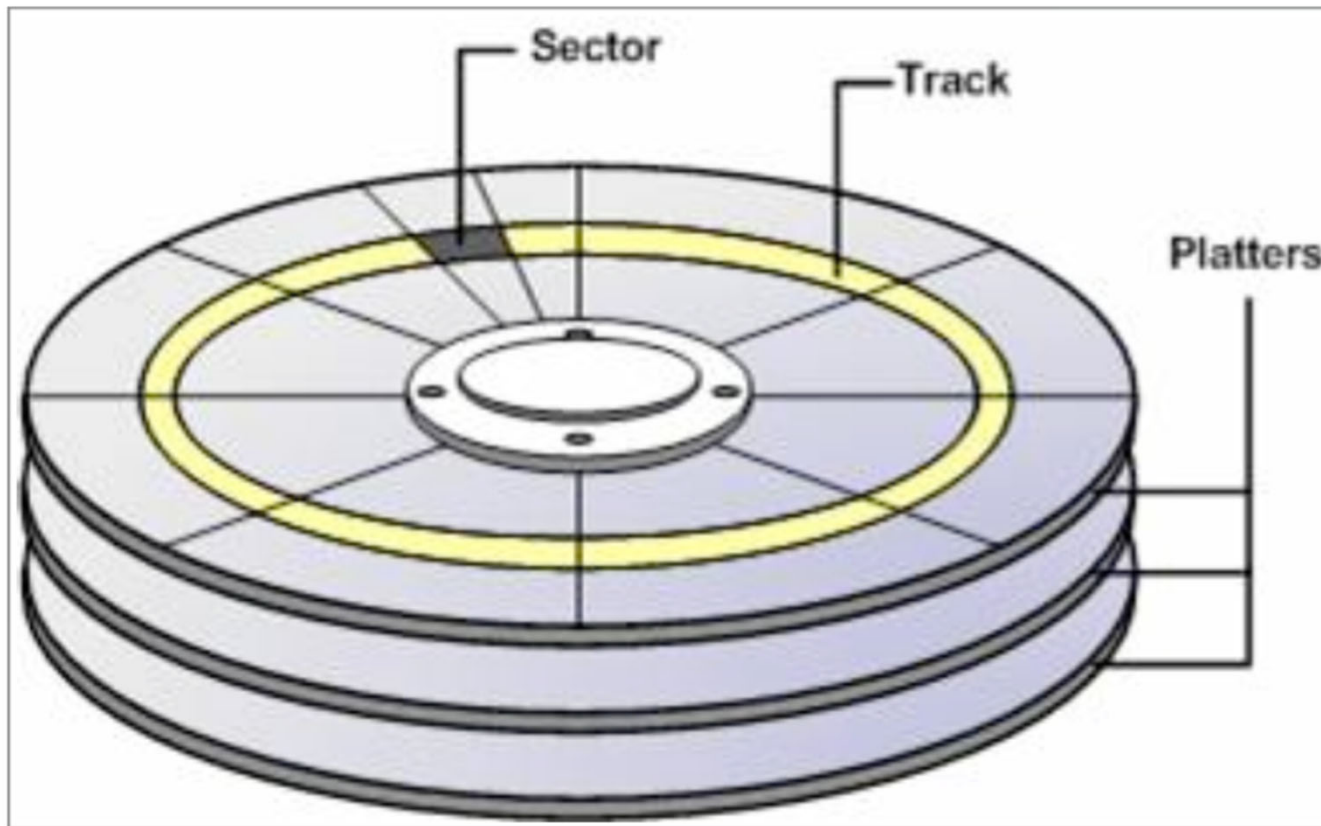


# Lecture 6

Disks – remember this diagram?



- Disks typically have surface defects resulting in unusable sectors.
- At the factory, the disk is tested and all bad sectors are identified.
- The disk then arranges to use alternate sectors for the bad locations.
- Some number of sectors are reserved for alternate sectoring.

- Once in operation a disk may develop sectors.
- If this happens the disk will normally adjust – however data/ file system may become corrupted.
- Systems normally provide a way to perform surface analysis of disk e.g. *format*.
  - Don't reformat new drives as a matter of course
- Such analysis can be destructive or non-destructive.

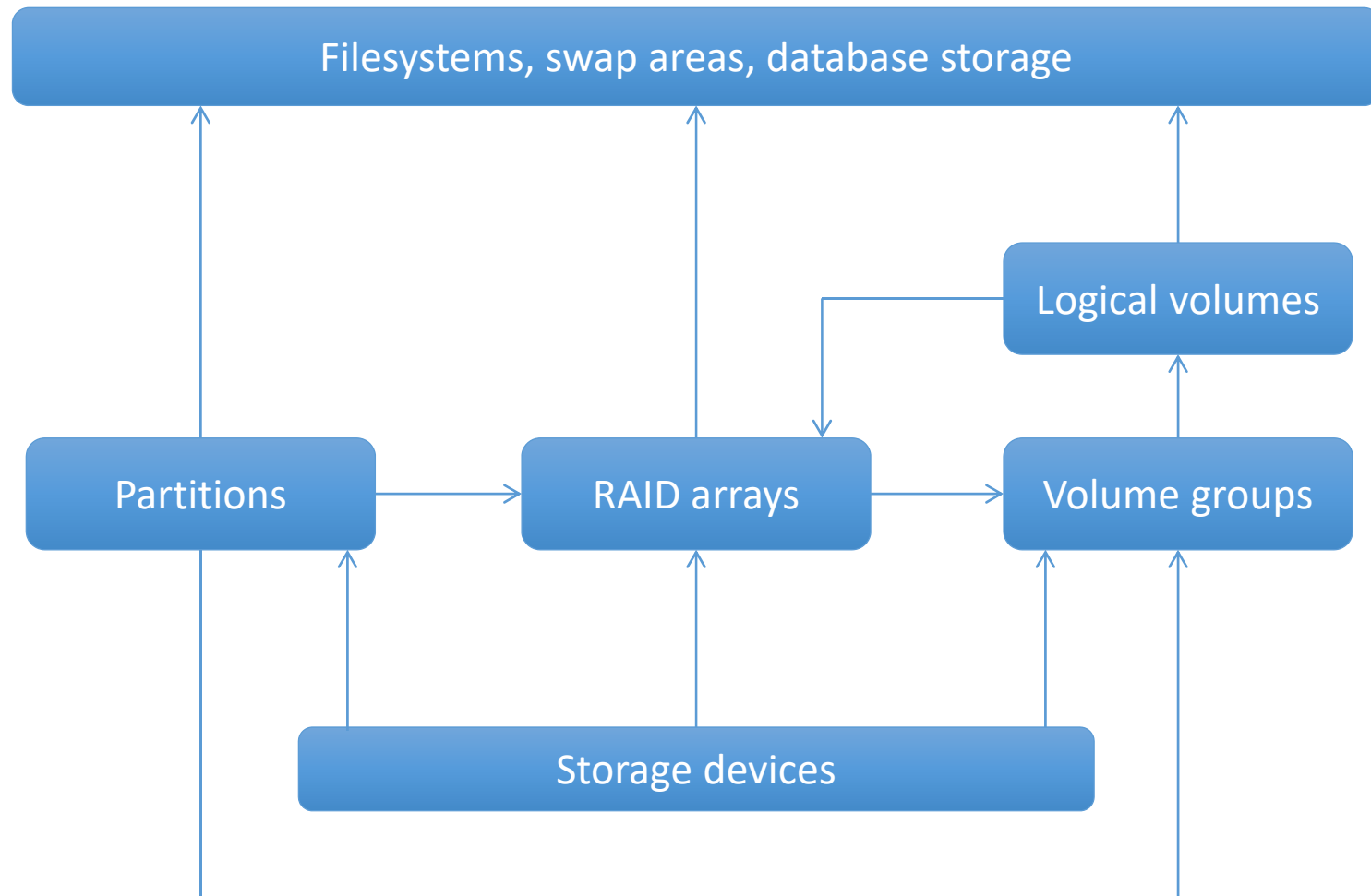
# Output of Solaris (UNIX) *format* command

## **FORMAT MENU:**

<b>disk</b>	- select a disk
<b>type</b>	- select (define) a disk type
<b>partition</b>	- select (define) a partition table
<b>current</b>	- describe the current disk
<b>format</b>	- format and analyze the disk
<b>repair</b>	- repair a defective sector
<b>show</b>	- translate a disk address
<b>label</b>	- write label to the disk
<b>analyze</b>	- surface analysis
<b>defect</b>	- defect list management
<b>backup</b>	- search for backup labels
<b>verify</b>	- read and display labels
<b>save</b>	- save new disk/partition definitions
<b>volname</b>	- set 8-character volume name
<b>!&lt;cmd&gt;</b>	- execute <cmd>, then return
<b>quit</b>	

# The software side of storage

- Software components mediate between the raw storage devices and the filesystem hierarchy
  - Device drivers
  - Partitioning conventions
  - RAID implementations
  - Logical volume managers
  - Systems for virtualising disks over a network
  - The filesystem implementations



- A storage device is anything that looks like a disk
  - A hard disk
  - A flash drive
  - An SSD
  - An external RAID array implemented in hardware
  - A network service that provides block-level access to a remote device
- As long as the device allows random access, handles block I/O, and represented by a device file.



- A partition is a fixed-size subsection of a storage device.
- Each partition has its own device file.
- Most partitioning schemes consume a few blocks at the start of the device to record the range of blocks that make up each partition.
- You probably are familiar with Windows-partitioned disks.

**Total disk cylinders available: 17660 + 2 (reserved cylinders)**

<b>Part</b>	<b>Tag</b>	<b>Flag</b>	<b>Cylinders</b>	<b>Size</b>	<b>Blocks</b>
0	root	wm	0 - 609	300.23MB (610/0/0)	614880
1	swap	wu	610 - 1650	512.37MB (1041/0/0)	1049328
2	backup	wm	0 - 17659	8.49GB 17660/0/0)	17801280
3	usr	wm	1651 - 4698	1.47GB (3048/0/0)	3072384
4	unassigned	wm	4699 - 5308	300.23MB (610/0/0)	614880
5	unassigned	wm	5309 - 6324	500.06MB (1016/0/0)	1024128
6	unassigned	wm	6325 - 6731	200.32MB (407/0/0)	410256
7	unassigned	wm	6732 - 17658	5.25GB (10927/0/0)	11014416

**partition>**

Above is the output from the Solaris *format* command during a *partition* process

# Linux partitions

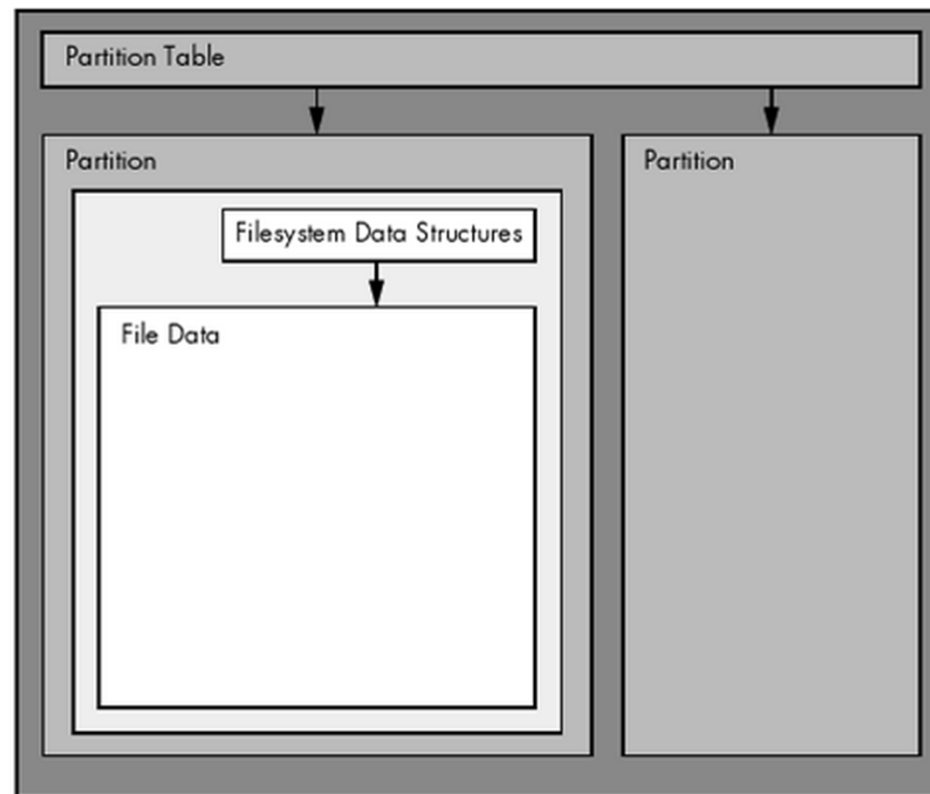


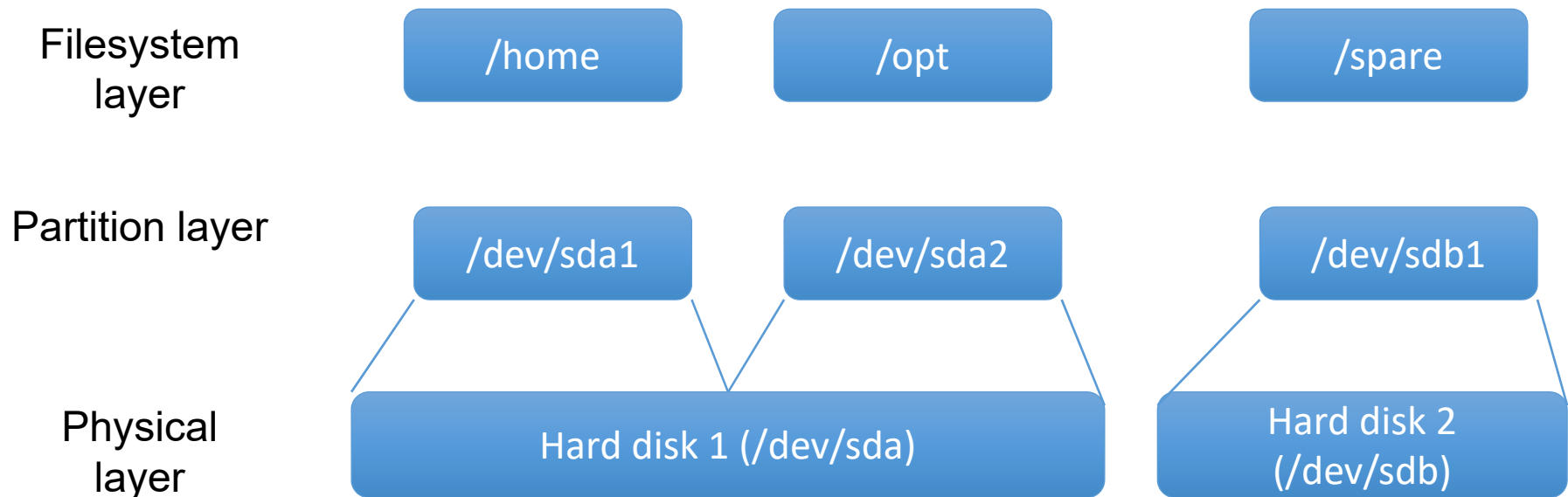
Figure 4-1: Typical linux disk schematic

- Partitions are useful to keep data types separate:
  - Operating system partition (don't want user data corrupting OS)
  - User data partition (don't want this to change when OS being patched)
  - Swap partitions – extend memory capability
  - Backup partitions
- Its possible to change partitions – however must be done with care
  - Its not easy to retrieve data that has been over written because of change of partition
  - Don't alter partitions that are currently in use

- A RAID array (a redundant array of inexpensive/independent disks) combines multiple storage devices into one virtualised device.
- This is to
  - Increase performance
  - Increase reliability
  - Or both
- RAID is typically conceived of as an aggregation of bare drives
- Modern implementations let you use as a component of a RAID array anything that acts like a disk

- Volume groups and logical volumes are associated with logical volume managers (LVMs)
- Physical devices are aggregated to form volume groups
- Volume groups can then be subdivided into logical volumes.
  - Very similar to disk partition
- For example, a 750GB disk and a 250GB disk could be aggregated into a 1TB volume group and then split into two 500GB logical volumes

# Traditional data disk partitioning scheme



# Disk devices on Linux

- Disk device files are shown as */dev/sda* etc.
  - sda: first registered device
  - sda1: first partition of sda
- Disk names are assigned in sequence as the kernel enumerates the various interfaces and devices on the system
- Subdirectories under */dev/disk* list disks by various stable characteristics such as their manufacturer ID
- *parted -l* lists the sizes, partition tables, model numbers, and manufacturers of every disk on the system



- Other OS use different naming conventions for device files
- For example, Solaris use `/dev/dsk/cWtXdYsZ`
  - W is the controller number
  - X is the SCSI target number
  - Y is the SCSI logical unit number (almost always 0)
  - Z is the partition number
  - You see something like `/dev/dsk/c2t1d0s1`

```
UbuntuServer1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
abc123@ubuntuserverseu:~$ sudo parted -l
[sudo] password for abc123:
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sda: 10.7GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name  Flags
  1      1049kB  2097kB  1049kB             bios_grub
  2      2097kB  10.7GB  10.7GB  ext4

abc123@ubuntuserverseu:~$
```

# Root partition

- All systems have a root partition that includes / and most of the local host's configuration data
- Everything needed to bring the system up to single-user mode is part of the root partition
- Various subdirectories (/var, /usr, /tmp, /home) may be broken out into their own partitions or volumes
- Most systems also have at least one swap area

# Some general partitioning points

- Have a backup root device that you can boot to
- Put /tmp on a separate filesystem to limit temporary files to a finite size
- It's a good idea for /var to be a separate disk partition
- Put users' home directories on a separate partition or volume.
- Split swap space among several physical disks
- Backups of a partition may be simplified if the entire partition can fit on one piece of media

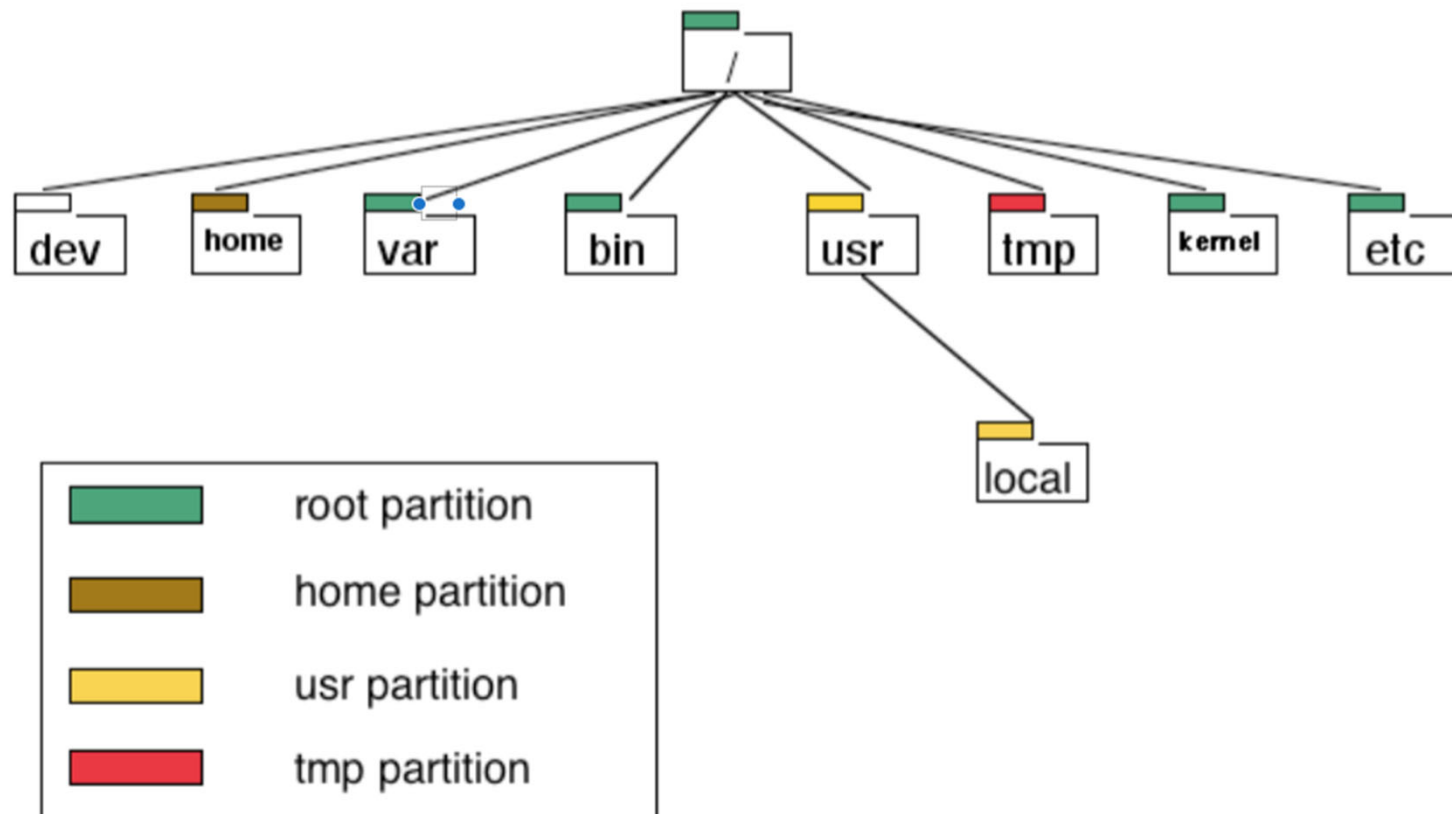
# Linux partitioning

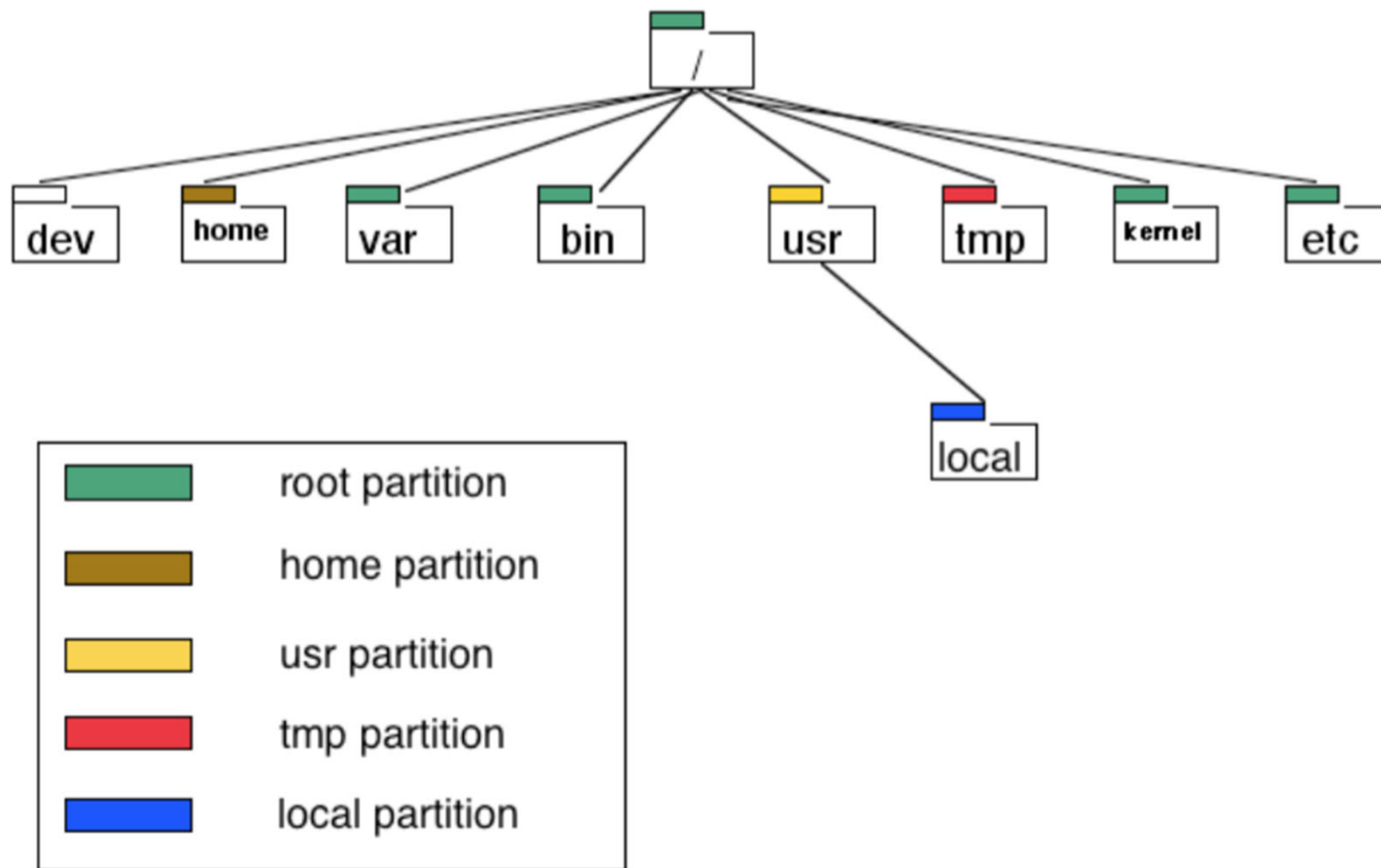
- The *fdisk* command is a basic command-line partitioning tool but it is not designed for large partitions
- The *parted* command is a fancier command-line tool that can move and resize partitions in addition to simply creating and deleting them
- A GUI version of *parted* is *gparted* which lets you specify the size of the partition
- You can also use *cedisk*

# I just want to add a disk (Linux recipe)

- Attach the disk and reboot
- Run *sudo fdisk -l* to list the system's disks and identify the new drive
- Run *fdisk* to add new partitions (you can also use *parted*)
- Format the newly created partitions using *mkfs* command, e.g.,
  - *mkfs -t ext4 /dev/sda4*
- Mount the filesystem (need to create mount point)
- Set mount options by editing */etc/fstab* (mount the filesystem at startup)

A sample filesystem in reference to the partitioning scheme on a disk





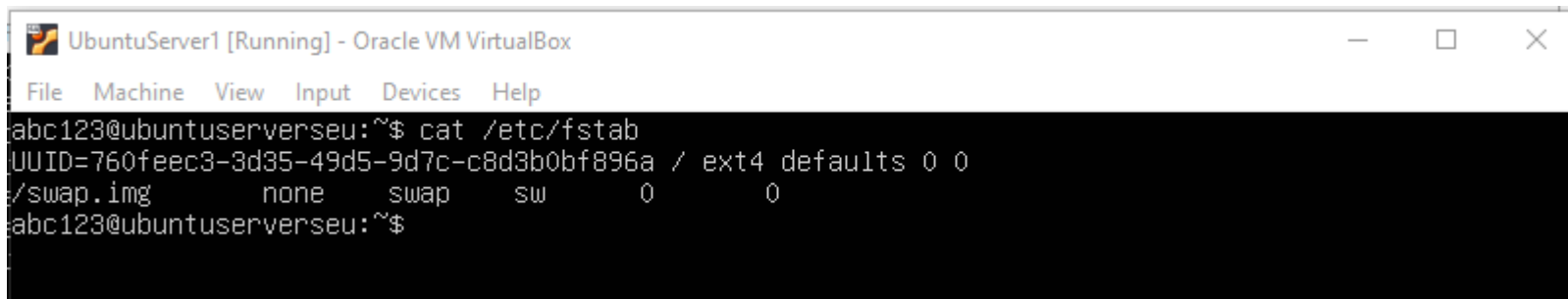
- Lets say `/usr/local` is full. We can simply purchase another disk, create a file system on it and mount it as `/usr/local`



- The mount command can take several options. In order to mount a filesystem, you must know the following:
  - The filesystem's device (such as a disk partition; where the actual filesystem data resides).
  - The filesystem type.
  - The mount point —that is, the place in the current system's directory hierarchy where the file system will be attached.
- I can even mount NFS file systems or even FTP services into the file system map.
  - `mount wraith:/open /mnt`
  - `mount_ftp ftp.seu.edu.cn /mnt`

# Mounting files systems on startup

- A list of file systems that are mounted on a particular system at startup are kept in fstab file:
  - Solaris `/etc/vfstab`
  - BSD/Linux `/etc/fstab`
- Building further from earlier LINUX partitioning and formatting exercise updating the `/etc/fstab` file.
- Open `/etc/fstab` file, enter:
  - `$ vi /etc/fstab`



```
UbuntuServer1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
abc123@ubuntuserverseu:~$ cat /etc/fstab
UUID=760feec3-3d35-49d5-9d7c-c8d3b0bf896a / ext4 defaults 0 0
/swap.img none swap sw 0 0
abc123@ubuntuserverseu:~$
```

- Append the following line to fstab file.
  - /dev/sdb1                /disk1                ext4   defaults        1 2
- Each line corresponds to one filesystem, each of which is broken into six fields.
  - The device or UUID. Most current Linux systems no longer use the device in /etc/fstab , preferring the UUID.
  - The mount point Indicates where to attach the filesystem.
  - The filesystem type
  - Options – default: This uses the mount defaults: read-write mode, enable device files, executables, the setuid bit, and so on
  - Backup information for use by the dump command.
  - The filesystem integrity test order

# Examples

/dev/hda1	/	ext2	defaults	1 1
/dev/hdb2	/home	ext2	defaults	1 2
/dev/cdrom	/media/cd	auto	ro,noauto,user,exec	0 0

---

```
proc /proc proc nodev,noexec,nosuid 0 0
UUID=70ccd6e7-6ae6-44f6-812c-51aab8036d29 / ext4 errors=remount-ro 0 1
UUID=592dcfd1-58da-4769-9ea8-5f412a896980 none swap sw 0 0
/dev/sr0 /cdrom iso9660 ro,user,nosuid,noauto 0 0
```

---

# Universally Unique Identifier (UUID)

- The method of mounting filesystems discussed in the preceding slides depends on device names. However, device names can change because they depend on the order in which the kernel finds the devices. To solve this problem, you can identify and mount file systems by their Universally Unique Identifier (UUID), a software standard. The UUID is a type of serial number, and each one should be different. File system creation programs like *mke2fs* generate a UUID identifier when initializing the file system data structure.

# More on Options List:

- **defaults** This uses the mount defaults: read-write mode, enable device files, executables, the setuid bit, and so on. Use this when you don't want to give the filesystem any special options but you do want to fill all fields in `/etc/fstab` .
- **errors** This ext2-specific parameter sets the kernel behavior when the system has trouble mounting a filesystem. The default is normally `errors=continue` , meaning that the kernel should return an error code and keep running. To have the kernel try the mount again in read-only mode, use `errors=remount-ro` . The `errors=panic` setting tells the kernel (and your system) to halt when there is a problem with the mount.
- **noauto** This option tells a `mount -a` command to ignore the entry. Use this to prevent a boot-time mount of a removable-media device, such as a CD-ROM or floppy drive.
- **user** This option allows unprivileged users to run `mount` on a particular entry, which can be handy for allowing access to CD-ROM drives

# RAID

- The idea is to use a cluster of small disks as a group to simulate a larger device and at the same time provide a mechanism whereby a single disk failure did not result in the data becoming unavailable.
- Hardware implementation has been predominate in the past but things are changing.

# RAID

- RAID can improve performance by striping data across multiple drives, thus allowing several drives to work simultaneously to supply or absorb a single data stream
- RAID can replicate data across multiple drives, decreasing the risk associated with a single failed disk
  - Mirroring: data blocks are reproduced “bit-by-bit”
  - Parity scheme: one of more drives contain an error-correcting checksum of the blocks on the remaining data drives

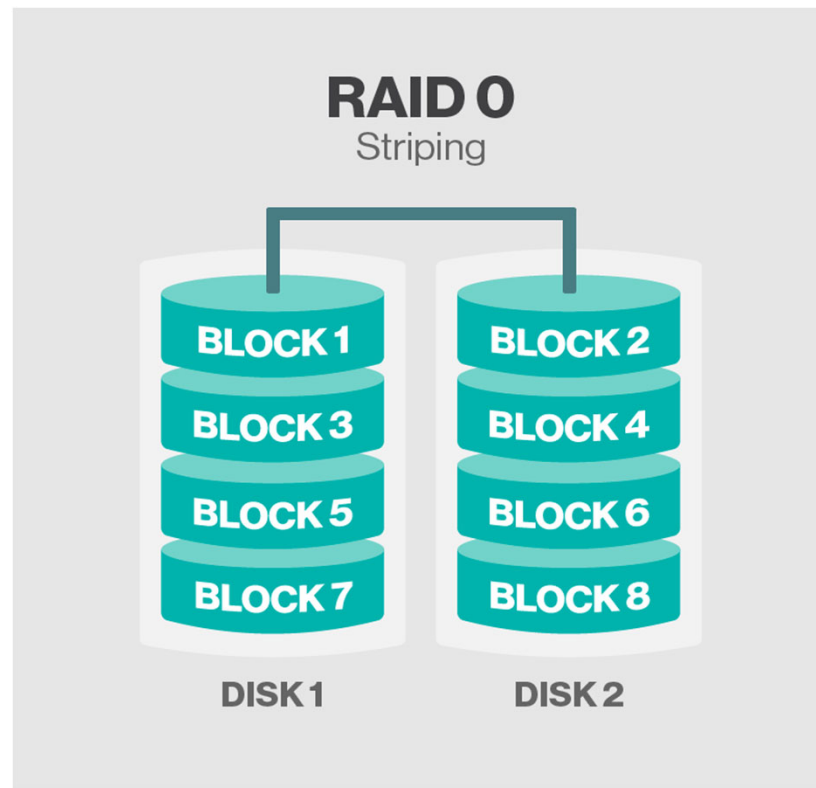


# RAID levels

- 0 striped disk (no redundancy)
- 1 mirrored disk
- 0+1 mirrored and striped (often called 10)
- 2 minor variation of 3 not used
- 3 striping by byte/word with parity
- 4 striping by block with parity drive
- 5 striping by block with rotating parity
- 6 striping by block with rotating parity in two parity disks

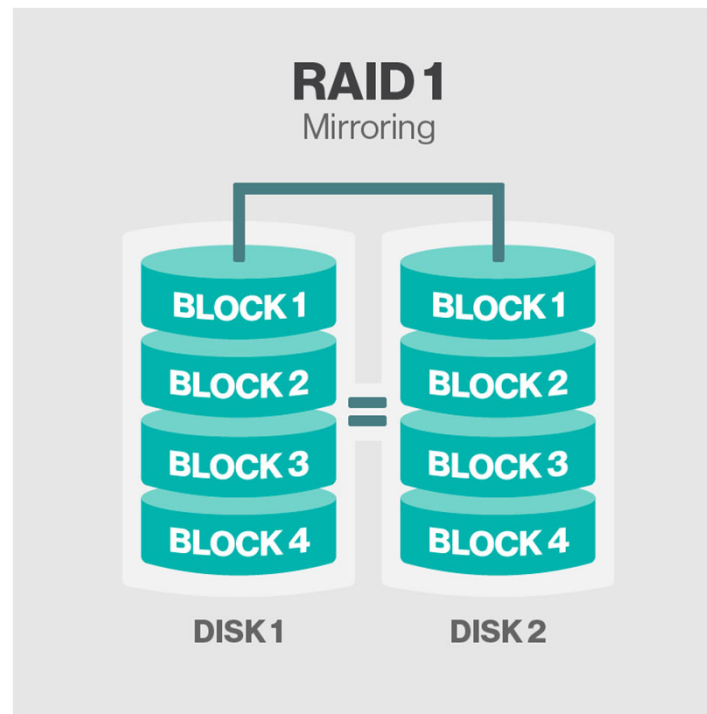
# RAID 0

- This configuration has striping but no redundancy of data. It offers the best performance but no fault-tolerance.



# RAID 1

- Also known as *disk mirroring*, this configuration consists of at least two drives that duplicate the storage of data. There is no striping. Read performance is improved since either disk can be read at the same time. Write performance is the same as for single disk storage.

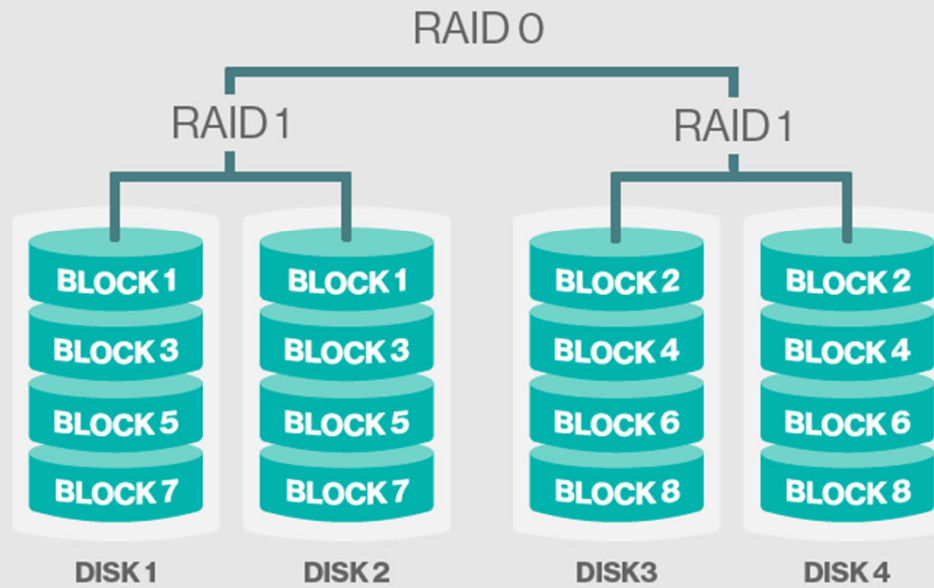


# Nested RAID Arrays

- Some RAID levels are referred to as *nested RAID* because they are based on a combination of RAID levels. Here are some examples of nested RAID level.
- RAID 10 (RAID 1+0): Combining RAID 1 and RAID 0, this level is often referred to as RAID 10, which offers higher performance than RAID 1 but at a much higher cost. In RAID 1+0, the data is mirrored and the mirrors are striped.

## RAID 10 (RAID 1+0)

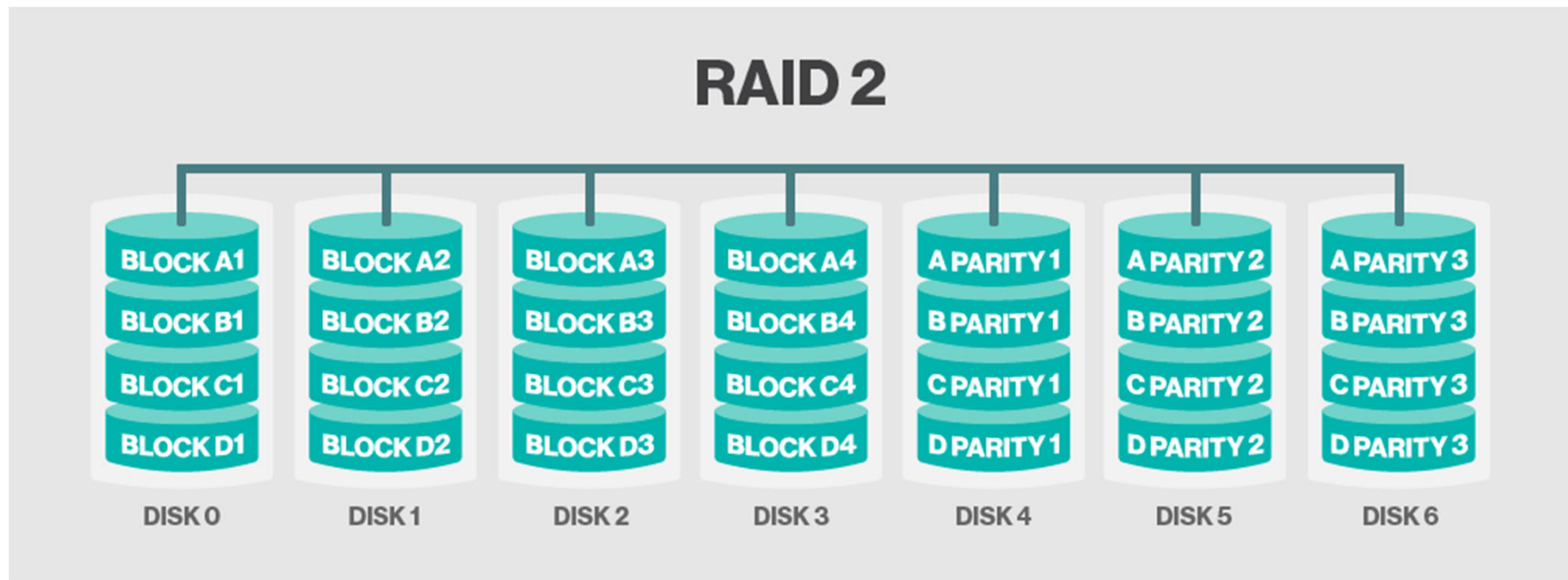
Stripe + Mirror



# ECC – Error Checking and Correcting

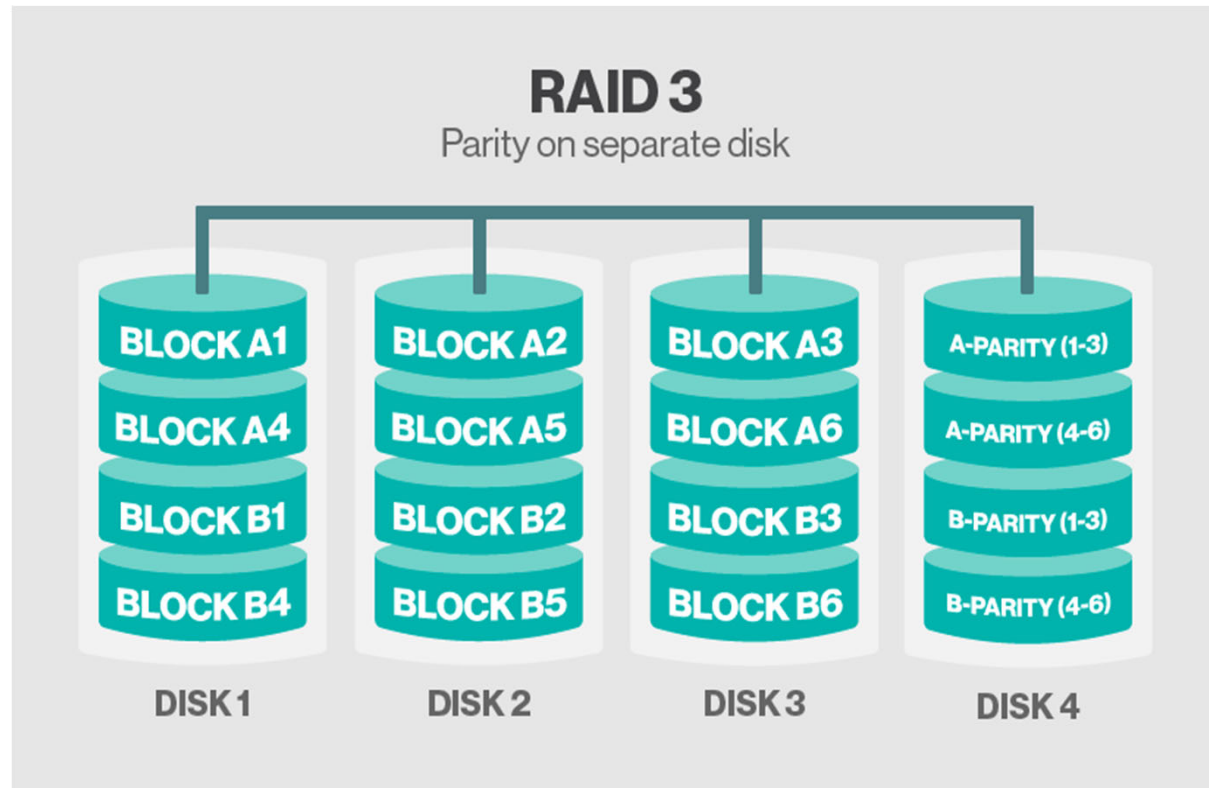
- Raid 2-6 include ECC techniques to provide additional data redundancy measures .
- Parity data is used by some RAID levels to achieve redundancy. A certain checksum is calculated which then is written to a disk. If a drive in the array fails, remaining data on the other drives can be combined with the parity data (e.g. using the Boolean XOR function or Reed Solomon code) to reconstruct the missing data.

# RAID 2 (no longer used)



# RAID 3

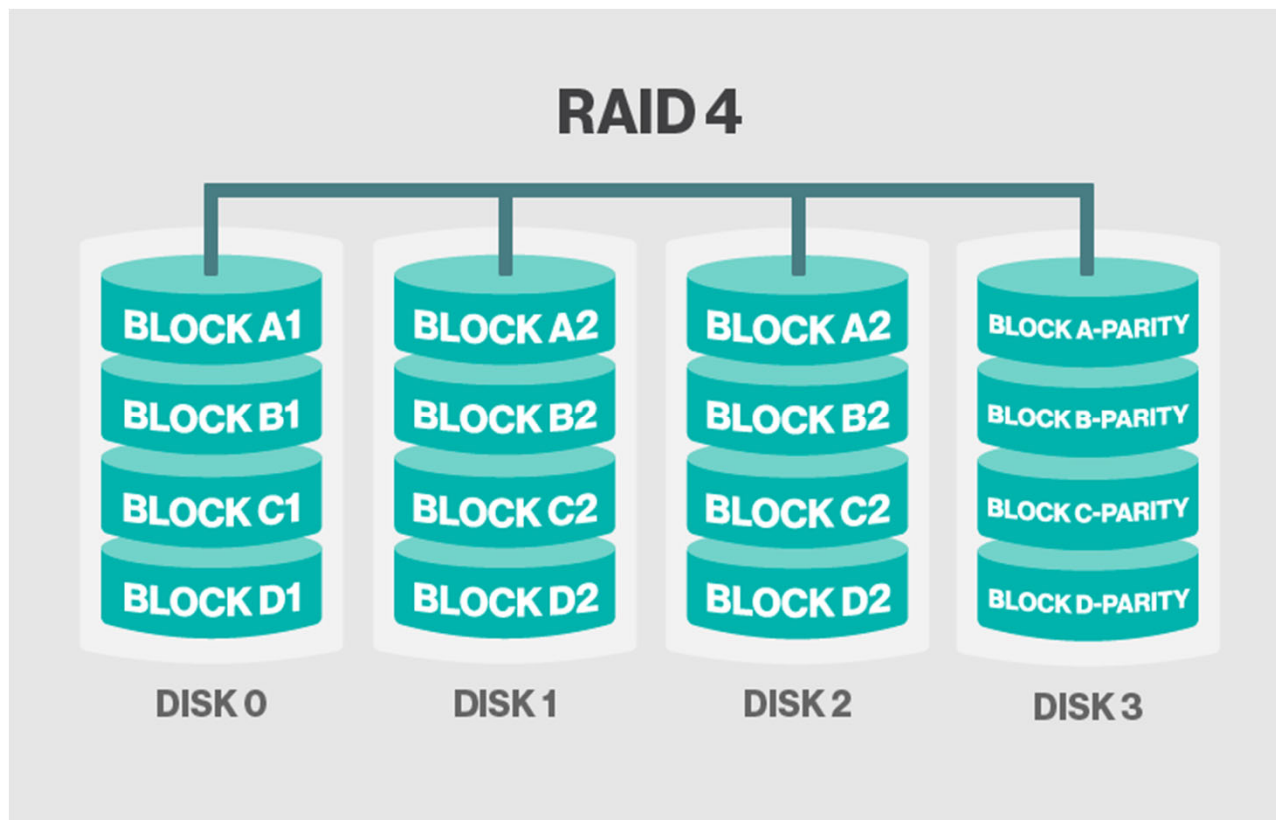
- RAID 3 is best for single-user systems with long record applications, such as streaming media, graphics and video editing.





# RAID 4

- RAID 4 offers no advantage over RAID 5



# RAID 4

- Imagine you have 9 disk drives.
- Your logical disk block size is 8K.
- Each logical disk block comprises 1K from each of the first 8 disk drives.
- When an 8K block is written, the first 1K goes to disk1, the second 1K goes to disk 2 and so on.
- To read back an 8K block, 1K is read from each disk drive and the 1K blocks concatenated to form the original 8K block.

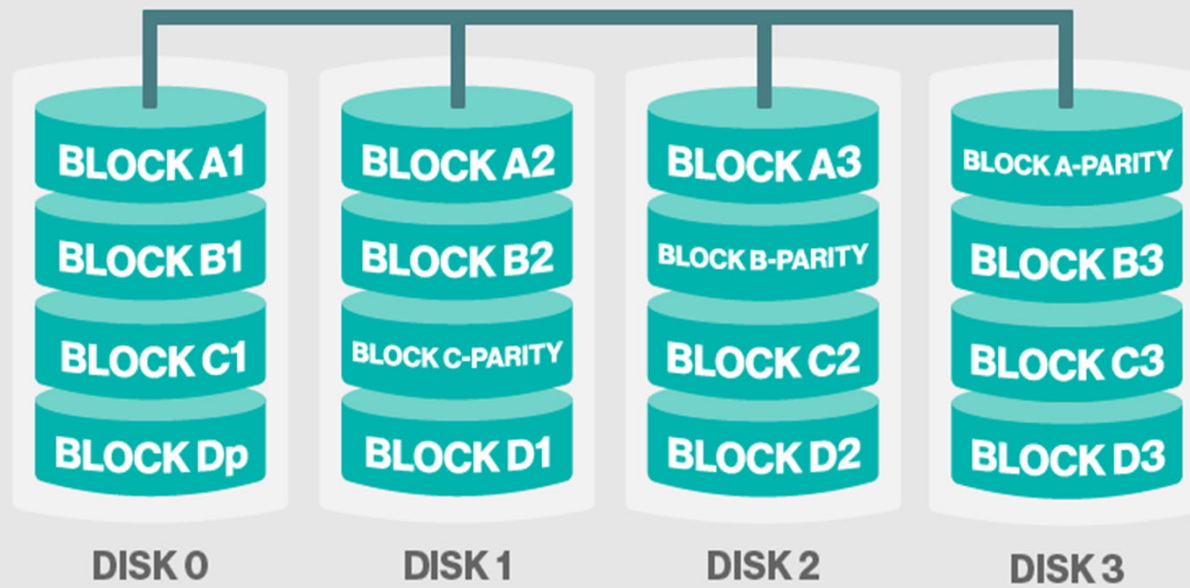
- Reading and writing to each of the 8 drives runs in parallel, so the time to write or read a block (8K) is improved.
- To protect against a single drive failure, the 9th drive is used to write a parity block of 1K as well.
- The parity block is not needed in the normal case.
- However, if one of the first 8 drives fails, the data from the other 7, plus the parity information can be used to reconstruct the original 8K block.

- This gives protection against a single drive failure.
- If 2 or more drives fail there is no protection unless more parity information is written.
- To protect against a single disk failure in the above example, 11% of the total storage was used for parity and therefore not available for normal use.
- This idea can be expanded to any number of disk drives.
- Some storage is always lost to parity information. The amount lost depends on the configuration and RAID level used. The more disks in the RAID group, the less capacity is lost to parity.
- Parity blocks are constructed by XORing the contents of all the other blocks. This operation is expensive and tends to make RAID writes slow.

# RAID 5

- RAID 5 is identical to RAID 4, except the parity information is spread over all the disk drives rather than just one.
- This is done to even out the I/O load on reads.
- When a drive fails, the system will continue to function normally, perhaps losing some performance.
- The failed drive can be replaced and the RAID system will re-create the lost data and return to normal operation.

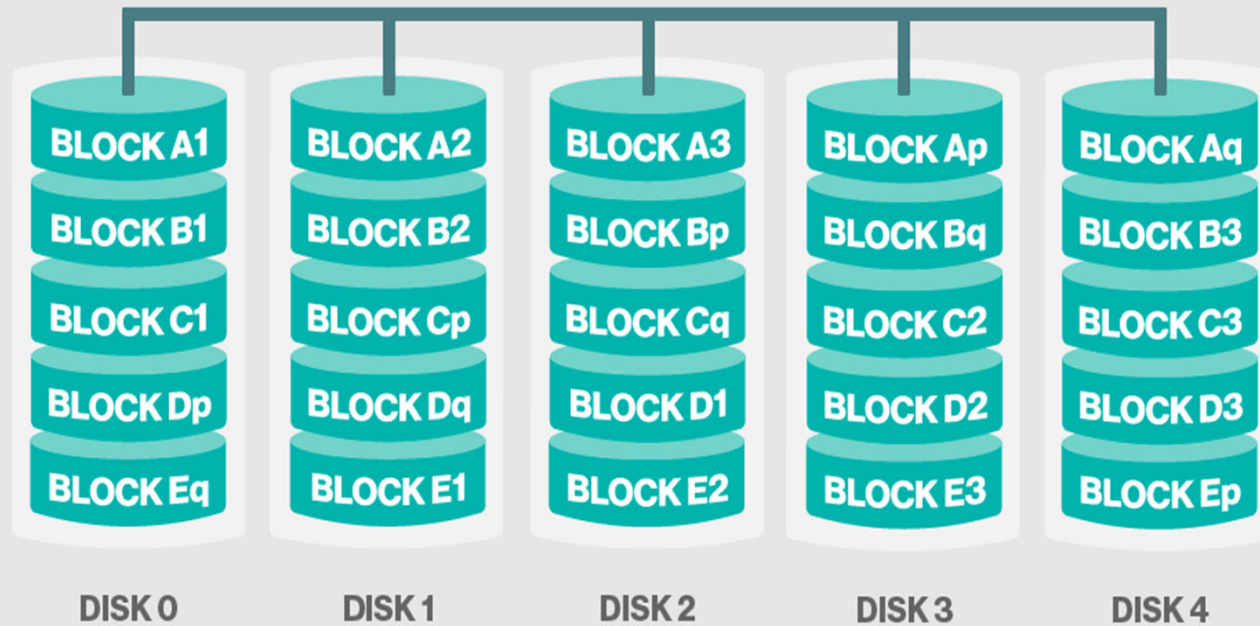
## RAID 5



# RAID 6

- This technique is similar to RAID 5 but includes a second parity scheme (in addition to Exclusive NOR) that is distributed across the drives in the array. The use of additional parity allows the array to continue to function even if two disks fail simultaneously. However, RAID 6 arrays have a higher cost per gigabyte (GB) and often have slower write performance than RAID 5 arrays

## RAID 6





- It is common to be using different RAID levels on different volumes in the same RAID environment, since different software systems have differing performance and reliability requirements..
  - Home directories may need reliability but not high performance, so use RAID 5 or 4.
  - Database system may need high read performance, so use 4 way RAID 1 (mirroring).

- In any environment where data is important or uptime is important, you should consider using RAID systems for your disk storage.
- Downtime due to failed disk can be very large.
- Restoring filesystems from tape is very slow.

# Logical volume management

- Logical volumes are more flexible and powerful than disk partitions
  - Move logical volumes among different physical devices
  - Grow and shrink logical volumes on the fly
  - Take copy-on-write “snapshots” of logical volumes
  - Replace on-line drives without interrupting service
  - Incorporate mirroring or striping in your logical volumes

# LVM implementation in Linux:

## LVM2

- You can use either a group of simple commands or the *lvm* command and its subcommands.
- Physical volume is a storage device that has had an LVM label applied
- Can be disks, disk partitions or RAID arrays
  - Create: *pvcreate*
  - Inspect: *pvdisplay*
  - Modify: *pvchange*
  - Check: *pvck*

- Volume group
  - Create: *vgcreate*
  - Modify: *vgchange*
  - Extend: *vgextend*
  - Inspect: *vgdisplay*
  - Check: *vgck*
  - Enable: *vgscan*
- Logical volume
  - Create: *lvcreate*
  - Modify: *lvchange*
  - Resize: *lvresize*
  - Inspect: *lvdisplay*

# LVM examples

- A simple illustration can be found at <https://www.thegeekstuff.com/2010/08/how-to-create-lvm>
- A more detailed introduction can be found at [https://www.howtoforge.com/linux\\_lvm](https://www.howtoforge.com/linux_lvm)

# Linux filesystems: the ext family


- The UNIX File System (UFS) formed the basis of several filesystem implementations
- UNIX kernel allows multiple types of filesystems to be active at once.
- In most cases, stick with the system's defaults.
- ext2 is functionally similar to UFS
- ext3 adds journaling capability to the existing ext2 code
- ext4 increase the performance and allows the use of “extents” (default on Ubuntu)

# Design of File Systems

- When building computer systems with storage we need to understand the structure and interactions of the file system. We also need to understand how users will use it as well. Here are a few rules which are useful for designing file systems that are scalable and more importantly manageable.
- Isolate the Operating System
  - Ensure that the file systems used to contain the operating system are separate from the file system used for other purposes.
  - This minimizes the problems when new OS releases are installed since the administrator knows that other file systems can be safely preserved across OS installs.



- Ensure areas written by applications or users are not on vital partitions.
  - Make sure areas such as home directories, log and spool areas are not in the OS partitions i.e. root.
  - This prevents some problems with the file system filling.
- Group Files used for like purposes.
  - Use separate partitions for home directories, software packages, logs and any other functional grouping you consider reasonable.
  - This makes designing backup strategies simpler and prevents problems in one area affecting others.
  - Different creation of mount options can be used on each file system to increase performance or security in way that make sense for the different use areas.

- Design for backups.
  - Do not create file systems larger than can be reasonably fit on tapes.
  - Allow for differing backup strategies of schedules between file systems.  e.g. areas used for software can be backed up less frequently than those containing home directories
- Design of File Systems
  - Try for flexibility - keep some disk up your sleeves. This can be difficult.
  - When change is needed, you can use the additional space without a major re-organisation.

- Spread Disk load.
  - This improves performance due to overlapped read/writes/seek.
  - Allocate busy file systems to separate disks and controllers where possible.
  - Place busy file systems in the middle of the drive. Busy file systems should be adjacent to one another.
- Use raw partitions for swap.
  - Don't skimp on memory - you do not want swap being over utilised.
  - In the event you can't have that luxury - do not skimp on the size of swap.
  - Monitor memory and swap usage e.g. *vmstat*.
  - Scan rate is the most important. It is the number of candidate pages being scanned per second as candidates are being freed.

# Server redundancy

- Remember that RAID protects against disk drive failure – not software or human error or failure of other components.
- Power supplies – there is much value in have two parallel power supplies. This means that the server should still be able to run if one power supply fails; this is called n+1 redundancy
- This means that is two power supplies are required to power the device you will need three power supplies.
- Each power supply should have a separate power chord
  - its possible that a single power chord can be accidentally disconnected;
  - very useful if you need to move a chord to a new power strip.
- If high availability is the order of the day it makes sense to draw power from two different UPS (Uninterruptable Power Supplies)

# Full versus n+1 redundancy

- N+1 redundancy is where selected components in a system have a second item to take over should it fail
- Full redundancy is where two sets of identical hardware are provided.
- Sometimes are linked in a “fail-over” configuration (either manual or automatic)
- Other times linked as a “load sharing” arrangement
- N+1 is cheaper than full redundancy. Careful choices need to be made about what is made redundant

# Disk and File Systems – ZFS Case study

- There are a number of issues with modern file systems.
  - Scalability - How much can I grow?
  - Reliability - What happens on failure?
  - Consistency - Is my data consistent?
  - Manageability - How do I manage the file system?

# MEET ZFS

- Zetabyte File System (ZFS)
- Originated with Sun Microsystems but has ports to other OSs
- ZFS represents a different way of thinking about disks.
- ZFS is a comprehensive approach to storage management that includes the functions of a logical volume manager and a RAID [LSEP]
- Meets modern filesystem requirements providing redundancy, scalability, data integrity and other feature such as Snapshots, Clones and data compression

# ZFS Features

- Dynamic Striping and Redundancy (Mirroring & Raid Z/RaidZ2)
- Pooled storage: physical storage devices are added to a pool, and storage space is allocated from that shared pool.
- Scrubbing (live file system checks and fixes using metadata checksum - compare with *fsck* utility)
- Data Compression.
- Copy on write transactions.
- Snapshots & Clones (writable snapshots).
- Ability to reconstruct a dead device from healthy devices (resilvering)



# Concept: Pools

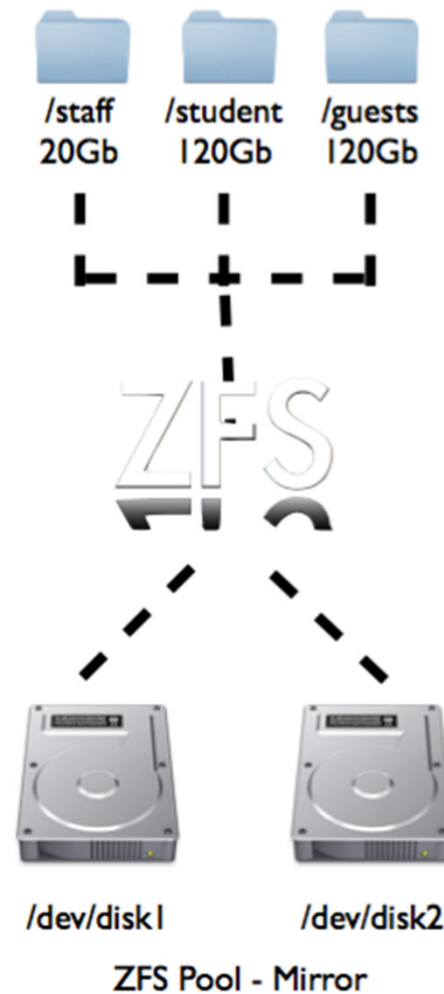
- Historically disks and partitions represented volumes.
  - Administrators would partition these devices initially to support their needs.
  - Did not allow for growth or redundancy



- To facilitate redundancy and scalability vendors produced *Volume Managers*.
- *Volumes Managers* would group devices and present virtual devices to handle growth and redundancy independent of the file system.



- ZFS aggregates devices (disks or files) into pools.
- The storage pool defines the characteristics of the storage i.e. redundancy.
- File systems are layered on top of the pool.
- The pool can grow and contract, so, too, can the file systems layered on top



# Concept Reliability

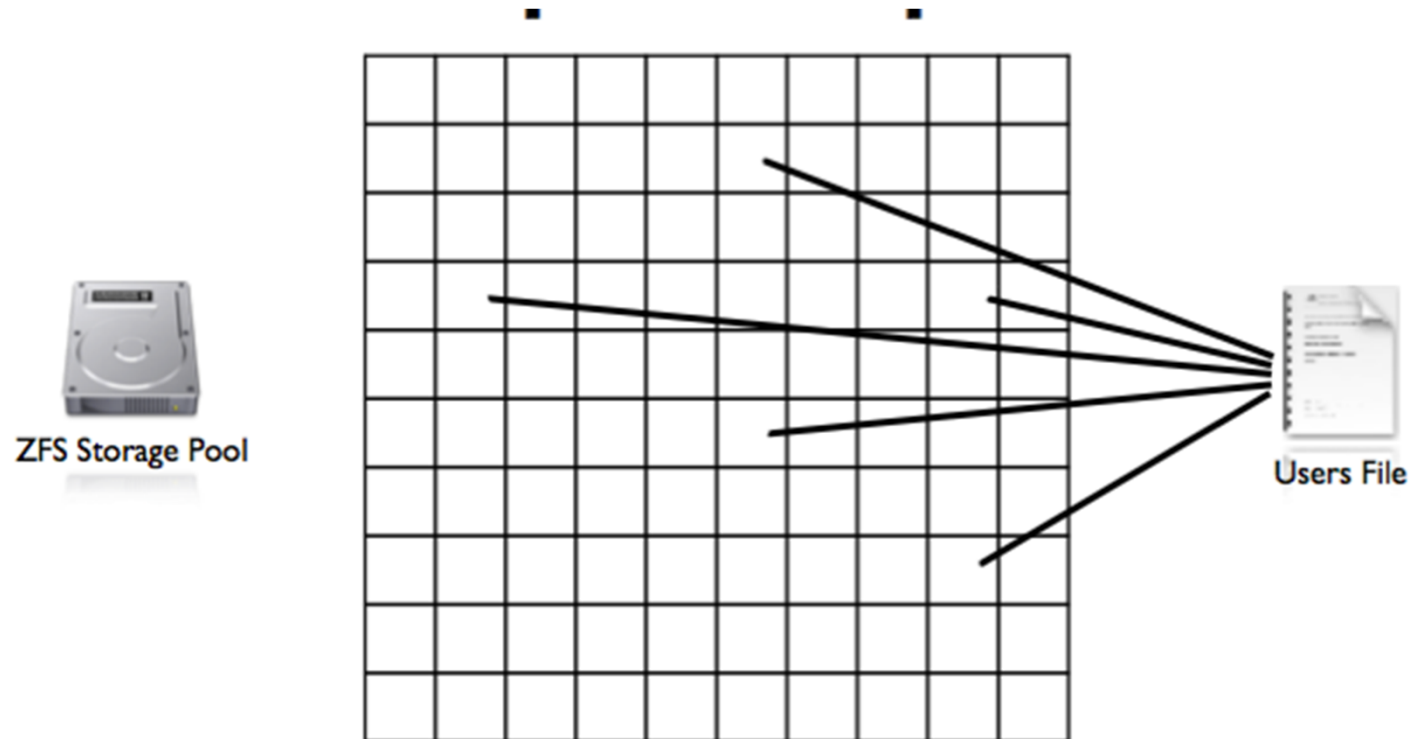
- ZFS is designed to be reliable. Pools can consist of multiple devices i.e. iSCSI and Fibre channel, real disks or even partitions. [SEP]
- These devices can be combined into virtual devices i.e. mirrors and RAIDZ if you have more than two disks. [SEP]
- Data on the devices comprising mirrors and RaidZ virtual devices are striped dynamically. [SEP]
- All Metadata in the file system is checksummed - if an error is detected it is fixed. [SEP][SEP]
- Each read is compared to a 256 bit checksum in meta data. [SEP]
- Also has a self-healing property - when errors are detected they are written to alternate blocks on the disk.
- If there is an error it is read from a different device and then fixed.

# Concept Transactions

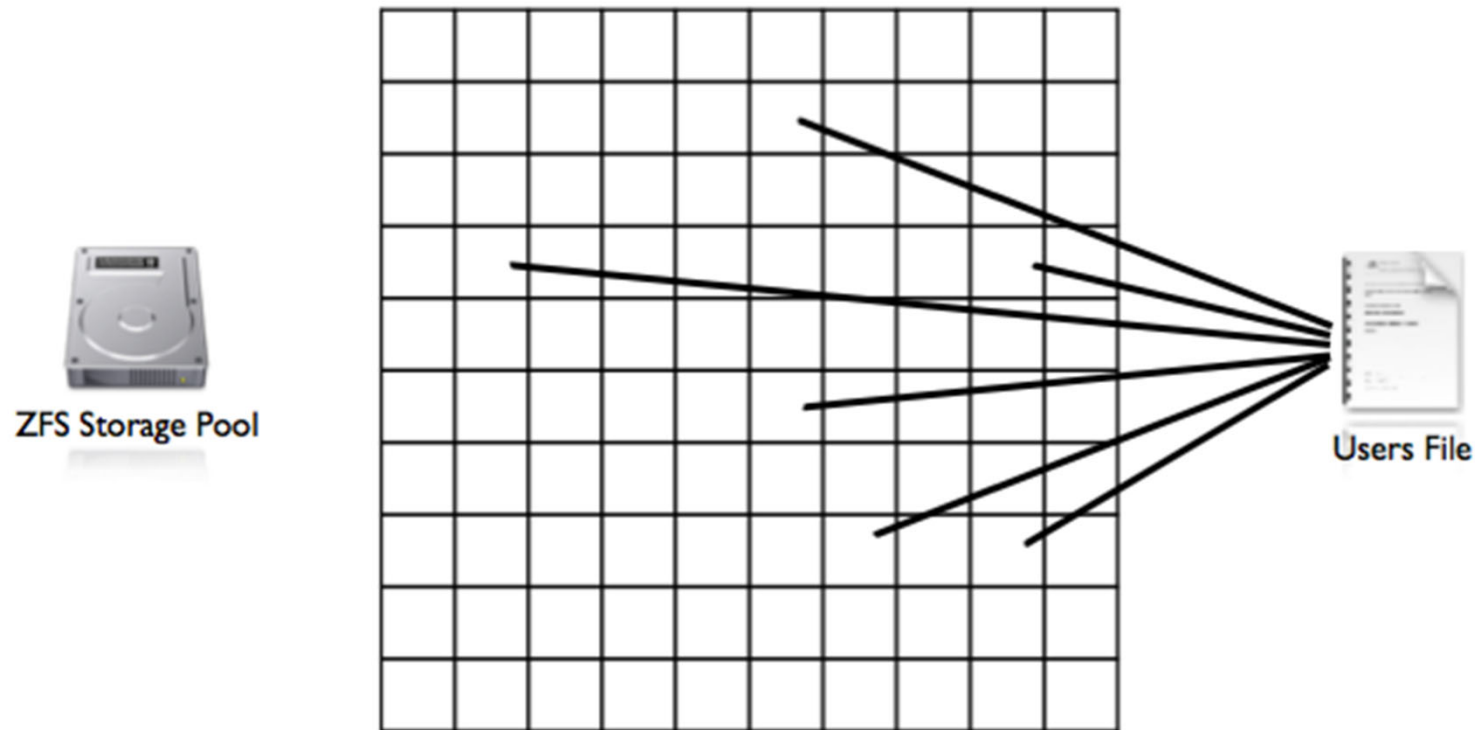
- Modern file systems implement journaling to keep transactions (blocks) on disks in a consistent state. [SEP]
- Failures can cause disks to be inconsistent i.e. blocks are allocated but not referenced by objects i.e. inodes on Unix File System (UFS) [SEP]
- By using *Copy on Write*, ZFS transactions are guaranteed to be consistent by always writing to new blocks (this is true for meta data blocks as well).
- Blocks containing active data are never overwritten in place; instead, a new block is allocated, modified data is written to it, then any metadata blocks referencing it are similarly read, reallocated, and written.

- By using Copy on Write, a copy of the old data blocks (and meta data) can be retained. [L][SEP]
- This means that a snapshot can be created at a specific point of the time in the file system. [L][SEP]
- Snapshots have no effect on space.
- Snapshots are implemented per filesystem rather than per-volume.

# Concept: Snapshots

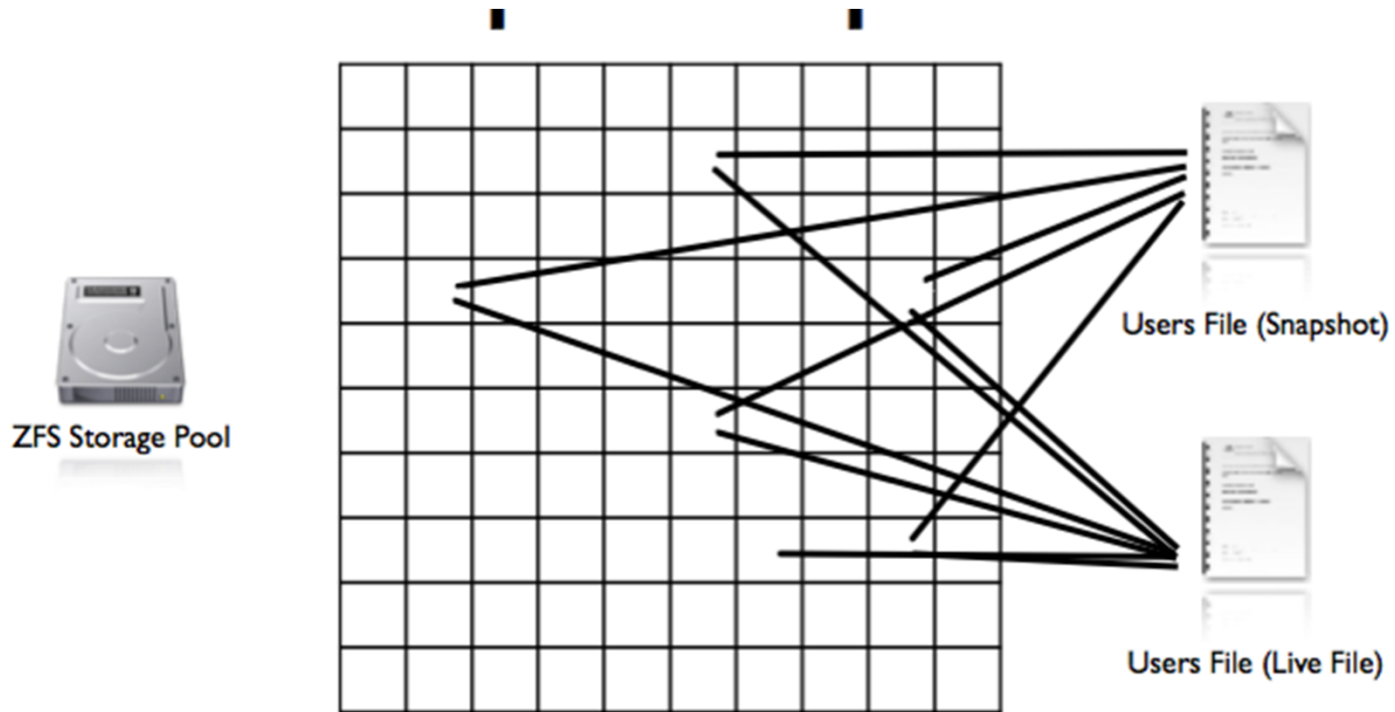


When a file is created blocks are allocated



On Copy on Write, changes mean a new block is allocated with meta data changes.





If a snapshot is made before the change then after the change we have two files.

# Concept Clone

- A clone is a 'promoted' snapshot. [L SEP]
- Writeable snapshots ("clones") can also be created, resulting in two independent file systems that share a set of blocks. As changes are made to any of the clone file systems, new data blocks are created to reflect those changes, but any unchanged blocks continue to be shared, no matter how many clones exist.
- As a result of being promoted it is a live file system which shares its base blocks with another. [L SEP]

# Create a pool with two disks

- Create a striped pool where a copy of data is stored across all drives
  - `sudo zpool create demo /dev/sdb /dev/sdc`
- Create a mirrored pool where a copy of data is stored across all drives
  - `sudo zpool create demo mirror /dev/sdb /dev/sdc`
- A new filesystem is automatically created and mounted at /demo
- You can check the status of the newly created pool
  - `sudo zpool status`
  - `sudo zpool list demo`
- You can also destroy a pool
  - `sudo zpool destroy demo`

# Create a RAIDZ pool

- RAID-Z is basically an improved version of RAID 5 which provides redundancy
- RAID-Z requires a minimum of three hard drives
- If a single disk in your pool dies, simply replace that disk and ZFS will automatically rebuild the data based on parity information from the other disks.
- To make things even more redundant, you can use RAID 6 (RAID-Z2 in the case of ZFS) and have double parity
- `sudo zpool create demo raidz /dev/sdb /dev/sdc /dev/sdd`

# Create filesystem

- A ZFS file system can be easily created
  - `sudo zfs create demo/fs1`
- To view a filesystem that is created
  - `sudo zfs list demo`
- To limit the usage of filesystem, we define reservation and quota
  - `zfs set quota=500m demo/fs1`
  - `zfs set reservation=200m demo/fs1`
- To change mount point for the filesystem
  - `zfs set mountpoint=/test demo/fs1`

# ZFS snapshots

- ZFS allows you to create instantaneous snapshots which are read-only copies of the file system
  - `zfs snapshot $fs@$snapshotname`
  - `zfs snapshot demo/fs1@version1`
- You can see all the snapshots created
  - `zfs list -t snapshot`
- A snapshot can be used to roll back to a previous state
  - `zfs rollback demo/fs1@version1`

# ZFS clones

- ZFS snapshots are not true filesystems. You can instantiate a snapshot as a full-fledged filesystem by cloning it.
- A clone is a read-write copy based on a snapshot.
  - `zfs clone demo/fs1@version1 demo/subclone`
- The new filesystem retains a link to both the snapshot and the filesystem on which it's based. Neither can be deleted as long as the clone exists.

# One filesystem per user

- If you keep users' home directories on a zfs storage pool, it is recommended that you make each home directory a separate filesystem.
  - You can easily set disk usage quotas on both individual users' filesystems and on the filesystem that contains all users
  - Snapshots are per filesystem and users can access snapshots through `./zfs` to service most of their own file restore needs



Questions?