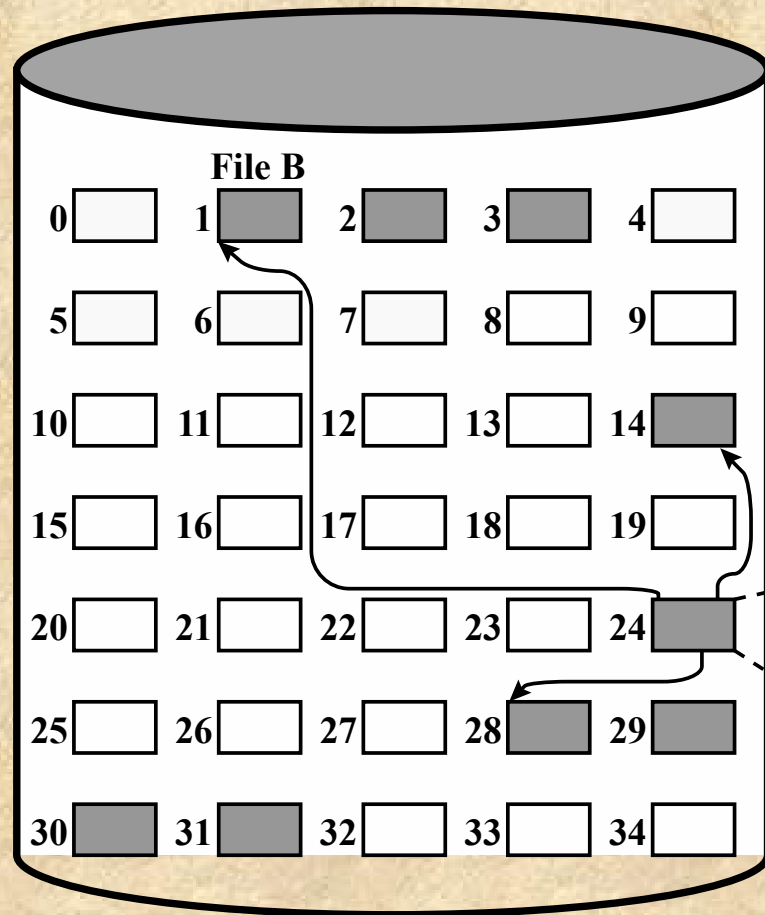


Figure 12.13 Indexed Allocation with Block Portions



File Allocation Table

File Name	Index Block
...	...
File B	24
...	...

Start Block	Length
1	3
28	4
14	1

Figure 12.14 Indexed Allocation with Variable-Length Portions

Free Space Management

- Just as allocated space must be managed, so must the unallocated space
- To perform file allocation, it is necessary to know which blocks are available
- A *disk allocation table* is needed in addition to a file allocation table

Bit Tables

- This method uses a vector containing one bit for each block on the disk
- Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use

Advantages:

- Works well with any file allocation method
- It is as small as possible

UNIX File Management

- In the UNIX file system, six types of files are distinguished:

Regular, or ordinary

- Contains arbitrary data in zero or more data blocks

Directory

- Contains a list of file names plus pointers to associated inodes (index nodes)

Special

- Contains no data but provides a mechanism to map physical devices to file names

Named pipes

- An interprocess communications facility

Links

- An alternative file name for an existing file

Symbolic links

- A data file that contains the name of the file to which it is linked



In Linux, a 'file' can be ... a

a file, or

a **directory**

special

- a character device file
- a block device file
- a socket
- a named pipe

a symbolic link (soft link)

(hard) link

(aka everything is a file)

(a regular file)

(a file of some specific content)

(a file used for character-based I/O)

(a file used for block-based I/O eg HDD)

(a file used for networked comm.)

(FIFO I/O;

unnamed pipe: `ps -ef | egrep userID`)

(a file of some specific content)

is not a file type is another name
entry for same file; **TEXTBOOK**
THOUGH CALLS IT a FILE TYPE



Example.

pipeONE is to become a FIFO queue

WINDOW₁ (CONSUMER)

```
% egrep 2020 <pipeONE
```

WINDOW₂ (PRODUCER)

```
% cd /tmp
```

```
% mkfifo pipeONE
```

```
% for n in {1..1000};do date ; done >/tmp/pipeONE
```

```
% rm /tmp/pipeONE
```

Note: At NJIT named pipes can only exist in directory /tmp.

Inodes

- All types of UNIX files are administered by the OS by means of inodes
- An inode (index node) is a control structure that contains the key information needed by the operating system for a particular file
- Several file names may be associated with a single inode
 - An active inode is associated with exactly one file
 - Each file is controlled by exactly one inode



Inodes (repeat)

Metadata

is information about the data

InodeTable/List or InodeT

The “array” referred to hereafter as **InodeT** in the file system that contains meta-data about files is known in Unix as inode table
An entry of InodeT (in Unix filesystems 128-256B typical size)

Inode table entry.

Inode number (index)

inode # : an index (offset) into InodeT that identifies a unique entry (inode# is a non-negative integer number)

InodeT[inode#]

The inode table entry for inode number inode#

InodeT.length

maximum number of files in filesystem!

Range of inode#

0 through InodeT.length -1

Name of a file (by filesystem)

The inode# (what is supposed to be a file name)

filename

An alias created by the UserOwner of the file to identify the file instead of using its inode# (aka name file): a file can have >1 aliases
It is the alias created and stored in directory into which file was created
It is a file of type directory that contains alias pairs for every file of any type created into it plus one alias for itself (. : current directory) and one alias for its parent (..: parent directory).

Pair (filename, inode#)

Directory

Root of the filesystem

The master (primary) directory of the filesystem; an ordinary directory that is only identified by the fact that its parent is itself!
That is parent of the root of the filesystem is the root itself!



What does **InodeTable[315239608]** contain?

device number	majorID-minorID
inode number	inode# (unique for filesystem)
type & mode	(12 bit: type of file(3bit), mode: accessrights)
link count	number of hard links (aliases)
userID, groupID	
device rep	if inode represents a device, its majorID-minorID
filesize	in B (for a symbolic link length of path only)
IOblksize	preferred block size for I/O
#blcks	less refined than filesize
LastAccess, FileCreation	timestamps
LastModification, LastStatusUpdate	timestamps
List of Blocks (15 pointers*)	More to follow (next page)

*** a pointer is a block# usually it is 8B**



List of Blocks (15 pointers) last line of previous page

(textbook in Figure 12.15 shows 16 pointers in error)

a pointer is in fact a block#

a block# can be a generic block number or

a block# is a block number of a datablock (aka datablock#)

[in this latter case: datablock = contains data of file]

12 of those 15 pointers are datablock#

13th ptr (single indirect) is block# of a block that contains datablock#

14th ptr (double indirect) is block# of a block that contains block# of
 blocks that contain datablock#

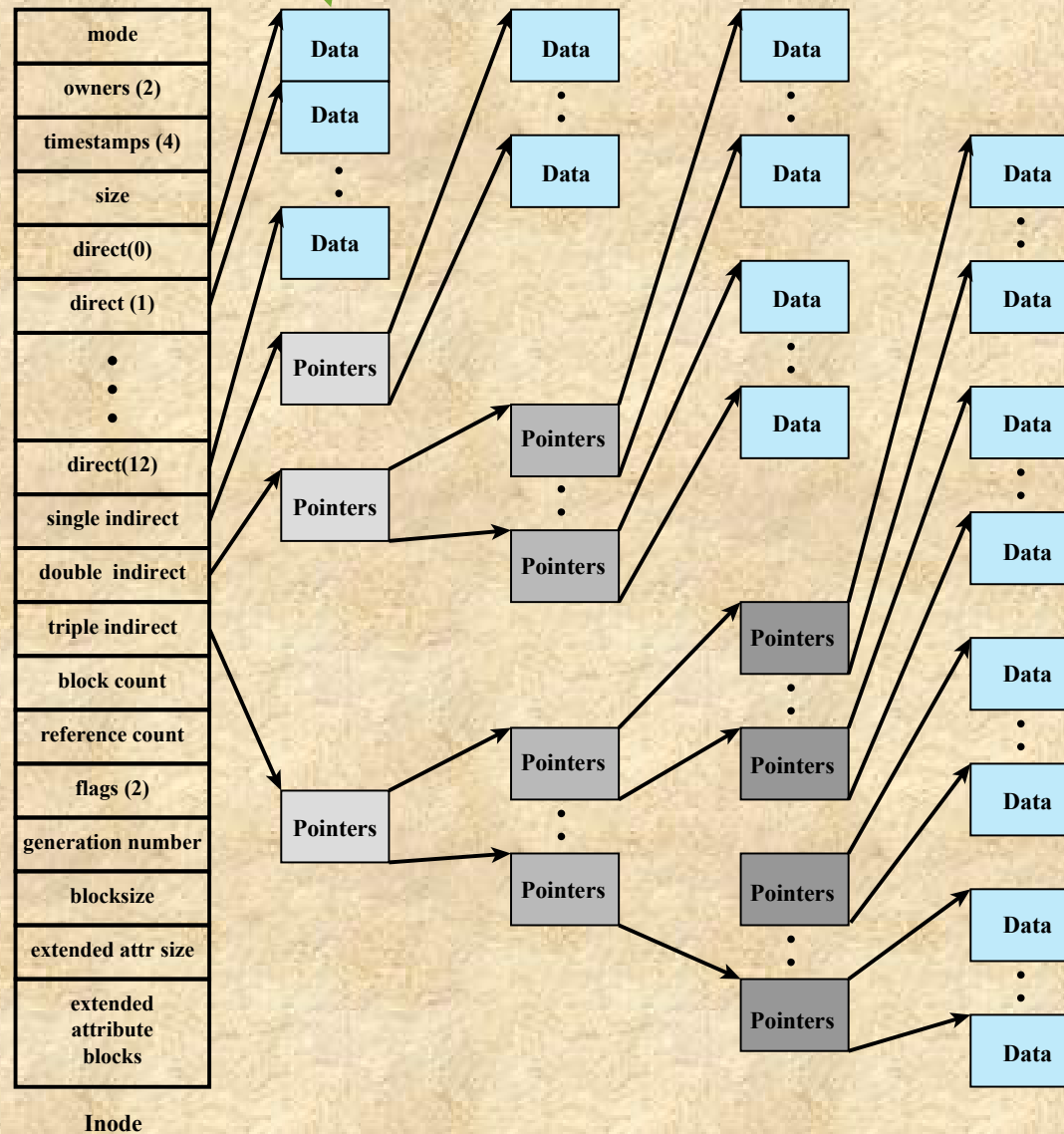
15th ptr (triple indirect) is block# of a block that contains block# of
 blocks that contain block# of
 blocks that contain datablock#

1 block is 4KiB == 4096B

Each block is either a datablock and contains data (4096B)
 or it contains 32-bit pointers i.e. block# i.e 1024 32-bit block#
 or it contains 64-bit pointers i.e. block# i.e. 512 64-bit block#

12 pointers to
data blocks

Fig 12.15
indicates 13
pointers from
direct(0)
through
direct(12)
They must be
12 not 13 from
direct(0)
through
direct(11)



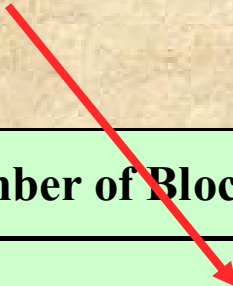
TYP0 in Textbook

Figure 12.15 Structure of FreeBSD inode and File

Table 12.3

Capacity of a FreeBSD File with 4-Kbyte Block Size

In this slide the textbook uses the correct 12 not a 13



Level	Number of Blocks	Number of Bytes
Direct	12	48K
Single Indirect	512	2M
Double Indirect	$512 \times 512 = 256K$	1G
Triple Indirect	$512 \times 256K = 128M$	512G

Moreover a 4KiB (aka 4-Kbyte) block holds 512 64-bit (8B) block#

Example: We are going to assume that Bsize=64B and Block# are 8B i.e. 16 hexadecimal digit but indicated as denary integers for lack of display space. Thus a block can contain $64B/block / 8B/Block\# = 8 Block\# / Block$ i.e. 8 Block# (block numbers)

Info extracted from
 InodeT[249]

Pointers 0-11 are block numbers.

Blocks with block numbers 100-111 contain data of the file with inode# 249.

Pointers 12 is also a block number (900).

The block with block numbers 900 contains 8 block# of blocks that contain data of the file with inode# 249.

Thus the block with block # 900 contains 8 block# 112-119 that contain additional data for the file with inode #249

Pointers 13 is also a block number (910).

The block with block numbers 910 contains 8 block# of block that each contains 8 block# of blocks that contain data of the file with inode# 249. (See next slide)

0	100
1	101
2	102
3	103
4	104
5	105
6	106
7	107
8	108
9	109
10	110
11	111
12	900
13	910
14	920

900

112
113
114
115
116
117
118
119

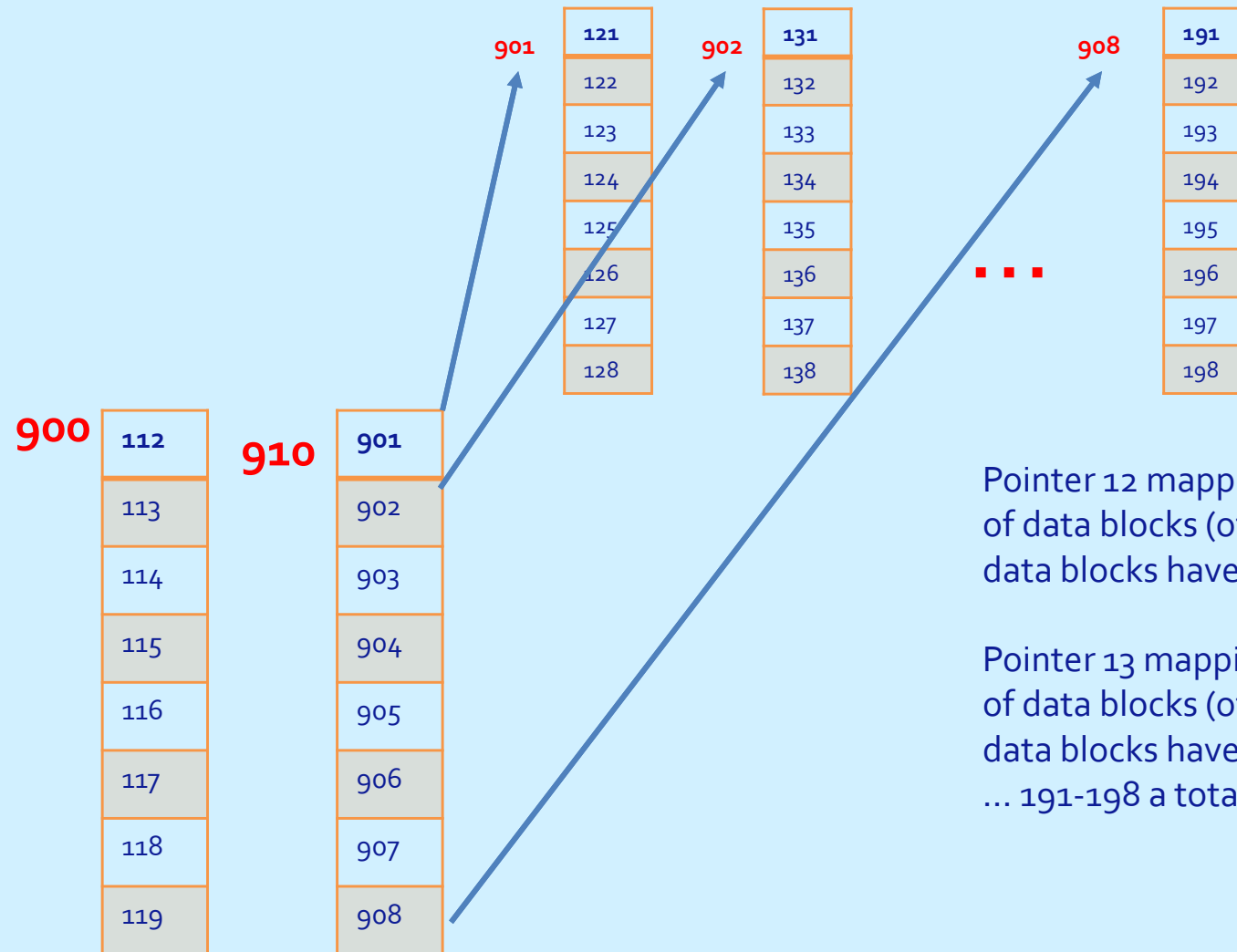
Part of the Inode (inode table entry) for a file with (eg) Inode# 249. All 15 ptrs (block numbers) are used and point to blocks of the file system



Pointers 13 is also a block number (910).

The block with block numbers 910 contains 8 block# of blocks that each contains 8 block# of blocks that contain data of the file with inode# 249.

0	100
1	101
2	102
3	103
4	104
5	105
6	106
7	107
8	108
9	109
10	110
11	111
12	900
13	910
14	920



Pointer 12 mapping to block# 900 avails 8 block# of data blocks (of the file with inode# 249). The data blocks have block# 112-119.

Pointer 13 mapping to block# 910 avails 64 block# of data blocks (of the file with inode# 249). The data blocks have block# 121-128, 131-138, 141-148, ... 191-198 a total of $8 \times 8 = 64$ data blocks

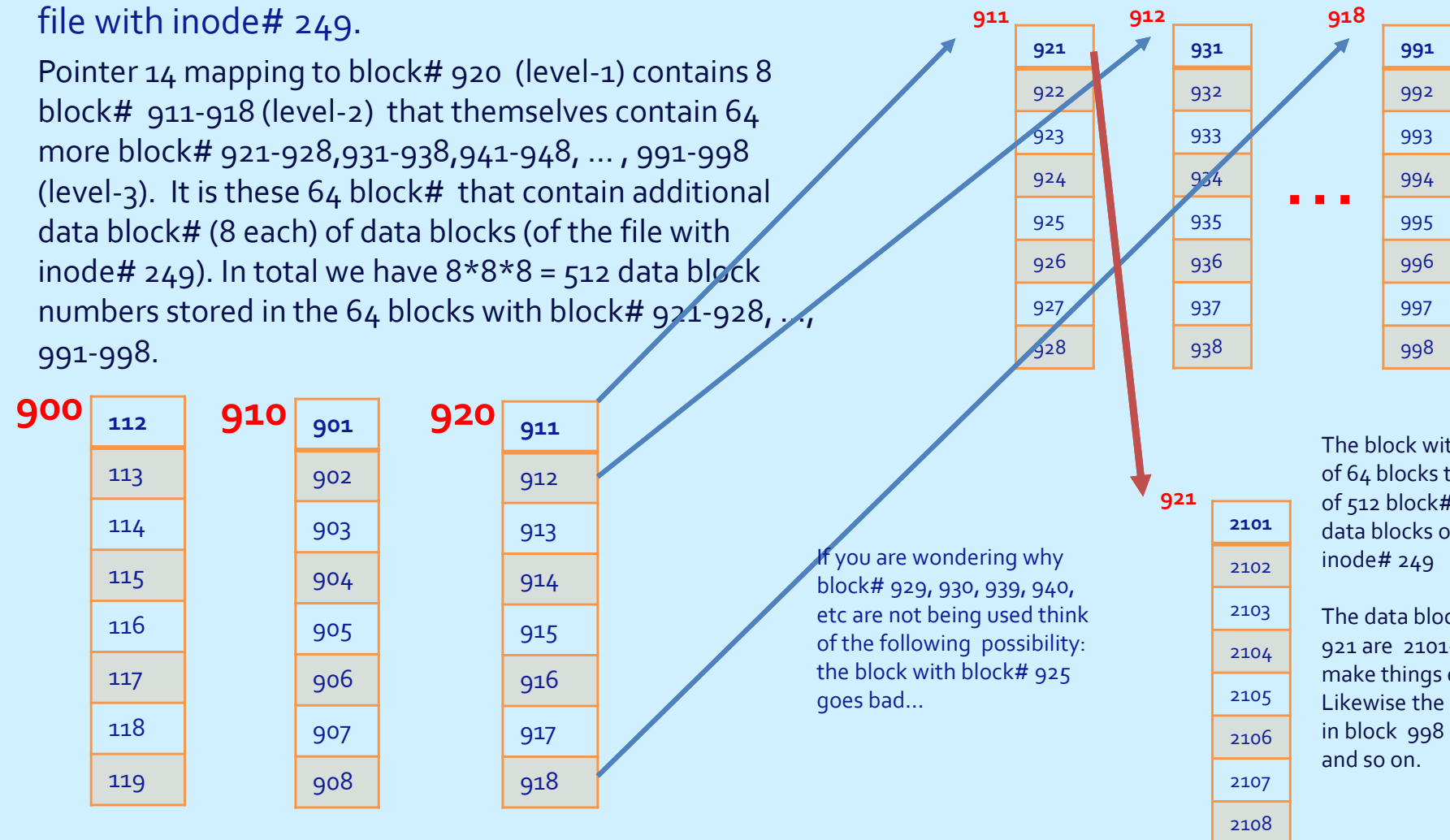
Example :Pointers aka Block# (Page 3/3)

Pointers 14 is also a block number (920).

The block with block number 920 contains 8 block# of block that each contains 8 block# of blocks that each contains 8 block# of block that each contains data of the file with inode# 249.

Pointer 14 mapping to block# 920 (level-1) contains 8 block# 911-918 (level-2) that themselves contain 64 more block# 921-928, 931-938, 941-948, ..., 991-998 (level-3). It is these 64 block# that contain additional data block# (8 each) of data blocks (of the file with inode# 249). In total we have $8 \times 8 \times 8 = 512$ data block numbers stored in the 64 blocks with block# 921-928, ..., 991-998.

0	100
1	101
2	102
3	103
4	104
5	105
6	106
7	107
8	108
9	109
10	110
11	111
12	900
13	910
14	920



If you are wondering why block# 929, 930, 939, 940, etc are not being used think of the following possibility: the block with block# 925 goes bad...

The block with block# 921 is one of 64 blocks that contains a total of 512 block# that identify 512 data blocks of the file with inode# 249

The data block# stored in block 921 are 2101-2108; this is to make things easy to the eye. Likewise the data block# stored in block 998 could be 9801-9808, and so on.



**A file does not use all the block numbers / pointers
that can possibly be defined.
How many they are used is defined by #blks in the inode table entry
for the file**

In Unix (and Linux) a file of type directory contains entries that are (filename, inode#) alias pairs

```
% rm -rf *
% ls
% ls -l
total 0
% pwd
/afs/cad/u/m/y/myucid/csabc
% touch my1 my2 my3                                # How many files in directory ?
% ls -li                                             # How many (alias,inode#) in directory file?
315239608 -rw----- 1 alexg users 0 Dec  4 14:14 my1
315258958 -rw----- 1 alexg users 0 Dec  4 14:14 my2
315258962 -rw----- 1 alexg users 0 Dec  4 14:14 my3
% ls -lai                                           # Watchout for hidden files; Use -a option as well
315170091 drwx----- 2 alexg users 2048 Dec  4 14:14 .
315168799 drwx----- 4 alexg users 2048 Dec  4 11:41 ..
315239608 -rw----- 1 alexg users  0 Dec  4 14:14 my1
315258958 -rw----- 1 alexg users  0 Dec  4 14:14 my2
315258962 -rw----- 1 alexg users  0 Dec  4 14:14 my3
% vi .                                             # . (period) is the current working directory
../
./
my1
my2
my3

#cleaned up some clutter; 5 entries in directory csabc
# three files created by us
# every directory contains a current directory alias (. or ../)
# as well as a parent directory alias (.. aka ../)
```



[1] Create (format) a file system

Action: Root of file system would be directory /
 Inode# of / is 128 by default (sometimes);
 Entry in InodeT is created; block assigned

Block 900

.	128
..	128

Report of activity

1. Inode number 128 is assigned to / and block 900 contains data of directory / which is the root of the filesystem

Directory structure:

alias name followed by inode number one entry per line

. and .. occupy first two entries of any directory block

. : is an alias to 'current directory'

.. : is an alias to 'parent directory'

For only one directory x of a filesystem is the parent of x equal to x. This is for x = / i.e. the root of the file system indicated by /

HardLinkCount of node# 128 is 2 because there are two aliases for 128 in the filesystem (. and .. Of directory / with info stored in block with block#900

Directory / is empty it has only the two default entries of a directory: . , ..

128,d, 2,?, B# 900

InodeTable entry aka inode detail explained on the left

Inode#, FileType, Link Count, FileSize(B), Block Number

Assumes: a file uses only one block number

Inode# is 8B

File size is ? because we will not bother sizing directories in this example

InodeTable : InodeT

0

...

128

128,d, 2, ?,B# 900

...

249

250

251

...



Example 1 continued

```
[2] % cd /
    % mkdir aa
```

Action: In directory / create a (sub)directory aa

Block 900

.	128
..	128
aa	249

Block 901

.	249
..	128

Report of Activity

Directory aa is a file of type d; inode# is 249 assigned by the filesystem

Inside directory / into which aa was created a new entry (alias name, inode#) is inserted.

This is (aa,249)

Block 901 will contain directory entries for directory aa in filesystem path /aa

(., 249) is the current directory alias and corresponding inode number (a reference to aa)

(.., 128) indicates parent of 249 (i.e. aa) is 128 (i.e. /)

Note link count of 128 is now a 3 not 2

Note link count of 249 is now a 2.

An ls -li aa

An ls -li /

Or stat aa or sta / will confirm actions, if they were possible to track our example

Directory aa is empty; directory / is not empty any more

InodeTable : InodeT

0

...

128

128,d, 3, ?,B# 900

...

249

249,d,2, ?, B# 901

250

251

...

Example 1 continued



```
[3] % cd /aa
    % echo "Hello" > myf
```

Action: In directory aa file myf is created; contents are Hello\n where \n is the newline character

Block 900		Block 901		Block 902
.	128	.	249	Hello\n
..	128	..	128	
aa	249	myf	250	

InodeTable : InodeT

0	
...	
128	128,d, 3,?,B# 900
...	
249	249,d,2,?, B# 901
250	250,-,1,6, B#902
251	
...	

Report of Activity

Directory aa info is stored in Block 901.

File myf that is being created is assigned inode# 250

An entry in the block (Block# 901) of directory aa is added : (myf, 250) with the name of the alias (myf aka filename) and the associated inode#

File myf is a regular file (- in inodeT entry); in block with block# 901 for myf we can read the first 6 byte containing H, e, l, l, o, \n assuming an ASCII representation. Thus filesize is 6B

There is one entry in filesystem containing inode# 250 in a directory entry; link count of 250 is 1

Example 1 continued



```
[4] % cd /aa
      % ln myf one
```

Action: In directory aa create a (hard link) to myf named one; i.e. create one more alias to the existing myf alias

Block 900

Block 901

Block 902

.	128	.	249	Hello\n
..	128	..	128	
aa	249	myf	250	
		one	250	

InodeTable : InodeT

0	
...	
128	128,d, 3,?,B# 900
...	
249	249,d,2,?, B# 901
250	250,-,2,6, B#902
251	
...	

Report of Activity

The syntax of `% ln old new` is as follows:

a new alias with name new is created for the file (i.e. for the inode#) whose alias in the current directory is old (directory the command ln is issued)

Thus the new alias new is associated with the inode# of myf which is 250.

The effect of `ln myf one` is minimal: a new entry (one,250) is added into the directory block of aa (block# 901) and the LinkCount of inode# 250 is now a two now: there are two aliases to 250 named myf and one and existing in directory aa

Do in directory aa an

```
% ls -li
```

And observe the output

Example 1 continued



```
[5] % ln one two
      % cat two
```

Action: Still in directory aa create a (hard link) to one named two; i.e. create one more alias for ... 250

Block 900		Block 901		Block 902
.	128	.	249	Hello\n
..	128	..	128	
aa	249	myf	250	
		one	250	
		two	250	

InodeTable : InodeT

0	
...	
128	128,d, 3,?,B# 900
...	
249	249,d,2,?, B# 901
250	250,-,3,6, B#902
251	
...	

Report of Activity

One more alias is created; hard link count of 250 is now a 3

The command `cat two`

Displays the contents of the file whose alias is two this file is the one with inode# 250. The command plus output is as follows

```
% cat two
Hello
```

Example 1 continued



```
[6] % rm myf
    % cat two
```

Action: Still in directory aa remove alias (AND NOT THE FILE) myf, then do a cat say of alias two or even one

Block 900		Block 901		Block 902
.	128	.	249	Hello\n
..	128	..	128	
aa	249			
		one	250	
		two	250	

InodeTable : InodeT

0	
...	
128	128,d, 3,?,B# 900
...	
249	249,d,2,?, B# 901
250	250,-,2,6, B#902
251	
...	

Report of Activity

Myf is an alias to file 250; rm myf decreases the link count of 250 from 3 into a 2
 The entry for 250 is going to get cleared when the link count becomes a 0
 It is not a zero yet
 The command cat two works fine; it associates name/alias two with inode# 250 and in the inodeT with index 250 the filesystem finds the block with block#902 containing the data of file 250 (aka one aka two ex aka myf!)
 % cat two
 Hello

Example 1 continued



```
[7] % rm one
      % cat two
```

Action: Still in directory aa remove the alias (AND NOT THE FILE) myf, then do a cat say of alias two

Block 900		Block 901		Block 902
.	128	.	249	Hello\n
..	128	..	128	
aa	249			
		two	250	

InodeTable : InodeT

0	
...	
128	128,d, 3,?,B# 900
...	
249	249,d,2,?, B# 901
250	250,-,1,6, B#902
251	
...	

Report of Activity

Myf is an alias to file 250; rm one decreases the link count of 250 from 2 into a 1
 The entry for 250 is going to get cleared when the link count becomes a 0
 It is not a zero yet
 The command cat two works fine; it associates name/alias two with inode# 250 and in the inodeT with index 250 the filesystem finds the block with block#902 containing the data of file 250 (aka one aka two ex aka myf!)

```
% cat two
Hello
```

Example 1 continued



[8] % ln -s two sft

Action: Still in aa create a symbolic link sft (soft link) to the file (with alias) two

Block 900		Block 901		Block 902	Block 903
.	128	.	249	Hello\n	two
..	128	..	128		
aa	249	sft	251		
		two	250		

InodeTable : InodeT

0	
...	
128	128,d, 3,?,B# 900
...	
249	249,d,2,?, B# 901
250	250,-,1,6, B#902
251	251, l ,1,3, B#903
...	

Report of Activity

A symbolic link uses the same command ln as the one used to create a (hard) link but it requires the additional option -s for symbolic(aka soft)

The first alias is the alias two of the file/alias to be linked with the new file to be created whose name would be sft. Therefore

1. The new symbolic link acquires an inode# 251,
2. an entry (sft, 251) is added) in directory aa into which the ln -s two sft was issued,
3. a block with block number 903 is assigned to 251, the file type is an l for symbolic link on the left of l and the right of 251 in the inode area (inode table entry) for inode#251.
4. What are the contents of the block of a symbolic link? The name of the file it is linked to!

Example 1 continued



```
[9] % cat two
      % cat sft
```

Action: Still in aa create a symbolic link sft (soft link) to the file (with alias) two

Block 900		Block 901		Block 902	Block 903
.	128	.	249	Hello\n	two
..	128	..	128		
aa	249	sft	251		
		two	250		

Report of Activity

```
% cat two
Hello
% cat sft
Hello
```

Where as the effect of the two cat commands is identical the actions required by the cat command are different
 In the former case two is resolved to inode# 250 and then access to block #902 generates the output
 In the latter case sft is resolved to inode# 251; because filetype is l, access to block #903 provides the name two and from that point on the action is just like the former case i.e. the resolution of the name/ alias two

InodeTable : InodeT

0	
...	
128	128,d, 3,?,B# 900
...	
249	249,d,2,?, B# 901
250	250,-,1,6, B#902
251	251, l ,1,3, B#903
...	

Example 1 ends



```
[10]% rm two
      % cat sft
```

Action: still in aa

Block 900		Block 901		Block 902	Block 903
.	128	.	249	Hello\n	two
..	128	..	128		
aa	249	sft	251		

Report of Activity

The command `rm two` first it eliminates the entry (two,250) from the block of directory aa.

Then it decrements the link count for inode# 250; after the decrement operation the count reaches 0.

At that point file with inode# 250 ceases to exist because its InodeTable entry has been cleared.

Many naïve users think that this deletes file two this is far from the truth; the information in block # 902 is still out there and reachable if you know the block number previously associated with the inode# 250 and the now defunct aliases myf, one, and two.

The second activity is fun! There is a dangling pointer over there. The alias sft resolves to inode# 251, which is a symbolic link so going to block#903 we find a name two. There is nore reference (i.e. no entry) to two in the block (block# 901) of directory aa. The system (filesystem/operating system) cannot resolve the name two A no such file or directory will then be output because cat sft cannot be resolved!

InodeTable : InodeT

0	
...	
128	128,d, 3,?,B# 900
...	
249	249,d,2,?, B# 901
250	--,-,0,6, B# 902
251	251, 1 ,1,3, B# 903
...	

Example 2.

```
% cd csabc
% touch file          # touch creates an empty file that is assigned an alias (filename)
% ls
csabc
% ls -li file
315239608 -rw----- 1 alexg users 0 Dec  4 12:52 file
% stat file
  File: file
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 28h/40d Inode: 315239608   Links: 1
Access: (0600/-rw-----)  Uid: (32108/   alexg)   Gid: ( 100/   users)
Access: 2022-12-04 12:52:25.000000000 -0500
Modify: 2022-12-04 12:52:25.000000001 -0500
Change: 2022-12-04 12:52:25.000000000 -0500
Birth: -
```

csabc

.	12800
..	2565
file	315239608

Operation of ... touch file

1. Filesystem finds an available index to the InodeTable i.e. inode# (315239608) and as
2. inode# entry in InodeTable is populated with information
3. An alias **file** is created and associated with inode#
4. Pair of (inode# 315239608 and alias file) is stored in directory into which file was created. (Do a pwd to find it; in our case it is csabc)

Operation of ... stat file

1. Finds in directory csabc the pair (315239608,file) matching the argument (file) of stat
2. Accesses InodeTable using the inode# (315239608) of the pair
3. Prints information available in the InodeTable[315239608]



Two files are different: inode# are different

Two files can have same 'filename' and they are different

Two files can have same 'filename' and are the same

Two files have different 'filename' and are the same

```
% touch file1                                # create file1
% vi file1
% ls -li file1                                # inode# shown; hardlink count is 1
315258962 -rw----- 1 alexg users 29 Dec  4 13:37 file1
% cat file1                                  # see contents of file1 (data)
This is file1. Or is it not?
% ln file1 file2                             # Link file2 to file1 (create new alias)
% ls -li file1 file2                         # Alias is associated with inode#)
315258962 -rw----- 2 alexg users 29 Dec  4 13:37 file1          #hard link count of 315258962
315258962 -rw----- 2 alexg users 29 Dec  4 13:37 file2          # is 2
% ln file1 file3                             # file3 vs file1
% ln file2 file4                             # file4 vs file2
% ls -li file1 file2 file3 file4
315258962 -rw----- 4 alexg users 29 Dec  4 13:37 file1
315258962 -rw----- 4 alexg users 29 Dec  4 13:37 file2
315258962 -rw----- 4 alexg users 29 Dec  4 13:37 file3
315258962 -rw----- 4 alexg users 29 Dec  4 13:37 file4
```

Example 3: More on links



```
% ls -li file1 file2 file3 file4
315258962 -rw----- 4 alexg users 29 Dec  4 13:37 file1
315258962 -rw----- 4 alexg users 29 Dec  4 13:37 file2
315258962 -rw----- 4 alexg users 29 Dec  4 13:37 file3
315258962 -rw----- 4 alexg users 29 Dec  4 13:37 file4
% rm file1                                     # Action: Erase pair (315258962,file1);
                                              # and decrement hardlink count by 1;
                                              # check if hardlink count is 0; if not 0 DONE

% ls -li
total 3
315258962 -rw----- 3 alexg users 29 Dec  4 13:37 file2
315258962 -rw----- 3 alexg users 29 Dec  4 13:37 file3
315258962 -rw----- 3 alexg users 29 Dec  4 13:37 file4
% cat file2                                   # data of 315258962 are still there alias file1 gone
This is file1. Or is it not?
% cat file1
cat: file1: No such file or directory
% rm file2; rm file3                         #hardlink count is now 1
% rm file4                                   #hardlink count is now 0; Release 315258962 to the
                                              # free inode# list; and if OS is security conscious
                                              # cleanup the InodeTable[315258962] entries
                                              # Datablocks are not ERASED' somewhere on the disk
                                              # you can still read "This is file1. Or is it not?"
```

% link oldfile newfile

**#Create a new alias named newfile for the existing file that
 # has alias oldfile**

% rm newfile

**# removes an alias and decrements hardlinkcount of inode#
 # associated with newfile; if hardlinkcount become 0 release inode#**

Example 4: More on links



```
% rm -rf *                                # start from scratch
% touch file1
% vi file1
% cat file1
ls this file1?
% ls -li file1
315258962 -rw----- 1 alexg users 15 Dec  4 13:59 file1
% ln file1 file2
% ls -li file1 file2
315258962 -rw----- 2 alexg users 15 Dec  4 13:59 file1
315258962 -rw----- 2 alexg users 15 Dec  4 13:59 file2
% touch fileOther
% ls -li
total 2
315258962 -rw----- 2 alexg users 15 Dec  4 13:59 file1
315258962 -rw----- 2 alexg users 15 Dec  4 13:59 file2
315239608 -rw----- 1 alexg users  0 Dec  4 13:59 fileOther
% rm file1 file2 fileOther
% touch file1 fileOther
% ls -li
% rm file1 file2 fileOther
rm: remove regular file file1? y
rm: remove regular file file2? y
rm: remove regular empty file fileOther? y
% touch file1 fileOther
% ls -li
total 0
315239608 -rw----- 1 alexg users  0 Dec  4 14:02 file1
315258958 -rw----- 1 alexg users  0 Dec  4 14:02 fileOther
```

#Using inode#

#Using inode#

#Reusing inode# ?



A symbolic link is also know as a soft link

In order to create a softlink do as follows; oldfile must exist

% ln -s oldfile newfile

```
% rm -rf *
```

```
% ls
```

```
% touch file1
```

```
% vi file1
```

```
% ls -li file1
```

```
315258962 -rw----- 1 alexg users 15 Dec 4 14:07 file1
```

% ln -s file1 file2

file1 and file2 have different inode# now

```
% ls -li file1 file2
```

```
315258962 -rw----- 1 alexg users 15 Dec 4 14:07 file1
```

```
315239608 lrwxr-xr-x 1 alexg users 5 Dec 4 14:07 file2 -> file1
```

#pointer!

```
% cat file1
```

```
Is this file1?
```

```
% cat file2
```

```
Is this file1?
```

```
% rm file1
```

```
rm: remove regular file file1? y
```

```
% cat file2
```

```
cat: file2: No such file or directory
```

```
% ls -l
```

```
total 1
```

```
lrwxr-xr-x 1 alexg users 5 Dec 4 14:07 file2 -> file1
```

Practice 2



2.1. In file allocation, what's the difference between compaction and consolidation?

2.2. Where do you store the name of a file? (a) in its inode, (b) in the directory that contains (i.e. in a datablock of the directory that contains it).

2.3. In a filesystem an inodetable is of length 10^{**7} . How many (regular) files can it support? How many directories can it support? How many (regular) files or directories can it support?

2.4 File x is a symbolic link to y. Do x and y have the same inode (number)?

2.5 File z is a (hard) link to w. Do z and w have the same inode (number)?

NJIT

A white curved line that starts under the 'N', arches over the 'J' and 'I', and ends under the 'T'.

New Jersey's Science &
Technology University

THE EDGE IN KNOWLEDGE

Alex. Gerbessiotis

CS Department