DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

COMPUTER ARCHITECTURES

Mass Storage Devices

Gábor Horváth

BUTE Department of Networked Systems and Services
ghorvath@hit.bme.hu

Budapest,
2023. 03. 07.

MŰEGYETEM 1782

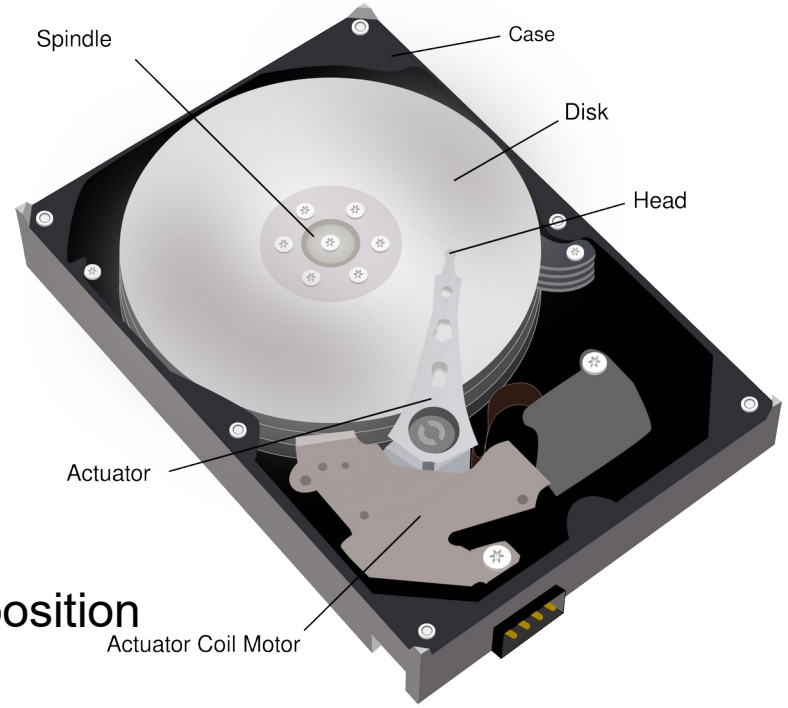
DATA STORAGE DEVICES

- All computers have data storage devices
- Their performance impacts the overall performance of the system
- They have a crucial role in virtual memory management
- We are going to cover:
  - HDD: **Hard disk drives**
  - SSD: **Solid state drives**
- There are others as well:
  - Optical drives: similar to HDDs at several aspects
  - Pendrives: are based on the same flash memory technology as SSDs
  - Etc.

Hard Disk Drives

DATA STORAGE ON ROTATING MEDIA

- Parts:
  - Disk(s)
    - 1 or more data recording surfaces
    - Stores the data
      → with magnetic signals
  - Head
    - Reads/writes the data
- Data access:
  1) The head is moved to the desired radial position
     - Called: **seek**
  2) The disk is rotated to the desired angular position
  3) Data is read/recorded from/to the data recording surface
     - HDD → in a magnetic way
     - CD/DVD/BR → in an optical way

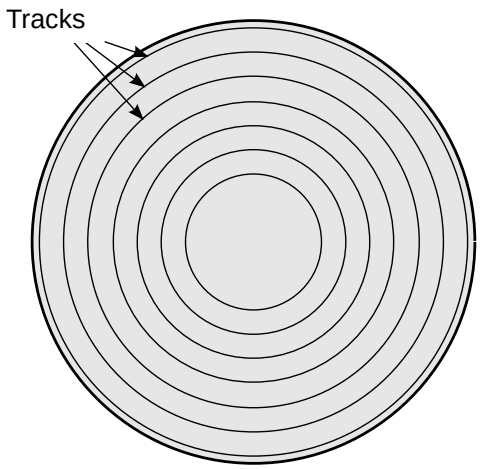


- https://youtu.be/3owqvmMf6No

DATA ORGANIZATION

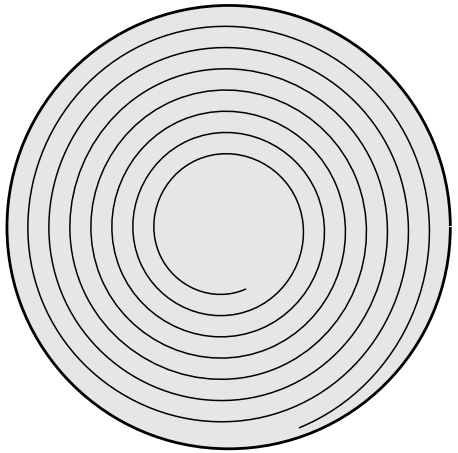
- Dominating solution: fix sized blocks
  → **sector** (typically 512 bytes)
- Referring to sectors: with linear addresses (from 0)
  - **Logical Block Addressing (LBA)**
  - The disk maps it to its own geometry

Hard disk drives:



- The organization of sectors:
  - On hard disk drives:
    - In concentric circles
      → called: **track**
    - Disk rotate with
      CAV: Constant Angular Velocity
  - On optical drives:
    - Spiral placement of sectors
    - Rotation: along the spiral line, with
      CLV: Constant Linear Velocity

Optical drives:

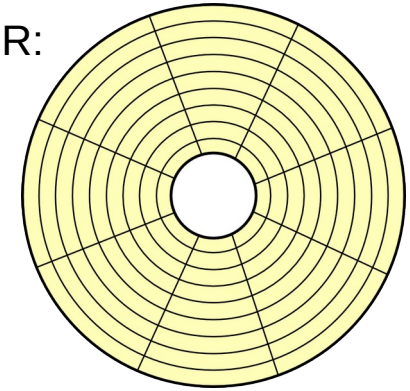


ZONED-BIT RECORDING

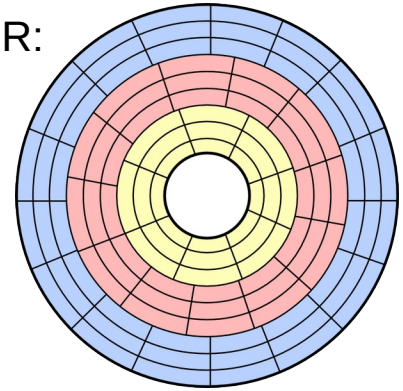
- **The same number of sectors per track** for the entire disk:
  - The reading time in always the same for every sector
  - The recording density is higher in the inner tracks → a waste of space in the outer tracks
- **The recording density is the same** for the entire disk:
  - Increasing number of sectors towards the outer tracks
  - Sectors need to be read faster towards the outer tracks
  - Chaning reading speed in every track is not feasible
- Compromise: don't change speed in every track, only at zone boundaries

  → **ZBR, Zoned Bit Recording**
  - In a zone, the number of sectors is the same in every track
  - In a zone, the reading speed of one sector is the same in every track

Without ZBR:

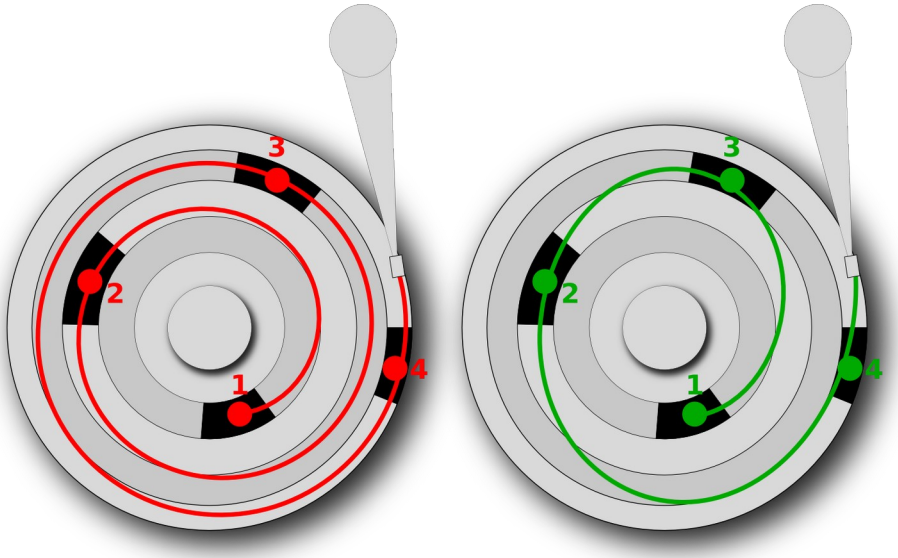


With ZBR:



PERFORMANCE OF HDDS

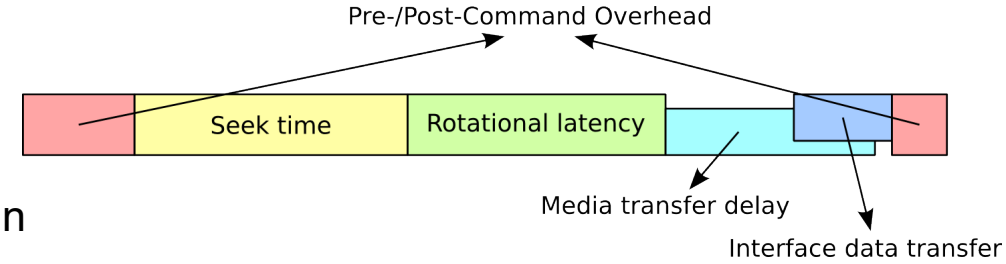
- We would like to transfer a sequence of sectors
  - The request is first put to a buffer (a queue)
    - Old times: the buffer was managed by the op. system
    - Nowadays: disk manages the buffer → **command queueing**
      - In the order which is the most convenient (and the fastest) for the disk



First-come-first-served strategy Optimal order or service

  - After taking out a request from the buffer, the disk serves it

PERFORMANCE OF HDDS



- **Service time of a request**
  - Has 5 components:
- Example:
  - Rotation speed: 10.000 rotation/min
  - Average seek time: 4.5 ms
  - Pre-/post-command delay: 0.3 ms
  - In the track accessed, there are 600 sectors containing 512 bytes each
  - Speed of the interface: 100 MB/s
- How long does it take to read and transmit a 4 kB block?
  - Rotational latency: 1 rotation: 6 ms.
    → At a random point of time, we need to wait 3 ms for the rotation on average
  - Reading from media: 4 kB = 8 sector. Reading 1 sector: 6 ms / 600 = 0.01 ms.
  - Data transmission on the interface: Can be overlapped! Media reading time + the transmission of the last sector: 512 byte / 100 MB/s = 0.005 ms
  - In total: 0.3 + **4.5** + **3** + 0.08 + 0.005 = **7.885 ms**
- The rotational latency and the seek time dominates!
- **It is worth transmitting a large amount of data at once!**
- This way the seek time and the rotational time count only once.

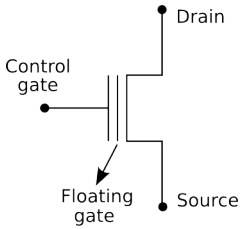
PERFORMANCE OF HDDS

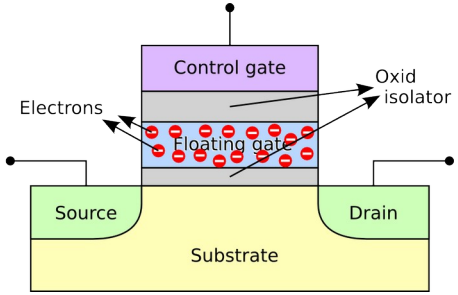
- **Data speed (throughput)**
  - Two different definitions:
    - Measured in requests: IOPS: [requests served/sec]
    - Measured in bytes: IOPS x average data amount transmitted [byte/sec]
  - Measuring it with small request sizes: **random throughput**
    - Seek time and rotational latency dominates
  - Measuring it with large, continuous blocks: **sequential throughput**
    - The media transfer delay dominates
- Example:
  - Random throughput (4 kB block size):
    - We have calculated before: 7.885 ms/request
      → 1000 ms/s ⁄ 7.885 ms/req = 126.823 IOPS/s
      → 126.823 IOPS/s * 4096 byte/req ≈ 0.52 MB/s → **Very slow!**
  - Sequential throughput (16 MB block size):
    - Like before, the service time is: 335.485 ms/request
      → 1000 ms/s ⁄ 335.485 ms/req = 2.98 IOPS/s
      → 2.98 IOPS/s * 16 MB/req ≈ 50 MB/s → **Much faster!**

Solid State Drives (SSD)

STORING A SINGLE BIT

- Based on floating gate transistors
  - The floating gate can store electrons
    - Electrons remain there permanently (without power)
    - Their presence change the behavior of the transistor
- Representing data:
  - 1 transistor → 1 bit (for a moment!)
  - Presence of electrons: **0**, no electrons: **1**
- Changing a bit from 1 to 0: **programming**
  - We push electrons to the floating gate
- Changing from 0 to 1: **erasing**
  - Removing electrons from the floating gate
- Programming and erasing is a big stress for the transistor
- **A side effect of programming and erasing:**
  - Some electrons may stuck into the isolator when going in and out (and they remain there forever)
  - The isolator will not isolate that well any more
  - **Ageing!**



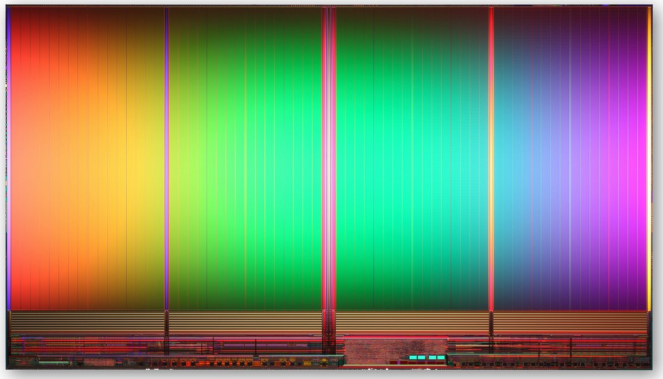


STORING SOME BITS

- So far we had 2 states: presence of electrons → 0, no electrons → 1
- Why not to distinguish multiple levels of charge as well?
  - **To store *N* bit in a single transistor 2^N charge levels have to be distinguished**
    - SLC flash: 1 transistor → 1 bit
    - MLC flash: 1 transistor → more bits
  - Terminology as used nowadays:
    - SLC: N=1 – fastest, most expensive
    - MLC: N=2
    - TLC: N=3
    - QLC: N=4 – slowest, cheapest
- The greater N is, the faster is the ageing effect! Tolerated number of programming/erase cycles:
  - SLC: ≈ 100.000
  - MLC: ≈ 10.000
  - TLC: ≈ 1.000 – 3.000
  - QLC: ≈ 100 – 750 (!)
  - Example: Intel SSD 670p 1TB, QLC, TBW = 320TB (Terabytes Written)

STORING A LARGE NUMBER OF BITS

- Transistors are arranged into a 2D grid
- But how?
- **NOR flash:**
  - Responds to byte level read/write requests
  - Typical application area: intruction memory (BIOS, firmware, etc.)
- **NAND flash:**
  - Responds to page level read/write requests
  - Unit of read and write requests: **pages**
    - Page: one row in the transistor grid
  - Name of the whole grid: **block**
  - **Pages can not be erased individually, only the whole block**
  - Storage hierarchy:
    - 1 floating gate transistor: 1 – 4 bits
    - 1 page: 512 bytes – 8 kB
    - 1 block: 128 – 256 pages
    - 1 plane: 1024 blocks
    - 1 silicon chip: 1 – 4 plane
    - 1 IC: 1 – 4 silicon chips



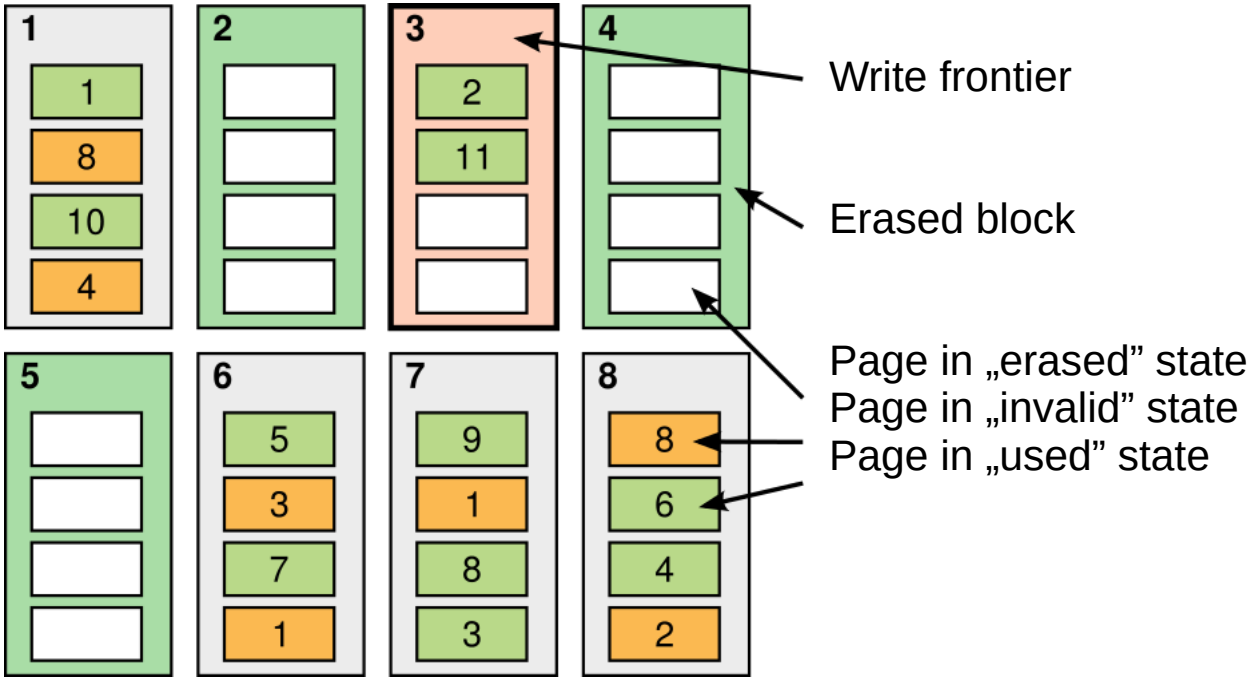
Intel 8 GB 2-bit MLC NAND flash memory

SECTOR LEVEL OPERATIONS

- **Reading**
  - Translation: LBA address → physical location, readout, done.
- **Writing**
  - HDD: Saving a sector several times → HDD overwrites the old one
  - SSD is not able to overwrite pages!!!
  - Changed pages are put onto a new, unused block, and the outdated data is marked as invalid
  - Creating new, unused blocks: by erasing blocks that contain no valid data (can be as large as 2 MB!)
  - A state is associated to each page:
    - „Used”
    - „Invalid”
    - „Erased”

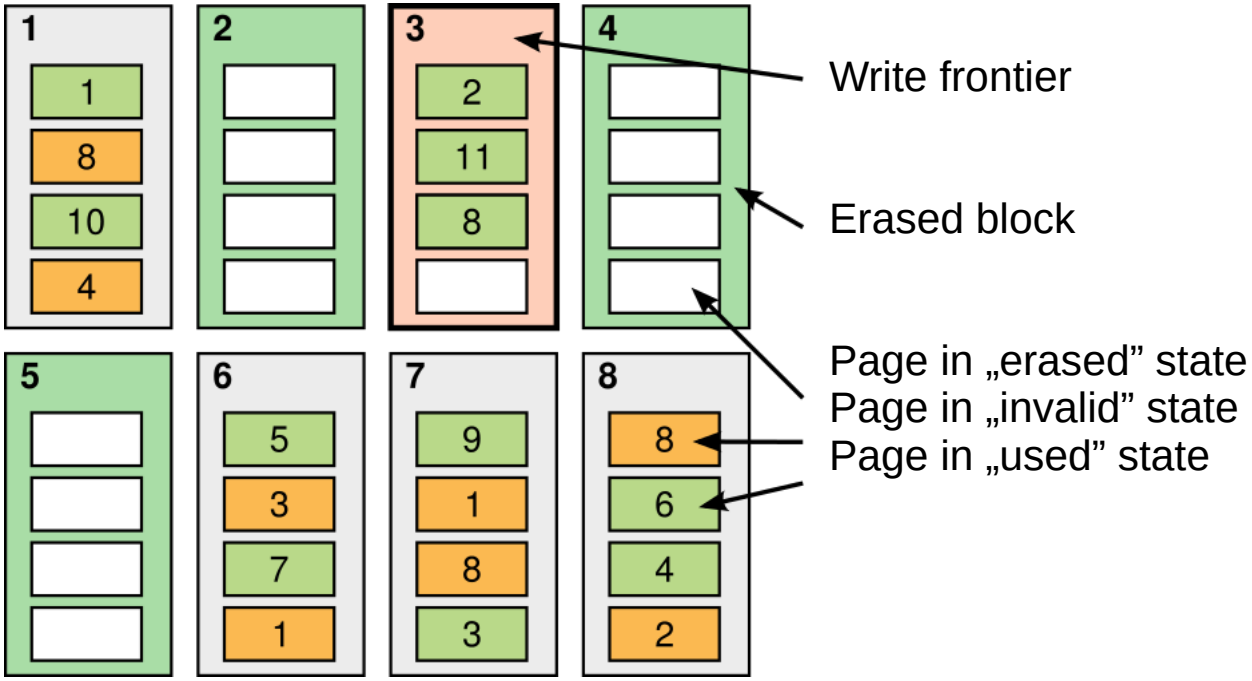
PAGE MANAGEMENT

- Every write operation goes to the **write frontier**
- When the write frontier gets full, a new is selected among the erased blocks
- Example: write requests to LBA addresses 8, 12, 1



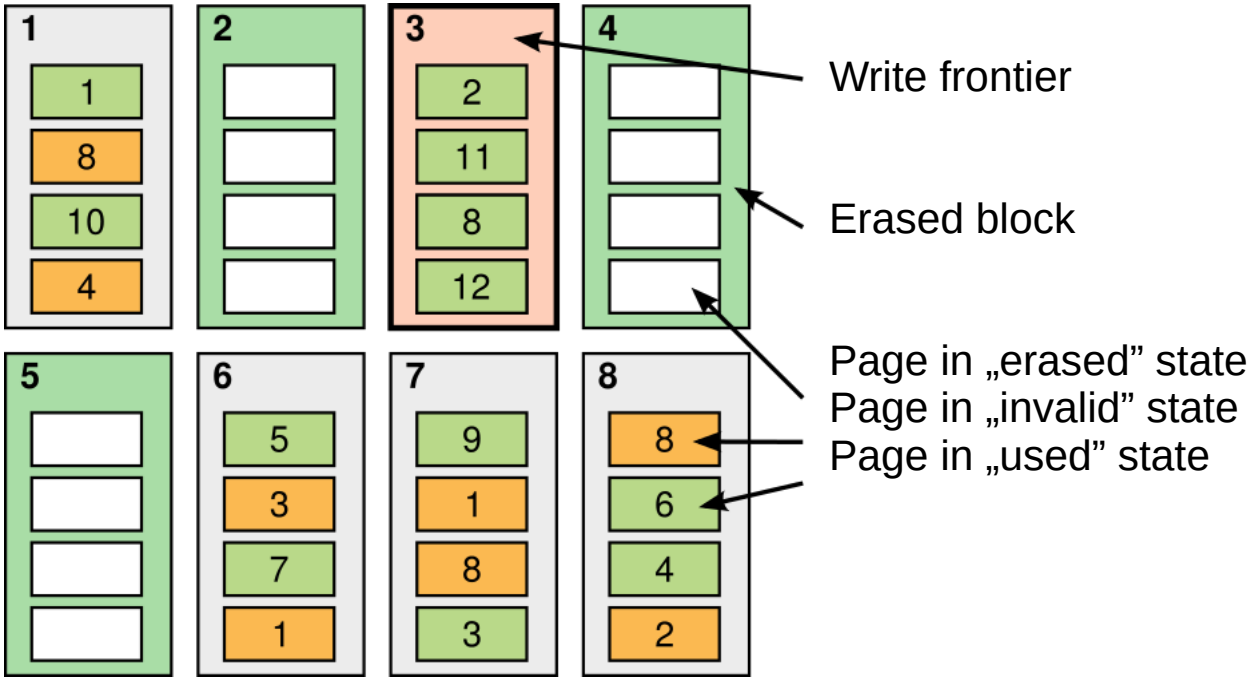
PAGE MANAGEMENT

- Every write operation goes to the **write frontier**
- When the write frontier gets full, a new is selected among the erased blocks
- Example: write requests to LBA addresses **8**, 12, 1



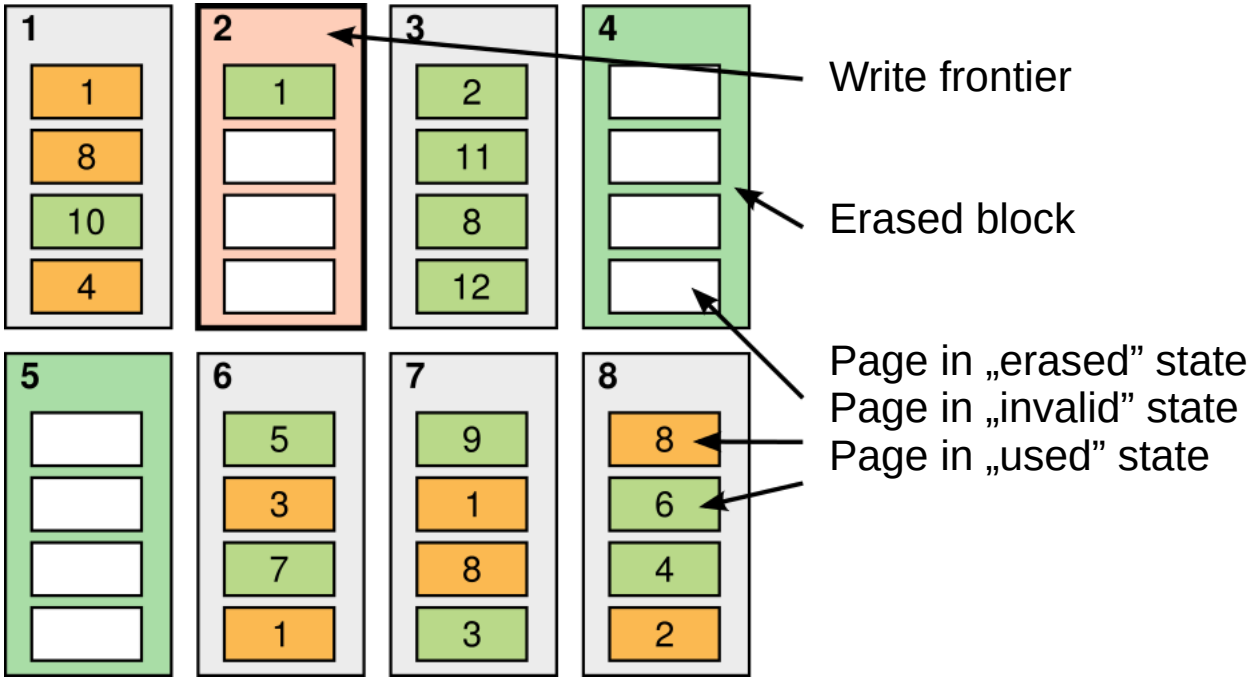
PAGE MANAGEMENT

- Every write operation goes to the **write frontier**
- When the write frontier gets full, a new is selected among the erased blocks
- Example: write requests to LBA addresses 8, **12**, 1



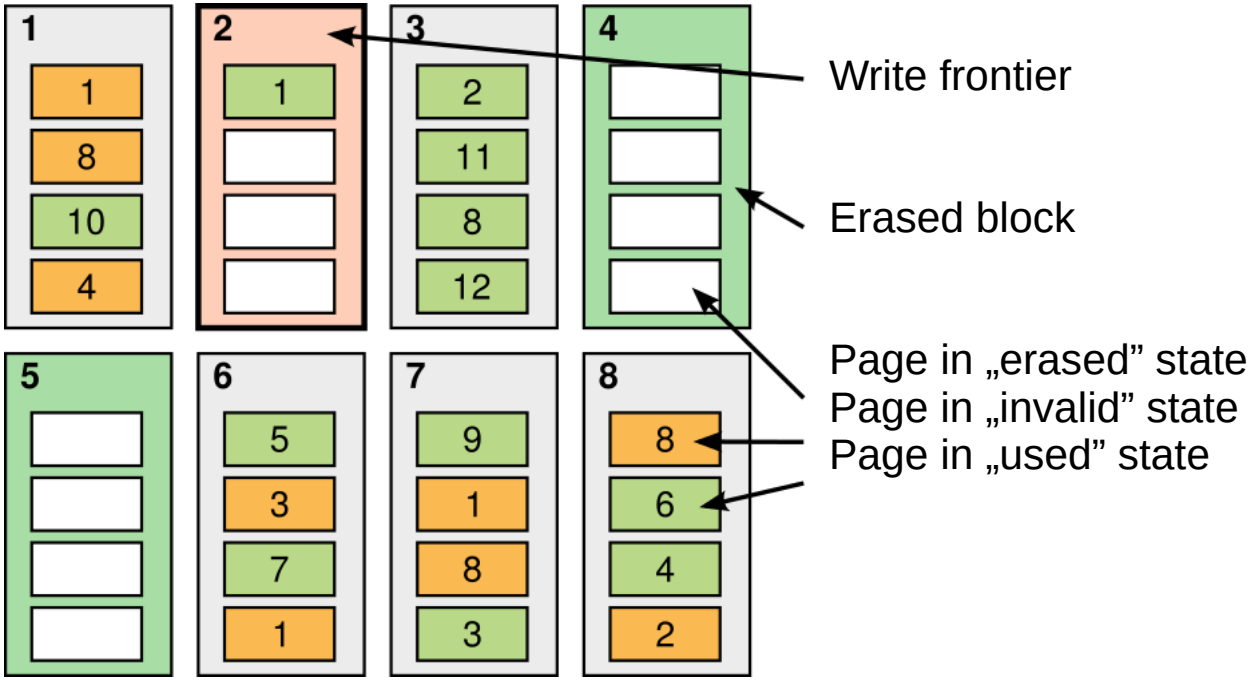
PAGE MANAGEMENT

- Every write operation goes to the **write frontier**
- When the write frontier gets full, a new is selected among the erased blocks
- Example: write requests to LBA addresses 8, 12, **1**



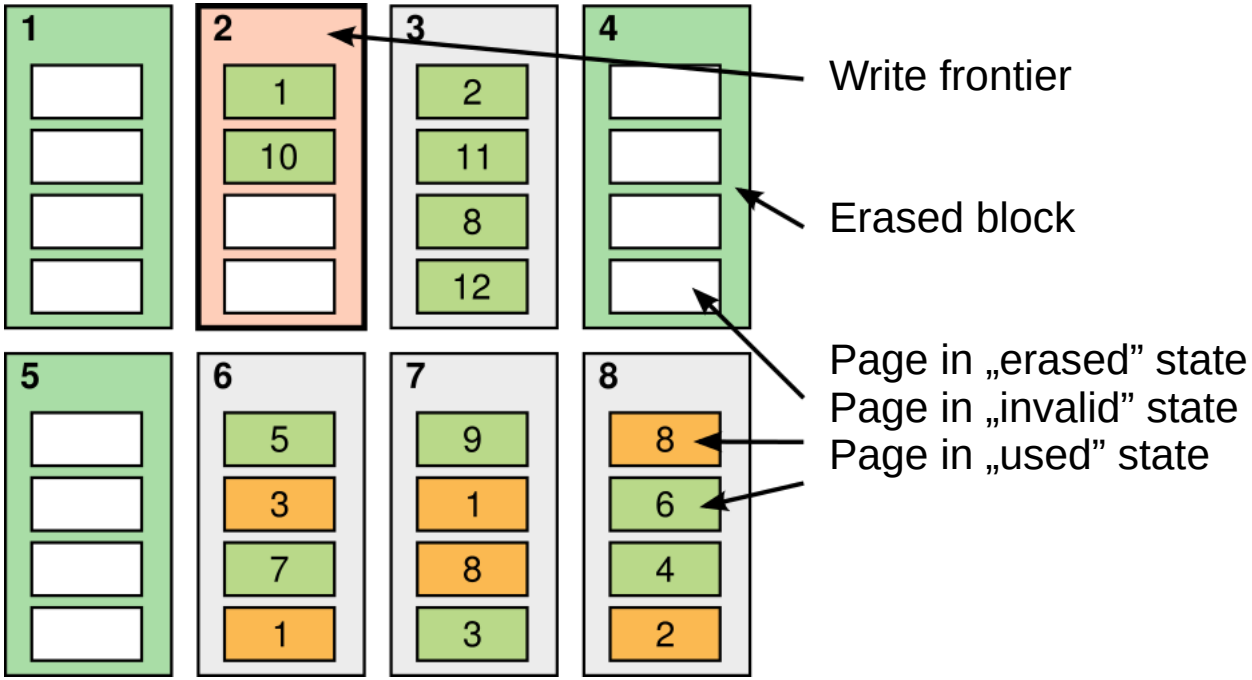
PAGE MANAGEMENT

- When the SSD runs out of erased blocks → **garbage collection**
  - A block is selected
  - The pages in „Used” state are copied to the write frontier
  - Erases the block
- Example:



PAGE MANAGEMENT

- When the SSD runs out of erased blocks → **garbage collection**
  - A block is selected
  - The pages in „Used” state are copied to the write frontier
  - Erases the block
- Example:



PAGE MANAGEMENT

- Consequences:
  - **The SSD drive gets slower with time!**
  - When it is brand new, it does not have to run the garbage collection
- Solution of modern SSDs:
  - *They run garbage collection in the idle time*
    - Doing garbage collection too frequently – makes the SSD older!
    - Effective only if there is idle time! Under high load (eg. in a server), the garbage collection slows down SSDs drastically
  - *Wear leveling*
    - To keep the age of each block approximately the same
  - *Data compression*
    - If the data is compressed, we have to write less (slows down ageing)
  - *Waiting for more write request to come*
    - Hoping that 1-2 further write requests arrive affecting the same page
    - These can be executed in a single write operation instead of multiple
- Consequence: *Write amplification*
  - The drive writes more than we request → **faster aging!**

Storage Arrays

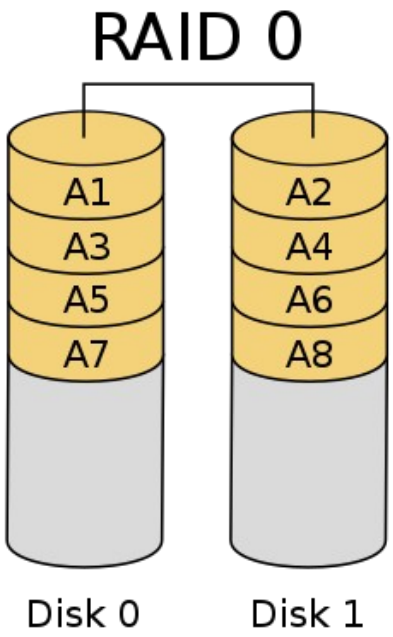
MOTIVATION

- SLED: Single Large Expensive Disk – Big storage capacity for big money
- Idea: Let us use many cheap disks and merge them virtually
- Issue: dependability
  - MTTF: Mean Time To Failure
  - If MTTF = 100.000 hours for 1 disk,
    then for an array of 100 disks MTTF = 100.000/100 = 1000 hours
    → there is a failure every 1000 hour in average!
- Solution: redundancy
- Let us give up some disk capacity for reliability
- This is RAID – Redundant Array of { Inexpensive | Independent } Disks
- Depending on how redundancy is implemented:
  - **RAID 0**, **RAID 1**, **RAID 2**, **RAID 3**, **RAID 4**, **RAID 5**, **RAID 6**
  - Combinations: RAID 10, RAID 0+1

RAID 0

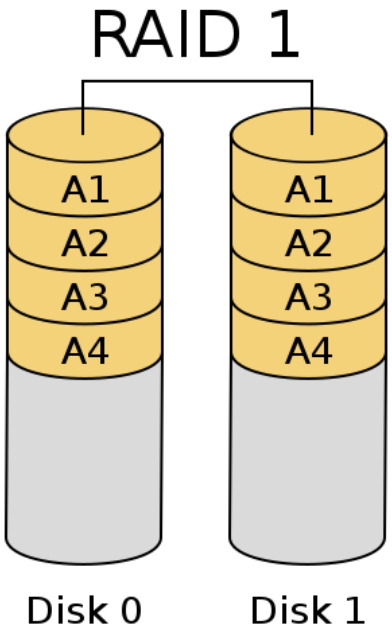
- RAID 0: pooling disks to increase capacity without redundancy
- ...in a special way: **striping**

  → Placing blocks alternately between discs
- Benefits:
  - Large storage capacity
    (capacity offered = sum of capacity of disks)
  - High throughput – load is shared between disks
  - Simple implementation
- Drawbacks:
  - Low availability (MTTF)
  - Even 1 disk failure is catastrophic



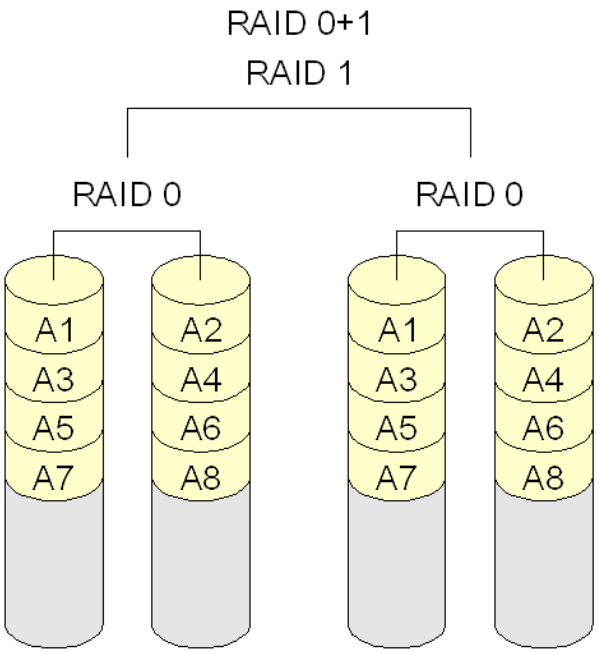
RAID 1

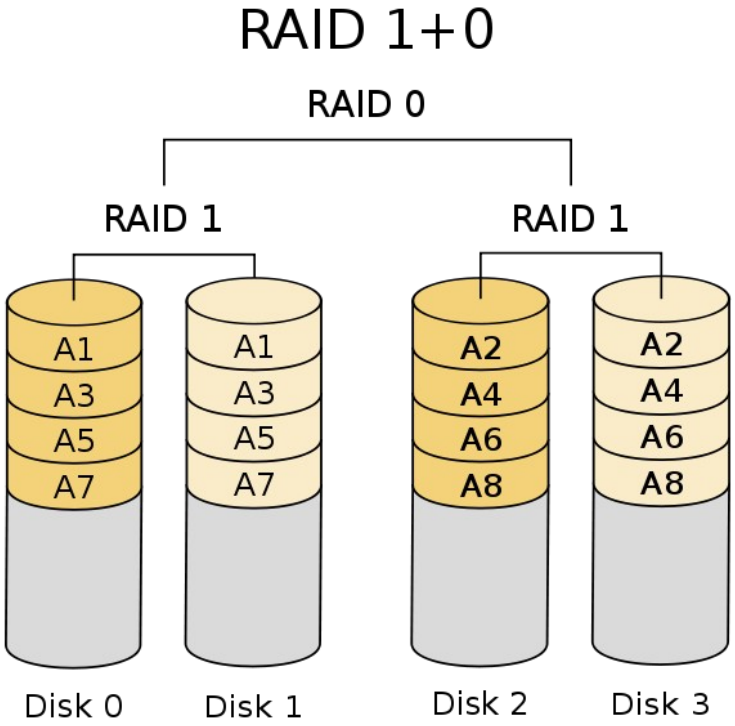
- RAID 1: mirroring data across multiple disks
- Advantages:
  - Read operations: N times the transfer speed
  - Write operations: as 1 disk (not faster)
  - Simple
- Disadvantages:
  - High (N times) overhead.
    → N disk capacity = 1 offered capacity



COMBINATIONS OF RAID 0 AND RAID 1

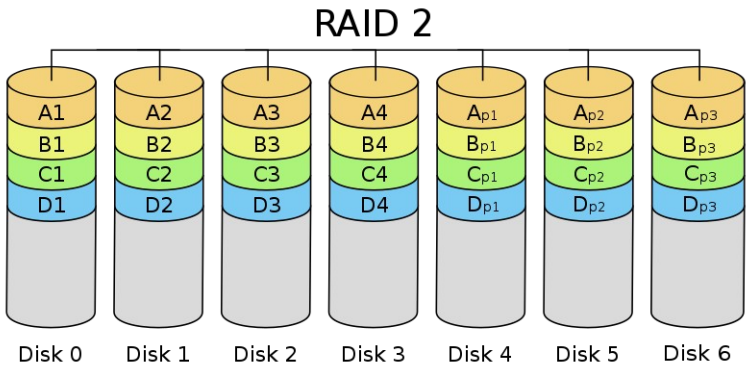
- RAID 0+1 and RAID 1+0 (=10)
- RAID 1+0 is the better choice, as failures affect only the RAID 1 block



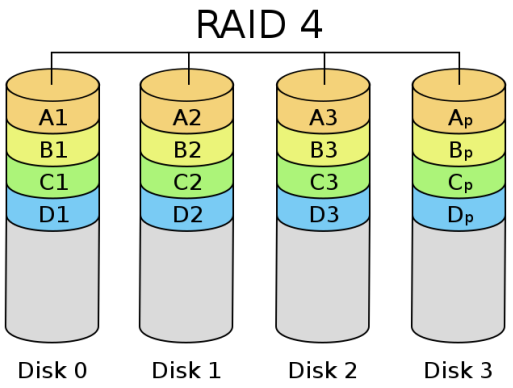


ERROR CORRECTION WITH LESS REDUNDANCY

- Mirroring: waste of capacity
- There are good error-correcting codes (see: information theory)
- E.g. Hamming code
  - (7, 4) encoding 4-bit messages into 7 bits any bit can be corrected
  - Less overhead can be achieved
  - RAID-2 basis
    - but no commercial implementation

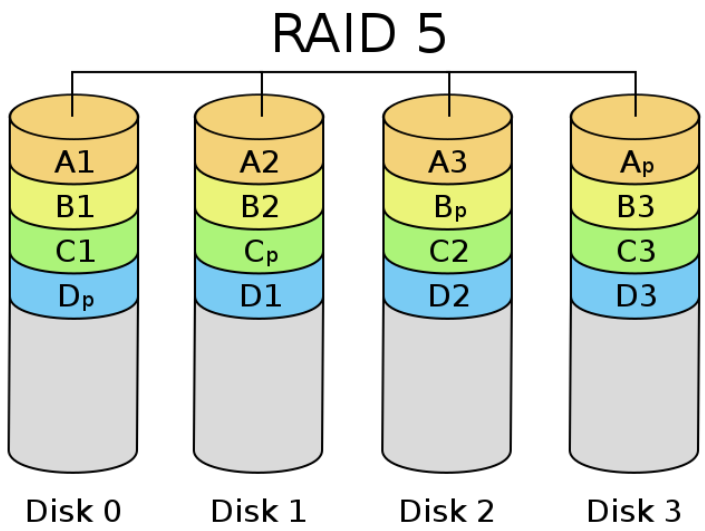


- Parity bit:
  - Not capable of error correction, only error indication
  - Unless we know the location of the error!
  - Because then the parity bit can fix 1 error
  - And disks tell you when an operation fails
- RAID-3 and 4
  - Dedicated parity disk (bottleneck)
  - Difference between 3 and 4: different data units



RAID 5

- RAID 5: error correction with parity bit
- N blocks of data stored on N+1 disks → the +1 is the parity
- Error correction:
  - We know when a disk fails
  - All blocks can be recovered from the other disks
    - Even if the parity disk fails
- Advantages:
  - Parity information is distributed across disks
    - No extra load on one disk
      → would be a bottleneck
  - Read operations: high throughput
    - N disks working in parallel
- Disadvantages:
  - Complex hardware implementation
  - Data overwriting: slower
    - Overwrite - read all disks - calculate parity - update parity
  - No protection against another error during restore
    - And for large disks, this is a real danger

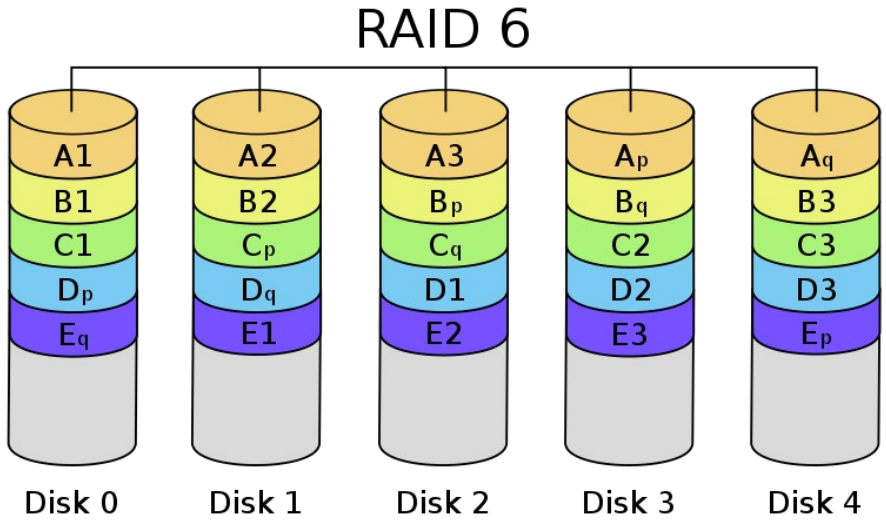


RAID 6

- RAID 6: like RAID 5, but with two parity bits
- N blocks of data stored on N+2 disks → the +2 are the parity

- Advantages:
  - Read operations: high throughput (parallel access to disks)
  - Protects against failure of two disks
    - There is protection during recovery

- Disadvantages:
  - Higher overhead
  - Overwrite: even slower (see RAID-5)
  - Complexity



CONTROLLERS

- Power failure while writing
  - RAID Write Hole Effect: Inconsistent state, array cannot be recovered
  - Solution of hardware RAID controllers:
    - Write cache protected by battery
- Hot Swap
  - Disk swap without shutting down the system
- Hot Spare
  - Spare disk in the RAID array
  - In case of failure, recovery starts immediately
    - No need to wait for the operator

DEPARTMENT OF
NETWORKED SYSTEMS
AND SERVICES