Contents

Chapter 1

Block Devices

All disks are just devices—like keyboards or GPUs—managed by the Linux kernel. What makes disks special is that they are block devices, meaning data is read and written in fixed-size chunks with support for random access. This structure enables filesystems, caching, and efficient IO.

1 Block

A block device enables:

caching	blocks numbered—kernel can cache blocks in ram (page cache).
filesystems	reading/writing data/metadata offsets—needs random block access
efficient io	dag

Definition 1: Source Of Truth

The same block can exist in memory or on disk, which begs the question—which is the source of truth? To write data, the block must be moved into memory, and then made "dirty", ie changed. The dirty block is the source of truth. For a clean block, the source of truth is memory or disk, though most blocks will not be in memory.

Definition 2: drive firmware

The disk controller (in the drive firmware) exposes the disk as a Linear Block Address (LBA) space—a flat array of block numbers like:

$$\mathrm{LBA}_0, \mathrm{LBA}_1, \ldots, \mathrm{LBA}_N$$

When the OS issues a read/write to block 42, the controller translates that to a physical location (e.g., cylinder/head/sector or NAND page). This abstraction is standardized, which is why all block devices—from spinning HDDs to NVMe SSDs—present the same numbered interface.

Type	Example	Description
Whole disk	`/dev/sda`	Entire physical disk
Partition	`/dev/sda1`	Subdivision of a disk
Loop device	`/dev/loop0`	File-backed block device
LVM logical volume	`/dev/mapper/vg-lv`	Device-mapper virtual volume
Software RAID	`/dev/md0`	Aggregated disk via mdadm
ZRAM device	`/dev/zram0`	Compressed RAM-backed block
DM-Crypt (LUKS)	`/dev/mapper/cryptroot`	Encrypted block device
RAM disk	`/dev/ram0`	Legacy RAM-based block device

Table 1.1: Common Block Device Types in Linux

2 Sector

Definition 3: sector

While the term block is the most commonly encountered one when talking about "block" devices, disk devices are actually addressed by sector. A sector is the smallest addressable unit of a block device—typically 512 B or 4 KiB.
Filesystems often operate on larger logical blocks built from sectors (e.g., 4 KiB), but the sector remains the atomic I/O unit for hardware and the kernel.
Just learn to keep that tension in your brain as you read this, as that inexact phrasing dominates nearly all literature about this topic.

name	typical size	who uses it	what for
page	4 KiB	memory (MMU, kernel)	virtual memory, paging
sector	512 B–4 KiB	storage hardware, kernel	atomic disk I/O
block	4 KiB–64 KiB	filesystem	allocation, structure
cluster	4 KiB–64 KiB	FAT/NTFS filesystems	group of blocks for allocation
extent	variable	modern filesystems (ext4, XFS, Btrfs)	contiguous allocation range

Table 1.2: Standard units in memory and storage

3 FileSystem

Definition 4: allocation

How a filesystem decides where on disk to place file data. Some use fixed-size blocks, others group them into extents. Some delay allocation until writeback, others allocate immediately. The strategy affects fragmentation, performance, and metadata overhead.

filesystem	category	allocation	metadata	recovery	snapshots	checksums	compression
ext4	local	block bitmap	journaling	journal	no	no	no
xfs	local	extents	b-tree + log	log replay	no	metadata only	no
btrfs	local	extents	cow b-trees	atomic cow	yes	yes	yes
f2fs	local	segments	nat/sit	log replay	no	optional	optional
zfs	local	extents	cow b-trees	trans. groups	yes	yes	yes
apfs	local	extents	b-trees + cow	atomic cow	yes	yes	yes
ntfs	local	clusters	mft	journal	shadow copy	metadata only	yes
fat32	local	clusters	fat table	none	no	no	no
nfs	network	n/a	server-side	n/a	server-side	server-side	server-side
sshfs	network	n/a	passthrough	n/a	no	no	no
cephfs	network	object-based	distributed	fs journal	yes	yes	yes
fuse	user-space	n/a	user-defined	n/a	user-defined	user-defined	user-defined
overlayfs	overlay	passthrough	union of dirs	volatile	upper layer	no	no
tmpfs	memory	ram pages	kernel-internal	none	no	no	yes (zram)
iso9660	read-only	fixed layout	static table	none	no	optional (crc)	optional (rare)
ipfs	p2p/content	blockstore	merkledag	immutable	yes	yes (by hash)	no
initramfs	memory	cpio archive	static layout	none	no	no	no

Table 1.3: Comparison of filesystems by key characteristics and category

Definition 5: extent

Contiguous range of blocks on disk treated as a single unit. Instead of tracking each block individually, the filesystem records the starting block and the length. This reduces metadata overhead and improves performance for large files.

1. **Structure & Organization**

   (a) **Filesystem:** The overall structure of how files and directories are arranged on storage.

   (b) **Metadata:** Information *about* files, such as names, sizes, timestamps, and permissions.

2. **Data Management & Efficiency**

   (a) **Allocation:** The method by which disk space is assigned to files.

   (b) **Compression:** Reducing file size to optimize storage space and transfer speeds.

3. **Resilience & Advanced Features**

   (a) **Recovery:** The ability to restore the file system to a consistent state following failures or data corruption.

   (b) **Checksums:** Mechanisms for verifying data integrity, detecting accidental changes or corruption.

   (c) **Snapshots:** Point-in-time copies of the file system, crucial for backups, versioning, and disaster recovery.

Definition 6: Filesystem

A **filesystem** is a kernel-resident function that maps *inode-level operations* to *block-level operations*:

$$\text{inode\_op} \Rightarrow [\text{block\_op}]$$

It translates high-level actions like `open`, `read`, `write`, and `mkdir` into precise sequences of reads and writes on a block device.
Once mounted, the filesystem hooks into the kernel's VFS tree, and all accesses under the mount point are routed through this translation layer.

Chapter 2

Partitions

1 Boot Firmware

The type of boot firmware a computer has changes the types of partitions that are valid on a disk for that computer. The main purpose of the firmware is to find the bootloader, and to find the bootloader, the firmware needs to look into partitions. Those partitions need a standard format, so that the firmware can access any partition on first communication. The older type of firmware is bios, and it expects Master Boot Record type partitions.

Definition 7: Master Boot Record

It's essentially a fixed location on a fixed sector of the disk.

Definition 8: GPT

It's essentially a fixed location on a fixed sector of the disk.

Type	Firmware	Description
Primary	BIOS	Up to 4 primary partitions.
Extended	BIOS	container for logical partitions.
Logical	BIOS	inside an extended partition.
Boot	Both	Contains the kernel, initrd, and bootloader files.
Swap	Both	for virtual memory (paging) when RAM is full.
Root (/)	Both	contains the system's core files and directories.
EFI System Partition (`ESP`)	UEFI	bootloaders and system files, FAT32 format.
LVM	Both	volume management, allowing resizing and snapshots.
RAID	Both	for software RAID configurations (e.g., `mdadm`).
Linux Filesystem (ext4, XFS)	Both	formatted with specific Linux filesystems.
Linux Reserved	Both	special purposes or system recovery.

Table 2.1: Partition Types in Linux and Firmware Compatibility

Definition 9: sample setup

* **EFI System Partition (ESP)**: `FAT32`, approximately 512MB, mounted at `/boot/efi` for boot files.
* **Root Partition**: `Ext4`, approximately 15-20GB, mounted at `/` for system files.
* **Home Partition**: `Ext4`, utilizes remaining space, mounted at `/home` for user data.
* **Swap Partition**: `Linux-swap`, approximately 4GB, for memory overflow.
* **Data Partition**: Optional, for shared storage (discussed later).

Chapter 3

Device Mapper

$$
\begin{aligned}
I &:= \text{inode ops} \\
S &:= \text{sector ops} \\
f &:= \text{file system sys calls} \\
d &:= \text{device mapper} \\
f &: I \to S \\
d &: S \to S \\
d \circ f &: I \to S
\end{aligned}
$$

Definition 10: Physical Volume

$$
\begin{aligned}
S_0 &:= \text{raw sector device} \\
S_1 &:= \text{lvm-tracked sector space} \\
d &:= \text{pv init} \\
d &: S_0 \to S_1
\end{aligned}
$$

* first wedge in the LVM stack
* just marks a sector as eligible for lvm
* stores metadata in reserved sectors
* created via `pvcreate`

Definition 11: Volume Group

$$
\begin{aligned}
S_1 &:= \text{lvm-tracked sector space (from PVs)} \\
S_2 &:= \text{allocatable pool of extents} \\
d &:= \text{vg init} \\
d &: \{S_1\} \to S_2
\end{aligned}
$$

* second wedge in the LVM stack

* pools multiple PVs into a single storage namespace
* abstracts physical layout into a unified extent map
* created via `vgcreate`, extended via `vgextend`

Block Device	LUKS	PV	VG	LV
/dev/nvme0n1	crypt-nvme0n1	pv-nvme0n1	fast_vg	home
/dev/nvme1n1	crypt-nvme1n1	pv-nvme1n1	fast_vg	swap
/dev/sda1	crypt-sda1	pv-sda1	slow_vg	media
/dev/sdb1	crypt-sdb1	pv-sdb1	slow_vg	media
/dev/sdc1	crypt-sdc1	pv-sdc1	slow_vg	media

Table 3.1: LVM layering from block devices through LUKS to logical volumes

Intuition 1: LVM

Physical Volumes get a block device eligible for LVM. Then you want to pool them together into Volume Groups. From there, you split them back into Logical Volumes, like `home`, `data`, `media`, etc.
If you have multiple disk types — like external USBs, SATA disks, NVMe — then you probably want one Volume Group per disk type. That way, you can ensure your Logical Volume goes on the right kind of disk — like `home` on a fast disk, and `media` on the big slow disk.

Definition 12: Logical Volume

$$
\begin{aligned}
S_2 &:= \text{allocatable pool of extents (from VG)} \\
S_3 &:= \text{linear address space for user data} \\
d &:= \text{lv init} \\
d &: S_2 \to S_3
\end{aligned}
$$

* final wedge in the LVM stack
* defines usable block devices (e.g., `/dev/vg0/lv0`)
* supports resizing, snapshots, and striping
* created via `lvcreate`, resized via `lvextend`, `lvreduce`

Definition 13: extent

$$
\begin{aligned}
e &:= \text{contiguous region of sectors} \\
e &\subset S \quad \text{where } S \text{ is a sector space}
\end{aligned}
$$

* atomic allocation unit in LVM
* same size across all PVs in a VG (default: 4 MiB)
* logical volumes are built from extents

Definition 14: LUKS

$$S_1 := \text{plain sector}$$
$$S_2 := \text{encrypted sector}$$
$$d := \text{luks}$$
$$d : S_1 \to S_2$$

* transparent sector-level encryption
* decryption happens on io path
* user never sees ciphertext; the disk never sees plaintext.
* implemented via `dm-crypt` and `cryptsetup`

Chapter 4

Disko

Definition 15: Purpose

The purpose of disko is to go from a raw block storage device to a fully configured, partitioned disk with installed filesystems, ready to mount or ready for boot.

device disk partition

Chapter 5

Tools

Definition 16: installer

An installer is a bootable disk that can create another bootable disk. The reason you must run it is because it needs to survey the hardware and query the user for preferences. You can possibly combine both of those steps in nix if you know your destination hardware and encode the user preferences in the configuration.

element	description
bootable disk	a disk you can boot from (like a usb).
installer program	a program that runs from the bootable disk.
target disk	the disk where the program installs the operating system

Definition 17: fdisk

dialogue-driven command-line utility that creates and manipulates partition tables and partitions on a hard disk. Hard disks are divided into partitions and this division is described in the partition table.

```
fdisk -l <block device>
```

Definition 18: fstab

About these things, and basically only these things:

(1) locate a block device (partition is a block device)

(2) find its filesystem

(3) mount the filesystem on a mount point in the root filesystem

Chapter 6

LUKS

Definition 19: LUKS

linux unified key setup

Intuition 2: LUKS

It's a way to encrypt block level disks at rest. Once you decrypt and mount it, it appears just like any other device or filesystem for an application that is using it.

(1) dm-crypt

(2) full disk encryption

```
sudo cryptsetup luksOpen /dev/sda4 my-drive
sudo mkfs.ext4 /dev/mapper/my-drive -L my-drive
sudo mkdir /mnt/data
sudo mount /dev/mapper/my-drive /mnt/data
sudo cryptsetup luksAddKey --key-slot 1
sudo cryptsetup luksHeaderRestore /dev/sda4
cryptsetup luksDump /dev/xvda5
cryptsetup luksHeaderBackup /dev/xvda5 --header-backup-file LuckyHeader.bin
cfdisk /dev/xvdb
cryptsetup luksFormat /dev/xvdb
cryptsetup luksOpen /dev/xvd DataDrive
mkfs.ext 4 /dev/mapper/DataDrive
fsck /dev/mapper/DataDrive


lsblk
fdisk /dev/sdd
cryptsetup open /dev/sdd1 drive
mount /dev/mapper/drive /mnt
cryptsetup close drive
```

Definition 20: dm-crypt

a linux device mapper and kernel module for encryption

Definition 21: device mapper

Framework provided by the Linux kernel for mapping physical block devices onto higher-level virtual block devices. It forms the foundation of the

(1) logical volume manager (LVM)

(2) software RAIDs

(3) dm-crypt disk encryption

(4) file system snapshots

$$\text{deviceMapper} : \text{block} \to \text{block}$$

(1) pvcreate

(2) vgcreate

(3) lvcreate

cryptsetup keyslot luks header