Operating Systems Notes - Chapter 11

July 27, 2025

Contents

1 Overview of mass-storage structure

1.1 Introduction

- Computer systems must provide mass storage for permanently storing files and data.
- Modern computers use secondary storage: hard disks (HDDs) and nonvolatile memory (NVM) devices.
- Secondary storage devices vary:
  - Transfer: character at a time vs. block of characters.
  - Access: sequentially vs. randomly.
  - Data transfer: synchronously vs. asynchronously.
  - Usage: dedicated vs. shared.
  - Read-only vs. read-write.
  - Speed: slowest major component of computer.
- OS needs to provide wide range of functionality for device control.
- Key OS I/O subsystem goals:
  - Simplest interface possible to rest of system.
  - Optimize I/O for maximum concurrency (devices are performance bottleneck).
- Mass storage: nonvolatile storage system of a computer.
- Main mass-storage system: secondary storage (HDDs and NVM devices).
- Some systems: slower, larger tertiary storage (magnetic tape, optical disks, cloud storage).
- This chapter focuses on HDDs and NVM devices.
- Discuss physical structure, scheduling algorithms (maximize performance), device formatting, boot blocks, damaged blocks, swap space, RAID systems.
- General term: **non-volatile storage** (NVS) or "storage drives" for all types.
- Things to learn:
  - Describe physical structures of various secondary storage devices and effect of structure on uses.
  - Explain performance characteristics of mass-storage devices.
  - Evaluate I/O scheduling algorithms.
  - Discuss operating-system services for mass storage, including RAID.

1.2 Hard disk drives

- Bulk of secondary storage: **hard disk drives** (*HDDs*) and **nonvolatile memory** (*NVM*) devices.
- This section: basic mechanisms, OS translation of physical properties to logical storage via address mapping.
- HDDs: conceptually simple.
- Each disk **platter**: flat circular shape (like a CD).
- Common platter diameters: 1.8 to 3.5 inches.
- Two surfaces of platter: covered with magnetic material.
- Information stored by recording magnetically, read by detecting magnetic pattern.
- Read-write head: "flies" just above each surface of every platter.
- Heads attached to a **disk arm**: moves all heads as a unit.
- Platter surface: logically divided into circular **tracks**.
- Tracks subdivided into **sectors**.
- Set of tracks at given arm position: **cylinder**.
- Thousands of concentric cylinders; hundreds of sectors per track.
- Each sector: fixed size, smallest unit of transfer.
- Sector size: commonly 512 bytes until 2010, then migrating to 4KB.
- Storage capacity: gigabytes and terabytes.
- Disk drive motor: spins at high speed (60 to 250 times/sec, RPM).
- Common RPM: 5,400, 7,200, 10,000, 15,000.
- Some drives: power down when not in use, spin up on I/O request.
- Rotation speed relates to **transfer rate**: rate data flows between drive and computer.
- Another performance aspect: **positioning time** or **random-access time**.
- Positioning time parts:
  - **seek time**: move disk arm to desired cylinder.

  - **rotational latency**: desired sector rotates to disk head.
- Typical disks: tens to hundreds of MB/sec transfer, several ms seek/rotational latency.
- Performance increase: DRAM buffers in drive controller.
- Disk head: flies on thin cushion (microns) of air/gas.
- Danger: head contact with disk surface (**head crash**).
- Head crash: normally irreparable, entire disk replaced, data lost (unless backed up/RAID protected).
- HDDs: sealed units.
- Some chassis: allow HDD removal without shutting down system (**removable**).
- Other removable media: CDs, DVDs, Blu-ray discs.

1.3 Disk transfer rates

- Published performance numbers for disks: not same as real-world.
- Stated transfer rates: always higher than **effective transfer rates**.
- Transfer rate (bits read from magnetic media) vs. rate blocks delivered to OS.

1.4 Nonvolatile memory devices

- NVM devices: growing importance, electrical rather than mechanical.
- Composed of: controller and flash NAND die semiconductor chips (store data).
- Other NVM technologies (DRAM with battery, 3D XPoint): less common, not discussed.

1.4.1 Overview of nonvolatile memory devices

- Flash-memory-based NVM: often in disk-drive-like container, called **solid-state disk** (*SSD*).
- Other forms: **USB drive** (thumb drive/flash drive), DRAM stick.
- Surface-mounted on motherboards (smartphones).
- All forms: act and treated similarly.
- NVM devices: more reliable (no moving parts), faster (no seek/rotational latency), consume less power than HDDs.
- Negative: more expensive per MB, less capacity than larger HDDs.
- Trend: NVM capacity increased faster, price dropped quicker; increasing use.
- SSDs: used in laptops for smaller, faster, energy-efficient systems.
- NVM devices faster than HDDs: standard bus interfaces can limit throughput.
- Some NVM devices: connect directly to system bus (PCIe).
- NVM technology: changing computer design (direct disk replacement, new cache tier).
- NAND semiconductors: storage/reliability challenges.
- Read/write in "page" increment (similar to sector).
- Data cannot be overwritten: NAND cells must be erased first.
- Erasure: "block" increment (several pages), much slower than read/write.
- NVM flash devices: many die, many datapaths; parallel operations possible.
- NAND semiconductors: deteriorate with every erase cycle (100,000 program-erase cycles).
- NVM lifespan: measured in **Drive Writes Per Day** (**DWPD**).
- DWPD: how many times drive capacity can be written per day before failure.
- Example: 1 TB NAND drive, 5 DWPD rating $\implies$ 5 TB/day written for warranty period.
- Limitations: led to ameliorating algorithms (usually in NVM device controller, transparent to OS).
- OS reads/writes logical blocks; device manages.
- NVM performance variations: based on operating algorithms.

1.4.2 NAND flash controller algorithms

- NAND semiconductors cannot be overwritten: pages contain invalid data.
- Example: file-system block written, then rewritten. Old data invalid, new data valid.
- Controller maintains **flash translation layer** (*FTL*): maps physical pages to currently valid logical blocks.
- FTL tracks physical block state: which blocks contain only invalid pages (can be erased).
- Full SSD, pending write:
  - If block contains no valid data: write waits for erase, then occurs.
  - If no free blocks but individual pages hold invalid data: **garbage collection** occurs.

  - Good data copied to other locations, freeing blocks for erase/writes.
- To solve problem and improve write performance: **over-provisioning**.
- Over-provisioning: device sets aside pages (e.g., 20% total) as always available write area.
- Blocks totally invalid by garbage collection/write operations: erased, placed in over-provisioning space or returned to free pool.
- Over-provisioning space: helps with **wear leveling**.
- **Wear leveling**: effort to select all NAND cells over time as write targets.
- Avoids premature media failure due to frequently erased blocks.
- Controller: uses algorithms to place data on less-erased blocks to level wear.
- Data protection: NVM devices provide error-correcting codes (calculated/stored with data, detect/correct errors).
- If page frequently has correctible errors: marked bad, not used in subsequent writes.
- Single NVM device: catastrophic failure possible (corrupts/fails read/write).
- Data recoverable: RAID protection used.

1.5 Volatile memory

- DRAM: frequently used as mass-storage device.
- **RAM drives** (RAM disks): created by device drivers, carve out DRAM section, present as storage device.
- Can be used as raw block devices or with file systems.
- Purpose of RAM drives (despite volatility): allow user/programmer to place data in memory for temporary safekeeping using standard file operations.
- Useful for temporary files, sharing data.
- Found in all major OS: Linux (`/dev/ram`), MacOS (`diskutil`), Windows (third-party tools), Solaris/Linux (`/tmp` as "tmpfs" RAM drive at boot).

1.6 Magnetic tapes

- Early secondary-storage medium.
- Nonvolatile, holds large data quantities.
- Access time: slow compared to main memory/drives.
- Random access: 1000x slower than HDDs, 100,000x slower than SSDs.
- Main uses: backup, infrequently used info, transferring info between systems.
- Tape: kept in spool, wound/rewound past read-write head.
- Moving to correct spot: can take minutes.
- Once positioned: tape drives read/write at speeds comparable to HDDs.
- Capacities: exceed several terabytes.
- Some tapes: built-in compression (double effective storage).
- Categorized by width (4, 8, 19mm; 1/4, 1/2 inch) or technology (LTO-6, SDLT).

1.7 Secondary storage connection methods

- Secondary storage device: attached to computer by system bus or **I/O bus**.
- Bus types: **advanced technology attachment** (*ATA*), **serial ATA** (**SATA**), **eSATA**, **serial attached SCSI** (*SAS*), **universal serial bus** (*USB*), **fibre channel** (*FC*).
- Most common: SATA.
- NVM devices faster than HDDs: special fast interface **NVM express** (*NVMe*).
- NVMe: directly connects device to system PCI bus (increases throughput, decreases latency).
- Data transfers on bus: by special electronic processors (**controllers** or **host-bus adapters** (*HBA*)).
- **Host controller**: controller at computer end of bus.
- **Device controller**: built into each storage device.
- Mass storage I/O operation: computer places command into host controller (memory-mapped I/O ports).
- Host controller sends command via messages to device controller.
- Device controller operates drive hardware.
- Device controllers: usually have built-in cache.
- Data transfer at drive: between cache and storage media.
- Data transfer to host: fast electronic speeds, between cache host DRAM via DMA.

1.8 Address mapping

- Storage devices: addressed as large one-dimensional arrays of **logical blocks**.
- **Logical block**: smallest unit of transfer.
- Each logical block: maps to physical sector or semiconductor page.
- One-dimensional array of logical blocks: mapped onto sectors/pages of device.
- HDD mapping example: Sector 0 = first sector of first track on outermost cylinder. Proceeds through track, then rest of tracks on cylinder, then rest of cylinders (outermost to innermost).
- NVM mapping: tuple (chip, block, page) to array of logical blocks.
- **Logical block address** (**LBA**): easier for algorithms than sector, cylinder, head tuple or chip, block, page tuple.
- Converting LBA to old-style disk address (cylinder, track, sector): difficult in practice. Reasons:
  - Defective sectors: mapping hides by substituting spare sectors (LBA sequential, physical changes).
  - Number of sectors per track: not constant on some drives.
  - Disk manufacturers manage LBA to physical address mapping internally (little relationship between LBA and physical sectors).
- Algorithms dealing with HDDs: assume logical addresses relatively related to physical addresses (ascending logical $\implies$ ascending physical).
- **Constant linear velocity** (CLV): bit density uniform per track.
- Farther track from center: greater length, more sectors.
- Outer zones to inner zones: sectors per track decreases.
- Outer zone tracks: typically 40% more sectors than innermost.
- Drive increases rotation speed: head moves outer to inner tracks (keep same data rate under head).
- Used in CD-ROM and DVD-ROM drives.
- Alternatively: disk rotation speed constant.
- Bit density decreases inner to outer tracks (keep data rate constant, performance same).
- Used in hard disks: **constant angular velocity** (CAV).
- Sectors per track and cylinders per disk: increasing with technology.
- Large disks: tens of thousands of cylinders.
- Other storage devices exist (shingled magnetic recording HDDs, combination NVM/HDD devices, volume managers).
- These devices: different characteristics, may need different caching/scheduling algorithms.

Term	Definition
hard disk drive (HDD)	Secondary storage device based on mechanical components (spinning magnetic media plat ters, moving read-write heads).
nonvolatile memory (NVM)	Persistent storage based on circuits and electric charges.
platter	HDD component with magnetic media layer for holding charges.
disk arm	HDD component holding read-write head, moves over cylinders of platters.
track	On HDD platter, medium under read-write head during platter rotation.
sectors	On HDD platter, fixed-size section of a track.
cylinder	On HDD, set of tracks under read-write heads on all platters in device.
transfer rate	Rate at which data flows.
positioning time	On HDD, time for read-write head to position over desired track.
seek time	On HDD, time for read-write head to position over desired cylinder.
rotational latency	On HDD, time for read-write head (once over desired cylinder) to access desired track.
head crash	On HDD, mechanical problem: read-write head touching platter.
effective transfer rate	Actual, measured transfer rate of data between two devices.
solid-state disk	Disk-drive-like storage device using flash-memory-based nonvolatile memory.
USB drive	Nonvolatile memory in form of device plugging into USB port.
flash translation layer (FTL)	For nonvolatile memory, table tracking currently valid blocks.
garbage collection	Recovery of space containing no-longer-valid data.
over-provisioning	In non-volatile memory, space set aside for data writes not counted in device free space.
wear leveling	In nonvolatile memory, effort to select all NAND cells over time as write targets to avoi premature media failure.
RAM drives	Sections of system's DRAM presented as secondary storage devices.
magnetic tape	Magnetic media storage device (tape spooled on reels, passing over read-write head); mostly for backups.
I/O bus	Physical connection of I/O device to computer system.
advanced technology attachment (ATA)	Older-generation I/O bus.
eSATA	Type of I/O bus.
serial-attached SCSI (SAS)	Common type of I/O bus.
universal serial bus (USB)	Type of I/O bus.
fibre channel (FC)	Type of storage I/O bus used in data centers to connect computers to storage arrays.
NVM express (NVMe)	High-speed I/O bus for NVM storage.
controller	Special processor managing I/O devices.
host bus adapter (HBA)	Device controller installed in host bus port to allow device connection to host.
host controller	I/O-managing processors within a computer (e.g., inside HBA).
device controller	I/O managing processor within a device.
logical blocks	Logical addresses used to access blocks on storage devices.
constant linear velocity (CLV)	Device-recording method: constant bit density per track by varying rotational speed.
constant angular velocity (CAV)	Device-recording method: medium spins at constant velocity, bit density decreases inner t outer tracks.

2 HDD scheduling

- OS responsibility: use hardware efficiently.
- For HDDs: minimize access time, maximize data transfer bandwidth.
- Access time (for HDDs/mechanical storage):
  - **seek time**: time for device arm to move heads to desired cylinder.
  - **rotational latency**: additional time for platter to rotate desired sector to head.
- Device **bandwidth**: total bytes transferred / total time (first request to last transfer completion).
- Improve access time and bandwidth: manage order of storage I/O requests.
- Process needs I/O: issues system call to OS.
- Request specifies: input/output, open file handle, memory address, amount of data.
- Drive/controller available: request serviced immediately.
- Drive/controller busy: new requests placed in queue.
- Multiprogramming system: device queue often has pending requests.
- Queue of requests: allows device drivers to improve performance via ordering (avoiding head seeks).
- Past HDD interfaces: host specified track/head; much effort on disk scheduling.
- Modern drives: do not expose these controls, map LBA to physical addresses internally.
- Current disk scheduling goals: fairness, timeliness, optimizations (e.g., bunching sequential reads/writes).
- Drives perform best with sequential I/O.
- Absolute knowledge of head/physical block/cylinder locations: generally not possible on modern drives.
- Approximation: increasing LBAs mean increasing physical addresses; close LBAs equate to physical proximity.

2.1 FCFS scheduling

- Simplest disk scheduling: First-Come, First-Served (FCFS) or FIFO.
- Intrinsically fair.
- Generally does not provide fastest service.
- Example: queue requests for cylinders 98, 183, 37, 122, 14, 124, 65, 67.
- Initial head at 53.
- Movement: $53 \rightarrow 98 \rightarrow 183 \rightarrow 37 \rightarrow 122 \rightarrow 14 \rightarrow 124 \rightarrow 65 \rightarrow 67$.
- Total head movement: 640 cylinders.
- Problem: wild swing (e.g., 122 to 14 then back to 124).
- Servicing requests for 37 and 14 together (before/after 122/124) would decrease head movement, improve performance.

2.2 SCAN scheduling

- **SCAN algorithm**: disk arm starts at one end, moves to other, servicing requests.
- Reaches other end: direction reversed, servicing continues.
- Head continuously scans back and forth.
- Also called **elevator algorithm**.
- Example: requests 98, 183, 37, 122, 14, 124, 65, 67.
- Initial head at 53, moving toward 0.
- Servicing order: $53 \rightarrow 37 \rightarrow 14 \rightarrow 0$ (reverse) $\rightarrow 65 \rightarrow 67 \rightarrow 98 \rightarrow 122 \rightarrow 124 \rightarrow 183$.
- Request just in front of head: serviced almost immediately.
- Request just behind head: waits until arm moves to end, reverses, comes back.
- Uniform distribution of requests: few requests immediately in front of head after reversal (recently serviced).
- Heaviest density of requests: at other end of disk (waited longest).

2.3 C-SCAN scheduling

- **Circular SCAN (C-SCAN) scheduling**: variant of SCAN for more uniform wait time.
- Moves head from one end to other, servicing requests.
- Reaches other end: immediately returns to beginning of disk, no servicing on return trip.
- Essentially treats cylinders as circular list.
- Example: requests 98, 183, 37, 122, 14, 124, 65, 67.
- Initial head at 53, moving from 0 to 199.

- Servicing order: $53 \rightarrow 65 \rightarrow 67 \rightarrow 98 \rightarrow 122 \rightarrow 124 \rightarrow 183 \rightarrow 199$ (return to 0) $\rightarrow 14 \rightarrow 37$.

2.4 Selection of a disk-scheduling algorithm

- Many disk-scheduling algorithms exist, rarely used.
- Optimal order: can be defined for any request list, but computation may not justify savings over SCAN.
- Performance depends heavily on number and types of requests.
- Example: queue with one outstanding request $\implies$ all algorithms behave like FCFS.
- SCAN and C-SCAN: better for heavy disk load, less likely to cause starvation.
- Starvation still possible: Linux created **deadline** scheduler.
- Deadline scheduler:
  - Maintains separate read and write queues.
  - Gives reads priority (processes more likely to block on read).
  - Queues sorted in LBA order (implements C-SCAN).
  - All I/O requests sent in batch in LBA order.
  - Keeps four queues: two read, two write (one sorted by LBA, other by FCFS).
  - Checks after each batch: FCFS requests older than configured age (default 500 ms)?
  - If so: LBA queue (read/write) with old request selected for next batch.
- Deadline I/O scheduler: default in Linux RedHat 7.
- RHEL 7 also includes:
  - NOOP: preferred for CPU-bound systems using fast storage (NVM devices).
  - **Completely Fair Queueing scheduler** (*CFQ*): default for SATA drives.
  - CFQ maintains three queues (insertion sort, LBA order): real time, best effort (default), idle.
  - Each has exclusive priority (real time > best effort > idle); starvation possible.
  - Uses historical data: anticipates if process will issue more I/O requests soon.
  - If so: idles waiting for new I/O, ignores other queued requests (minimizes seek time, assumes locality of reference per process).

Term	Definition
bandwidth	Total amount of data transferred divided by total time between first request for service and completion of last transfer.
SCAN algorithm	HDD I/O scheduling algorithm: disk head moves from one end to other, performing I/O then reverses.
Circular SCAN (CSCAN) scheduling	HDD I/O scheduling algorithm: disk head moves from one end to other, performing I/O then returns to beginning without servicing.
Completely Fair Queueing (CFQ)	In Linux, default I/O scheduler in kernel 2.6 and later.

3 NVM scheduling

- Disk-scheduling algorithms (HDDs): minimize disk head movement.
- NVM devices: no moving disk heads, commonly use simple FCFS policy.
- Linux **NOOP** scheduler: FCFS policy, merges adjacent requests.
- NVM device behavior: read service time uniform, write service time not uniform (due to flash memory properties).
- Some SSD schedulers: merge only adjacent write requests, service all read requests in FCFS order.
- I/O: sequential or random.
- Sequential access: optimal for mechanical devices (HDD, tape); data near read/write head.
- Random-access I/O: measured in **input/output operations per second** (*IOPS*).
- Random access I/O: causes HDD disk head movement.
- Random access I/O: much faster on NVM (HDD hundreds IOPS, SSD hundreds of thousands IOPS).
- NVM devices: less advantage for raw sequential throughput (HDD head seeks minimized).
- Sequential reads: NVM performance equivalent to 10x advantage over HDD.
- Writing to NVM: slower than reading, decreases advantage.
- HDD write performance: consistent throughout device life.
- NVM write performance: varies based on device fullness (garbage collection, over-provisioning) and "wear".
- Worn NVM device: much worse performance than new device.
- Improve NVM lifespan/performance: file system informs device when files deleted (device erases blocks).
- Impact of garbage collection on performance:
- NVM device under random read/write load, full but with free space.
- **Garbage collection** must occur to reclaim space from invalid data.
- One write might cause: read of pages, write of good data to overprovisioning space, erase of all-invalid-data block, placement of block into overprovisioning space.
- Summary: one write request $\implies$ page write (data) + one or more page reads (by GC) + one or more page writes (good data from GC blocks).
- Creation of I/O requests by NVM device (GC, space management): **write amplification**.
- Write amplification: can greatly impact write performance.
- Worst case: several extra I/Os triggered with each write request.

Term	Definition
NOOP	Linux NVM scheduling algorithm: FCFS with adjacent requests merged.
input/output operations per second	Measure of random access I/O performance: number of inputs + outputs per second.
write amplification	Creation of I/O requests by NVM devices (GC, space management), impacting write performance.

4 Error detection and correction

- Error detection and correction: fundamental to memory, networking, storage.
- **Error detection**: determines if problem occurred (e.g., bit change in DRAM, network packet change, data block change).
- By detecting issue: system can halt operation, report error, warn of failing/failed device.
- Memory systems: detect errors using parity bits.
- Each byte: associated parity bit (records even/odd number of 1s).
- Single-bit error: parity changes, does not match stored parity $\implies$ detected.
- Stored parity bit damaged: does not match computed parity $\implies$ detected.
- All single-bit errors detected.
- Double-bit error: might go undetected.
- Parity: calculated by XORing bits.
- Requires extra bit of memory per byte.
- Parity: one form of **checksums**.
- **Checksums**: use modular arithmetic to compute, store, compare values on fixed-length words.
- Another error-detection method: **cyclic redundancy check** (**CRCs**).
- CRCs: use hash function to detect multiple-bit errors.
- **Error-correction code** (*ECC*): detects and corrects problems.
- Correction: uses algorithms and extra storage.
- Codes vary: based on extra storage needed, number of errors correctable.
- Disk drives: use per-sector ECC.
- Flash drives: use per-page ECC.
- Controller writes sector/page: ECC written with value calculated from data.
- Sector/page read: ECC recalculated, compared with stored value.
- Mismatch: data corrupted, storage media may be bad.
- ECC is error correcting: contains enough info to identify changed bits and calculate correct values (if few bits corrupted).
- Reports recoverable **soft error**.
- Too many changes, ECC cannot correct: non-correctable **hard error** signaled.
- Controller automatically performs ECC processing on read/write.
- Error detection/correction: frequently differentiators between consumer and enterprise products.
- ECC: used in some systems for DRAM error correction and data path protection.

Term	Definition
error detection	Determining if a problem has occurred (e.g., data corruption).
checksum	General term for an error detection and correction code.
cyclic redundancy check (CRCs)	Error-detection method using a hash function to detect multiple-bit errors.
error-correcting code (ECC)	Value calculated from data bytes, recalculated later to check for changes, and can correc errors.
soft error	Recoverable error by retrying the operation.
hard error	Unrecoverable error, possibly resulting in data loss.

5 Storage device management

- OS responsible for several aspects of storage device management.
- Discuss: drive initialization, booting from drive, bad-block recovery.

5.1 Drive formatting, partitions, and volumes

- New storage device: blank slate (platter of magnetic material, uninitialized semiconductor cells).
- Before storing data: device must be divided into sectors (controller can read/write).
- NVM pages: must be initialized, FTL created.
- Process: **low-level formatting** or **physical formatting**.
- Low-level formatting: fills device with special data structure for each storage location.
- Data structure (sector/page): header, data area, trailer.
- Header/trailer: controller info (sector/page number, error detection/correction code).
- Most drives: low-level formatted at factory (manufacturing process).
- Enables manufacturer: test device, initialize mapping from logical block numbers to defect-free sectors/pages.
- Choice of sector sizes: 512 bytes, 4KB.
- Larger sector size: fewer sectors per track, fewer headers/trailers, more space for user data.
- Some OS: handle only one specific sector size.
- OS needs to record own data structures on device (3 steps):
- **Step 1: Partitioning**
  - Divide device into one or more groups of blocks/pages.
  - OS can treat each partition as separate device.
  - Example: one partition for OS executable code (file system), another for swap space, another for user files.
  - Some OS/file systems: partition automatically when entire device managed by file system.
  - Partition info: written in fixed format at fixed location on storage device.
  - Linux: `fdisk` command manages partitions.
  - Device recognized by OS: partition info read, OS creates device entries (`/dev` in Linux).
  - Configuration file (`/etc/fstab`): tells OS to **mount** each partition (containing file system) at specified location, with mount options (e.g., read-only).
  - **Mounting** a file system: making it available for use.
- **Step 2: Volume creation and management**
  - Implicit: file system placed directly within a partition $\implies$ **volume** ready to be mounted.
  - Explicit: multiple partitions/devices used as RAID set (one or more file systems spread across devices).
  - Linux volume manager `lvm2` provides features; commercial third-party tools.
  - ZFS: volume management and file system integrated.
  - "Volume" can also mean any mountable file system (e.g., CD image file).
- **Step 3: Logical formatting** or creation of a file system.
  - OS stores initial file-system data structures onto device.
  - Data structures: maps of free/allocated space, initial empty directory.
- Partition info: indicates if partition contains bootable file system (OS).
- Partition labeled for boot: used to establish root of file system.
- Once mounted: device links for all other devices/partitions created.
- Computer's "file system": consists of all mounted volumes.
- Windows: separately named via letter (C:, D:, E:).
- Linux: boot file system mounted at boot, other file systems mounted within tree structure.
- Windows: file system interface clear when device used.
- Linux: single file access might traverse many devices.
- File systems group blocks into larger chunks: **clusters**.
- Device I/O: via blocks; file system I/O: via clusters.
- Assures: more sequential-access, fewer random-access characteristics.
- File systems: group file contents near metadata (reduces HDD head seeks).
- Some OS: allow special programs to use partition as large sequential array of logical blocks (no file-system data structures).
- Array: **raw disk**; I/O: **raw I/O**.

- Used for swap space, some database systems (control exact record location).
- Raw I/O: bypasses file-system services (buffer cache, file locking, prefetching, space allocation, file names, directories).
- Allows applications to implement own special-purpose storage services on raw partition.
- Most applications: use provided file system.
- Linux: generally no raw I/O, but similar access via `DIRECT` flag to `open()` system call.

5.2 Boot block

- Computer starts (powered up/rebooted): must have initial program to run.
- Initial **bootstrap** loader: tends to be simple.
- Bootstrap: stored in NVM flash memory firmware on motherboard, mapped to known memory location.
- Can be updated by manufacturers, but also written to by viruses.
- Initializes: CPU registers, device controllers, main memory contents.
- Tiny bootstrap loader: brings in full bootstrap program from secondary storage.
- Full bootstrap program: stored in "boot blocks" at fixed location on device.
- Device with boot partition: **boot disk** or **system disk**.
- Bootstrap NVM code: instructs storage controller to read boot blocks into memory (no device drivers loaded), then executes code.
- Full bootstrap program: more sophisticated, loads entire OS from non-fixed location, starts OS.
- Windows boot process example:
  - Drive divided into partitions.
  - One partition: **boot partition** (contains OS, device drivers).
  - Windows boot code: first logical block on hard disk/first page of NVM device.
  - Termed: **master boot record** (*MBR*).
  - Booting begins: runs code in system's firmware.
  - Firmware code: directs system to read boot code from MBR (understands enough about controller/device to load sector).
  - MBR: contains boot code, table listing partitions, flag for boot partition.
  - System identifies boot partition: reads first sector/page (**boot sector**).
  - Boot sector: directs to kernel.
  - Continues boot process: loads subsystems, system services.

5.3 Bad blocks

- Disks: prone to failure (moving parts, small tolerances).
- Complete failure: disk replaced, contents restored from backup.
- More frequent: one or more sectors become defective.
- Most disks: come from factory with **bad blocks**.
- Bad blocks handled in various ways (depending on disk/controller).
- Older disks (e.g., IDE controllers): bad blocks handled manually.
- Strategy: scan disk for bad blocks during formatting.
- Discovered bad blocks: flagged as unusable (file system doesn't allocate).
- Blocks go bad during operation: special program (Linux `badblocks`) run manually to search/lock away.
- Data on bad blocks: usually lost.
- More sophisticated disks: smarter about bad-block recovery.
- Controller maintains list of bad blocks (initialized at factory, updated over life).
- Low-level formatting: sets aside spare sectors (not visible to OS).
- Controller: replaces each bad sector logically with spare sector.
- Scheme: **sector sparing** or **forwarding**.
- Typical bad-sector transaction:
  - OS tries to read logical block 87.
  - Controller calculates ECC, finds sector bad, reports I/O error to OS.
  - Device controller replaces bad sector with spare.
  - After that: system requests logical block 87 $\implies$ translated to replacement sector's address by controller.
- Redirection by controller: could invalidate OS disk-scheduling optimization.
- Most disks: formatted with few spare sectors in each cylinder, and a spare cylinder.

- Bad block remapped: controller uses spare sector from same cylinder if possible.
- Alternative to sector sparing: **sector slipping**.
- Example: logical block 17 defective, first spare after sector 202.
- Sector slipping: remaps all sectors from 17 to 202, moving them down one spot.
- Frees up space of sector 18, sector 17 mapped to it.
- Recoverable soft errors: may trigger device activity (copy block data, spare/slip block).
- Unrecoverable hard error: lost data. File using block must be repaired (e.g., from backup), requires manual intervention.
- NVM devices: bits, bytes, pages nonfunctional/go bad.
- Management simpler than HDDs (no seek time performance loss).
- Multiple pages set aside as replacement locations, or space from over-provisioning area used.
- Controller maintains table of bad pages, never sets them as available to write to.

Term	Definition
low-level formatting	Initialization of a storage medium for computer storage.
physical formatting	Initialization of a storage medium for computer storage.
partition	Logical segregation of storage space into multiple areas.
mounting	Making a file system available for use by logically attaching it to the root file system.
volume	Container of storage; often a device with a mountable file system.
logical formatting	Creation of a file system in a volume to ready it for use.
cluster	In Windows storage, a power-of-2 number of disk sectors collected for I/O optimization.
raw disk	Direct access to a secondary storage device as an array of blocks with no file system.
bootstrap	Steps taken at computer power-on to bring system to full operation.
boot disk	Disk with a boot partition and kernel to load for booting.
system disk	Storage device with a boot partition, can store OS for booting.
boot partition	Storage device partition containing an executable operating system.
master boot record (MBR)	Windows boot code, stored in the first sector of a boot partition.
boot sector	First sector of a Windows boot device, containing bootstrap code.
bad block	Unusable sector on an HDD.
sector sparing	Replacement of unusable HDD sector with another sector elsewhere on device.
sector slipping	Renaming of sectors to avoid using a bad sector.

6 Swap-space management

- Swapping: moving entire processes between secondary storage and main memory (Section 9.5).
- Occurs when physical memory critically low, processes moved to swap space to free memory.
- Modern OS: very few implement swapping this way.
- Systems now combine swapping with virtual memory (Chapter Virtual Memory), swap pages, not entire processes.
- Terms "swapping" and "paging" used interchangeably.
- **Swap-space management**: low-level OS task.
- Virtual memory uses secondary storage as extension of main memory.
- Drive access much slower than memory access $\implies$ swap space significantly decreases system performance.
- Main goal for swap space design/implementation: provide best throughput for virtual memory system.
- Discuss: swap space use, location, management.

6.1 Swap-space use

- Swap space used differently by OS, depending on memory-management algorithms.
- Swapping systems: may hold entire process image (code, data segments).
- Paging systems: store pages pushed out of main memory.
- Amount of swap space needed: varies (few MB to GB).
- Depends on: physical memory, virtual memory backing, virtual memory usage.
- Safer to overestimate than underestimate swap space.
- Running out of swap space: may abort processes or crash.
- Overestimation: wastes secondary storage space (no other harm).
- Some systems recommend swap space amount.
- Solaris: swap space = virtual memory exceeding pageable physical memory.
- Past Linux: swap space = double physical memory.
- Today's Linux: paging algorithms changed, considerably less swap space used.
- Some OS (Linux): allow multiple swap spaces (files, dedicated partitions).
- Multiple swap spaces: usually on separate storage devices.
- Purpose: spread I/O load from paging/swapping over system's I/O bandwidth.

6.2 Swap-space location

- Swap space can reside in two places:
  - Carved out of normal file system (large file).
  - In a separate **raw partition**.
- Swap space as file: normal file-system routines create, name, allocate space.
- Swap space in raw partition:
  - No file system or directory structure.
  - Separate swap-space storage manager allocates/deallocates blocks.
  - Manager uses algorithms optimized for speed (not storage efficiency).
  - Swap space accessed more frequently than file systems.
  - Internal fragmentation may increase (acceptable trade-off, data life shorter).
  - Fragmentation short-lived (reinitialized at boot time).
  - Raw-partition approach: fixed amount of swap space during disk partitioning.
  - Adding more swap space: repartitioning device (moving/destroying other partitions) or adding another swap space elsewhere.
- Some OS: flexible, can swap in raw partitions and file-system space (e.g., Linux).
- Trade-off: convenience of file system allocation/management vs. performance of raw partitions.

6.3 Swap-space management: an example

- Evolution of swapping/paging in UNIX systems:
- Traditional UNIX kernel: copied entire processes between contiguous disk regions and memory.
- Later UNIX: evolved to combination of swapping and paging (as paging hardware available).
- Solaris 1 (SunOS): changed standard UNIX methods for efficiency/tech developments.
- Process executes: text-segment pages (code) brought from file system, accessed in memory, thrown away if selected for pageout.
- More efficient to reread page from file system than write to swap and reread.
- Swap space used only as backing store for pages of **anonymous** memory.
- **Anonymous** memory: not backed by any file (stack, heap, uninitialized data of process).
- Later Solaris versions: biggest change $\implies$ allocates swap space only when page forced out of physical memory (not when virtual memory page first created).
- Scheme: better performance on modern computers (more physical memory, page less).
- Linux: similar to Solaris, swap space only for anonymous memory.
- Linux: allows one or more swap areas (swap file on regular file system or dedicated swap partition).
- Each swap area: series of 4-KB **page slots** (hold swapped pages).
- Associated with each swap area: **swap map** (array of integer counters).
- Each counter: corresponds to page slot.
- Counter value 0: page slot available.
- Counter value > 0: page slot occupied by swapped page.
- Counter value: indicates number of mappings to swapped page (e.g., 3 $\implies$ mapped to 3 processes if shared memory).

Term	Definition
swap-space management	Low-level OS task of managing space on secondary storage for swapping and paging.
raw partition	Partition within a storage device not containing a file system.
anonymous memory	Memory not associated with a file; dirty pages paged out are stored in swap space.
page slot	In Linux swap-space management, part of data structure tracking swap-space use.
swap map	In Linux swap-space management, part of data structure tracking swap-space use.

7 Storage attachment

- Computers access secondary storage in three ways: host-attached, network-attached, and cloud storage.

7.1 Host-attached storage

- **Host-attached storage**: accessed through local I/O ports (most common: SATA).
- Typical system: one or few SATA ports.
- More storage access: individual storage device, device in chassis, or multiple drives in chassis connected via USB, FireWire, or Thunderbolt.
- High-end workstations/servers: need more/shared storage, use sophisticated I/O architectures.
- **Fibre channel** (*FC*): high-speed serial architecture (optical fiber or copper cable).
- FC benefits: large address space, switched communication, multiple hosts/storage devices attach to fabric (flexibility in I/O communication).
- Wide variety of devices suitable for host-attached storage: HDDs, NVM devices, CD/DVD/Blu-ray/tape drives, storage-area networks (**SANs**).
- I/O commands for host-attached storage: reads/writes of logical data blocks directed to specifically identified storage units (bus ID or target logical unit).

7.2 Network-attached storage

- **Network-attached storage** (*NAS*): provides access to storage across a network.
- NAS device: special-purpose storage system or general computer system providing storage to other hosts.
- Clients access NAS via remote-procedure-call (RPC) interface: NFS (UNIX/Linux), CIFS (Windows).
- RPCs carried via TCP/UDP over IP network (usually same LAN).
- NAS unit: usually implemented as storage array with RPC interface software.
- CIFS and NFS: provide locking features, allowing file sharing between hosts accessing NAS.
- NAS: convenient way for LAN computers to share storage pool (ease of naming/access).
- Downside: less efficient, lower performance than some direct-attached storage.
- **iSCSI**: latest network-attached storage protocol.
- Uses IP network protocol to carry SCSI protocol.
- Networks (not SCSI cables) used as interconnects between hosts and storage.
- Hosts treat storage as directly attached, even if distant.
- NFS/CIFS: present file system, send parts of files.
- iSCSI: sends logical blocks across network, client uses blocks directly or creates file system.

7.3 Cloud storage

- **Cloud storage**: similar to network-attached storage, access across network.
- Unlike NAS: accessed over Internet/WAN to remote data center (storage for fee/free).
- Difference from NAS: how storage accessed/presented.
- NAS: accessed as another file system (CIFS/NFS) or raw block device (iSCSI). OS integrates these protocols.
- Cloud storage: API based; programs use APIs to access.
- Examples: Amazon S3, Dropbox, Microsoft OneDrive, Apple iCloud.
- Reason for APIs vs. existing protocols: WAN latency and failure scenarios.
- NAS protocols: designed for LANs (lower latency, less connectivity loss).
- LAN connection failure (NFS/CIFS): system might hang.
- Cloud storage: failures more likely, application pauses access until connectivity restored.

7.4 Storage-area networks and storage arrays

- Drawback of NAS: storage I/O consumes data network bandwidth, increases network communication latency.
- Acute in large client-server installations: server-client communication competes with server-storage communication.
- **Storage-area network** (*SAN*): private network (storage protocols, not networking protocols) connecting servers and storage units.
- SAN power: flexibility.
- Multiple hosts/storage arrays: attach to same SAN.
- Storage: dynamically allocated to hosts.
- Storage arrays: RAID protected or unprotected drives (**Just a Bunch of Disks** (**JBOD**)).

- SAN switch: allows/prohibits access between hosts and storage.
- Example: host low on disk space, SAN configured to allocate more storage.
- SANs: clusters of servers share same storage, storage arrays include multiple direct host connections.
- SANs: typically more ports, higher cost than storage arrays.
- SAN connectivity: short distances, typically no routing. NAS can have more connected hosts than SAN.
- Storage array: purpose-built device (includes SAN/network ports or both).
- Contains: drives to store data, controller(s) to manage storage/access.
- Controllers: CPUs, memory, software (implement array features: network protocols, UIs, RAID, snapshots, replication, compression, deduplication, encryption).
- Some storage arrays include SSDs.
- Array with only SSDs: maximum performance, smaller capacity.
- Mix of SSDs/HDDs: array software/administrator selects best medium, or SSDs as cache/HDDs as bulk storage.
- FC: most common SAN interconnect.
- iSCSI: increasing use due to simplicity.
- Another SAN interconnect: **InfiniBand** (*IB*).
- IB: special-purpose bus architecture, hardware/software support for high-speed interconnection networks (servers, storage units).

Term	Definition
host-attached storage	Storage accessed through local I/O ports (directly attached to a computer).
fibre channel (FC)	Storage I/O bus used in data centers to connect computers to storage arrays.
network-attached storage (NAS)	Storage accessed from a computer over a network.
iSCSI	Protocol using IP network to carry SCSI protocol for distant storage access.
cloud storage	Storage accessed over Internet/WAN to a remote, shared data center.
storage-area network (SAN)	Local-area storage network allowing multiple computers to connect to storage devices.
Just a Bunch of Disks (JBOD)	Unprotected drives in a storage array.
InfiniBand (IB)	High-speed network communications link for servers and storage units.

8 RAID structure

- Economically feasible to attach many drives to a computer.
- Large number of drives: opportunities for improving data read/write rate (parallel operation).
- Potential for improving data storage reliability: redundant information on multiple drives.
- Failure of one drive: does not lead to data loss.
- **Redundant arrays of independent disks** (*RAIDs*): disk-organization techniques for performance and reliability.
- Past: RAIDs of small, cheap disks as cost-effective alternative to large, expensive disks.
- Today: RAIDs used for higher reliability and data-transfer rate.
- "I" in RAID: now stands for "independent" (was "inexpensive").

8.1 Improvement of reliability via redundancy

- Reliability of HDD RAID:
- Chance of some disk failing in N disks > chance of single disk failing.
- Example: **mean time between failures** (*MTBF*) of single disk = 100,000 hours.
- MTBF of some disk in 100-disk array = 1,000 hours (41.66 days).
- Storing only one copy: each disk failure $\implies$ significant data loss (unacceptable).
- Solution: introduce **redundancy** (store extra info not normally needed, used to rebuild lost info).
- Even if disk fails, data not lost.
- RAID applicable to NVM devices (less likely to fail than HDDs, no moving parts).
- Simplest (most expensive) redundancy: duplicate every drive (**mirroring**).
- Mirroring: logical disk = two physical drives; every write on both drives.
- Result: **mirrored volume**.
- One drive fails: data read from other. Data lost only if second drive fails before first replaced.
- MTBF of mirrored volume (failure = data loss) depends on:
  - MTBF of individual drives.
  - **mean time to repair**: average time to replace failed drive and restore data.
- Example: single drive MTBF = 100,000 hours, mean time to repair = 10 hours.
- **Mean time to data loss** of mirrored system: $(100,000^2)/(2 \times 10) = 500 \times 10^6$ hours (57,000 years).
- Cannot assume independent drive failures: power failures, natural disasters, manufacturing defects can cause correlated failures.
- Drives age: probability of failure grows, increasing chance of second failure during repair.
- Despite considerations: mirrored-drive systems offer much higher reliability.
- Power failures: particular concern (more frequent than natural disasters).
- Writes in progress to same block in both drives, power fails before both written: inconsistent state.
- Solution: write one copy first, then next; or add solid-state nonvolatile cache to RAID array.
- Write-back cache: protected from data loss during power failures (write considered complete).
- Assumes cache has error protection/correction (ECC, mirroring).

8.2 Improvement in performance via parallelism

- Parallel access to multiple drives: improves performance.
- Mirroring: read request rate doubled (reads sent to either drive).
- Transfer rate per read: same as single drive; but reads per unit time doubled.
- Multiple drives: improve transfer rate by **striping data**.
- **Data striping**: splitting data across drives.
- Simplest form: **bit-level striping** (splitting bits of each byte).
- Example: 8-drive array, bit i of each byte to drive i.
- Array of 8 drives: treated as single drive with 8x normal sector size, 8x access rate.
- Every drive participates in every access (read/write).
- Number of accesses per second: same as single drive.
- Each access: reads 8x data in same time.
- Bit-level striping generalized: number of drives multiple of 8 or divides 8.
- Example: 4-drive array, bits i and $4 + i$ to drive i.

- Striping not just bit level: **block-level striping** (blocks of file striped across multiple drives).
- With n drives, block i of file goes to drive $(i \pmod n) + 1$.
- Other striping levels possible (bytes of sector, sectors of block).
- Block-level striping: only commonly available striping.
- Parallelism in storage system (via striping) goals:
  1. Increase throughput of multiple small accesses (page accesses) by load balancing.
  2. Reduce response time of large accesses.

8.3 RAID levels

- Mirroring: high reliability, expensive.
- Striping: high data-transfer rates, no reliability improvement.
- Schemes for redundancy at lower cost: disk striping + "parity" bits.
- Different cost-performance trade-offs, classified as **RAID levels**.
- Figure 11.8.1: P indicates error-correcting bits, C indicates second copy of data.
- All cases in figure: 4 drives' worth of data stored; extra drives for redundant info.
- **RAID level 0**: drive arrays with block-level striping, no redundancy.
- **RAID level 1**: drive mirroring.
- **RAID level 4**: memory-style error-correcting-code (ECC) organization. Also in RAID 5 and 6.
  - Idea of ECC used directly in storage arrays via striping blocks across drives.
  - Example: 1st data block to drive 1, 2nd to drive 2, ..., Nth to drive N.
  - ECC calculation result stored on drive $N + 1$ (error-correction block).
  - One drive fails: ECC recalculation detects, prevents data pass, throws error.
  - RAID 4 corrects errors with one ECC block (drive controllers detect correct read).
  - If one sector damaged: know which sector, disregard data, use parity to recalculate.
  - For every bit: determine 1 or 0 by computing parity of corresponding bits from other drives.
  - Block read: accesses only one drive, others process requests.
  - Large reads: high transfer rates (all disks read in parallel).
  - Large writes: high transfer rates (data and parity written in parallel).
  - Small independent writes: not parallel.
  - OS write smaller than block: requires read-modify-write cycle.
  - **Read-modify-write cycle**: block read, modified, written back; parity block updated.
  - Single write: 4 drive accesses (2 read old, 2 write new).
  - WAFL uses RAID 4: allows seamless drive addition (added drives with zeros $\implies$ parity unchanged).
  - RAID 4 advantages over RAID 1 (equal data protection):
    - Storage overhead reduced: one parity drive for several regular drives (vs. one mirror for every drive).
    - Reads/writes of series of blocks spread over multiple drives (N-way striping): transfer rate N times faster than level 1.
  - Performance problem with RAID 4 (and parity-based RAIDs): expense of computing/writing XOR parity (slower writes).
  - Modern CPUs fast vs. drive I/O: minimal performance hit.
  - Many RAID storage arrays/HBAs: hardware controller with dedicated parity hardware (offloads computation).
  - Array has NVRAM cache: stores blocks while parity computed, buffers writes.
  - Buffering: avoids most read-modify-write cycles (gathers data into full stripe, writes concurrently).
  - Hardware acceleration + buffering: parity RAID almost as fast as non-parity RAID, often outperforms non-caching non-parity RAID.
- **RAID level 5**: block-interleaved distributed parity.
  - Spreads data and parity among all $N + 1$ drives (not parity in one drive).
  - For each set of N blocks: one drive stores parity, others store data.
  - Example: 5-drive array, parity for nth block stored in drive $(n \pmod 5) + 1$.
  - Parity block cannot store parity for blocks in same drive (failure $\implies$ data + parity loss).
  - Spreading parity: avoids overuse of single parity drive (RAID 4 problem).
  - RAID 5: most common parity RAID.
- **RAID level 6**: **P + Q redundancy scheme**.

  - Like RAID 5, but stores extra redundant info for multiple drive failures.
  - XOR parity not used on both parity blocks (would be identical).
  - Uses error-correcting codes (e.g., **Galois field math**) to calculate Q.
  - Example: 2 blocks redundant data for every 4 blocks data (vs. 1 parity block in level 5).
  - System tolerates two drive failures.
- **Multidimensional RAID level 6**: amplifies RAID level 6.
  - Array with hundreds of drives: RAID 6 stripe $\implies$ many data drives, two logical parity drives.
  - Logically arranges drives into rows and columns (2+ dimensional arrays).
  - Implements RAID 6 horizontally (rows) and vertically (columns).
  - System recovers from any/multiple failures using parity blocks in any location.
  - Figure 11.8.1(f): shows dedicated parity drives (in reality, scattered).
- **RAID levels 0 + 1 and 1 + 0**: combinations.
  - RAID 0 + 1: RAID 0 (performance) + RAID 1 (reliability).
  - Better performance than RAID 5, common where both important.
  - Doubles drives needed (expensive).
  - Set of drives striped, then stripe mirrored to equivalent stripe.
  - RAID 1 + 0: drives mirrored in pairs, then mirrored pairs striped.
  - Theoretical advantages over RAID 0 + 1.
  - RAID 0 + 1 single drive failure: entire stripe inaccessible, only other stripe left.
  - RAID 1 + 0 single drive failure: single drive unavailable, but mirror available, rest of drives available.
- Numerous variations to basic RAID schemes exist.
- Confusion about exact definitions of RAID levels.
- Implementation of RAID:
  - Volume-management software: implements RAID in kernel or system software layer. Storage hardware minimal features.
  - HBA hardware: implements RAID. Only directly connected drives part of RAID set. Low cost, not flexible.
  - Storage array hardware: creates RAID sets, slices into volumes. OS only implements file system on volumes. Arrays can have multiple connections/part of SAN.
  - SAN interconnect layer: implements RAID via drive virtualization devices. Device sits between hosts/storage, manages access. Provides mirroring by writing to two devices.
- Other features (snapshots, replication) implemented at these levels.
- **Snapshot**: view of file system before last update.
- **Replication**: automatic duplication of writes between separate sites (redundancy, disaster recovery).
- Replication: synchronous (block written locally/remotely before complete) or asynchronous (writes grouped, written periodically).
- Asynchronous replication: data loss if primary site fails, but faster, no distance limitations.
- Replication increasingly used within data center/host.
- Alternative to RAID: protects against data loss, increases read performance (reads from replicas). Uses more storage than most RAID.
- Implementation differences: software RAID $\implies$ each host manages replication; storage array/SAN interconnect $\implies$ host data replicated regardless of OS.
- Hot spare drive(s): **hot spare** not used for data, configured as replacement for drive failure.
- Example: rebuild mirrored pair if one drive fails.
- RAID level reestablished automatically, without waiting for replacement.
- More than one hot spare: more than one failure repaired without human intervention.

8.4 Selecting a RAID level

- System designers/administrators choose RAID level based on considerations.
- Rebuild performance: time to rebuild data if drive fails.
- Important if continuous data supply needed (high-performance/interactive databases).
- Rebuild performance influences MTBF.
- Rebuild performance varies with RAID level.
- Easiest for RAID 1 (data copied from another drive).
- Other levels: need to access all other drives to rebuild.
- Rebuild times: hours for large RAID 5 sets.

- RAID level 0: used in high-performance apps where data loss not critical (e.g., scientific computing).
- RAID level 1: popular for high reliability with fast recovery (e.g., small databases).
- RAID 0 + 1 and 1 + 0: used where both performance and reliability important (e.g., small databases).
- RAID 1: high space overhead.
- RAID 5: often preferred for moderate data volumes.
- RAID 6 and multidimensional RAID 6: most common in storage arrays (good performance/protection, no large space overhead).

8.5 The InServ storage array

- Innovation blurs lines between technologies.
- InServ storage array (HP 3Par):
- Does not require drives configured at specific RAID level.
- Each drive broken into 256-MB "chunklets".
- RAID applied at chunklet level.
- Drive participates in multiple/various RAID levels (chunklets used for multiple volumes).
- Provides snapshots (similar to WAFL file system).
- InServ snapshots: read-write and read-only.
- Allows multiple hosts to mount copies of file system without own copies.
- Host changes in own copy: copy-on-write, not reflected in other copies.
- Further innovation: **utility storage**.
- Some file systems: do not expand/shrink (original size only, changes require copying data).
- Administrator configures InServ: provide host with large logical storage, initially small physical storage.
- Host uses storage: unused drives allocated up to original logical level.
- Host believes large fixed storage space, creates file systems.
- Drives added/removed by InServ without file system noticing.
- Feature: reduces drives needed by hosts, delays drive purchase.

8.6 Extensions

- RAID concepts generalized to other storage devices: arrays of tapes, broadcast data over wireless.
- Arrays of tapes: recover data even if one tape damaged.
- Broadcast data: block split into short units, broadcast with parity unit.
- If unit not received: reconstructed from other units.
- Tape-drive robots: stripe data across drives (increase throughput, decrease backup time).

8.7 Problems with RAID

- RAID does not always assure data availability.
- Problems not protected by RAID: wrong file pointer, wrong pointers within file structure, incomplete writes ("torn writes"), accidental overwrite of file system structures.
- RAID protects against physical media errors, not other hardware/software errors.
- Hardware RAID controller failure or software RAID bug: total data loss.
- Numerous potential perils for data due to software/hardware bugs.
- **Solaris ZFS** file system: innovative approach using checksums.
- ZFS: maintains internal checksums of all blocks (data, metadata).
- Checksums: not kept with checksummed block, stored with pointer to block.
- Example: **inode** (data structure for file system metadata) with pointers to data.
- Inode contains checksum of each data block.
- Problem with data: checksum incorrect, file system knows.
- Data mirrored, one block correct checksum, one incorrect: ZFS automatically updates bad block with good one.
- Directory entry pointing to inode: has checksum for inode.
- Problem in inode: detected when directory accessed.
- Checksumming throughout ZFS structures: higher consistency, error detection, error correction than RAID/standard file systems.
- Extra overhead (checksum calculation, extra block read-modify-write): not noticeable (ZFS overall performance very fast).
- Similar checksum feature: Linux BTRFS file system.

- Another issue with most RAID implementations: lack of flexibility.
- Example: 20 drives, four 5-drive RAID 5 sets $\implies$ four separate volumes.
- Problem: file system too large for 5-drive set? Another needs little space?
- If not known ahead: drive use/requirements change over time.
- Even if 20 drives as one large RAID set: other issues.
- Several volumes of various sizes built on set.
- Some volume managers: do not allow changing volume size.
- Mismatched file-system sizes.
- Some volume managers allow size changes, but some file systems don't allow growth/shrinkage.
- Volumes change size, but file systems need recreation.
- ZFS: combines file-system management and volume management.
- Drives/partitions of drives: gathered via RAID sets into **pools** of storage.
- Pool: holds one or more ZFS file systems.
- Entire pool's free space: available to all file systems within that pool.
- ZFS uses memory model of `malloc()` and `free()` to allocate/release storage.
- No artificial limits on storage use, no need to relocate/resize.
- ZFS provides quotas (limit size) and reservations (assure growth).
- Variables changed by file-system owner at any time.
- Other systems (Linux): volume managers allow logical joining of multiple disks for larger volumes.

8.8 Object storage

- General-purpose computers: typically use file systems.
- Another approach: start with storage pool, place objects in pool.
- Differs from file systems: no way to navigate pool and find objects.
- Object storage: computer-oriented, designed for programs.
- Typical sequence:
  1. Create object in storage pool, receive object ID.
  2. Access object via object ID.
  3. Delete object via object ID.
- Object storage management software (**Hadoop file system** (*HDFS*), **Ceph**): determines where to store objects, manages protection.
- Typically on commodity hardware, not RAID arrays.
- Example: HDFS stores copies of object on different computers.
- Lower cost than storage arrays, fast access to object (on systems with copy).
- All systems in Hadoop cluster access object, but only systems with copy have fast access.
- Computations on data occur on those systems, results sent across network.
- Other systems: need network connectivity to read/write object.
- Object storage: usually for bulk storage, not high-speed random access.
- Advantage: **horizontal scalability**.
- Storage array: fixed max capacity.
- Object store: add capacity by adding more computers with internal/external disks to pool.
- Object storage pools: can be petabytes.
- Key feature: each object self-describing (includes content description).
- Also known as **content-addressable storage**: objects retrieved based on contents.
- No set format for contents: **unstructured data**.
- Not common on general-purpose computers, but huge amounts of data stored in object stores.
- Examples: Google search contents, Dropbox, Spotify songs, Facebook photos.
- Cloud computing (Amazon AWS): generally uses object stores (S3) to hold file systems/data objects for customer apps.

Term	Definition
redundant arrays of independent disks (RAID)	Disk organization technique: two or more storage devices work together, usually with pro tection from device failure.
mean time between failure (MTBF)	Statistical mean time a device is expected to work correctly before failing.
mirroring	Storage RAID protection: two physical devices contain same content; if one fails, read from other.
mirrored volume	Volume in which two devices are mirrored.
mean time to repair	Statistical mean of time to repair a device.
mean time to data loss	Statistical mean of time until data is lost.
data striping	Splitting of data across multiple devices.
bit-level striping	Splitting of data at bit level, each bit stored on separate device.
block-level striping	Splitting of data at block level, each block stored on separate device.
RAID levels	Various types of RAID protection.
read-modify-write cycle	Write of data smaller than block requires entire block to be read, modified, and written back.
Galois field math	Advanced error-correcting calculation in some RAID forms.
snapshot	In file systems, a read-only view of a file system at a particular point in time.
replication	In file systems, duplication and synchronization of data over network to another system.
hot spare	Unused storage device ready to be used to recover data (e.g., in RAID set).
utility storage	InServ feature: storage space can be increased as needed.
Solaris ZFS	Advanced file system, first included as part of Solaris.
inode	In many file systems, a per-file data structure holding most of the metadata of the file.
pool	In ZFS, drives, partitions, or RAID sets that can contain one or more file systems.
Hadoop file system	Example of object storage management software.
Ceph	Brand of object storage management software.
horizontal scalability	Ability to scale capacity by adding more items (computers) rather than expanding one.
content addressable storage	Another term for object storage; objects retrieved based on their contents.
unstructured data	Data not in a fixed format but rather free-form.

9 Summary

- Hard disk drives (HDDs) and nonvolatile memory (NVM) devices: major secondary storage I/O units.
- Modern secondary storage: structured as large one-dimensional arrays of logical blocks.
- Drives attached to computer:
  1. Through local I/O ports on host.
  2. Directly connected to motherboards.
  3. Through communications network or storage network connection.
- Requests for secondary storage I/O: generated by file system and virtual memory system.
- Each request: specifies device address as logical block number.
- Disk-scheduling algorithms: improve HDD effective bandwidth, average response time, variance in response time.
- Algorithms (SCAN, C-SCAN): improve via disk-queue ordering strategies.
- HDD performance: varies greatly with scheduling algorithms.
- Solid-state disks (SSDs): no moving parts, performance varies little among algorithms.
- SSDs: often use simple FCFS strategy.
- Data storage/transmission: complex, frequently result in errors.
- **Error detection**: attempts to spot problems, alert system for corrective action, avoid error propagation.
- **Error correction**: detects and repairs problems (depends on correction data, corruption amount).
- Storage devices: partitioned into one or more chunks of space.
- Each partition: can hold a volume or be part of a multidevice volume.
- File systems: created in volumes.
- OS manages storage device's blocks.
- New devices: typically pre-formatted.
- Device partitioned, file systems created.
- Boot blocks: allocated to store system's bootstrap program (if device contains OS).
- Block/page corrupted: system must lock out or logically replace with spare.
- Efficient swap space: key to good performance in some systems.
- Some systems: dedicate raw partition to swap space.
- Others: use file within file system.
- Still others: provide both options (user/admin decision).
- Large systems storage: secondary storage devices frequently made redundant via RAID algorithms.
- RAID algorithms: allow more than one drive for operation, allow continued operation/automatic recovery from drive failure.
- RAID algorithms: organized into different levels (each provides reliability/high transfer rates combination).
- **Object storage**: used for big data problems (e.g., Internet indexing, cloud photo storage).
- Objects: self-defining collections of data, addressed by object ID (not file name).
- Typically uses replication for data protection.
- Computes based on data: on systems where copy of data exists.
- Horizontally scalable: for vast capacity and easy expansion.