Command Line Interpretation of Shell

- Command execution is carried out in six sequential steps.

1. Command Parsing
2. Variable Evaluation
3. Command Substitution
4. Redirection
5. Wildcard Expansion
6. Path Determination

STEPS	DESCRIPTION
Command Parsing	The shell first parses the command into words. In this step, it uses whitespaces as delimiters between the words. It also replaces sequences of two or more spaces or table with a single space.
Variable Evaluation	After completely parsing the command, the shell looks for variable names. When a variable name is found, its value replaces the variable name.
Command Substitution	The shell then looks for a command substitution. If found, the command is executed and its output string replaces the command, the dollar sign and the parentheses.
Redirection	At this point, the shell checks the command for redirected files. Each redirected files is verified by opening it.
Wildcard Expansion	When filenames contain wildcards, the shell expands and replaces then with their matching filename. The step create file list.
Path Determination	In this last step, the shell uses the PATH variable to locate the directory containing the command code. The command is now ready for execution.

Command Interpretation of Shell

- Example: cat $var report* 1> file3 2> file2

  Here we use space and tab in command.

- In the first step, the shell parses words. It replaces all tabs and multiple space with single space. The result is shown in below

  $ cat $var report* 1>file3 2>file2

- The second step replaces the variable ($var) with its value, For example, file*. The result is

  $ cat file* report* 1>file3 2>file2

- The third step is skipped because there is no command substitution.

Command Interpretation of Shell

- In the forth step, the redirected files are handled: file3 is opened for output and file2 is opened for error.
- In the fifth step, the shell expand the wildcards. The command is now

```
$cat fileA fileB fileC report1 report2 1>file3 2>file2
```

- In the last step, the shell find the /bin directory in PATH variable. It completes the command as in the next example.

```
$/bin/cat  fileA fileB fileC report1 report2 1>file3 2>file2
```

Character Device File

- A character device file represents a physical device that transmit data character by character.
- Example: keyboard, line printer and modem
- Command : “mknod” command is used to create a block device file.

```
 $ mknod /dev/cfile   c  115  5
```

- In above command, “cfile” is the block device file being created under /dev directory with major number 115 and minor number 5.
- Argument c specify that the file to be created is a character device file.

Device File

- In UNIX each and every hardware device treated as a file.
- A device file allows to accesses hardware devices so that end users do not need to get technical details about hardware.
- In short, a device file (also called as a special file) is an interface for a device driver that appears in a file system as if it were an ordinary file.
- This allows software to interact with the device driver using standard input/output system calls, which simplifies many tasks.
- It is store in "/dev" directory.
- There are two types of device files :
  - Character special files or Character devices
  - Block special files or Block devices

Character Device File

- A character device file represents a physical device that transmit data character by character (1 Byte at a time).
- Example: Terminal, Modems, keyboard, mouse
- It is marked with “c” as the first letter of the permission string.

Block Device File

- A block device file represents a physical device that transmit data block at a time.
- Example: Hard disk, floppy disk drive .
- It is marked with “B” as the first letter of the permission string.
- To find out device file , type following command.

```
$ ls –l /dev
```

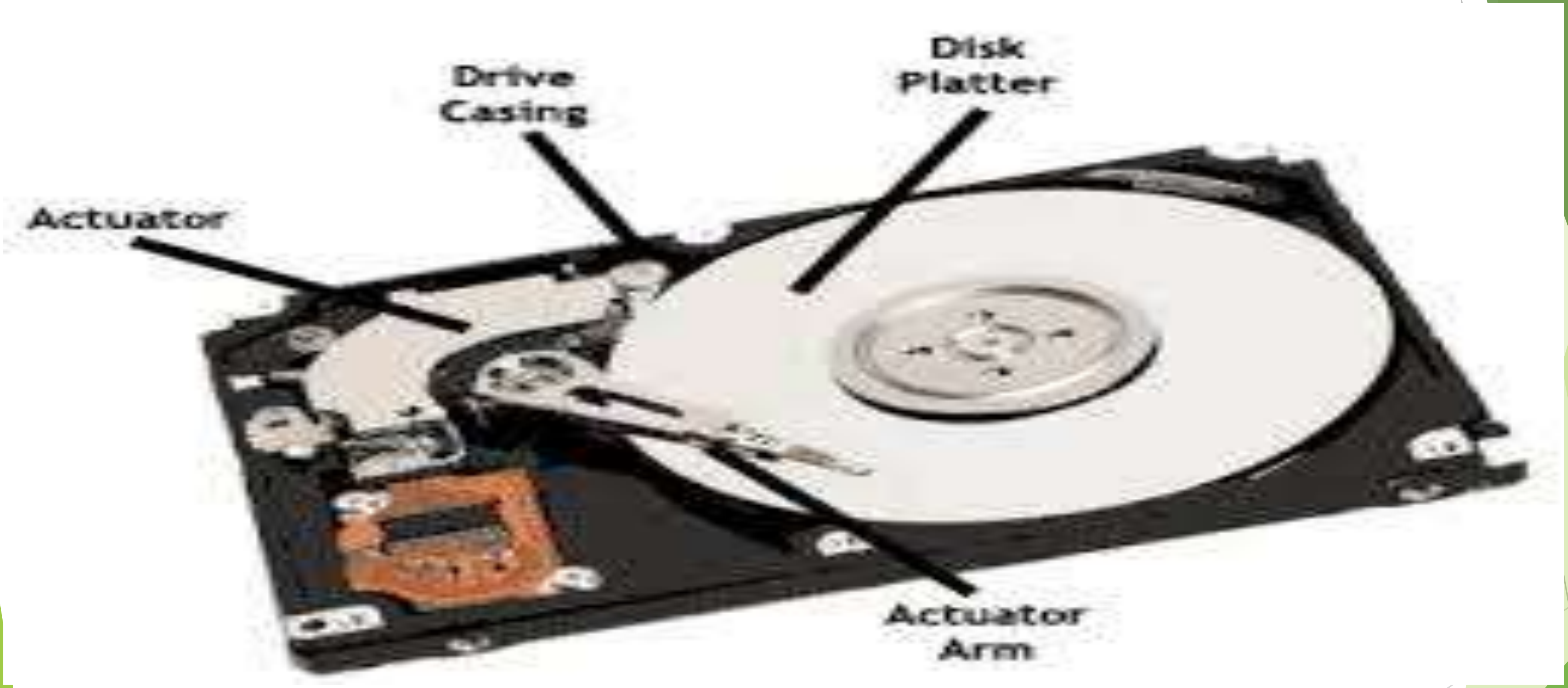
File System

- A file system is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk.
- In a computer, a file system is the way in which files are named and where they are placed logically for storage and retrieval.
- For Example, in DOS, Windows, Mac, Unix all have file system in which files are placed somewhere in hierarchical structure.
- It specify convention for naming files, including maximum number of character used . It also include a format for specifying the path to a file through the directories.
- Whole HD may comprise a single file system or it may be partition to house of several file system.

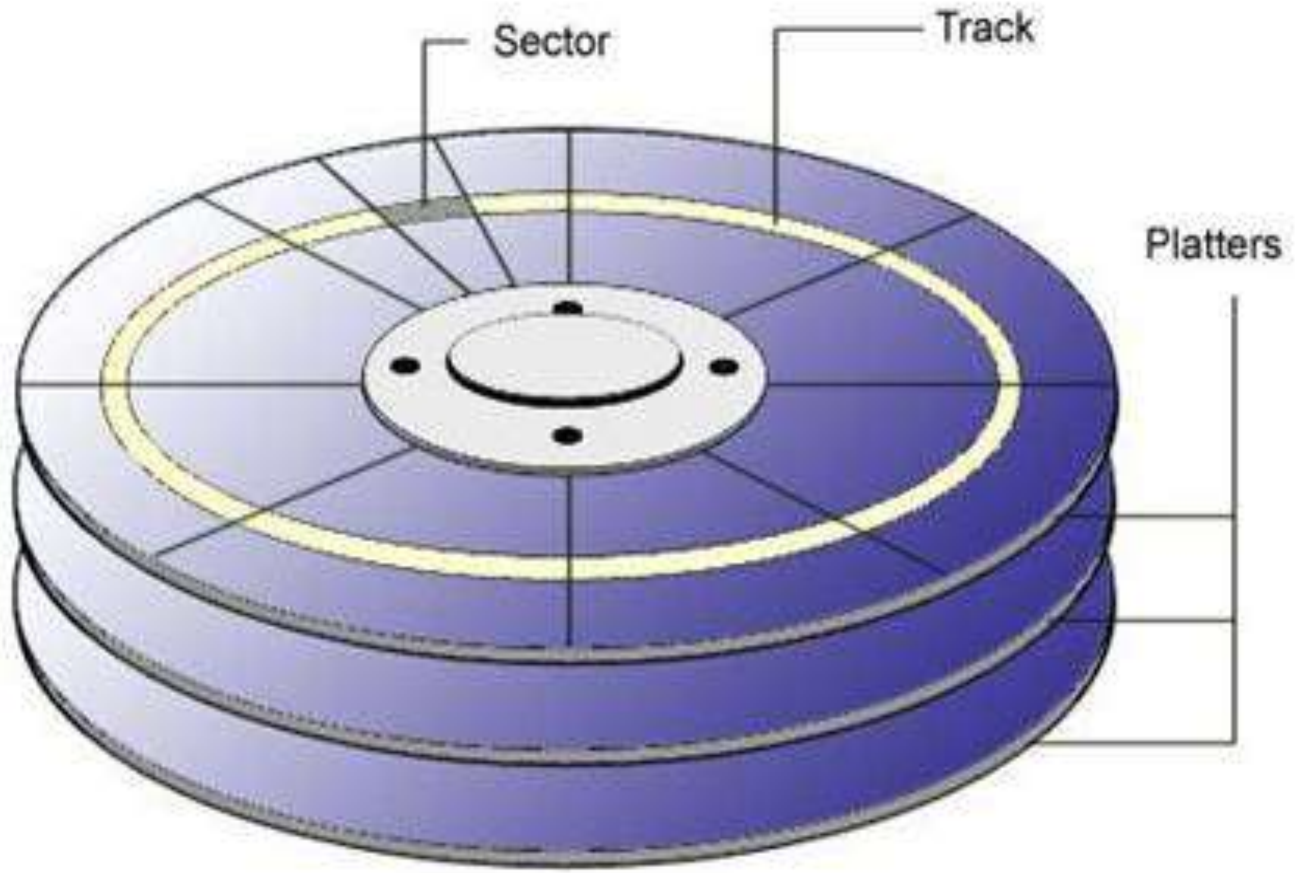
File System

- No file system can be split over two different disk.
- The disk space allocated to unix file system is made up of “blocks” which contain 512 bytes data.
- Block size depend upon the how file system has been implemented .
- File system divided into two categories:
  - User data - stores actual data contained in files
  - Metadata - stores file system structural information such as date created, date modified, file size etc.

File System



File System



Internal Structure of File System

- All the block belonging to the file system are logically divided into four parts.
  - Boot block
  - Super block
  - Inode table
  - Data block

BOOT BLOCK	SUPER BLOCK	INODE TABLE	DATA BLOCK

Internal Structure of File System

Boot Block

- The first block numbered 0 is the **boot block**, which is normally unused by the file system. The boot block contains the code to bootstrap the OS.

Super Block (What is the purpose of super block?)

- The second block, numbered 1 is called the **super block** and is used to control allocation of disk blocks.
- It contains following data…

Internal Structure of File System

It contains the following information:

- size of the file system

- number of free blocks in the file system + the list of free blocks

- index of the next free block

- size of the inode list

- number of free inodes

- list of free inodes

- flag indicating that the super block has been modified

- lock fields for the free block and free inode lists

Internal Structure of File System

Inode Table

- The third segment includes block 2 onwards, up to a number determined during the creation of the file system.
- Every file in the system will have an entry in this area identified by a 64-bit structure called the i-node.
- The complete list of i-nodes is known as the **i-list or inode table**.
- Every i-node is identified by its position in the list called the **i-number**.
- The information regarding each files are store in inode table.
- For each file there is a inode entry.
- Use **-i** option with **ls -l** to show i-number of a file.

Internal Structure of File System

Inode Table

- When users search for or access a file, the UNIX system searches through the inode table for the correct inode number.
- When the inode number is found, the command in question can access the inode and make the appropriate changes if applicable.
- Take, for example, editing a file with vi. When you type vi <filename>, the inode number is found in the inode table, allowing you to open the inode. Some attributes are changed during the edit session of vi, and when you have finished and typed :wq, the inode is closed and released.

Internal Structure of File System

Data Block

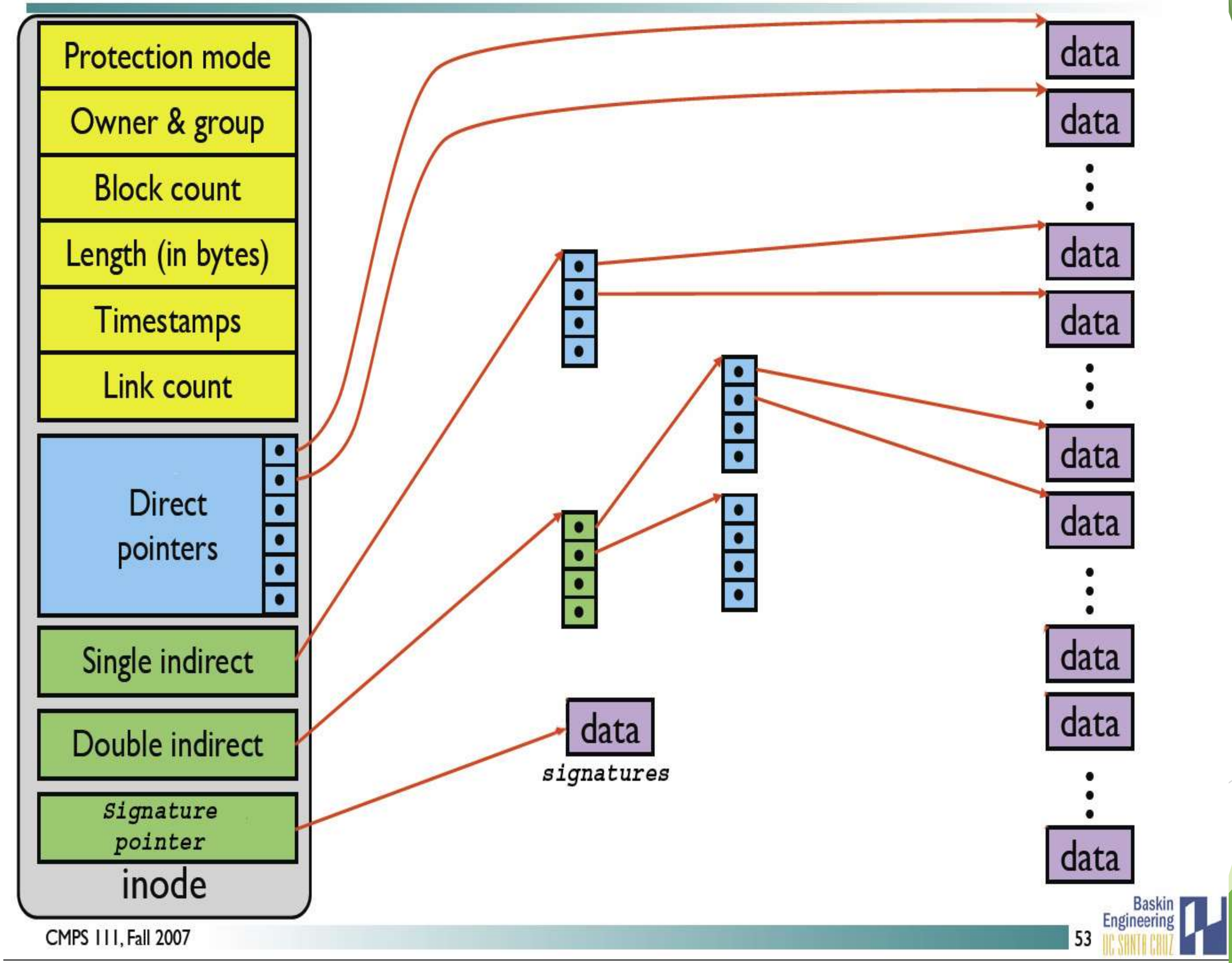
- The fourth and final segment contains a long chain of blocks for storing the contents of files (physical blocks of 512 bytes).
- These data blocks begin from the point the i-node blocks terminate.

INODE

- Definition: INODE is Data structure that keeps track of all the information about a file.
- We store information in a file, and the operating system stores the information about a file in an inode(sometimes called as an inode number).
- Information about files are sometimes called metadata. So you can even say it in another way, "An inode is metadata of the data."
- Whenever a user or a program needs access to a file, the operating system first searches for the exact and unique inode (inode number), in a table called as an inode table.
- To reach a particular file with its "name" needs an inode number corresponding to that file.
- Internally file is identified by the unix by a unique inode-number associated with it.
- Inode number is nothing but an index into the inode table.

INODE

- Directory is also represented as a file in unix.
- Unix directory entry contain one entry for each file in that directory.
- Here we will be discussing the contents of an inode.


Protection mode
Owner & group
Block count
Length (in bytes)
Timestamps
Link count
Direct pointers
Single indirect
Double indirect
Signature pointer
inode
data
data
data
data
data
data
data
data
data
data
signatures
CMPS 111, Fall 2007
53
Baskin Engineering
UC SANTA CRUZ

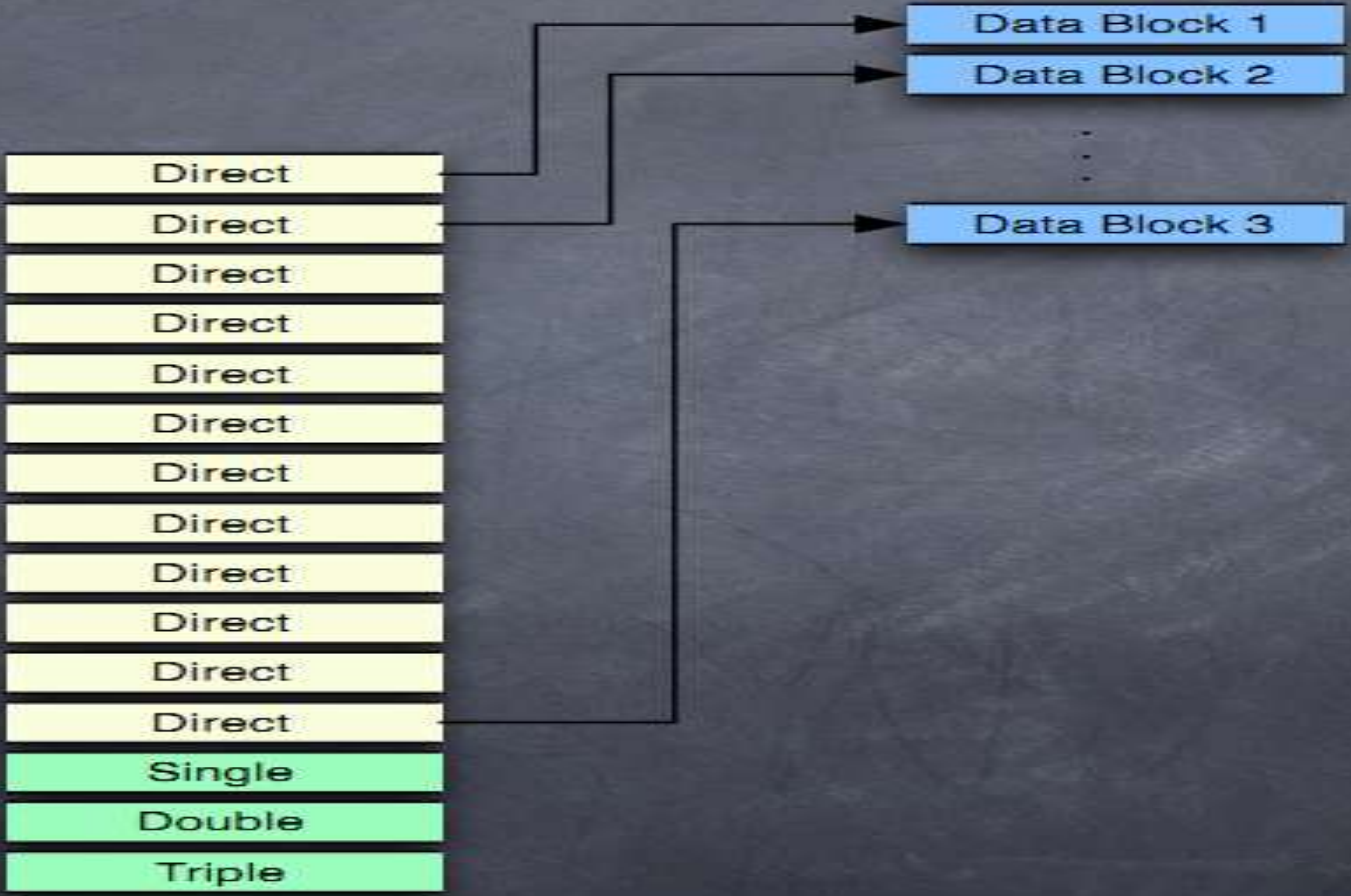
INODE

- **File mode:** This keeps information about two things, one is the permission information, the other is the type of inode, for example an inode can be of a file, directory or a block device etc.
- **Owner:** The user-id number of owner
- **Group:** The group-id number
- **Size** : Total size in bytes of the data contained in file
- **Time Stamp:** Time when contents of file were created, last modification time ; displayed by **ls -l**
- **Number of links:** Total number of hard links to file

INODE

- **Block size:** Whenever a partition is formatted with a file system. It normally gets formatted with a default block size. Now block size is the size of chunks in which data will be spread. So if the block size is 4K, then for a file of 15K it will take 4 blocks(because 4K*4=16K)

- Direct Block Pointer: In an ext2 file system an inode consists of only 15 block pointers. The first 12 block pointers are called as Direct Block pointers. Which means that these pointers point to the address of the blocks containing the data of the file. 12 Block pointers can point to 12 data blocks. So in total the Direct Block pointers can address only 48K(12 * 4K) of data. Which means if the file is only of 48K or below in size, then inode itself can address all the blocks containing the data of the file.

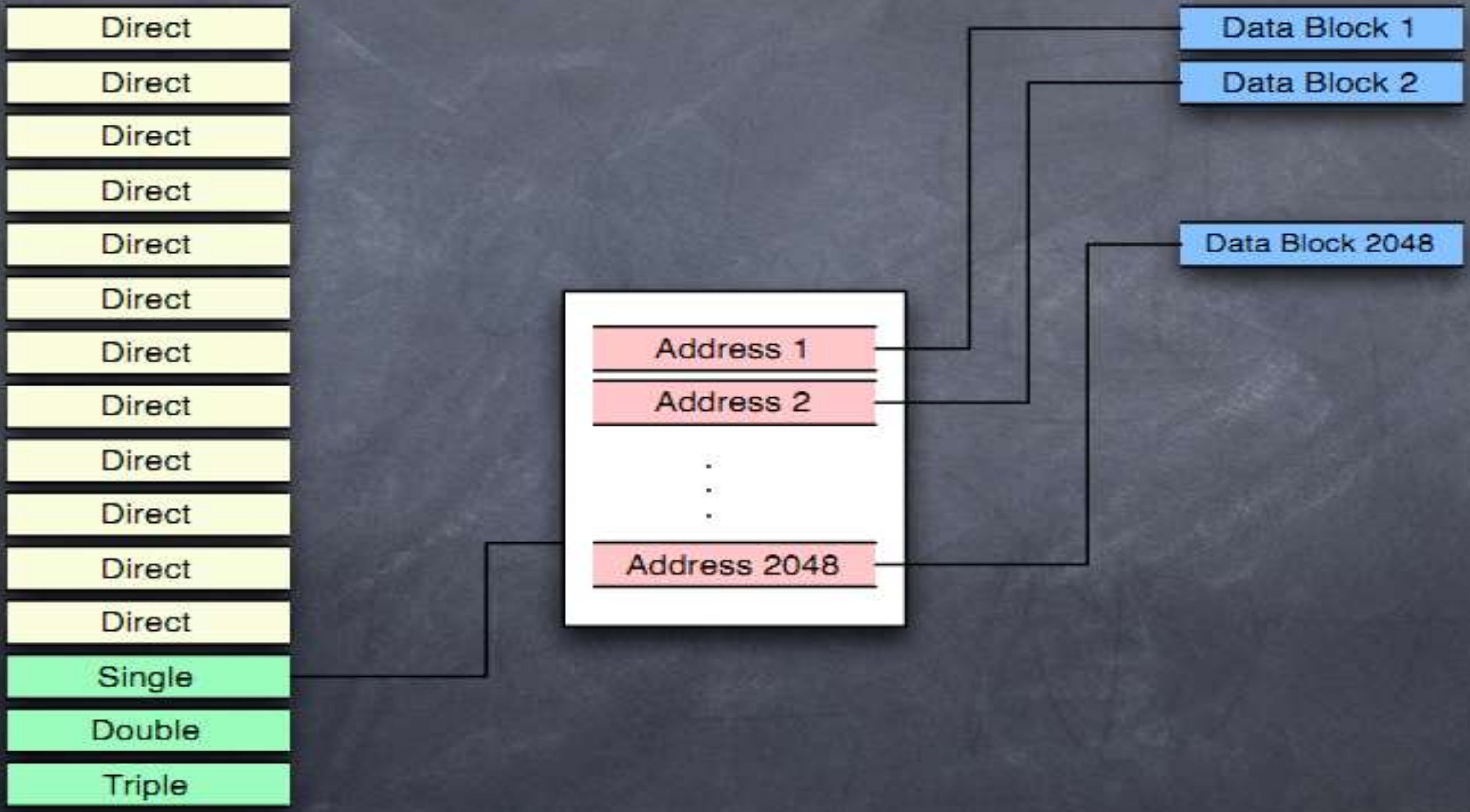

Data Block 1
Data Block 2
Data Block 3
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Single
Double
Triple

INODE

- InDirect Block Pointer: Whenever the size of the data goes above 48k(by considering the block size as 4k), the 13th pointer in the inode will points to a block of pointers that then point to blocks of the file's data

- Double indirect Block Pointer: Now if the size of the file is above 4MB + 48K then the inode will start using Double Indirect Block Pointers, to address data blocks. It is a pointer that points to a block of pointers that point to other blocks of pointers that then point to blocks of the file's data.

- Triple Indirect Block Pointers: Now this triple Indirect Block Pointers can address upto 4G * 1024 = 4TB, of file size. The fifteenth block pointer in the inode will point to a block of pointers that point to other blocks of pointers that point to other blocks of pointers that then point to blocks of the file's data



Direct

Direct

Direct

Direct

Direct

Direct

Direct

Direct

Direct

Direct

Direct

Direct

Single

Double

Triple

Address 1

Address 2

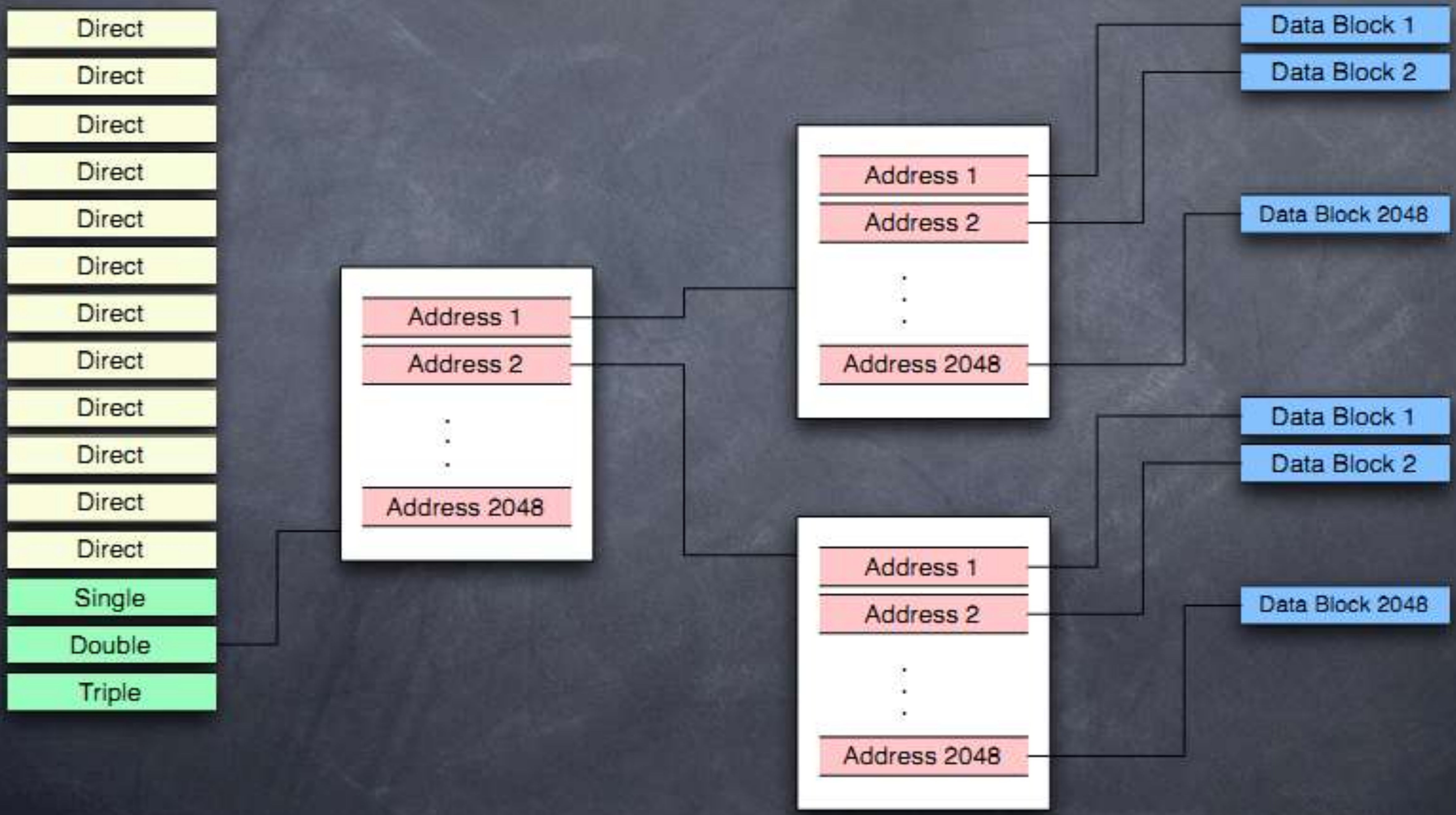
⋮

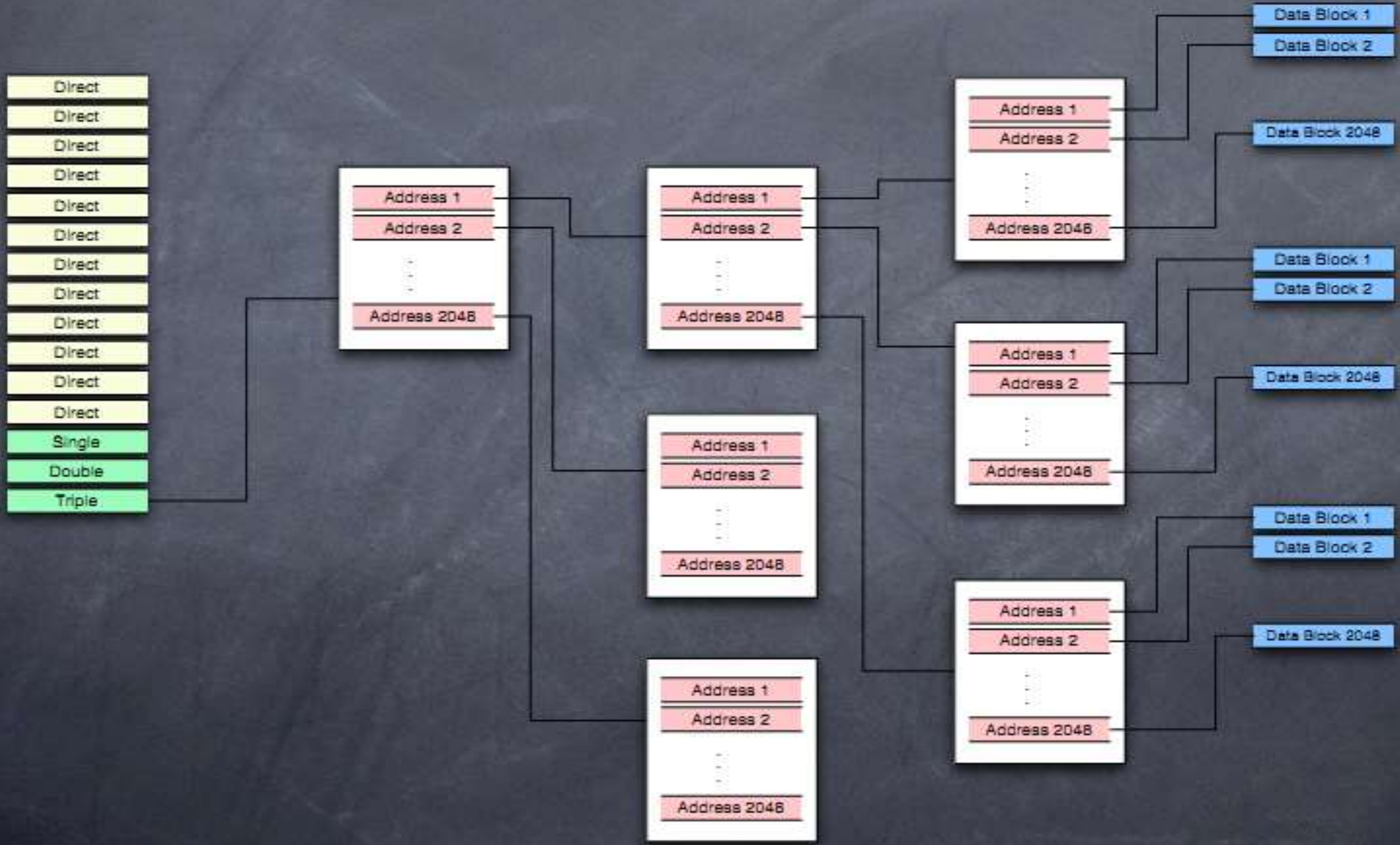
Address 2048

Data Block 1

Data Block 2

Data Block 2048


Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Single
Double
Triple
Address 1
Address 2
Address 2048
Address 1
Address 2
Address 2048
Address 1
Address 2
Address 2048
Data Block 1
Data Block 2
Data Block 2048
Data Block 1
Data Block 2
Data Block 2048



Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Direct
Single
Double
Triple

Address 1
Address 2
Address 2048

Address 1
Address 2
Address 2048

Address 1
Address 2
Address 2048

Address 1
Address 2
Address 2048

Address 1
Address 2
Address 2048

Address 1
Address 2
Address 2048

Address 1
Address 2
Address 2048

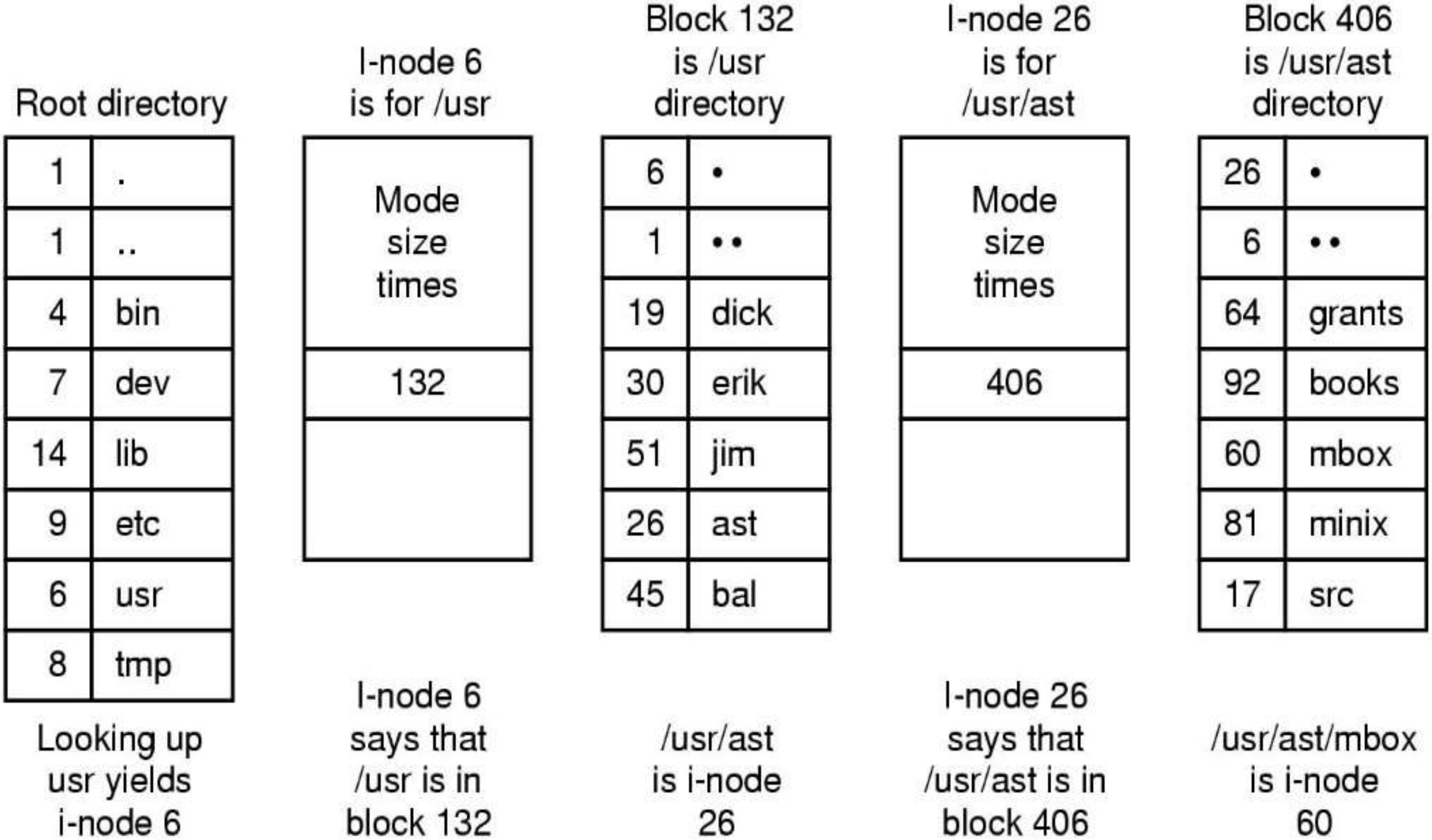
Data Block 1
Data Block 2
Data Block 2048

Data Block 1
Data Block 2
Data Block 2048

Data Block 1
Data Block 2
Data Block 2048

INODE

- So after the 12 direct block pointers, 13th block pointer in inode is for Indirect block pointers, and 14th block pointer is for double indirect block pointers, and 15th block pointer is for triple indirect block pointers.



Root directory

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

Looking up usr yields i-node 6

I-node 6 is for /usr

Mode size times
132

I-node 6 says that /usr is in block 132

Block 132 is /usr directory

6	•
1	••
19	dick
30	erik
51	jim
26	ast
45	bal

/usr/ast is i-node 26

I-node 26 is for /usr/ast

Mode size times
406

I-node 26 says that /usr/ast is in block 406

Block 406 is /usr/ast directory

26	•
6	••
64	grants
92	books
60	mbox
81	minix
17	src

/usr/ast/mbox is i-node 60

How to access inode number?

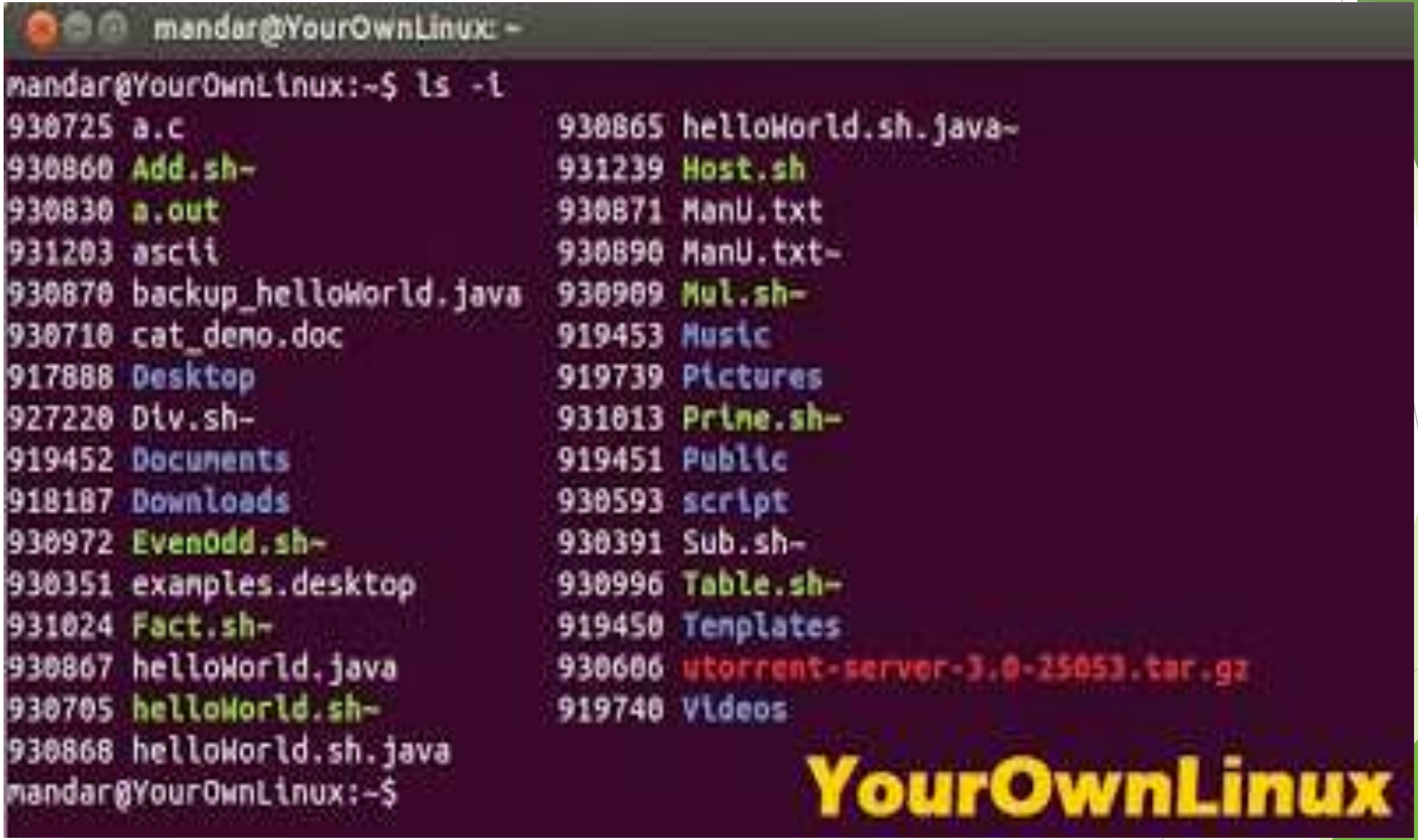
- Following command used to access inode number
  - ls –l
  - Stat
  - Du

ls -i

- ls is used for listing all the files within a directory.
- ls command when used with option -i, will display the list of files along with their inode numbers.

```
$ ls -i
```

ls -i

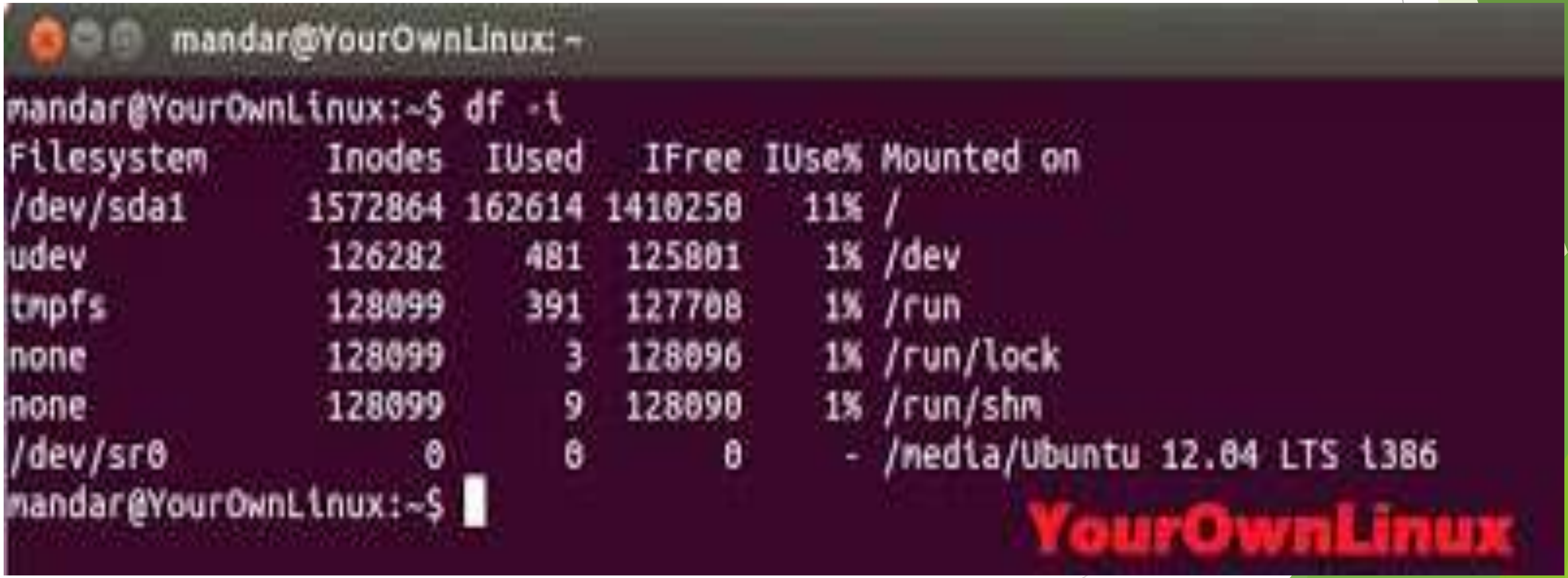


```
mandar@YourOwnLinux:~$ ls -i
930725 a.c                      930865 helloWorld.sh.java~
930860 Add.sh~                  931239 Host.sh
930830 a.out                    930871 ManU.txt
931203 ascii                    930890 ManU.txt~
930870 backup_helloWorld.java   930909 Mul.sh~
930710 cat_demo.doc             919453 Music
917888 Desktop                  919739 Pictures
927220 Div.sh~                  931013 Prime.sh~
919452 Documents                919451 Public
918187 Downloads                930593 script
930972 EvenOdd.sh~              930391 Sub.sh~
930351 examples.desktop         930996 Table.sh~
931024 Fact.sh~                 919450 Templates
930867 helloWorld.java          930606 utorrent-server-3.0-25053.tar.gz
930705 helloWorld.sh~           919740 Videos
930868 helloWorld.sh.java
mandar@YourOwnLinux:~$
```

df -i

- The df command when used with option -i, will display Inode information like the number of inodes used and free on the file system.

$ df –I



```
mandar@YourOwnLinux:~$ df -i
Filesystem      Inodes  IUsed   IFree IUse% Mounted on
/dev/sda1      1572864 162614 1410250   11% /
udev            126282    481  125801    1% /dev
tmpfs           128099    391  127708    1% /run
none            128099      3  128096    1% /run/lock
none            128099      9  128090    1% /run/shm
/dev/sr0             0      0       0     - /media/Ubuntu 12.04 LTS i386
mandar@YourOwnLinux:~$ 
```

stat

- Stat command is very useful in displaying the file statistics. This command also shows inode number of a file.

stat [file-name]

```
$ stat 101hacks.txt
  File: `/home/sathiyamoorthy/101hacks.txt'
  Size: 854         Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 1058122      Links: 1
Access: (0600/-rw-------)  Uid: ( 1000/ sathiya)   Gid: ( 1000/ sathiya)
Access: 2009-06-28 19:29:57.000000000 +0530
Modify: 2009-06-28 19:29:57.000000000 +0530
Change: 2009-06-28 19:29:57.000000000 +0530
```

Details of stat command output

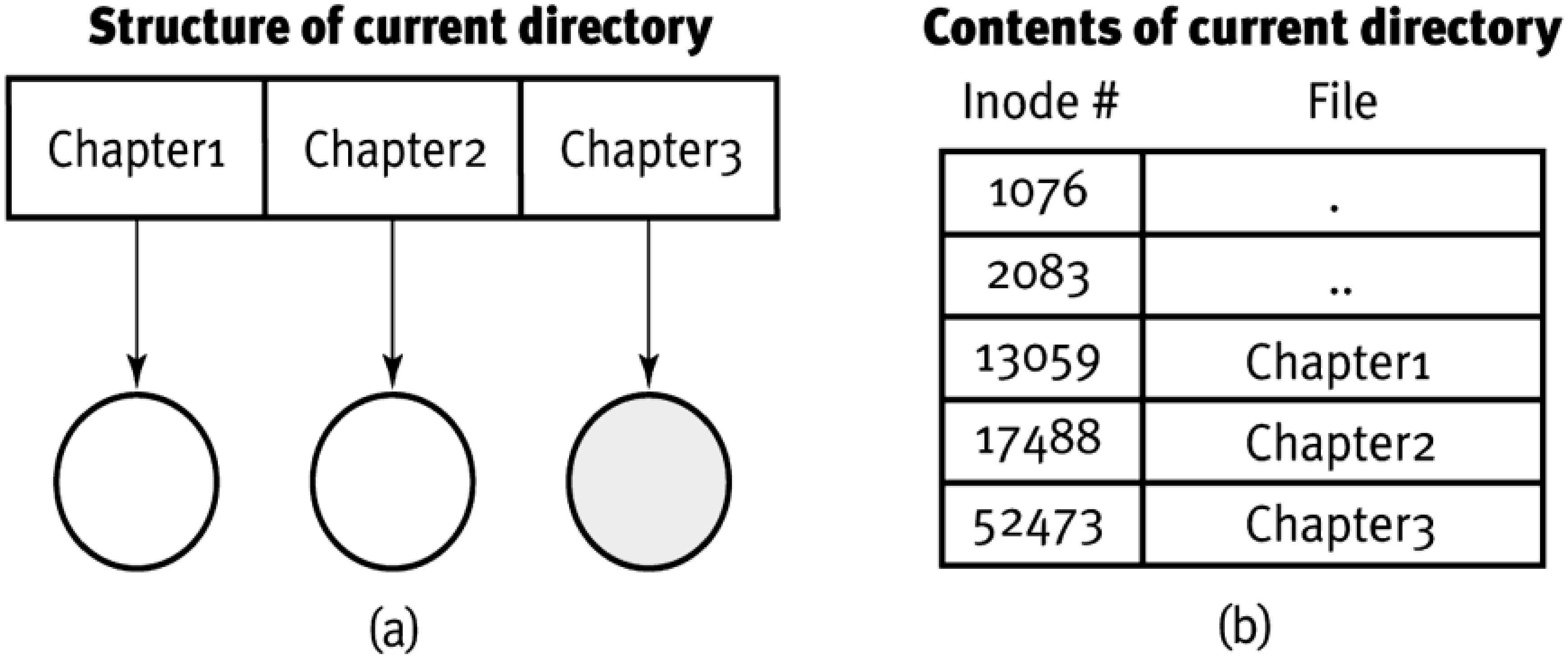
- **File:** `/home/sathiyamoorthy/101hacks.txt’ – Absolute path name of the file.
- **Size:** 854 – File size in bytes.
- **Blocks:** 8 – Total number of blocks used by this file.
- **IO Block:** 4096 – IO block size for this file.
- **Regular file** – Indicates the file type. This indicates that this is a regular file. Following are available file types.
  - regular file. (ex: all normal files).
  - directory. (ex: directories).
  - socket. (ex: sockets).
  - symbolic link. (ex: symbolic links.)
  - block special file (ex: hard disk).

Details of stat command output

- **Device:** 801h/2049d – Device number in hex and device number in decimal
- **Inode:** 1058122 – Inode number is a unique number for each file which is used for the internal maintenance by the file system.
- **Links:** 1 – Number of links to the file
- **Access:** (0600/-rw——-): Access specifier displayed in both octal and character format. Let us see explanation about both the format.
- **Uid:** (1000/ sahil) – File owner's user id and user name are displayed.
- **Gid:** (1000/ sahil) – File owner's group id and group name are displayed.
- **Access:** 2009-06-28 19:29:57.000000000 +0530 – Last access time of the file.
- **Modify:** 2009-06-28 19:29:57.000000000 +0530 – Last modification time of the file.
- **Change:** 2009-06-28 19:29:57.000000000 +0530 – Last change time of the inode data of that file.

Links

- A link in UNIX is a pointer to a file.
- Like pointers in any programming languages, links in UNIX are pointers pointing to a file or a directory.
- Creating a link is a kind of shortcut to access a file.
- 2 types of links exist:
  - Hard links
  - Soft (symbolic) links



Contents of current directory

Inode #	File
1076	.
2083	..
13059	Chapter1
17488	Chapter2
52473	Chapter3

(b)

Figure (a) Logical structure of current directory;

(b) contents of current directory

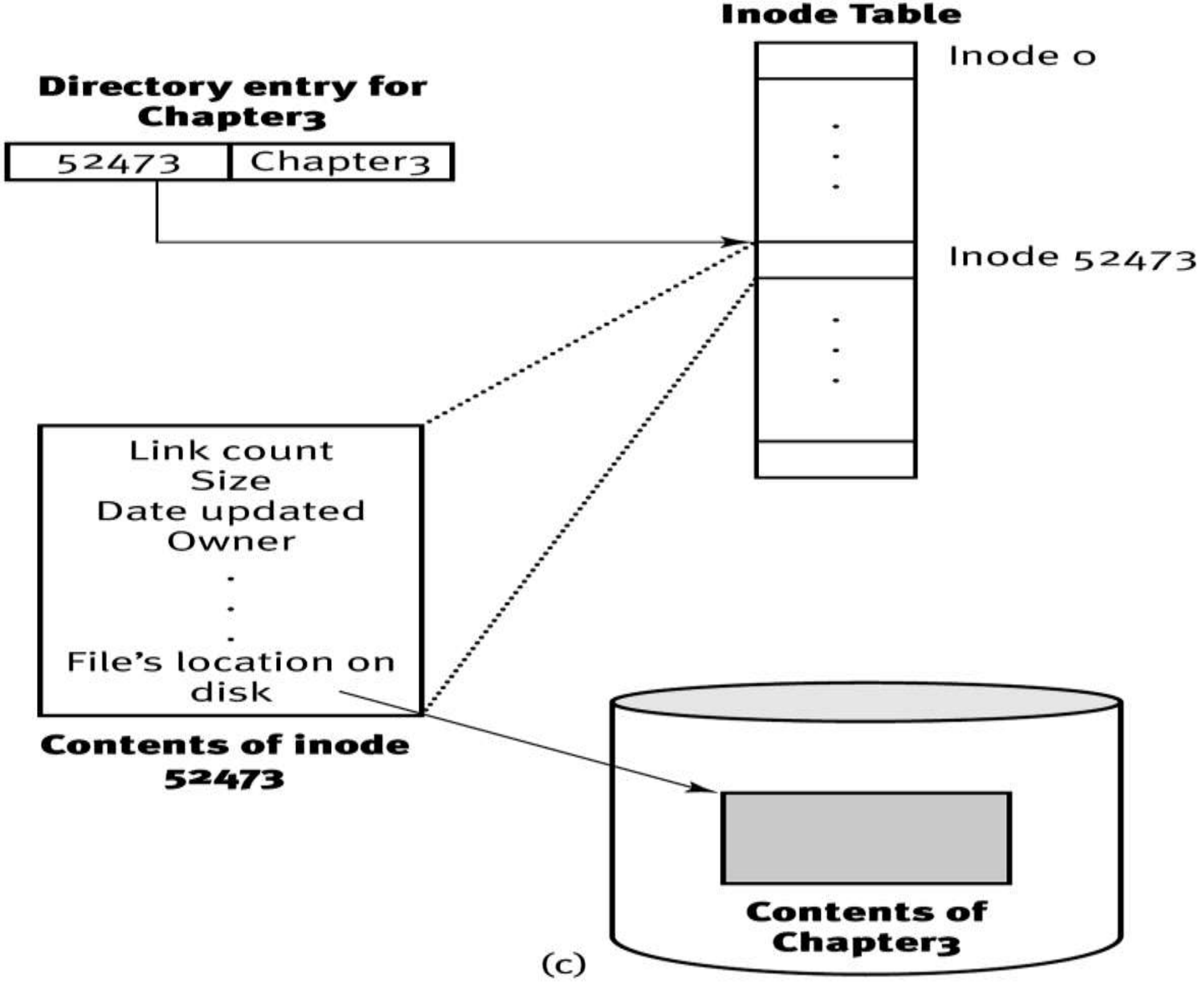


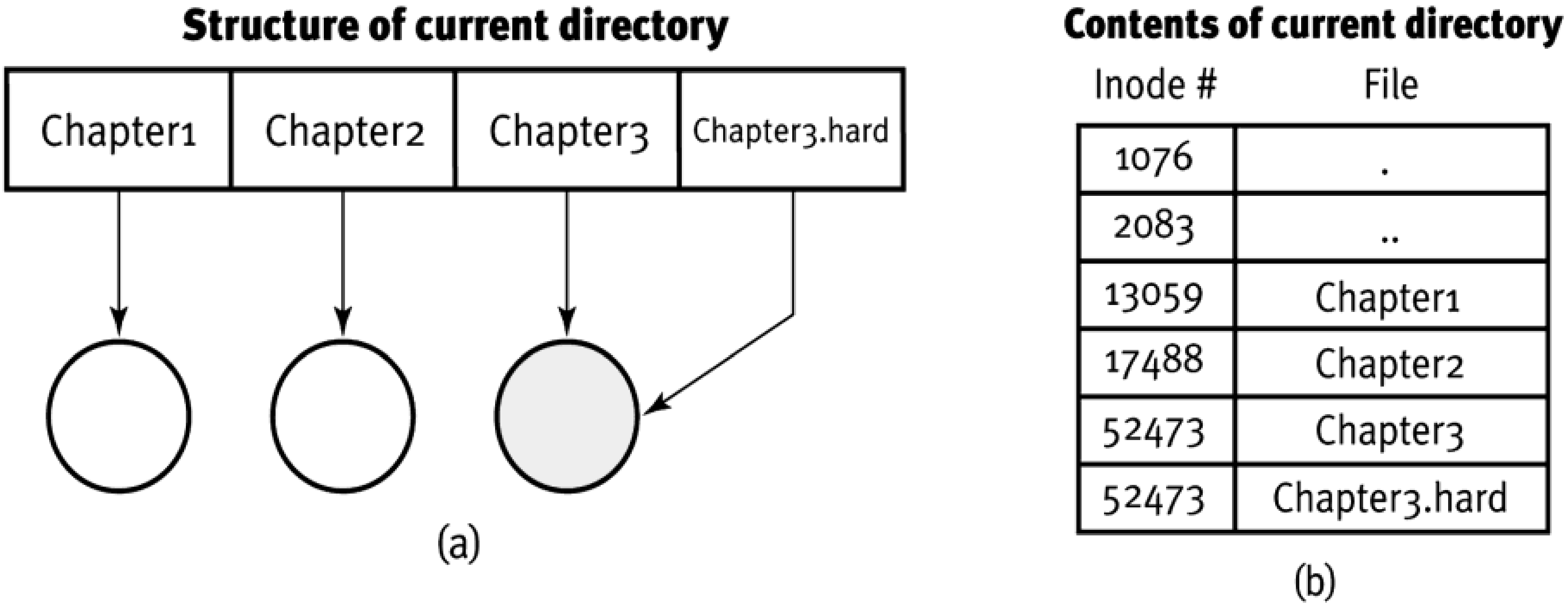
Figure (c) relationship among a directory entry, inode, and file contents

Hard Links

- Hard link is a reference to the physical data on a file system
- A hard link is a pointer to the same inode of a file and established using the ln command
- More than one name can be associated with the same physical data.
- If you change the name of the other file, a hardlink still points to the file.
- The link count of the file is incremented when create hard link.
- When deleted, the link count is decremented, and the file is only deleted if the resulting link count is zero
- This has the effect of creating multiple names for the same file, causing an aliasing effect.

Establishing a hard link:

```
$ ln  Chapter3  hapter3.hard
```



(a)

Contents of current directory

Inode #	File
1076	.
2083	..
13059	Chapter1
17488	Chapter2
52473	Chapter3
52473	Chapter3.hard

(b)

ln command

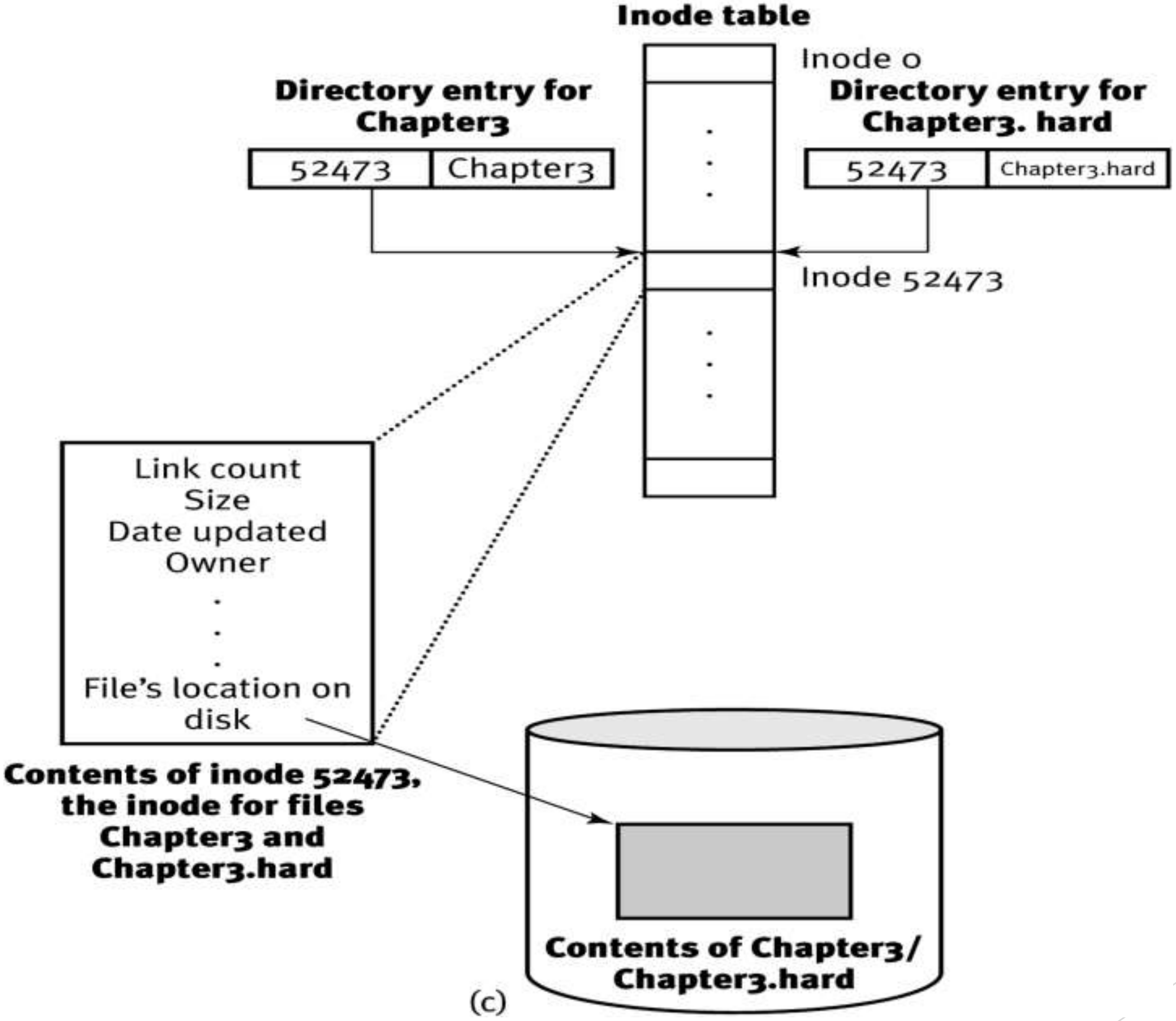
- Used: To create a link of file.
- Syntax

```
ln [options]  oldfile  newfile

ln [options] old-file-list  directory
```

- Options

OPTION	DESCRIPTION
-f	Force creation
-n	Don’t force
-s	Create soft link

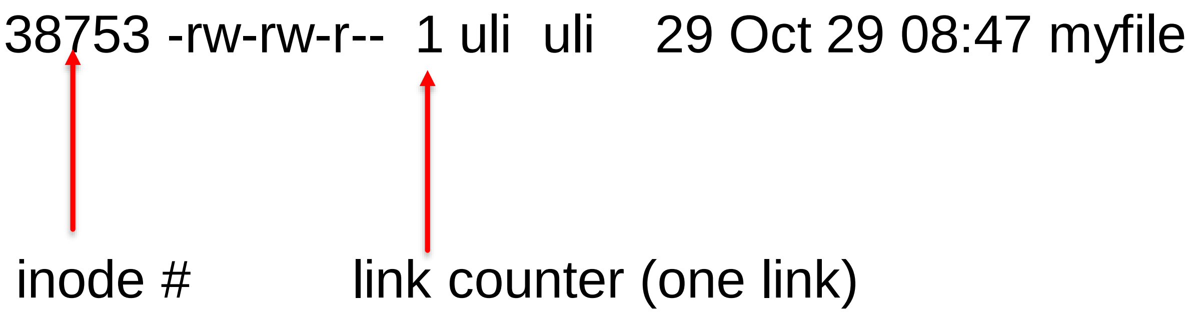

Inode table
Inode 0
Directory entry for Chapter3
52473
Chapter3
Directory entry for Chapter3. hard
52473
Chapter3.hard
Inode 52473
Link count
Size
Date updated
Owner
File's location on disk
Contents of inode 52473, the inode for files Chapter3 and Chapter3.hard
Contents of Chapter3/ Chapter3.hard
(c)

Hard link implementation by establishing a pointer to inode of the file

Display Hard Links info

- Create a new file called “myfile”
- Run the command “ls -il” to display the *i-node number* and *link counter*

```
$ ls –il myfile
```



```
38753 -rw-rw-r--  1 uli  uli    29 Oct 29 08:47 myfile
```

inode #

link counter (one link)

Display Hard Link Info

- Create a 2nd link to the same data:

```
$ ln myfile mylink
```

- Run the command : `$ ls -il`

```
38753    -rw-rw-r--    2     uli     uli    29 Oct 29 08:47 myfile

 38753   -rw-rw-r--    2     uli     uli    29 Oct 29 08:47 mylink
```

^

inode # **link counter (2 links)**

Add the 3rd Link

- Create a 2nd link to the same data:

```
$ ln myfile newlink
```

- Run the command : `$ ls -il`

```
38753 -rw-rw-r-- 3 uli  uli    29 Oct 29 08:47 myfile
38753 -rw-rw-r-- 3 uli  uli    29 Oct 29 08:47 mylink
38753 -rw-rw-r-- 3 uli  uli    29 Oct 29 08:47 newlink
```

inode #

link counter (3 links)

Removing a Hard Link

- When a file has more than one link, you can remove any one link and still be able to access the file through the remaining links.
- Hard links are a good way to backup files without having to use the copy command!

Limitations of Hard Links

- Links cannot be established across file systems
- If one of the files is moved to a different file system, it is copied instead, and the link counts of both files adjusted accordingly
- Only superusers can create hard links to directories

Soft (Symbolic) Links

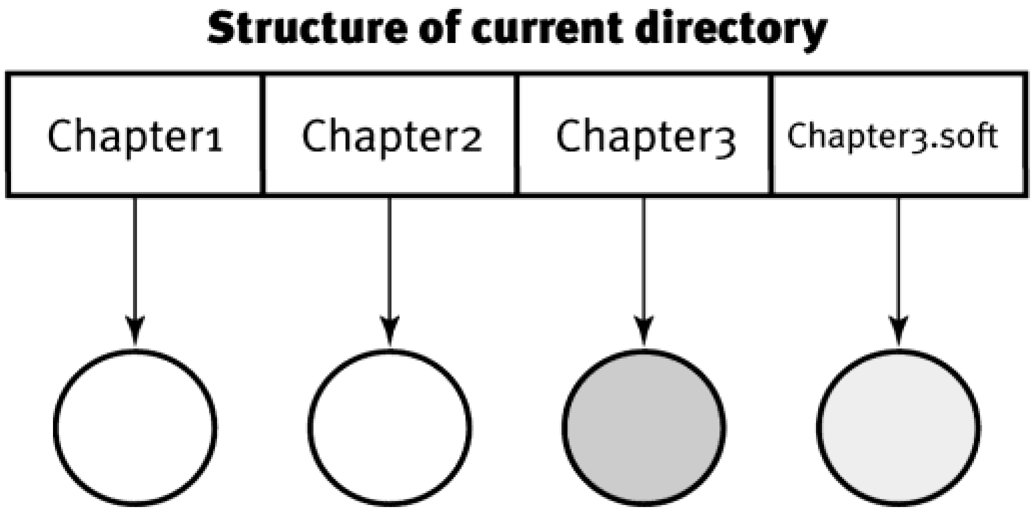
- A soft link (symbolic link or a symlink) makes it possible to associate one file with another.
- It is similar to shortcut in MS Windows where the actual file is resident somewhere in the directory structure but you may have multiple shortcuts or pointers with different names pointing to it.
- This means accessing the file via the actual file name or any of the shortcuts would yield an identical result.
- Each soft link has a unique inode number.
- Established using the ln -s command.
- The link count of the file is not incremented.

Soft (Symbolic) Links

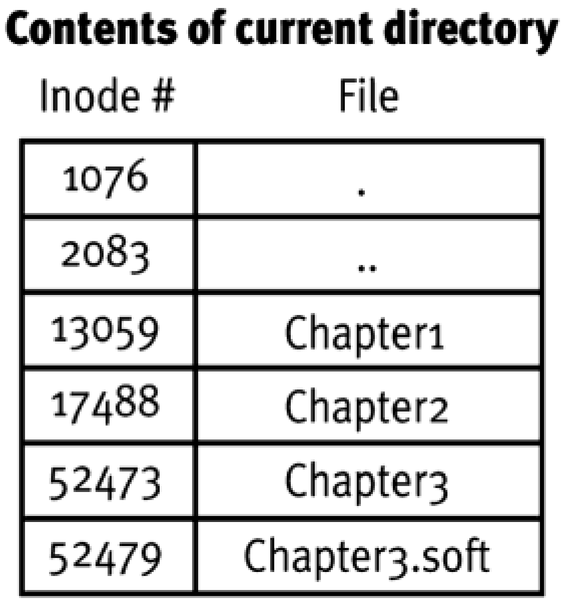
- The created file is of the special type “link” denoted by “l” in directory listings.
- The linked file is an actual file that contains the path to the original file.
- Symbolic links can be created across file systems.
- Symbolic links to directories can be created by any user.

Establishing a soft link:

```
$ ln  -s  Chapter3  Chapter3.soft
```



(a)



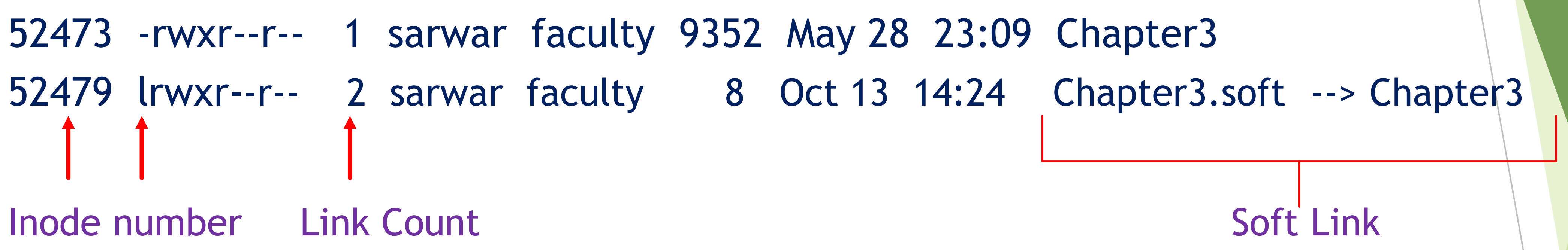
Contents of current directory

Inode #	File
1076	.
2083	..
13059	Chapter1
17488	Chapter2
52473	Chapter3
52479	Chapter3.soft

(b)

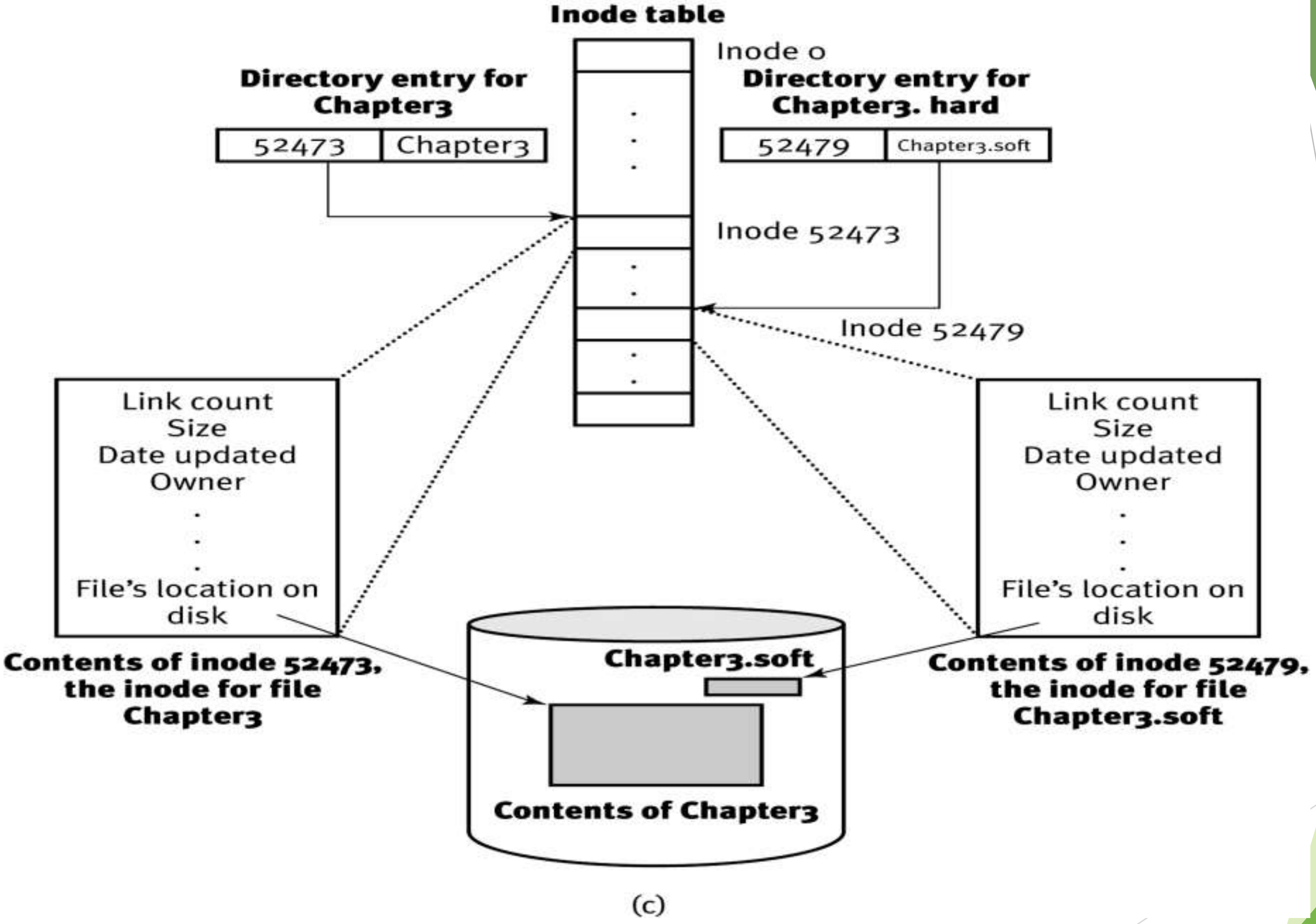
Soft links in directory listing

```
$ ln  -s  Chapter3  Chapter3.soft
$ ls  – il
```



```
52473  -rwxr--r--   1  sarwar  faculty  9352  May 28  23:09  Chapter3
52479  lrwxr--r--   2  sarwar  faculty     8   Oct 13  14:24   Chapter3.soft  --> Chapter3
```

- We create soft line using command ls –s for file chapter3.
- Here we see, in second line of output, that link count is 2 and inode number is different for both file means for softlink, new inode is created. Also check the first character “l” in permission column.
- Also notice an arrow pointing from the linked file to the original file. This indicates that Chapter3.soft is merely a pointer to Chapter3.



soft link implementation by establishing a "pointer" to (pathname of) the existing file in the link file

Drawbacks of Soft Link

- If the original file is moved to a different location, it can no longer be accessed via the symbolic link (dangling link)
- Extra space on disk and extra inode to store the link file
- Extra time required for access to the original file: the link file has to be read first, then path followed to target file

Hard Link Vs Soft Link

- A softlink will have a different Inode number than the source file but hardlink will be using the same Inode number as the source file.
- Soft link can used over the file system but hard link can not.
- Soft link take a space on disk but hard link can not .
- Soft link created on directory but hard link can not.
- To access file using Soft link is slower but we can access it speedy using hard link.
- If remove the file, we can not access file using soft link, but we can access data once original file is deleted , using hard link.

Types of Files

File Type	Description
Regular File	• This is the most common type of a file in Unix. Regular files hold data and executable programs. A large majority of the files found on UNIX and Linux systems are ordinary files. Ordinary files contain ASCII (human-readable) text, executable program binaries, program data, and more. • In long-format output of ls, this type of file is specified by the "-" symbol.

Directory	• Directories are files that contain other files and sub-directories. Directories are used to organize the data by keeping closely related files in the same place.
	• The directories are just like the folders in windows operating system.
	• In long-format output of ls, this type of file is specified by the "d" symbol:
	`$ ls -ld *`
	`-rw-r--r--  1 greys greys  1024 Mar 29 06:31 text`
	`drwxr-xr-x 2 greys greys     4096 Aug 21 11:00 mydir`

Special Or Device File	• Device or special files are used for device I/O on UNIX systems. They appear in a file system just like an ordinary file or a directory. • On UNIX systems there are two flavors of special files for each device 1 Character special files 2 Block special files. • When a character special file is used for device I/O, data is transferred one character at a time. This type of access is called raw device access. It is marked with “c”. • When a block special file is used for device I/O, data is transferred in large fixed-size blocks. This type of access is called block device access. It is mark with “b”.
Symbolic Link	• A symbolic link is a reference to another file. This special file is stored as a textual representation of the referenced file's path (which means the destination may be a relative path, or may not exist at all). • A symbolic link is marked with an l (lower case L) as the first letter of the mode string

Socket	• A socket file is used to pass information between applications for communication purpose
	• In long-format output of ls, Unix sockets are marked by "s" symbol:
	`$ ls -al /dev/log`
	`srw-rw-rw- 1 root root 0 Sep  7 05:04 /dev/log`

Named Pipe	• It is use for Unix interprocess communication.
	• There are circumstances where the communicating processes must use named pipes. One such circumstance is that the processes have to be executed under different user names and permissions.
	• These named pipes are special files that can exist anywhere in the file system.
	• These named pipe special files are made with the command mkfifo as in mkfifo mypipe.
	• In long-format output of ls, named pipes are marked by the "p" symbol:
	`$ ls -al /dev/xconsole`
	`prw-r----- 1 root adm 0 Sep 25 08:58 /dev/xconsole`

Disk Related Command

- One of the major concern of the system administrator of a unix installation is efficient hard disk management.
- Since Unix file system is installed on a hard disk its upkeep is primary importance.
- System administrator has to regularly monitor the integrity of the file system and amount of disk space available.
- Following command can be used to manage space efficiently
  - df
  - dfspace
  - du
  - ulimit

Disk Related Command - df

- df command is used to see how much of the disk is being used and what part of it lies free.
- This command reports the free as well as the used disk space for all the file systems installed on our machine

```
$ df
/ (/dev/root  ):   12970 blocks   27857 inodes
```

- We have on our machine only one file system installed, the root file system.
- If reports the number of free disk blocks and free inodes for this file system.
- If we want a more detailed information about disk usage we use

Disk Related Command - df

```
$ df –ivt

Mount Dir Filesystem  Blocks   Used      Free  %used   iused  ifree   %iused

/           /dev/root   282098  269146  12952   95%    7410   27854   21%
```

- Now available blocks and inodes are reported numerically as well as percentages of total available blocks and inodes. This gives better idea of how much disk space is free.

Disk Related Command - dfspace

- **dfspace command** reports the free disk space in terms of megabytes and percentages of total disk space.

```
$dfspace

Dfspace: not found
```

- It gives error because dfspace command is present in /etc directory. This directory doesn’t get searched when we execute any command. So to execute it we need

```
 $/etc/dfspace

 : Disk space: 6.32 MB of 137.74 available (4.59%)

  Total Disk Space: 6.32 MB of 137.74 available (4.59%)
```

- Now dfspace does all the mathematics internally and reports free disk space for the root file system.

Disk Related Command – dfspace

- If other file system installed their free space would also have been reported.
- Additionally it also report the total disk space available.

Disk Related Command – du

- **du command** reports the disk space used by specified files and directories.

```
$ du
226    ./backup
418    ./fa/backup
1182   ./fa
16     ./dbf
1658
```

- Here **du** is reporting the number of blocks used by the current directory and those used by sub-directories within the current directory. Thus, when invoked without any arguments it assumes that blocks occupied by current directory and the directories lying within it are to be reported.

Disk Related Command – du

- If we specify a directory then du descends down this directory locating any sub-directories lying in it and reports the block used by the directory and sub directories.

```
$ du /dev
2      /dev/string
4    /dev/rdsk
4     /dev/dsk
2     /dev/mouse
20   /dev
```

- **The** number of block occupied by each sub-directory within /dev as well as those occupied by /dev are displayed.

Disk Related Command – du

- If we want to displayed only block occupied by directory not sub-directory, use following command

```
$ du -s /dev

20 /dev
```

Disk Related Command - ulimit

- **ulimit stands** for “User Limit” and contains a value which signifies the largest file that can be created by the user in the file system.
- Sometimes things might take bad turn that the file might occupy several megabytes of disk space and ultimately harm the file system. To avoid creation of such files Unix uses a variable called “ulimit”.
- The current value of the ulimit variable is know by following command.

```
$ulimit
2097152
```

- This implies that the user cannot create a file whose size is bigger than 2097152 bytes or 2048KB. If you happen to create a file which exceeds this size, its size would be curtailed to 2048KB and the program creating this file would be aborted.

Disk Related Command - ulimit

- A user can reduce this value by

```
$ulimit 1
```

- Here onwards no file can be created whose size is bigger than 512 bytes. Once reduced this value remains effective till the user doesn’t log out.
- An ordinary user can only reduce the ulimit value and is never permitted to increase it.

Time Stamp in UNIX

- Unix filesystems store a number of timestamps for each file.
- This means that you can use these timestamps to find out when any file or directory was last accessed (read from or written to), changed (file access permissions were changed) or modified (written to).
- Three times tracked for each file and directory in Unix are these:
  - access time – **atime**
  - change time – **ctime**
  - modify time – **mtime**

Time Stamp in UNIX

atime – File Access Time

- Access time shows the last time the data from a file was accessed – read by one of the Unix processes directly or through commands and scripts.
- The atime gets updated when you open a file but also when a file is used for other operations like grep, sort, cat, head, tail and so on.

ctime – File Change Time

- ctime also changes when you change file's ownership or access permissions. It will also naturally highlight the last time file had its contents updated.

Time Stamp in UNIX

mtime – File Modify Time

- Last modification time shows time of the last change to file's contents.
- It does not change with owner or permission changes, and is therefore used for tracking the actual changes to data of the file itself.
- Most of the times ctime and mtime will be the same, unless only the file attributes are updated. In that case only the ctime gets updated.

Time Stamp in UNIX

- The simplest way to confirm the times associated with a file is to use ls command.
- Timestamps are shown when using the long-format output of ls command, ls -l:

```
$ ls -l /tmp/file1

-rw-r--r-- 1 greys root 9 2008-04-05 07:10 /tmp/file1
```

- This is the default output of ls –l.
- It shows you the time of the last file modification – mtime. In our example, file /tmp/file1 was last changed around 7:10am.

Time Stamp in UNIX

- If we want to see the last access time for this file, atime – you need to use -lu options for ls. The output will probably show some later time:

```
$ ls -lu /tmp/file1
-rw-r--r-- 1 greys root 9 2008-04-05 07:27 /tmp/file1
```

- In the example, it's 7:27am.
- Lastly, ls -lc will show you the last time our file was changed, ctime:

```
$ ls -lc /tmp/file1
-rw-r--r-- 1 greys root 9 2008-04-05 07:31 /tmp/file1
```

- To show how this works, change the ownership of the file and then run the same 3 ls commands to show you that only the ctime had been updated.

How to view atime, ctime and mtime?

- Using stat command is probably the easiest way to look at all the three timestamps associated with each file:

```
$ stat ./try
 File: `./try'
 Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 801h/2049d      Inode: 655596      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/   greys)   Gid: (  113/   admin)
Access: 2008-11-17 05:01:16.000000000 -0600
Modify: 2008-11-17 05:01:16.000000000 -0600
Change: 2008-11-17 05:01:16.000000000 -0600
```

Touch command

- It is also used to change the timestamps (i.e., dates and times of the most recent access and modification) on existing files and directories.
- The touch command is the easiest way to create new, empty files.
- Syntax: $ touch [option] [expression] file_name(s)
- There are following options available with this command.

OPTION	DESCRTIPTION
-a	Change the access time of file.
-c	Do not create a specified file if it does not exist.
-m	Change the modification time of file. Do not change the access time unless -a is also specified.
-d	Update access and modification time.
-t	Create a file using specified time

Touch command

- When used without any options, touch creates new files for any file names that are provided as arguments (i.e.,input data) if files with such names do not already exist.
- Touch can create any number of files simultaneously.
- Thus, for example, the following command would create three new, empty files named file1, file2 and file3:

```
$ touch file1 file2 file3
```

Touch command

Change File access and modification time: To change or update the last access and modification times of a file called leena, use the -a option as follows.

- The following command sets the current time and date on a file. If the leena file does not exist, it will create the new empty file with the name.

```
$ touch -a leena
```

How to Avoid Creating New File

Using -c option with touch command avoids creating new files. For example the following command will not create a file called leena if it does not exists.

```
$ touch -c leena
```

Touch command

Change modification time: If you like to change the only modification time of a file called leena, then use the -m option with touch command.

- Please note it will only updates the last modification times (not the access times) of the file.

```
$ touch -m leena
```

Explicitly Set the Access and Modification times: You can explicitly set the time using –a or -m and -t option with touch command. The format would be as follows.

```
$ touch -a -t YYDDHHMM leena
```

- For example the following command sets the access and modification date and time to a file leena as 17:30 (17:30 p.m.) December 10 of the current year (2012).

```
$ touch -c -t 12101730 leena
```

Touch command

- Next verify the access and modification time of file leena, with ls -l command.

```
$ ls -l

total 2

-rw-r--r--.  1 root    root   0 Dec 10 17:30 leena
```

find command

- Find command used to search and locate list of files and directories based on conditions you specify for files that match the arguments.
- Find can be used in variety of conditions like you can find files by permissions, users,groups, file type, date, size and other possible criteria.

Syntax: find [pathnames] [option] [filename]

- Find all filenames in specified pathnames

-atime n	File was accessed n days ago
-mtime n	File was modified n days ago
-size n	File is n blocks big (a block is 512 bytes)
-type c	Specifies file type: f=plain text, d=directory,b=block,c=character,p=pipe, l=link,s=socket
-fstype typ	Specifies file system type: 4.2 or nfs
-name nam	The filename is nam
-user usr	The file's owner is usr
-group grp	The file's group owner is grp
-perm p	The file's access mode is p (where p is an integer)

find command

- Find all the files under /home directory with name tecmint.txt.

```
$ find /home -name tecmint.txt

/home/tecmint.txt
```

Find Files Using Name and Ignoring Case: Find all the files whose name is tecmint.txt and contains both capital and small letters in /home directory.

```
$ find /home -iname tecmint.txt

./tecmint.txt

./Tecmint.txt
```

For ignoring case use –I option

find command

- Find file name using name: Find all the files whose name is tecmint.txt in a current working directory.

  ```
  $ find . -name tecmint.txt  [ . Represent pathname, -name is option , techmint is filename]
  ./tecmint.txt
  ```

- Use –name to find file using its name

OR

```
$ find –name tecmint.txt
```

find command

Find Directories Using Name: Find all directories whose name is Tecmint in / directory

```
$ find / -type d -name Tecmint

/Tecmint
```

Find PHP Files Using Name: Find all php files whose name is tecmint.php in a current working directory.

```
$ find . -type f -name tecmint.php

./tecmint.php
```

Find all PHP Files in Directory: Find all php files in a directory.

```
$ find . -type f -name "*.php"

./tecmint.php

./login.php
```

find command

Find Files With 777 Permissions: Find all the files whose permissions are 777.

```
$ find . -type f -perm 0777
```

- Use –perm option with –type

Find Files Without 777 Permissions: Find all the files without permission 777.

```
$ find / -type f ! -perm 777
```

- Use – perm option with ! Mark which point neglation .

Find Read Only Files: Find all Read Only files.

```
$ find / -perm /u=r
```

Find Executable Files: Find all Executable files.

```
$ find / -perm /a=x
```

find command

Find all Empty Files: To find all empty files under certain path.

```
$ find /tmp -type f –empty
```

- Use “f” to find file with –type option

Find all Empty Directories: To find all empty directories under certain path.

```
$ find /tmp -type d –empty
```

- Use “d” to find directory with –type option.

File all Hidden Files: To find all hidden files, use below command.

```
$ find /tmp -type f -name ".*"
```

find command

Find by user: Search for all the files with name test.txt and the owner of this file is Surendra

```
find / -user Surendra –name test.txt
```

Find by Group: find all the files whos name is test.txt and owned by a group called redcluster

```
find / -group redcluster –name test.txt
```

Find Based on size:Search for files whose size is more than 10bytes

```
find / -size +10c
```

USe “c” for bytes , “k” for Kilobytes,”m” for megabytes,”g” for gigabytes

Find Based on size:Search for files whose size is exactly 10Kilobytes

```
find / -size 10k
```

find command

Find by modification time: List file which is modify last day

```
find / -mtime 1
```

Find all the files which is modify in last seven days.

```
find / -mtime -7
```

Find Based on size:Search for files whose size is more than 10bytes

```
find / -size +10c
```

USe “c” for bytes , “k” for Kilobytes,”m” for megabytes,”g” for gigabytes

Find Based on size:Search for files whose size is exactly 10Kilobytes

```
find / -size 10k
```