Computer System & Network Administration

Lecture 04. Disk Management & File System

refer to: https://nasa.cs.nctu.edu.tw/sa/2020/

Outline

- Disk
  - Interface
  - Geometry
- GPT
- UEFI
- RAID
- File System
  - Architecture
  - inode and file
  - File access mode

Disk

Disk Interfaces

- SCSI (since 1986)
  - Small Computer Systems Interface
- PATA/IDE (ATA) (since 1986)
  - Integrated Device Eletronics (or Advanced Technology Attachment)
  - Low cost
  - Become acceptable for enterprise with the help of RAID
- SATA (since 2003)
  - Serial ATA
- SAS (since 2004)
  - Serial Attached SCSI
- PCIe (since 2013)
  - Being widely used in modern laptop / server

Disk Interfaces - ATA & SATA

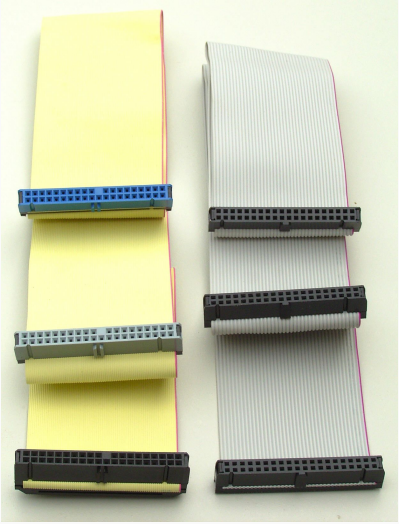
- ATA (AT Attachment)
  - ATA2
    - PIO, DMA
    - LBA (Logical Block Addressing)
  - ATA3, Ultra DMA/33/66/100/133
  - ATAPI (ATA Packet Interface)
    - CDROM, TAPE
  - Only one device can be active at a time
    - SCSI support overlapping commands, command queuing, scatter-gather I/O
  - Master-Slave
  - 40-pin ribbon cabble

Primary Master (0) / Slave (1)
Secondary Master (2) / Slave (3)

- SATA
  - Serial ATA
  - SATA-1 1.5Gbits/, SATA-2 3Gbit/s, SATA-3 6Gbit/s

Disk Interfaces - ATA & SATA

IDE Interface



IDE cable

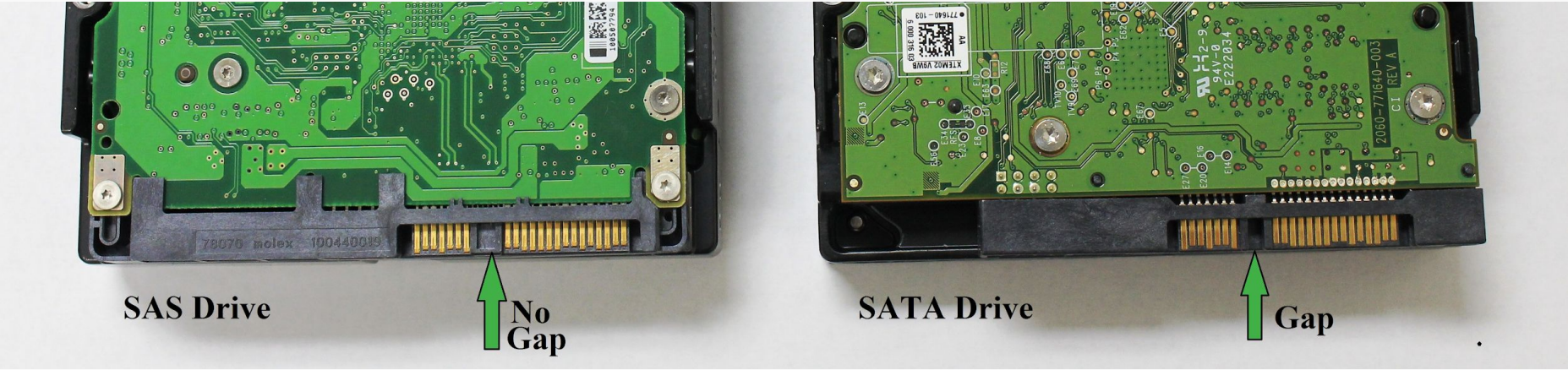
SATA Interface

SATA cable

Disk Interfaces - SAS

- SAS
  - Replaced SCSI
  - Optional compatibility with SATA-2 and later
  - Can connect SATA drive to SAS backplanes
  - Can’t connect SAS drive to SATA backplanes
  - Higher RPM than SATA
    - SAS: 15000rpm
    - SATA: 7200rpm
  - Faster speed
    - SAS-1 3Gbit/s, SAS-2 6Gbit/s, SAS-3 12Gbit/s, SAS-4 22.5Gbit/s
  - Full-Duplex
    - SATA is Half-Duplex

Disk interfaces - SAS



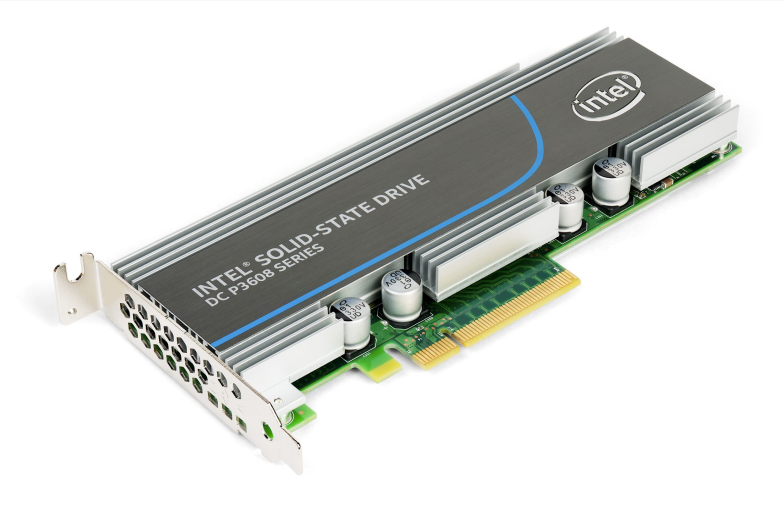
SAS cable (SFF8482)

Internal Mini-SAS (SFF8087)

Disk interfaces - PCIe

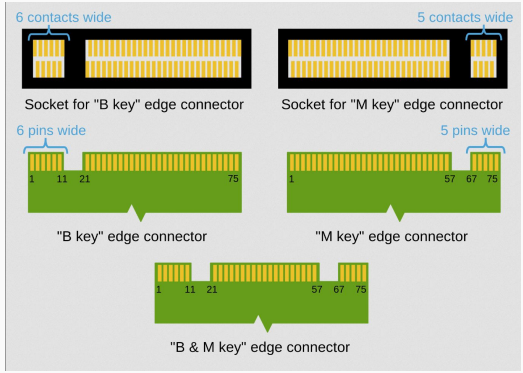
- PCIe
  - PCI Express
  - Normally uses NVMe protocol
    - NVM Express, Non-Volatile Memory Host Controller Interface Specification
    - Reduces I/O overhead
    - Multiple long command queues
    - Reduced latency
    - SATA can’t fully utilize SSD performance

Disk Interfaces - PCIe



PCIe NVMe SSD, PCIe x 4 interface

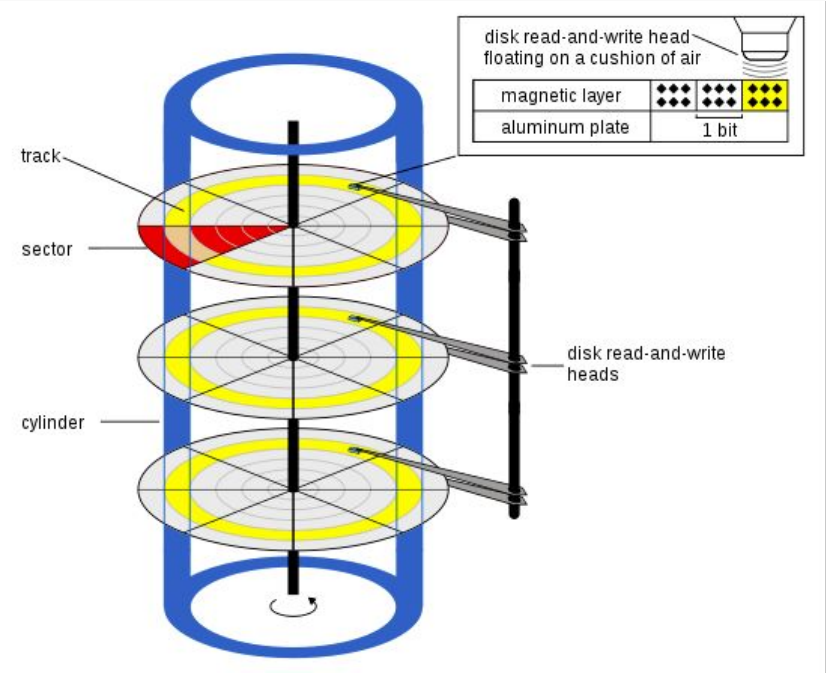
PCIe NVMe SSD, M.2 interface



M.2 key

Disk Geometry

- Sector
  - Individual data block
- Track
  - Circle
- Cylinder
  - Circle on all plattrers
- Position
  - CHS
    - Cylinder
    - Head
    - Sector



Disk Geometry

- 40G HDD
  - 4866 cylinders, 255 heads
  - 63 sector per track, 512 bytes per sector
  - 512 * 63 * 4866 * 255 = 40,024,212,480 bytes

    G M K
  - 1KB = 1024 bytes
  - 1MB = 1024KB = 1048576 bytes
  - 1GB = 1024MB = 1073741824 bytes
  - 40024212480 / 1073741824 ≒ 37.725GB
    - Why?

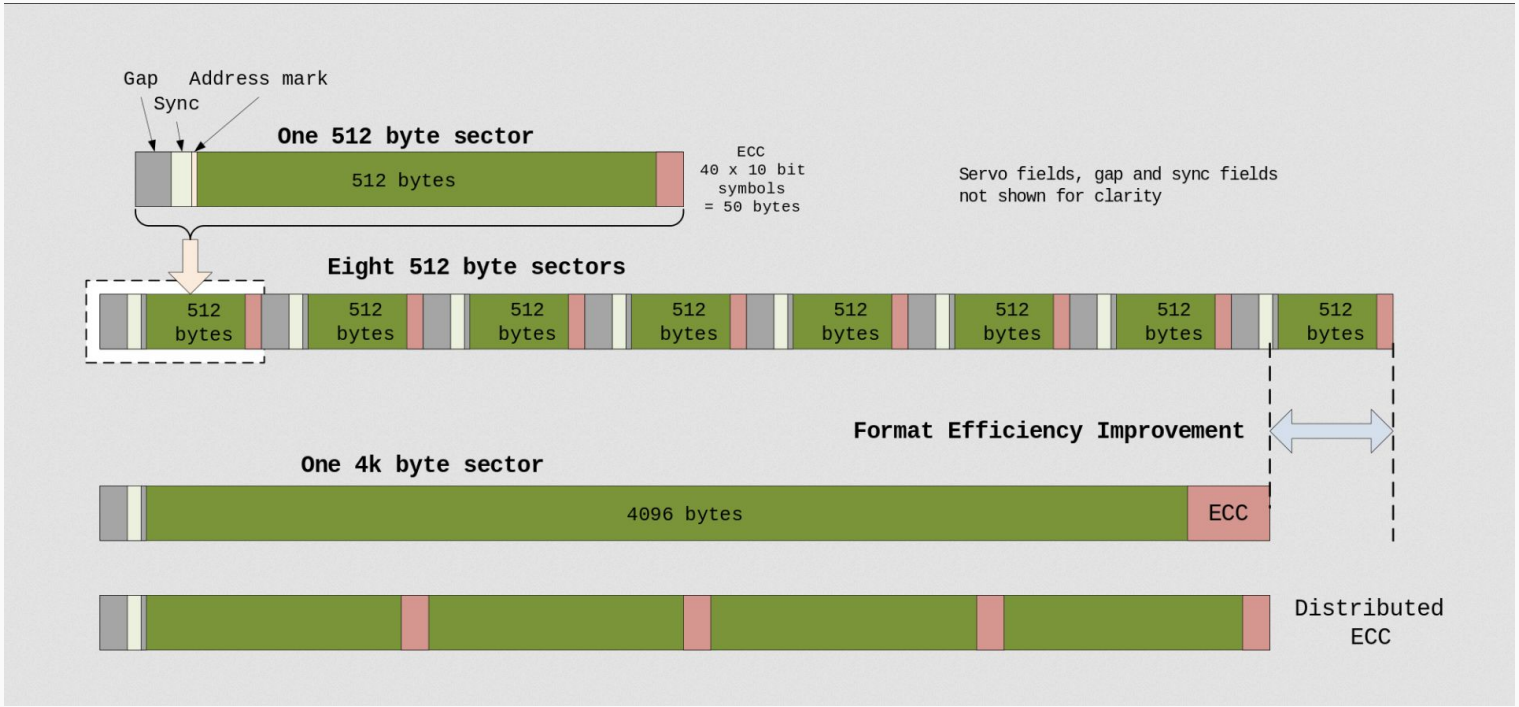
Disk Geometry

- SI vs IEC 60027-2
  - SI
    - 1 kilobyte = 10^3
    - kB, MB, GB, TB, PB, EB, ZB, YB
  - IEC 60027-2
    - 1 kibibyte = 2 ^ 10
    - KiB, MiB, GiB, TiB, PiB, EiB, ZiB, YiB
  - JEDEC
    - Define KB, MB, GB same as IEC standard
- Different software uses different standard

Disk Geometry - Sector - Advanced Format

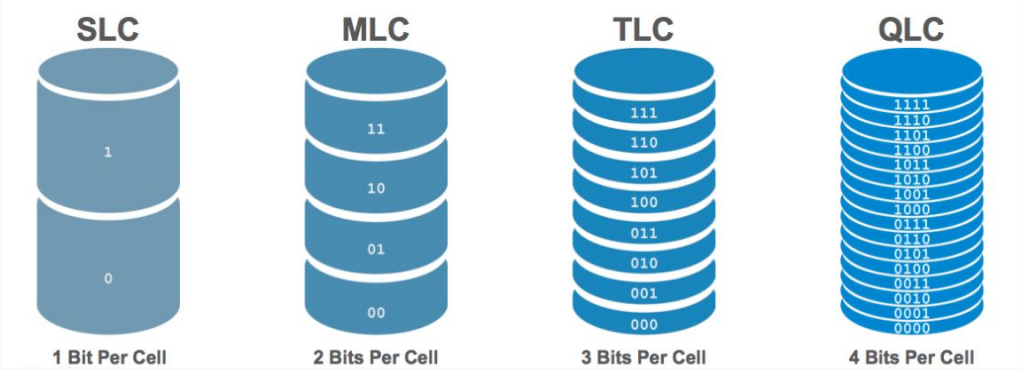
- Traditionally, hard disks uses 512 bytes per sector
- IDEMA defined 1024 bytes, 2048 bytes and 4096 bytes with hard drive manufacturers
  - Advanced Format is 4096 bytes
- Higher space efficiency
- Larger ECC data block size

Disk Geometry - Sector - Advanced Format

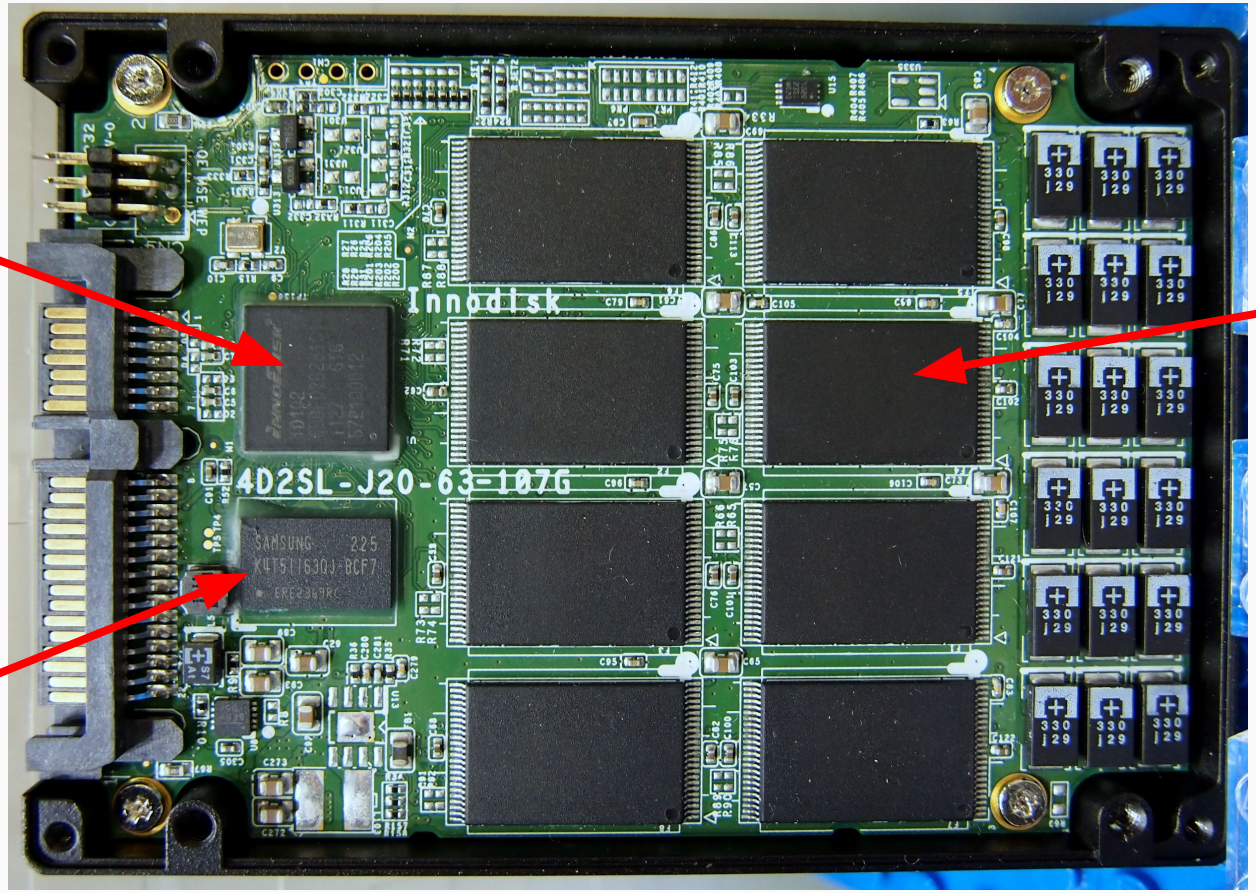


Disk Geometry - Solid-state Drive

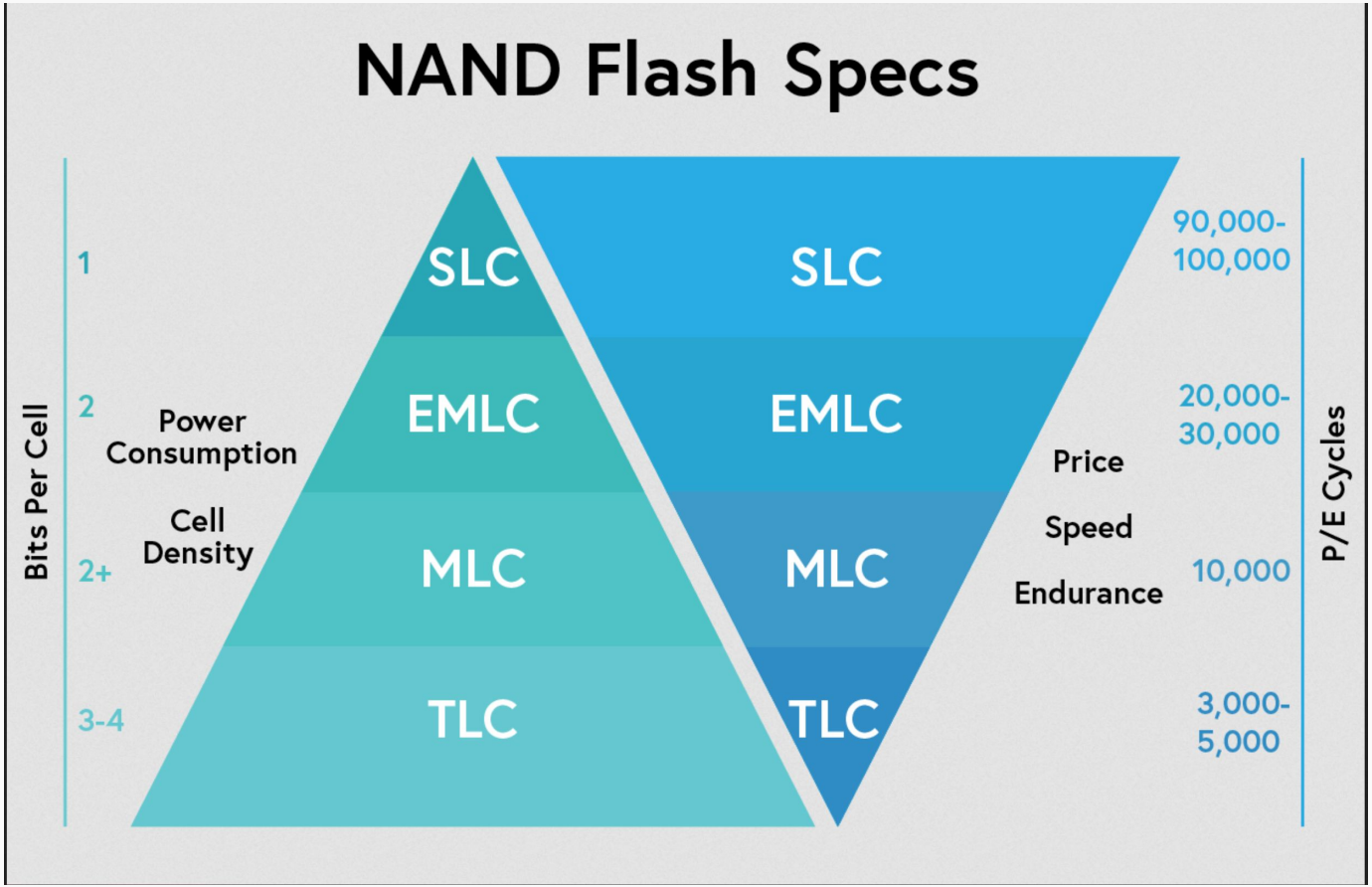
- Controller
- Memory
  - SLC
  - MLC
  - TLC
  - QLC
- Cache
  - QLC combined with SLC
  - Faster IO speed with Larger usable data



Disk Geometry - Solid-state Drive



Disk Geometry - Solid-state Drive - NAND



Disk Installation Procedure

- Connect disk to PC
  - IDE: Master/Slave
  - SATA
  - SAS
  - Power
- Device files should be automatically created

```
F74076310@F74076310:~$ ls /dev/vd*
/dev/vda  /dev/vda1  /dev/vda14  /dev/vda15  /dev/vdb  /dev/vdc
F74076310@F74076310:~$
```

Disk Installation Procedure

- Create partition table
- Create partition
  - MBR vs. GPT
- Format partition
- Mount partition

Disk Installation Procedure - GPT

Create a partition first (Partition table is automatically generated on a new drive)

Format the hard drive

Mount the partition

```
F74076310@F74076310:~$ sudo gdisk /dev/vdc
GPT fdisk (gdisk) version 1.0.5

Partition table scan:
  MBR: not present
  BSD: not present
  APM: not present
  GPT: not present

Creating new GPT entries in memory.

Command (? for help): n
Partition number (1-128, default 1):
First sector (34-16777182, default = 2048) or {+-}size{KMGTP}:
Last sector (2048-16777182, default = 16777182) or {+-}size{KMGTP}:
Current type is 8300 (Linux filesystem)
Hex code or GUID (L to show codes, Enter = 8300):
Changed type of partition to 'Linux filesystem'

Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? (Y/N): Y
OK; writing new GUID partition table (GPT) to /dev/vdc.
The operation has completed successfully.
F74076310@F74076310:~$ sudo mkfs.ext4 /dev/vdc1
mke2fs 1.45.5 (07-Jan-2020)
Discarding device blocks: done
Creating filesystem with 2096891 4k blocks and 524288 inodes
Filesystem UUID: 11bf0f30-5e89-4bbc-af8d-b39fa422f273
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

F74076310@F74076310:~$ sudo mount /dev/vdc1 /mnt
F74076310@F74076310:~$ mount | grep /dev/vdc1
/dev/vdc1 on /mnt type ext4 (rw,relatime)
F74076310@F74076310:~$ ls /mnt
lost+found
F74076310@F74076310:~$
```

Partition Table - MBR

- Master boot record (DOS partition table)
- First 512 bytes of a storage device

Offset	Length	Contents
0	446 bytes	Boot code area
446	64 bytes	Partition tables, each has 16 bytes
510	2 bytes	Boot signature (0x55AA)
512	Total	

Partition Table - MBR

- Drawbacks
  - (4 primary partitions) or (3 primary partitions + 1 extended partition)
    - Arbitrary number of logical partitions within the extended partition
  - Logical partitions meta-data is stored in a linked-list structure
  - One byte partition type codes which leads to many collisions

Partition Table - MBR (cont.)

- Drawbacks
  - Maximum addressable size is 2TiB
    Any space beyond 2TiB can't be defined as a partiton
    - MBR stores partition sector information using 32-bit LBA values
      - Logical Block Addressing
      - A common scheme used for specifying the location of blocks of data stored on computer storage devices
    - 512 bytes per sector
    - (2^32) * 512byte = 2TiB

GPT

GPT

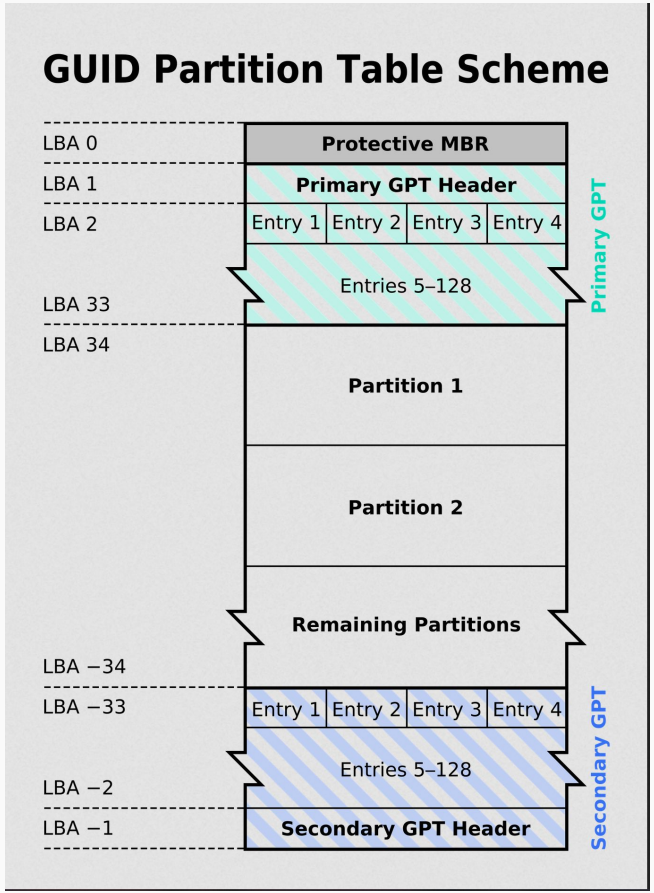
- GUID Partition Table
- GUID
  - Globally Unique Identifier
- Part of UEFI specification
- Solve some legacy problem with MBR, but also may have compatibility issues
- Can also be used on BIOS system via protective MBR

GPT - Advantages

- Filesystem-independent
- No partition type collision because of GUIDs
- 8 ZiB
  - GPT uses 64-bit LBA
  - 512 bytes per sector
  - 2^64 * 512bytes = 8ZiB
- Backup header and partition table at the end of the disk
- CRC32 checksums for header and partition table

GPT Scheme

- LBA 0: Legacy MBR / Protective MBR
- LBA 1: GPT Header
- LBA 2 ~ 33: Partition entries
  - Up to 128 partitions
- LBA 34 ~ : Partitions
- LBA -34 ~ -1: Secondary GPT data



GPT - LBA 0

- Legacy MBR / Protective MBR
  - A single partition type of 0xEE
  - For OSes can’t read GPT disks: Unknown type, no usable space
  - For GPT-aware OSes: Check the protective MBR

GPT - LBA 1

- GPT header (LBA 1)

Offset	Length	Contents
0	8 bytes	Signature ("EFI PART", 45 46 49 20 50 41 52 54)
8	4 bytes	Revision (For GPT version 1.0 (through at least UEFI version 2.3.1), the value is 00 00 01 00)
12	4 bytes	Header size in little endian (in bytes, usually 5C 00 00 00 meaning 92 bytes)
16	4 bytes	CRC32 of header (0 to header size), with this field zeroed during calculation
20	4 bytes	Reserved; must be zero
24	8 bytes	Current LBA (location of this header copy)
32	8 bytes	Backup LBA (location of the other header copy)
40	8 bytes	First usable LBA for partitions (primary partition table last LBA + 1)
48	8 bytes	Last usable LBA (secondary partition table first LBA - 1)
56	16 bytes	Disk GUID (also referred as UUID on UNIXes)
72	8 bytes	Partition entries starting LBA (always 2 in primary copy)
80	4 bytes	Number of partition entries
84	4 bytes	Size of a partition entry (usually 128)
88	4 bytes	CRC32 of partition array
92	*	Reserved; must be zeroes for the rest of the block (420 bytes for a 512-byte LBA)

GPT - LBA 1 (cont.)

```
F74076310@F74076310:~$ sudo dd if=/dev/vda bs=512 count=1 skip=1 | hd
00000000  45 46 49 20 50 41 52 54  00 00 01 00 5c 00 00 00  |EFI PART....\...|
00000010  a0 98 0f ee 00 00 00 00  01 00 00 00 00 00 00 00  |................|
00000020  ff 5f 06 01 00 00 00 00  22 00 00 00 00 00 00 00  |._......".......|
00000030  de 5f 06 01 00 00 00 00  87 4f c2 25 a5 ec 7d 4e  |._.......O.%..}N|
00000040  a0 67 ae 03 14 95 d5 40  02 00 00 00 00 00 00 00  |.g.....@........|
00000050  80 00 00 00 80 00 00 00  38 67 30 d8 00 00 00 00  |........8g0.....|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
1+0 records in
1+0 records out
512 bytes copied, 0.00107389 s, 477 kB/s
00000200
```

- Partition entries (LBA 2)

Offset	Length	Contents
0	16 bytes	Partition type GUID
16	16 bytes	Unique partition GUID
32	8 bytes	First LBA (little-endian)
40	8 bytes	Last LBA (inclusive, usually odd)
48	8 bytes	Attribute flags (e.g. bit 60 denotes read-only)
56	72 bytes	Partition name (36 UTF-16LE code units)
	128 bytes	Total

GPT - LBA 2 (cont.)

Partition Type	GUID
EFI system partition	C12A7328-F81F-11D2-BA4B-00A0C93EC93B
BIOS boot partition	21686148-6449-6E6F-744E-656564454649
Linux filesystem data	0FC63DAF-8483-4772-8E79-3D69D8477DE4
Linux swap partition	0657FD6D-A4AB-43C4-84E5-0933C84B4F4F
LVM partition	E6D6D379-F507-44C2-A23C-238F2A3DF928
OpenZFS	6A898CC3-1DD2-11B2-99A6-080020736631
Unix File System partition	516E7CB6-6ECF-11D6-8FF8-00022D09712B
FreeBSD swap partition	516E7CB5-6ECF-11D6-8FF8-00022D09712B
Microsoft Reserved Partition	E3C9E316-0B5C-4DB8-817D-F92DF00215AE
Microsoft Basic data partition	EBD0A0A2-B9E5-4433-87C0-68B6B72699C7

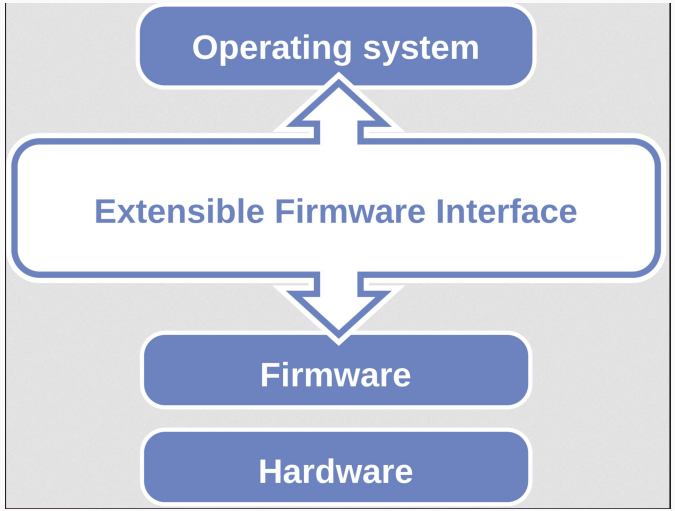
GPT - LBA 2 (cont.)

```
F74076310@F74076310:~$ sudo dd if=/dev/vda bs=512 count=10 skip=1 | hd
00000000  45 46 49 20 50 41 52 54  00 00 01 00 5c 00 00 00  |EFI PART....\...|
00000010  a0 98 0f ee 00 00 00 00  01 00 00 00 00 00 00 00  |................|
00000020  ff 5f 06 01 00 00 00 00  22 00 00 00 00 00 00 00  |._......".......|
00000030  de 5f 06 01 00 00 00 00  87 4f c2 25 a5 ec 7d 4e  |._.......O.%..}N|
00000040  a0 67 ae 03 14 95 d5 40  02 00 00 00 00 00 00 00  |.g.....@........|
00000050  80 00 00 00 80 00 00 00  38 67 30 d8 00 00 00 00  |........8g0.....|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000200  af 3d c6 0f 83 84 72 47  8e 79 3d 69 d8 47 7d e4  |.=....rG.y=i.G}.|
00000210  0c 9b e2 04 16 71 73 4d  bc c1 cd 25 5f 12 94 bb  |.....qsM...%_...|
00000220  00 78 03 00 00 00 00 00  de 5f 06 01 00 00 00 00  |.x......._......|
00000230  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000880  48 61 68 21 49 64 6f 6e  74 4e 65 65 64 45 46 49  |Hah!IdontNeedEFI|
00000890  30 59 ab 95 c7 66 5f 4c  b9 85 50 c8 2c c1 16 27  |0Y...f_L..P.,..'|
000008a0  00 08 00 00 00 00 00 00  ff 27 00 00 00 00 00 00  |.........'......|
000008b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000900  28 73 2a c1 1f f8 d2 11  ba 4b 00 a0 c9 3e c9 3b  |(s*......K...>.;|
00000910  dc 17 f0 8e fe fb fc 46  9c eb c3 e0 e8 4b a9 10  |.......F.....K..|
00000920  00 28 00 00 00 00 00 00  ff 77 03 00 00 00 00 00  |.(.......w......|
00000930  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

UEFI

UEFI

- Unified Extensible Firmware Interface
- Legacy BIOS limitations
  - 16-bit processor mode
  - 1MB addressable space
- Advantages
  - 32-bit/64-bit processor mode
  - Ability to boot from larger disk with help from GPT
  - Flexible pre-OS environment, including network capability
  - Modular design
- Compatibility Support Module (CSM)
  - BIOS-MBR
  - BIOS-GPT



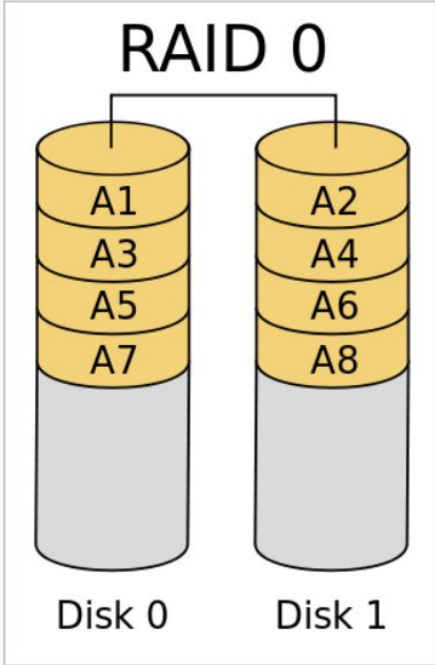
RAID

RAID

- Redundant Array of Inexpensive Disks
- Combine multiple physical drive into one or more logical units for the purpose of data redundancy or performance improvement, or both
- Common RAID levels
  - RAID 0
  - RAID 1
  - RAID 5
  - RAID 6
  - RAID 10
  - RAID 50

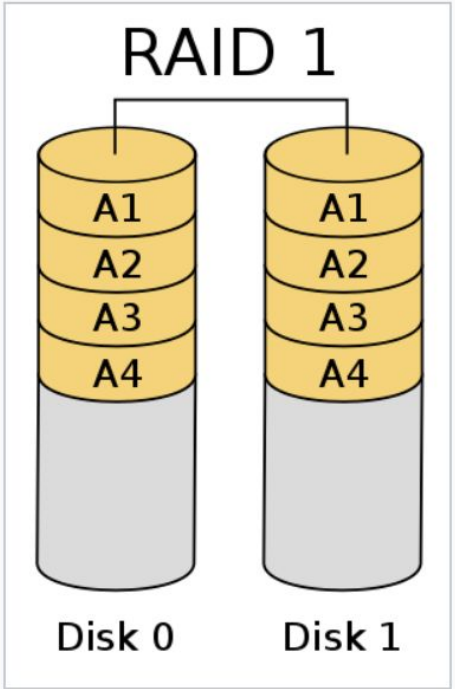
RAID 0

- No redundancy
- Data can be written into separate disks, resulting in fastest performance
- Maximum usable space



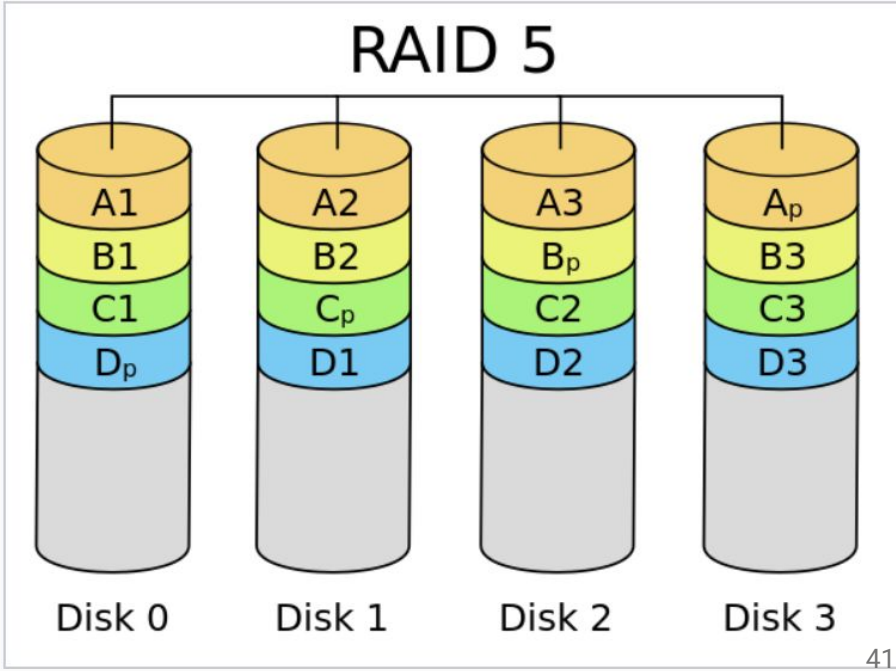
RAID 1

- Maximum redundancy
- Data need to be written into every single disk, resulting in slowest write performance
- Read performance is better, because you can read from any disk and getting same data
- Minimum usable space



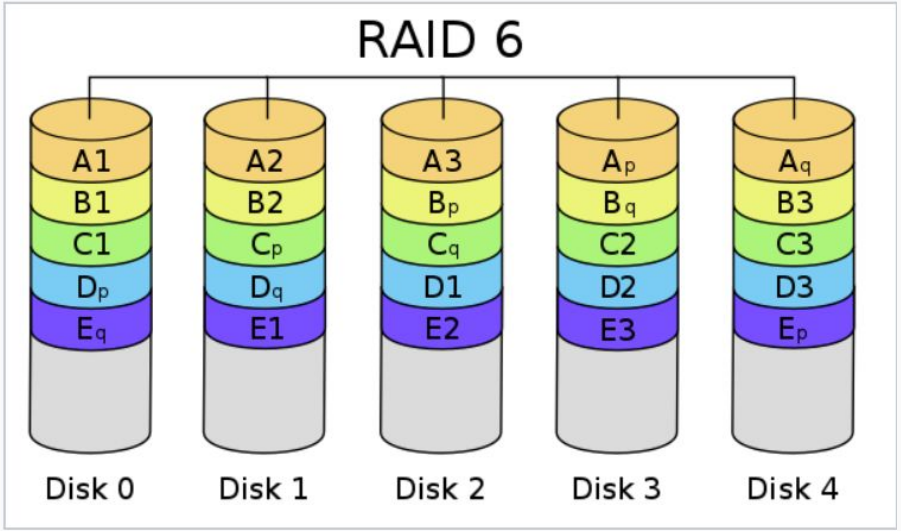
RAID 5

- Fault tolerance is 1 disk
- A data would be divided into N-1 parts, written into N-1 disks, and the last disk would have this data's parity bit
- In case of failed disk, you can use parity and remaining data to restore the lost data



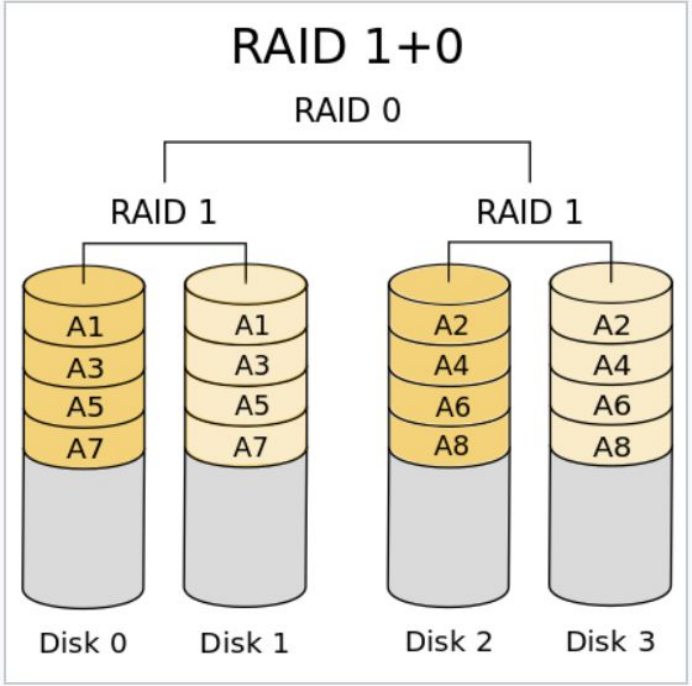
RAID 6

- Fault tolerance is 2 disk
- Dual check data computations (parity and Reed-Solomon)
- In case of failed disk, you can use parity, Reed-Solomun and remaining data to restore the lost data

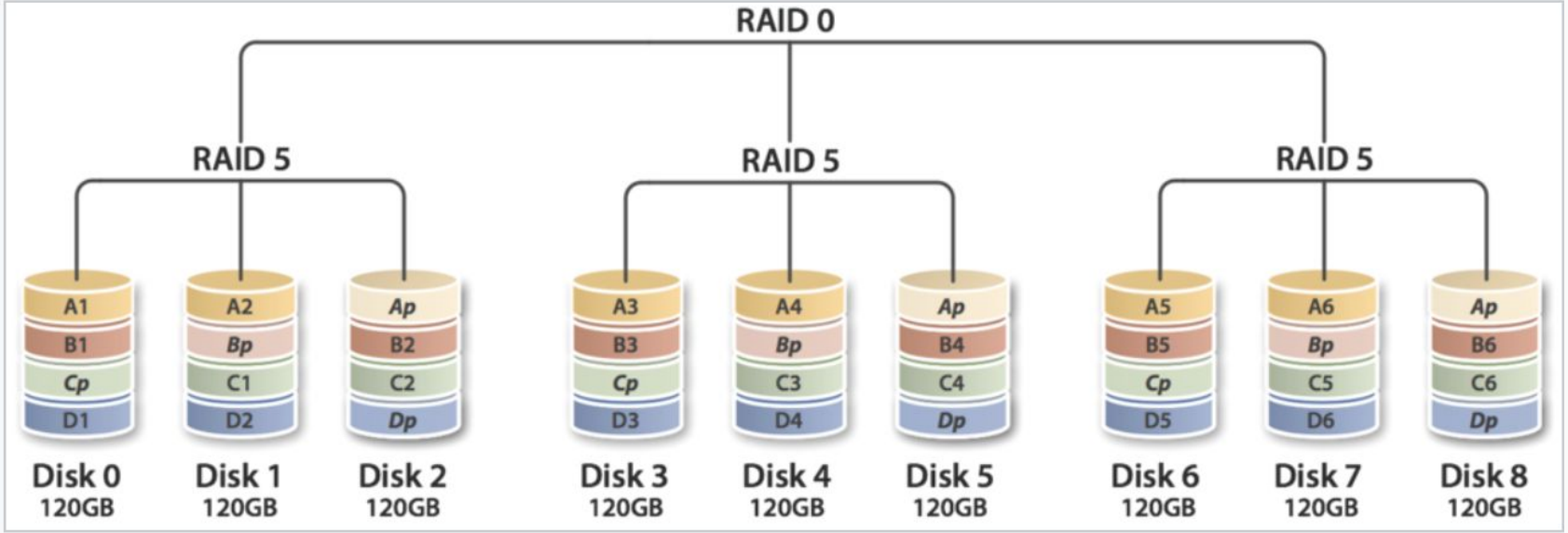


RAID 10

- Use RAID 1 to create base logical unit
- Combine those logical unit into RAID 0



RAID 50


RAID 0
RAID 5
RAID 5
RAID 5
A1
B1
Cp
D1
A2
Bp
C1
D2
Ap
B2
C2
Dp
A3
B3
Cp
D3
A4
Bp
C3
D4
Ap
B4
C4
Dp
A5
B5
Cp
D5
A6
Bp
C5
D6
Ap
B6
C6
Dp
Disk 0
120GB
Disk 1
120GB
Disk 2
120GB
Disk 3
120GB
Disk 4
120GB
Disk 5
120GB
Disk 6
120GB
Disk 7
120GB
Disk 8
120GB

RAID - Restore failed array

- Normally in RAID5 and RAID6, if a disk failed, you replace the failing disk with a new disk, and let array rebuild
  - What's the problem with this behavior?
- Other disks may fail during array rebuilding process, thus **causing permanent data loss**
  - Why?
- Be **extra** careful when handling arrays

- **Redundant is not backup**

File System

Files

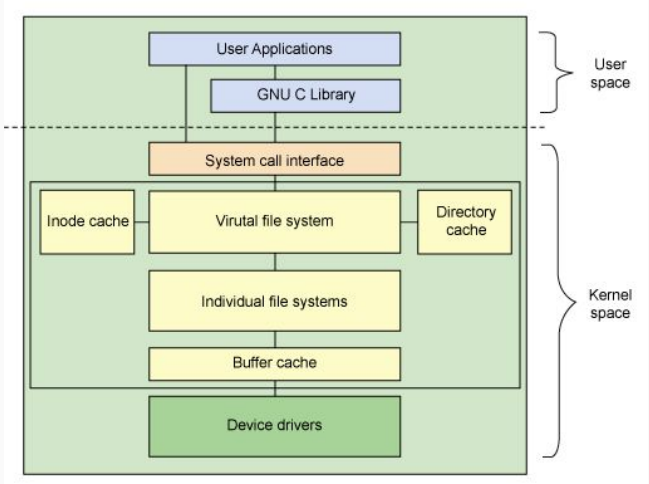
- ls -l

```
F74076310@F74076310:~$ ls -l
total 0
-rw-rw-r-- 1 F74076310 F74076310 0 Mar 13 16:20 bar
-rw-rw-r-- 1 F74076310 F74076310 0 Mar 13 16:19 foo
```

File Type	-/d (File / Directory)
File Access Mode	rw-rw-r-- (664)
inodes	1
File User Owner	F74076310
File Group Owner	F74076310
File Size	0
File Last Modify Time	Mar 13 16:20
File Name	bar / foo ...

File System Architecture

- Application / Kernel / Hardware
  - Applications call system-calls to request service
  - Kernel invokes corresponding drivers to fulfill this service



File System Architecture (cont.)

- Basic purpose of filesystem
  - Represent and organize the system's storage
  - Four main components
    - Namespace
      - A way of naming things and arranging them in a hierarchy
    - Application Programming Interface
      - A set of system calls for navigating and manipulating nodes
    - Security Model
      - A scheme for protecting, hiding and sharing things
    - Implementation
      - Code that ties the logical model to an actual disk

File System Architecture (cont.)

- Objects in the filesystem:
  - Files and directories
  - Hardware device files
  - Processes Information
  - Interprocess communication channel (IPC)
  - Shared memory segments (SHM)

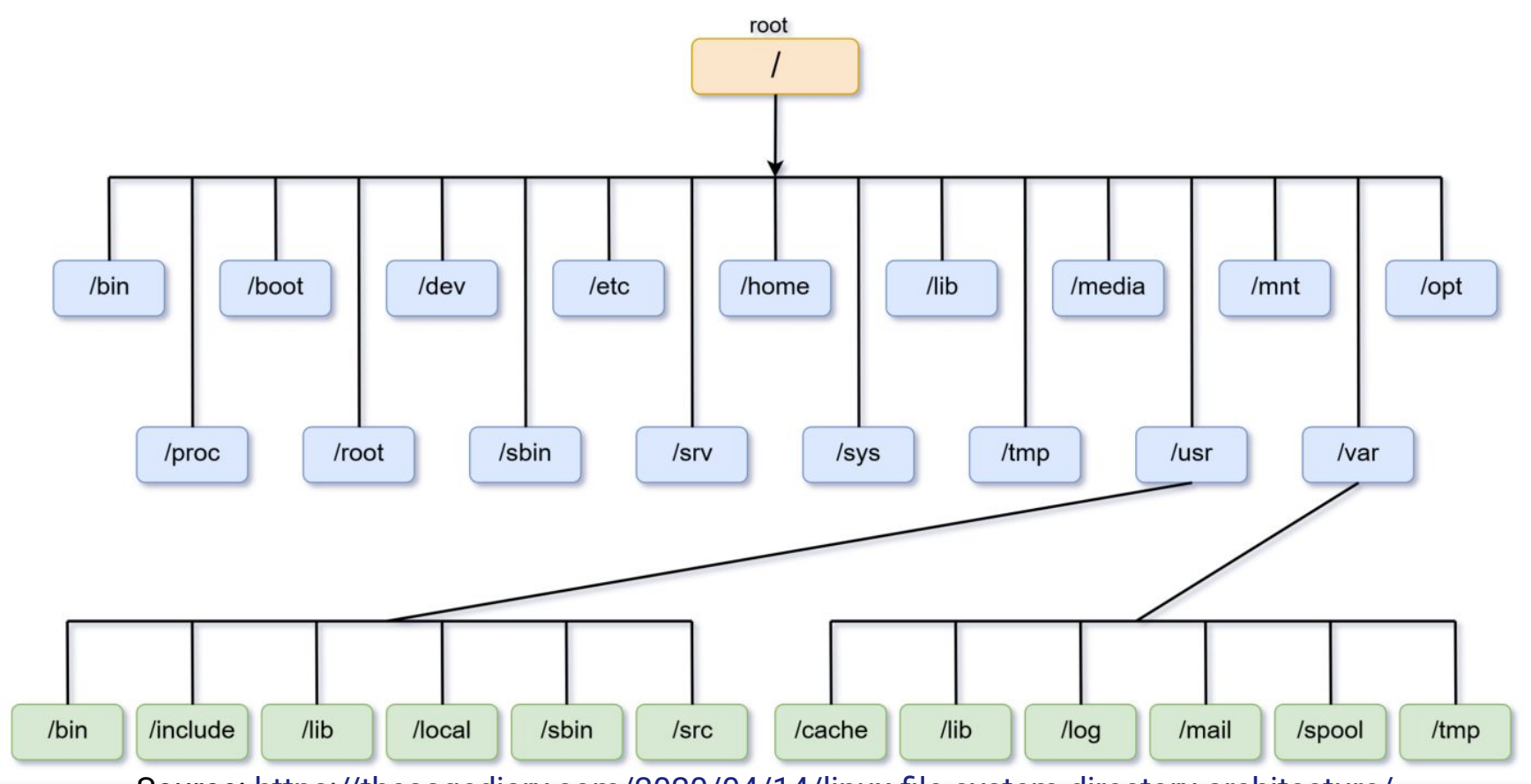
File System Architecture (cont.)

- UNIX philosophy: “Everything is a file”
  - Universality of I/O
    - One of the distinguishing features of the UNIX I/O model is the concept of universality of I/O. This means that the same four system calls—open(), read(), write(), and close()—are used to perform I/O on all types of files, including devices such as terminals.
    - A device special file corresponds to a device on the system. Within the kernel, each device type has a corresponding device driver, which handles all I/O requests for the device. A device driver is a unit of kernel code that implements a set of operations that (normally) correspond to input and output actions on an associated piece of hardware. The API provided by device drivers is fixed, and includes operations corresponding to the system calls open(), close(), read(), write(), mmap(), and ioctl().

Pathname

- Two kinds of path
  - Absolute path -> start from /
  - Relative path -> start from your current directory
- Constraints of pathname
  - Defined in include/uapi/linux/limits.h
  - NAME_MAX => 255
  - PATH_MAX => 4096

File System Directory Architecture



Source: https://thesagediary.com/2020/04/14/linux-file-system-directory-architecture/

man hier

Path Name	Description
/	Root directory
/bin	Contains executable programs which are needed in single user mode and to bring the system up or repair it
/boot	Contains static files for the boot loader. Holds only the files which are needed during the boot process
/dev	Special or device files, which refer to physical devices
/etc	Contains configuration files which are local to the machine
/home	Contains home directory for local users
/lib	Contains shared libraries that are necessary to boot the system and to run the commands in the root filesystem
/lost+found	Contains items lost in the fileystem. Usually chunks of files mangled as a consequence of a faulty disk or a system crash

man hier

Path Name	Description
/mnt	A mount point for a temporarily mounted filesystem.
/opt	Contains add-on packages that contain static files
/proc	The mount point for the proc filesystem, which provides information about running processes and the kernel
/root	The home directory for user root
/sbin	Contains commands needed to boot the system, but which are usually not executed by normal users
/sys	The mount point for the sysfs filesystem, which provides information about the kernel like /proc, but better structured, following the formalism of kobject infrastructure
/tmp	Contains temporary files which may be deleted with no notice
/usr	Contains only shareable, read-only data
/usr/local	This is where programs which are local to the site
/var	This directory contains files which may change in size, such as spool and log files.

Mounting file system

- mount
  - Map the mount point of the existing file tree to the root of the newly attached filesystem
  - `mount /dev/vdb1 /mnt`
  - Previous contents of the mount point become inaccessible
- Common types of file systems
  - Most are disk partitions
  - Network file servers
  - Memory disk emulators
  - Kernel components
  - ...etc

Mounting file system (cont.)

- umount
  - `umount { node | device }`
    - umount /mnt
    - umount /dev/vdc2
  - Busy file system
    - Someone’s current directory is there or there are opened files
    - `umount -f` to force unmount
    - Use `lsof` to check filesystem

umount - lsof

```
F74076310@F74076310:~$ lsof                                                                                    [1361/1397]
COMMAND    PID  TID TASKCMD              USER   FD      TYPE             DEVICE SIZE/OFF       NODE NAME
systemd      1                           root  cwd   unknown                                          /proc/1/cwd (rea
dlink: Permission denied)
systemd      1                           root  rtd   unknown                                          /proc/1/root (re
adlink: Permission denied)
systemd      1                           root  txt   unknown                                          /proc/1/exe (rea
dlink: Permission denied)
systemd      1                           root NOFD                                                      /proc/1/fd (open
dir: Permission denied)
kthreadd     2                           root  cwd   unknown                                          /proc/2/cwd (rea
dlink: Permission denied)
kthreadd     2                           root  rtd   unknown                                          /proc/2/root (re
adlink: Permission denied)
kthreadd     2                           root  txt   unknown                                          /proc/2/exe (rea
dlink: Permission denied)
kthreadd     2                           root NOFD                                                      /proc/2/fd (open
dir: Permission denied)
rcu_gp       3                           root  cwd   unknown                                          /proc/3/cwd (rea
dlink: Permission denied)
rcu_gp       3                           root  rtd   unknown                                          /proc/3/root (re
adlink: Permission denied)
rcu_gp       3                           root  txt   unknown                                          /proc/3/exe (rea
dlink: Permission denied)
```

Mounting file system (cont.)

- fstab
  - Filesystem Table
- Automatically mounted at boot time
- /etc/fstab
  - Filesystem in this file will be checked and mounted automatically at boot time

```
F74076310@F74076310:~$ cat /etc/fstab
LABEL=cloudimg-rootfs   /        ext4   defaults        0 1
LABEL=UEFI      /boot/efi       vfat    defaults        0 1
```

File Types

Symbol	File Types
-	Regular file
b	Block device file
c	Character device file
d	Directory
l	Symbolic link
s	UNIX domain socket
p	Named pipe

File Types (cont.)

- file
  - Determine file type
  - file /bin/bash
    - /bin/bash: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=a6cb40078351e05121d46daa768e271846d5cc54, for GNU/Linux 3.2.0, stripped
  - file /etc/fstab
    - /etc/fstab: ASCII text
  - file /dev/vda
    - /dev/vda: block special (252/0)

File Types (cont.)

- Directory
  - . and ..
  - mkdir / rmdir

File Types (cont.)

- UNIX domain socket
  - Created by socket()
  - Local to a particular host
  - Be referenced though a filesystem object rather than a network port

man socket

man7.org > Linux > man-pages Linux/UNIX system programming training

socket(2) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | RETURN VALUE | ERRORS | CONFORMING TO | NOTES | EXAMPLES | SEE ALSO | COLOPHON

Search online pages

```
SOCKET(2)                Linux Programmer's Manual                SOCKET(2)
```

NAME top

socket - create an endpoint for communication

SYNOPSIS top

```
#include <sys/types.h>          /* See NOTES */
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

DESCRIPTION top

socket() creates an endpoint for communication and returns a file descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

The *domain* argument specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in *<sys/socket.h>*. The formats currently understood by the Linux kernel include:

Name	Purpose	Man page
AF_UNIX	Local communication	unix(7)
AF_LOCAL	Synonym for **AF_UNIX**	
AF_INET	IPv4 Internet protocols	ip(7)
AF_AX25	Amateur radio AX.25 protocol	**ax25**(4)
AF_IPX	IPX - Novell protocols	
AF_APPLETALK	AppleTalk	ddp(7)
AF_X25	ITU-T X.25 / ISO-8208 protocol	x25(7)
AF_INET6	IPv6 Internet protocols	ipv6(7)
AF_DECnet	DECet protocol sockets	
AF_KEY	Key management protocol, originally developed for usage with IPsec	

File Types (cont.)

- Named Pipes
  - Let two processes do “FIFO” communication
  - mkfifo pipe

```
F74076310@F74076310:~$ ls -l > pipe
F74076310@F74076310:~$
```

```
F74076310@F74076310:~$ cat < pipe
total 0
-rw-rw-r-- 1 F74076310 F74076310 0 Mar 13 16:20 bar
-rw-rw-r-- 1 F74076310 F74076310 0 Mar 13 16:19 foo
prw-rw-r-- 1 F74076310 F74076310 0 Mar 13 17:58 pipe
F74076310@F74076310:~$
```

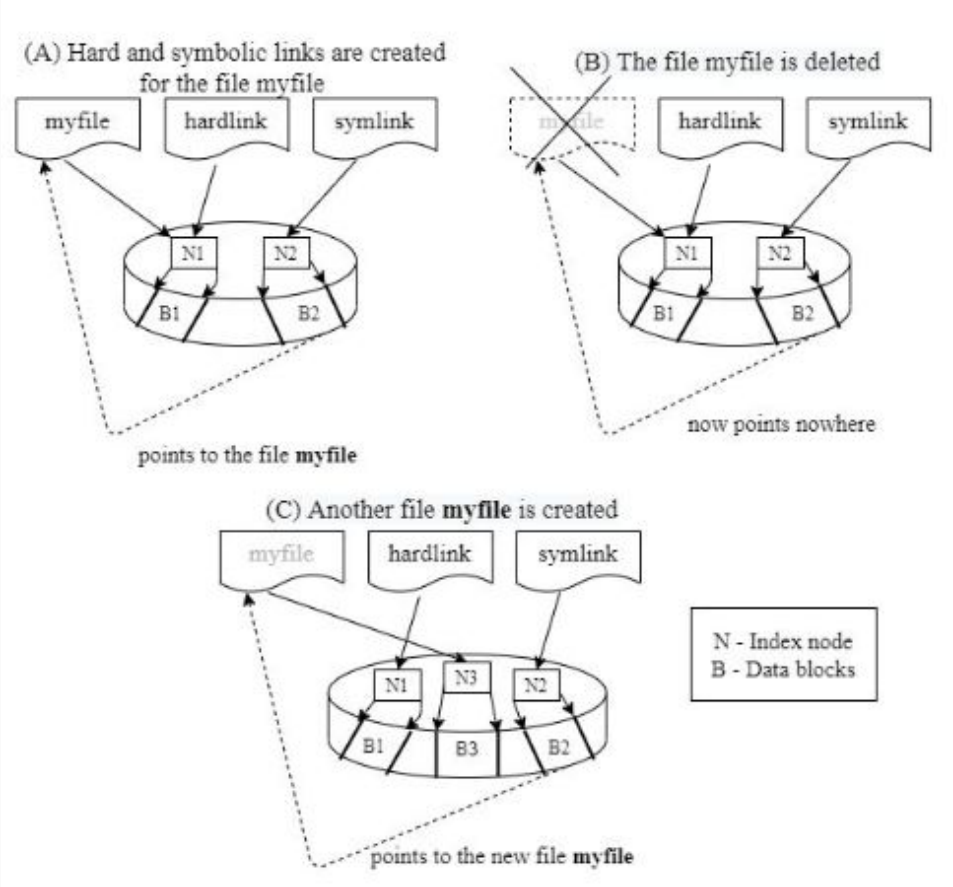
File Types (cont.)

- Symbolic Link
  - A file which points to another pathname
  - ln -s <ORIGINAL_FILE> <SOFT_FILE>
  - Like “shortcut”

Link - Hard Link & Symbolic Link

- Link
  - Hard Link
    - Associate two or more filenames with the same inode
      - Must in the same partition
    - ln <ORIGINAL_FILE> <HARD_FILE>
  - Symbolic (Soft) Link
    - A file which points to another pathname
    - ln -s <ORIGINAL_FILE> <SOFT_FILE>

Link - Hard Link & Symbolic Link

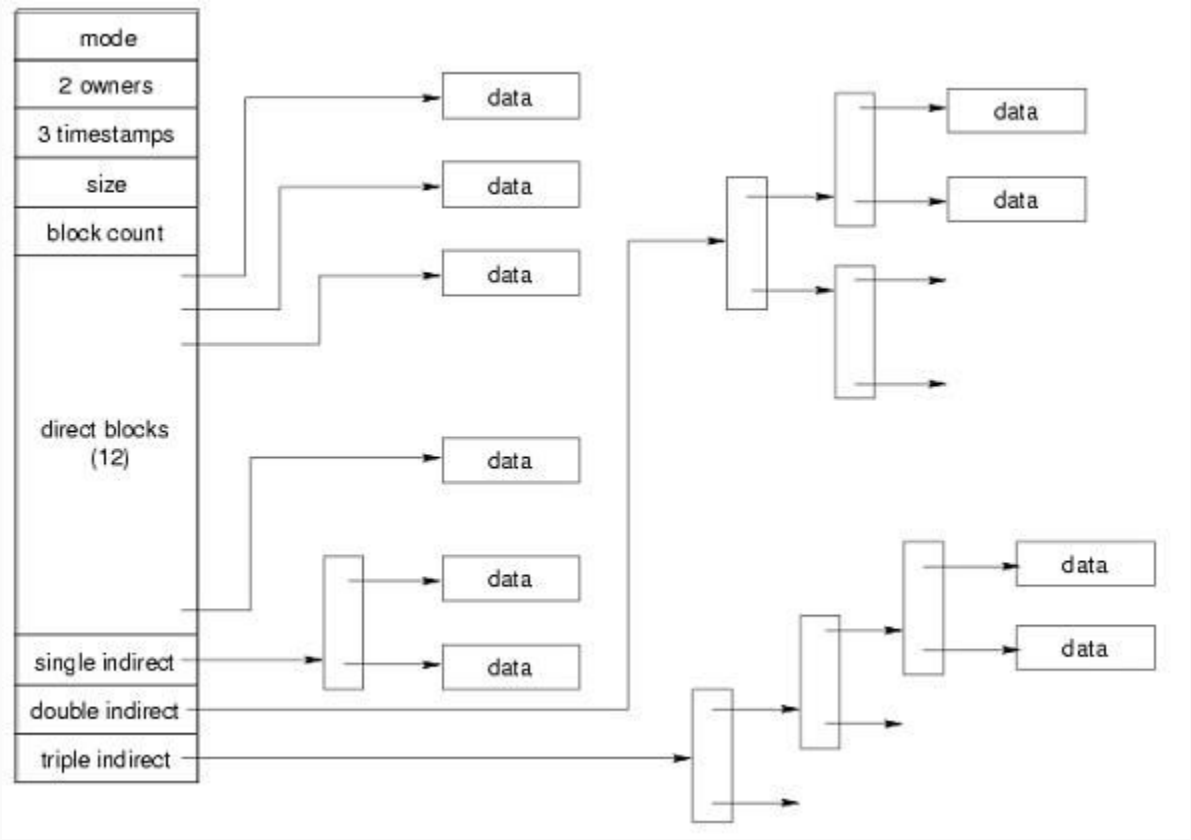


inode

- inode
  - A structure that records information of a file
    - Use `ls -i` to see each file's inode number

```
F74076310@F74076310:~$ ls -li
total 0
263418 -rw-rw-r-- 1 F74076310 F74076310 0 Mar 13 16:20 bar
259749 -rw-rw-r-- 1 F74076310 F74076310 0 Mar 13 16:19 foo
```

inode structure



Source: https://selvamvasu.wordpress.com/2014/08/01/inode-vs-ext4/

inode and filesystem

- Filesystem
  - Boot blocks
  - Superblocks
    - Total of block and inode
    - Unused/Used inode/block amount
    - Block / inode size
    - Filesystem-related information
      - Mount time
      - Last time data was written
      - Last time performed fsck
    - Valid bit
      - If mounted, 0
      - 1 othterwise
  - Inode list
  - Data block

```
F74076310@F74076310:~$ sudo mkfs.ext4 /dev/vdb1
mke2fs 1.45.5 (07-Jan-2020)
Discarding device blocks: done
Creating filesystem with 2096896 4k blocks and 524288 inodes
Filesystem UUID: f8af9b86-fcf2-4589-9242-c8921c8dabc5
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

inode - Directory / Data

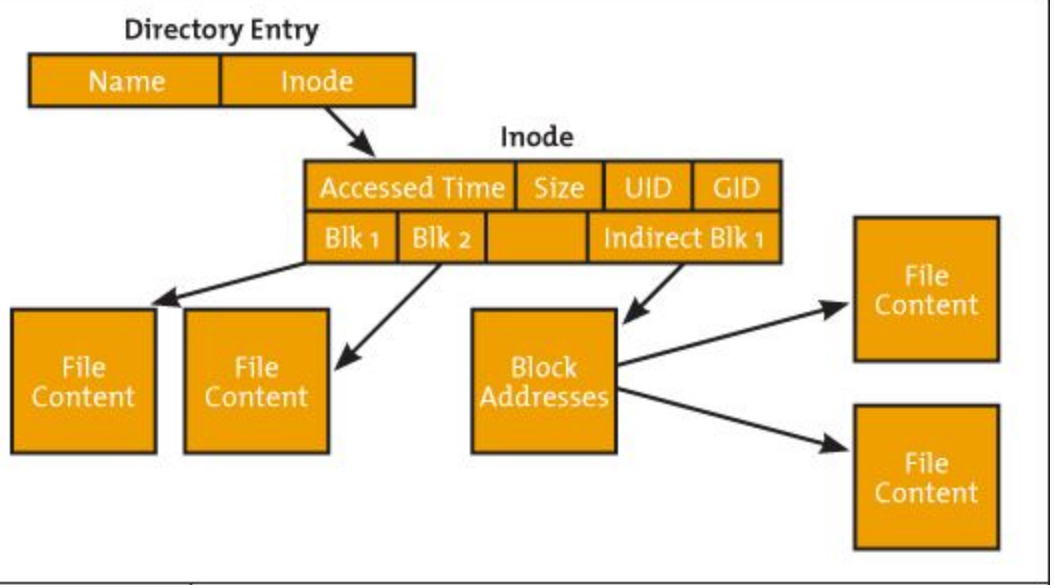


FIGURE 1 RELATIONSHIP BETWEEN THE DIRECTORY ENTRY, AN INODE, AND BLOCKS OF AN ALLOCATED FILE

Source: https://pc-freak.net/blog/find-filesystem-directory-eating-filesystem-inodes-linux/

File Access Mode

- setuid, setgid, sticky bit
  - SUID/SGID
    - The effective uid/gid of resulting process will be set to the UID/GID of the file
  - Sticky on directory (/tmp)
    - Do not allow to delete or rename a file unless you are
      - The owner of the file
      - The owner of the directory
      - root

File Access Mode (cont.)

- Decimal argument of chmod
  - setuid: 4000
  - setgid: 2000
  - sticky : 1000

Mode	Attribute	Mode	Attribute
755	- rwx r-x r-x	644	- rw- r-- r--
4755	- rws r-x r-x	600	- rw- --- ---
2755	- rwx r-s r-x	400	- r-- r-- r--
2775	d rwx rws r-x	1777	d rwx rwx rwt
755	d rwx r-x r-x	4555	- r-s r-x r-x
750	d rwx r-x ---	711	- rwx --x --x
700	d rwx --- ---	711	d rwx --x --x

File Access Mode (cont.)

- Assign default permissions: umask
  - Shell built-in command
  - Inference the default permissions given to the files newly created.
  - The newly created file permission:
  - Use full permission bit (file: 666, dir: 777) xor umask value.
  - Ex:

umask	New File	New Dir
022	- rw- r-- r--	d rwx r-x r-x
033	- rw- r-- r--	d rwx r-- r--
066	- rw- --- ---	d rwx --x --x
000	- rw- rw- rw-	d rwx rwx rwx
477	- r-- --- ---	d r-x --- ---
777	- --- --- ---	d --- --- ---

Journaling File System

- Write operational logs to the journal first, then commit it asynchronously
- If system crashed, check the log
  - Fully committed: skip
  - Partial committed: rollback or commit
  - Non-committed: Ignore or commit
- Reduce “fsck” time and data inconsistency
- Example
  - ext3 / ext4
  - xfs
  - btrfs

CoW (Copy on Write) File System

- If some data is copied but not modified, they will be referred to the same physical address in the storage
- Pro
  - Reduce the space used
- Con
  - Data inconsistency
  - Not “real” used space on file
- Example
  - ZFS deduplication

File Attribute Extension

- Associate files with metadata not interpreted by the filesystem
- Key-Value pairs, saved in the inode
- Example
  - mime_type
  - md5/sha1 checksum
  - security attributes

File Systems

ext4

- Extents
  - An extent is a range of contiguous physical blocks, improving large-file performance and reducing fragmentation.
- Presistent pre-allocation
  - By using function fallocate() , the allocated space would be guaranteed and likely contiguous
- Unlimited number of subdirectories
- Journal checksums
  - Improve filesystem reliability
- Lazy initialization
  - Allows to clean inode tables in the background, speeding initialization when creating new ext4 file system

xfs

- Direct I/O
  - Allows non-cached I/O operations to be applied directly to the userspace. Data is transferred between the buffer of the application and the disk using DMA
- Guaranteed-rate I/O
  - Provides an API allows applications to reserve bandwidth to the filesystem
- Online defragmentation
- Online resizing
  - xfs_growfs to perform online resizing of XFS filesystem

btrfs

- Online block device addition and removal
- Online balancing
- Online data scrubbing
- RAID 0, RAID 1 and RAID 10
- Subvolumes
- Compression via zlib, LZO and ZSTD
- Atomic writable or read-only Snapshots of subvolumes
- Send/Receive snapshots

ZFS

- Integrity checking for both data and metadata
- Continuous integrity verification and automatic “self-healing” repair
- Up to 256 trillion yobibytes
- Compression using LZ4, GZIP and ZSTD
- Hardware-accelerated native encryption
- Several levels of caching mechanism
- Snapshots and Copy-on-Write clones
- Send / Receive snapshots

Setup a ZFS storage pool | Ubuntu

CANONICAL

We are hiring Products

ubuntu Enterprise Developer Community Download Search

Tutorials Setup a ZFS storage pool

1 Overview

2 Installing ZFS

3 Creating a ZFS Pool

4 Checking Pool Status

5 Removing the pool

6 That's all!

1. Overview

ZFS is a combined file system and logical volume manager originally designed and implemented by a team at Sun Microsystems led by Jeff Bonwick and Matthew Ahrens. Features of ZFS include protection against data corruption, high storage capacity (256 ZiB), snapshots and copy-on-write clones and continuous integrity checking to name but a few. If you are dealing with large amounts of data, or providing a backing filesystem for virtualisation, ZFS is a great choice.

This guide will go through the process of installing ZFS on Ubuntu 16.04 LTS and setting up a storage pool.

What you'll learn

- How to install ZFS
- How to create a storage pool

What you'll need

- Ubuntu Server 16.04 LTS

Ready? Let's head over to the next step!

Originally authored by Aden Padilla.

Suggest changes › about 7 minutes to go