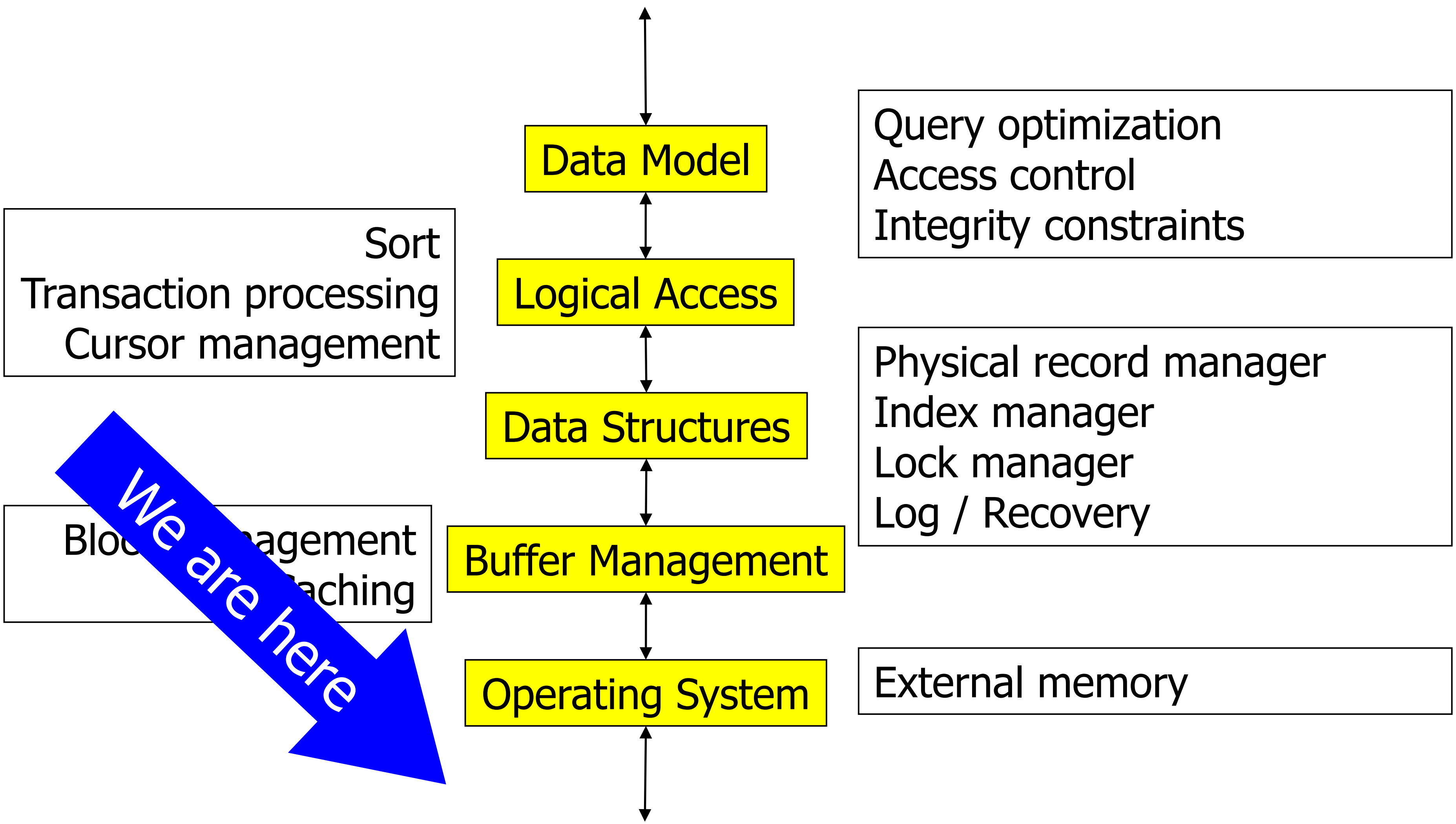
# Datenbanksysteme II: Storage, Discs, and Raid

Ulf Leser

# Tasks

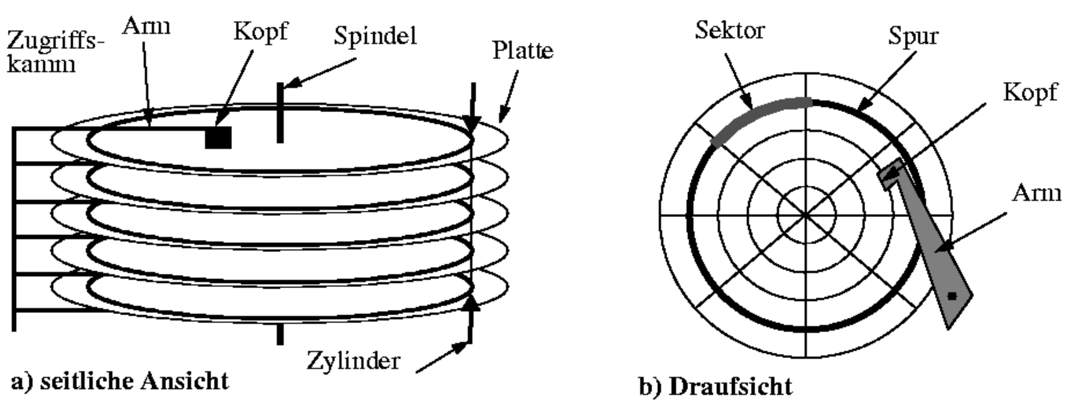

Data Model
Logical Access
Data Structures
Buffer Management
Operating System
Query optimization
Access control
Integrity constraints
Sort
Transaction processing
Cursor management
Physical record manager
Index manager
Lock manager
Log / Recovery
Bloc agement
aching
External memory
We are here

# Content of this Lecture

- Discs
- RAID level
- Some guidelines

# Magnetic Discs

- Preferred mass-storage since ~1970 until … now?
  - Multiple rotating discs, each with a separate head
  - Discs: Tracks, sectors, blocks
  - Formatting: Determining (fixed) block size
  - As blocks with fixed size: Tracks cannot have fixed number of blocks
    - Tracks are physically shorter toward the center
- More and more replaced by SSD
- Error-correcting codes: Single bit errors can be corrected



# Reading from Discs

- Seek time: $t_s$
  - 5-20ms: Move head to right track
- Wait time: $t_w$
  - 3-10ms: Wait for sector to rotate to head
  - On average: ½ rotation
  - Typical speed: 6.000 – 10.000 rotations / minute
- Reading blocks: At rotation speed
  - Do not forget caching within disc controller
- Transfer rate: u
  - Data volume read per time and put into main memory
  - Typical today: 100-300MB/sec (sequential reads)
    - That's more theoretical …

(Seek time and wait time together: Latency)

# Development


Source: https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/

# Random versus Sequential IO

- Task: Read 1000 blocks each 32KB (=32MB)
- Parameter: $T_s$= 10ms, $T_w$ = 6ms, u = 100MB/s
- Random I/O
  - For each block: Latency
  - t = 1000 * (10 ms + 6 ms) + 1000*32KB/100MB*1000 ms
  - t= 16000 ms + 320ms ~ 16s
- Sequential I/O
  - Once latency
  - 10 ms + 6ms + 1000*32KB/100MB*1000 ms
  - T= 16ms + 320 ms ~ 1/3 s
- One can read a lot sequentially before RA makes sense
- Reading few large files much faster than many small ones
  - Only when files are stored sequentially!

# Recent Technologies: SSD

- Solid state disks (SSD)
  - No moving objects, no mechanics
- Still smaller (~1-4TB) and more expensive
  - 500-1000€ in 2023: 4 TB SSD, but 18TB hard disc
- Five to ten times faster read/writes than HDD
  - Depending on interface, SATA* versus PCI*
  - Very small latency (<0.1ms)
  - Random access as fast as sequential reads (no defragementation!)
- Assume 500MB/s, lat=0.1ms, previous example
  - t = 1000 * 0.1 + 1000*32KB/500MB*1000 ms = 164ms ~ 1/6 sec
- Consume less energy
- Roughly same error rate, SSD probably with longer lifetime

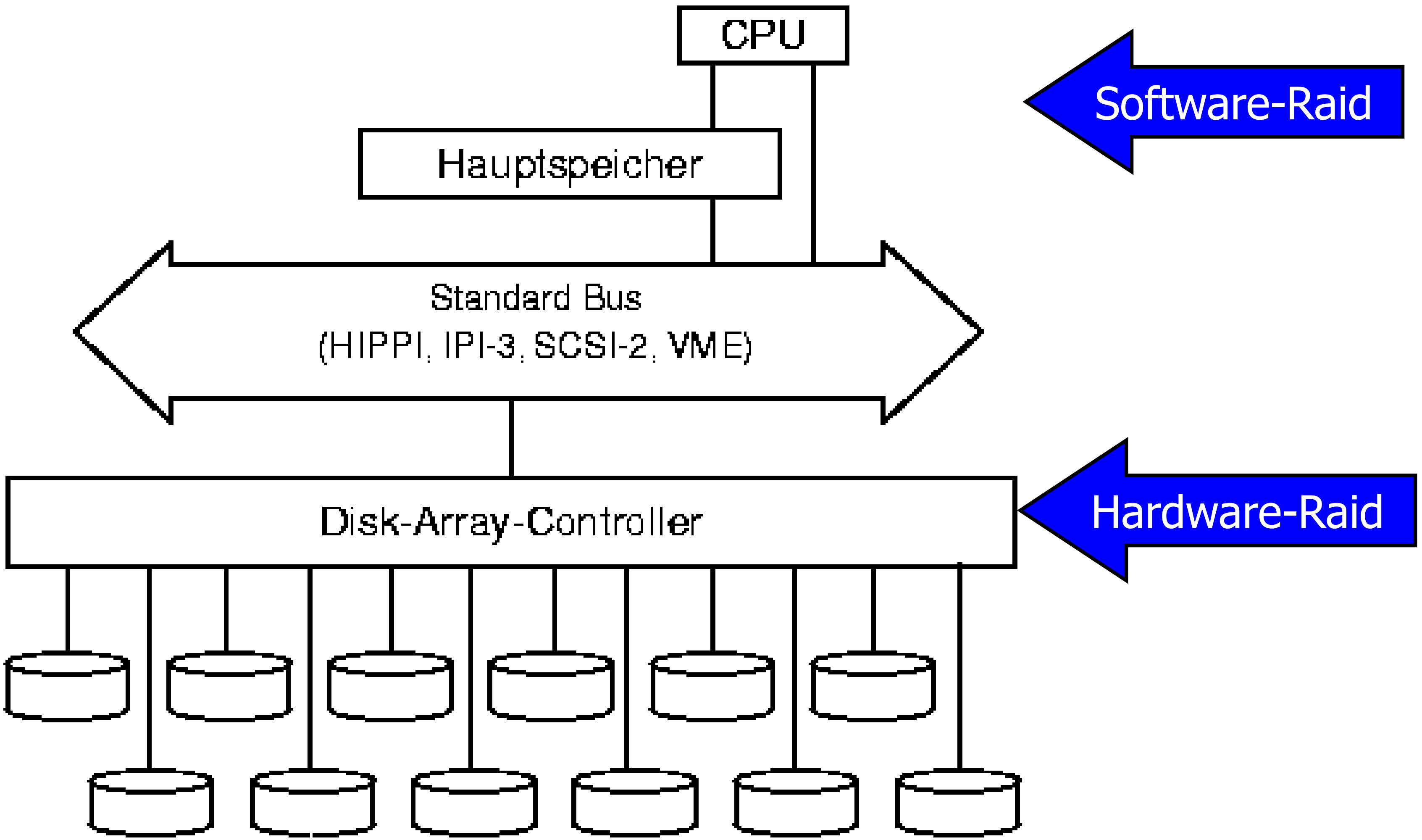
# Recent Technologies: NVA, RDMA

- Non-volatile memory (NVM, or storage-class memory)
  - Roughly same read speed as DRAM, same write speed as SSD
    - Different technologies
  - Many implications for database systems
    - Difference in read/write quite unusual for main memory
    - What is a reboot if all memory is non-volatile?
    - Arulraj/Pavlo. "How to build a non-volatile memory database management system.“, SIGMOD 2017
- Remote direct memory access (RDMA)
  - CPU’s read remote DRAM at network speed without foreign CPU
  - Combined with high-speed networks, remote access as fast as local
    - E.g. Infiniband (very expensive)
  - Beware: External processes are writing into your DRAM!

# How to get Faster with HDD?

- Fast IO is vital for an DBMS: Avoid SAN, NFS, HDFS, …
- Parallelize storage access (read and write)
  - Distribute files over multiple disks
  - Needs proper infrastructure: Controller, memory access channels
- RAID: Redundant Array of Independent Discs
  - Or: „Redundant array of inexpensive discs“
  - Idea: Buy many yet cheap disks
    - In contrast to more expensive disk with faster rotations and less errors
  - Different RAID level
  - May allow faster access (parallelization)
  - May allow higher fault tolerance (redundancy)
  - Always reduces net space
    - The space available for application data

# Architectures



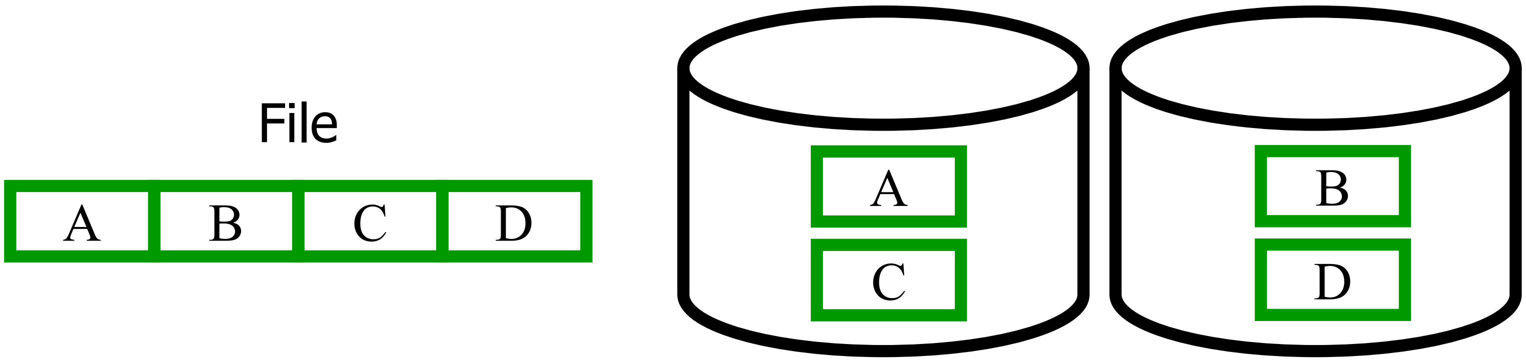
# Measuring Fault Tolerance

- One disc: If a head crashes, data is gone
  - Beware: Often the entire cylinder is gone
- With n non-redundant independent disks
  - Let d be the average number of days until a disk crashes
    - When will a disk fail (one is enough for data loss)?
    - If bought at the same time - after ~d days – all crash “at once”
  - Let p be the probability per day that a disk crashes
    - What is the probability per day that at least one disk crashes?
    - $1-(1-p)^n$
  - Example: 500 discs, p=1/1000: ~40% of at least one crash / day
- Redundancy: probability of non-maskable faults drops
  - May reduce latency, read throughput, write throughput
  - Increases total space

# Content of this Lecture

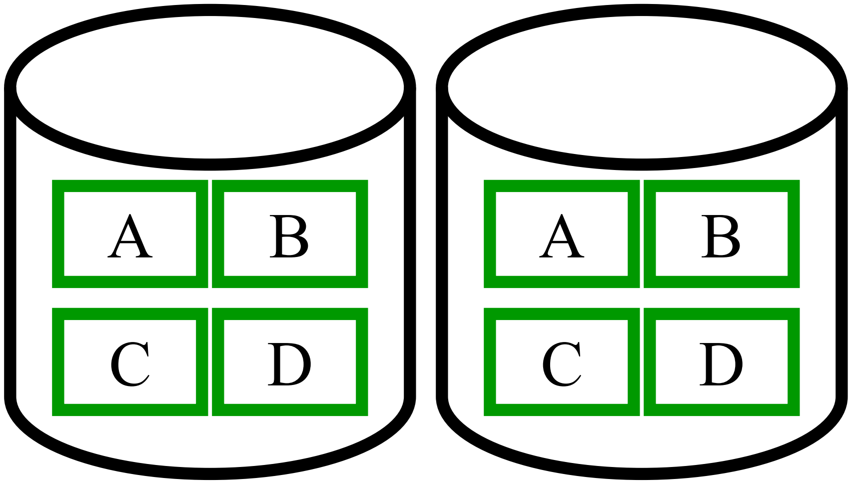
- Discs
- RAID level
- Some guidelines

# RAID 0: Striping




- Up to double throughput for sequential reads and writes
  - If a large file is perfectly distributed and completely read
- Small files not accelerated much, single blocks not at all
  - Latency dominate
  - Again: Large sequential scans are your friend
- Decreased fault tolerance
  - Distributed files (for throughput) are at risk from two discs
- Same net space

# RAID 1: Mirroring

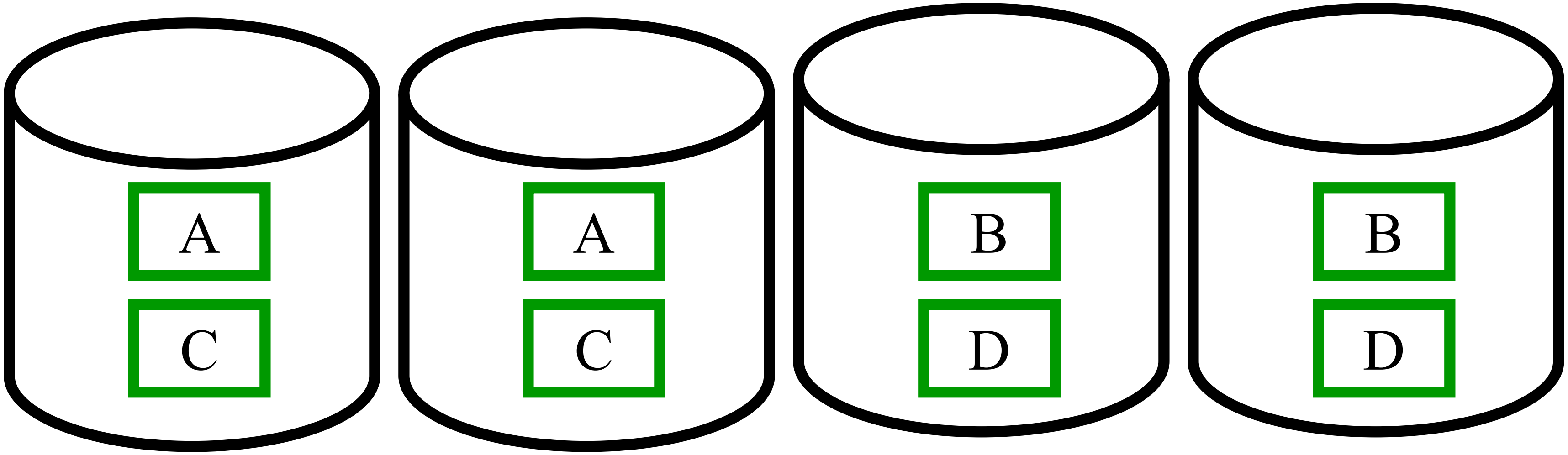



- Doubled throughput for sequential file reads
- Writes are not accelerated
- Single block read might be slightly better
  - Read from both disks, faster disk wins (better head position)
- Increased fault tolerance
- 50% net space less

# RAID0 versus RAID1

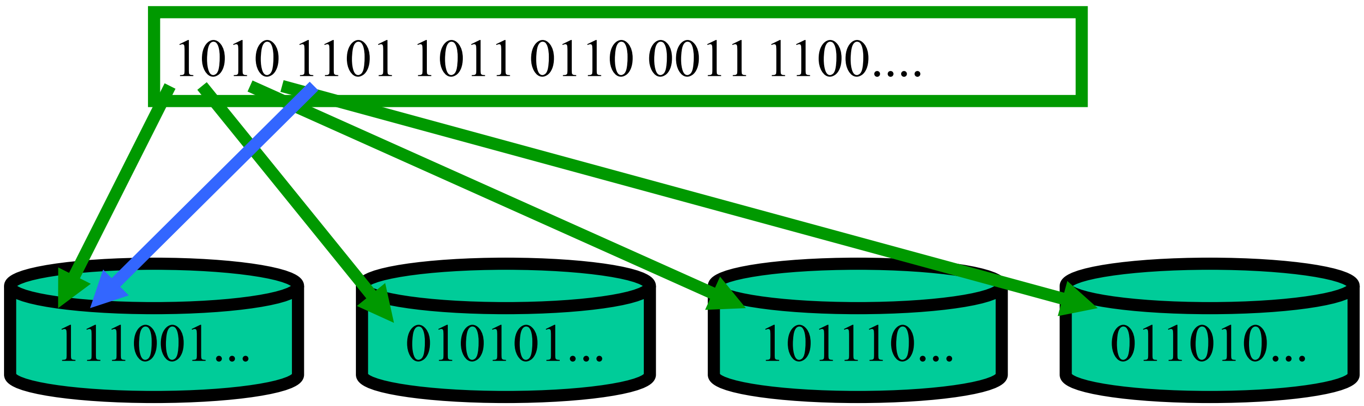
- Abbreviations
  - MTTF = Mean time to (between) failure of a disk
  - MTTDL = Mean time to data loss of a system (non-maskable crash)
    - Data needs to be restored from backup
- Example: MTTF = 3650 days
  - RAID0 with 2 disks bought at arbitrary points in time
    - Every crash destroys data
    - Expected $MTTDL_1$ = 3650/2 = 1825 days
  - RAID1 with 2 disks bought at arbitrary points in time
    - Both must crash at the same time to destroy data
    - $MTTDL_2 = MTTDL_1 * MTTDL_1 \sim$ 9.000 years
  - Assuming statistical independence of events (disks)
  - But: Shared room (fire, flood), shared power (outage), shared building (earthquake), shared age, …

# RAID 0+1: Striping and Mirroring



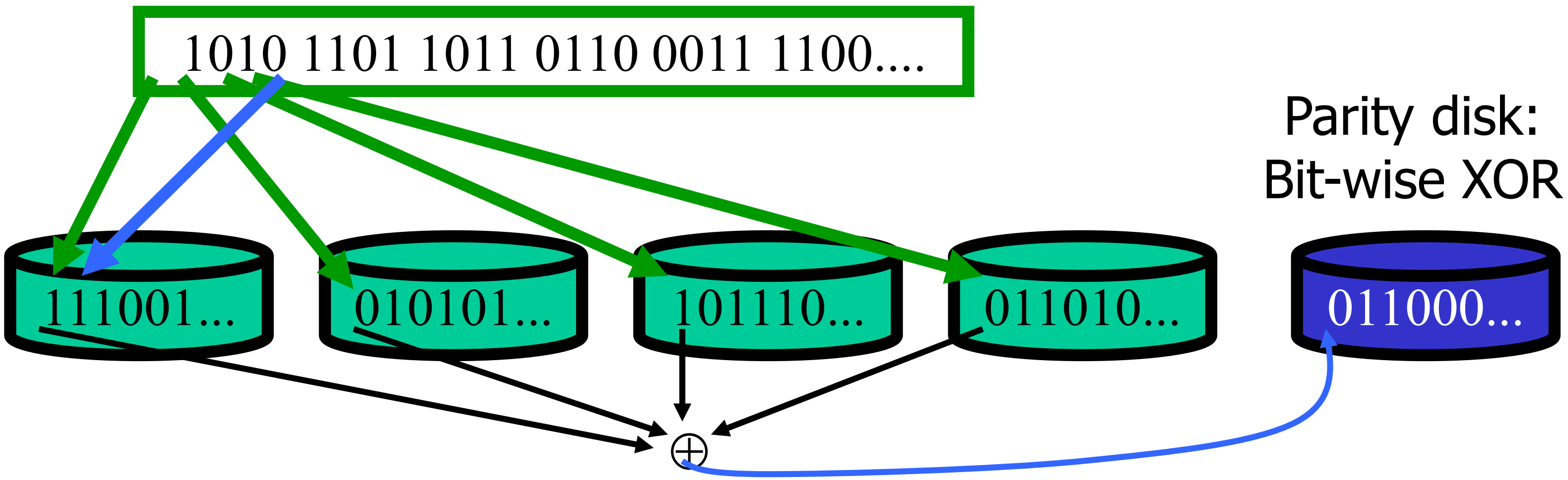

- Quadruple speed for sequential read
- Doubled speed for sequential writes
- 50% net space
- Increased fault tolerance

# RAID 2: Striping Bits (not Blocks)



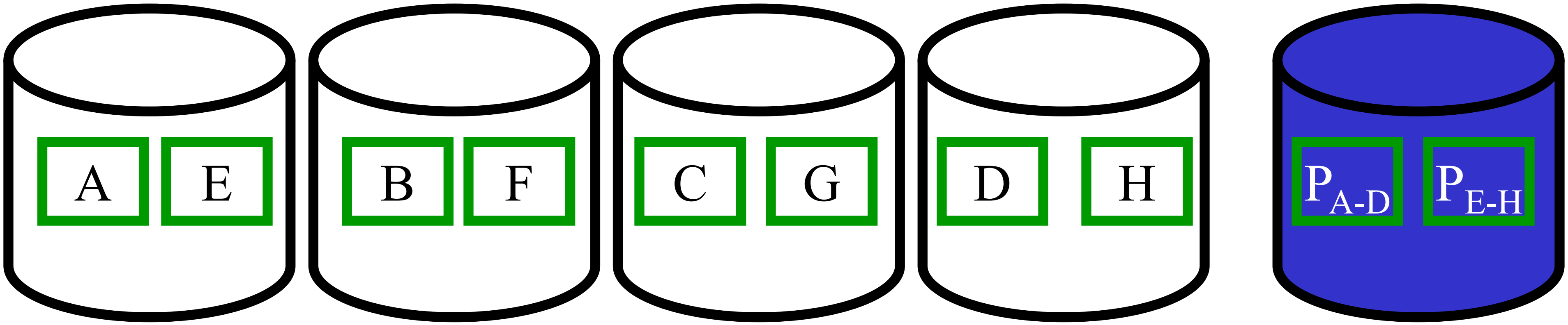

- Much disadvantage compared to RAID0
  - On block devices, reading a byte is as expensive as reading a block
- And more complex management
  - OS / DBs cache blocks, not parts of blocks
- Irrelevant for disks

# RAID 3: RAID2 + Parity



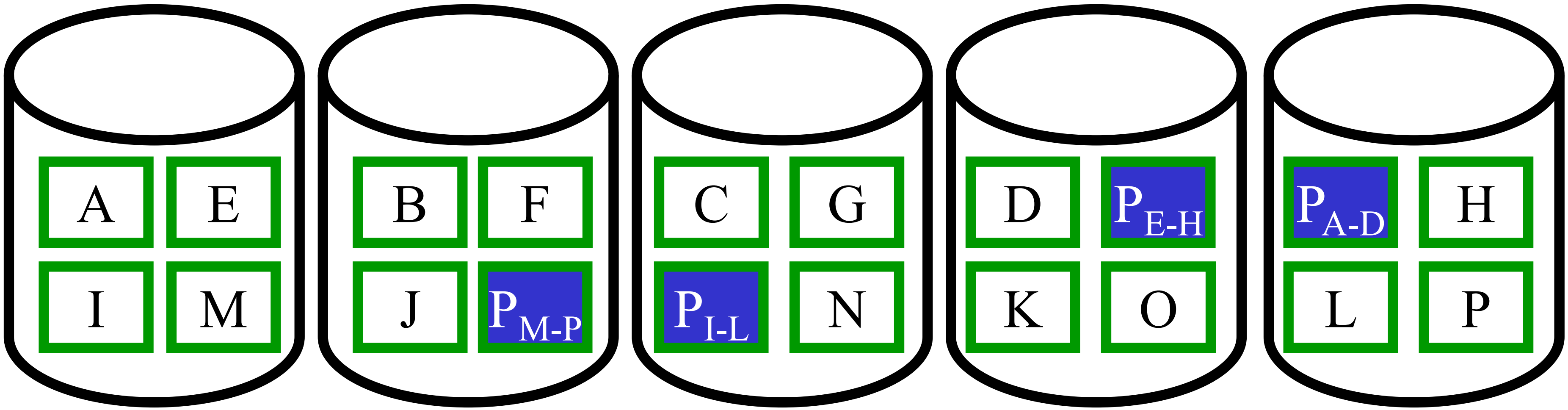

- Increased fault tolerance: One disk crash can be tolerated
  - Crashed data can be restored from other disks
  - Flipped bits can be detected, but not repaired
  - Same robustness, but much better space utilization than RAID1
- (n-1) times faster for sequential reads of large files
  - But not if flipped bits should be detected: parity disk is bottleneck
- Writes unchanged (parity disk) or even slower
  - If multiple processes write, parity disk becomes bottleneck

# RAID 4: Block Striping + Parity




- Same idea as RAID 3, but striping at block, not bit, level
- Easier management
- Parity remains bottleneck for parallel reads and writes
  - Every net block write incurs one parity write
  - Leads to locking if multiple processes write concurrently

# RAID 5: RAID4 with distributed Parity




- Parity blocks are evenly spread over disks
- Many benefits
  - Parity accesses distributed among all disks – no more bottleneck
  - Up to (n-1) times faster reads of large files
  - Writes slightly slower than RAID0 (still synch work on parity disk)
  - Not much space wasted: Net space is (n-1) times capacity
  - One disk crash can be repaired

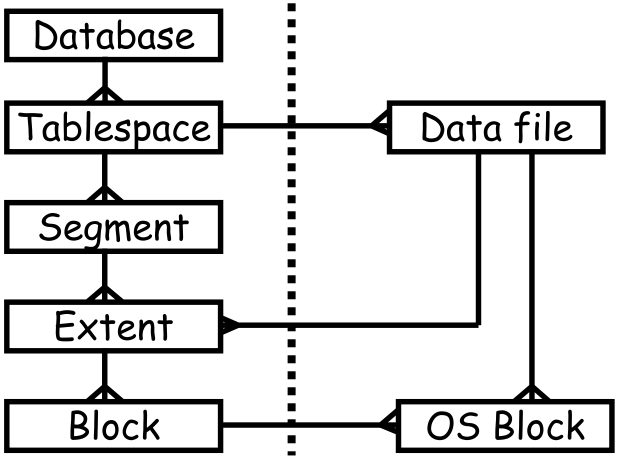
# Summary

| | 0 | 1 | 0+1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| Striping blockweise | √ | | √ | | | √ | √ |
| Striping bitweise | | | | √ | √ | | |
| Kopie | | √ | √ | | | | |
| Parität | | | | √ | √ | √ | √ |
| Parität dediz. Platte | | | | | √ | √ | |
| Parität verteilt | | | | | | | √ |
| Erkennen mehrerer Fehler | | | | | | | |

- Further RAID Level defined, e.g.: 6=5+1, …
  - Save as RAID5, fast as RAID1 – but 50% less capacity
- Typical scenarios
  - Increase write speed needs striping (e.g. RAID 0)
  - RAID1: Simple, fast, safe, but needs lots of space
  - RAID5: More complex, safe, fast, requires more space, requires at least three disks

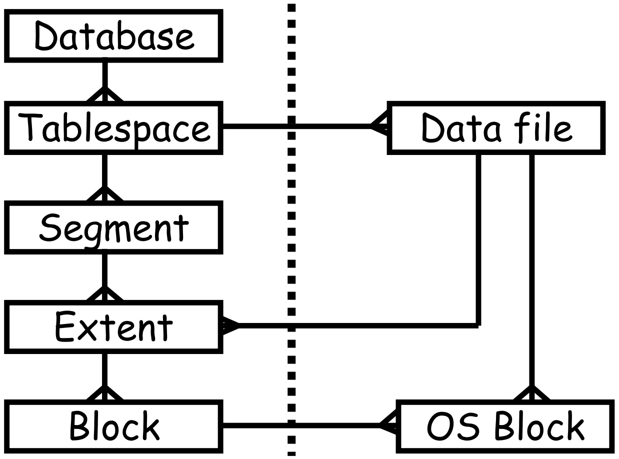
# Oracle: Options without RAID

- Parallelization by distributing tablespaces
  - System tablespace on separate disk
    - Or: Tablespace-managed data dict.
  - Separate tablespaces for data / index
  - Separate disk for REDO Logs




- .. by distributing one tablespace over multiple disks
- ... by distributing a single table over multiple disks
  - Extends in different, distributed files of the same tablespace
  - Partitioning – value-based distribution of data
    - All sales prior to 2005 on one disk, all younger sales on another disk
    - One disk for sales in 2005, 2004, 2003, …
    - See "Data Warehousing"

# Interference with RAID




- File layout and RAID interfere
- Multi-file distributed tablespace might not have an effect if files are RAID-distributed over the same physical disks
- Proper RAID design might make file distribution obsolete
- Need to consider both to prevent advantage-cancelling effects
- Note: Parallel reads must be consumed on upper levels – parallel memory access, parallel processing units, …

# Some guidelines (Oracle handbooks)

- „Tsps should stripe over at least as many devices as CPUs"
- "You should stripe tablespaces for tables, indexes, rollback segments, and temporary tablespaces. You must also spread the devices over controllers, I/O channels, and internal buses"
  - Queries can run in parallel (inter-query parallelization)
  - Single disk is bottleneck – multiple processors become useless
  - Ideally, each disk becomes a "feed" for one processor (thread)
- Disadvantages
  - Data spread over multiple disks leads to higher failure chances – use redundant RAID levels
  - Recovery (hot swap) of a disk might stop operations
    - All disks must be access at the same time for repair

# Guidelines 2

- „In high-update OLTP systems, the redo logs are write-intensive. Moving the redo log files to disks that are separate from other disks and from archived redo log files has … benefits …"
  - Every transaction generates REDO information
  - REDO is written in batches before commit, data blocks are written sporadically with mostly random access
  - Both should not interfere (too many seeks)
    - Hence: Put REDO log files away from data files
    - Disk crash can only effect REDO or only data files – built in redundancy
  - Redo data is extremely important (rollback, roll-forward)
    - Hence: Spread REDO itself redundantly over many disks
    - By system (RAID) or by database (REDO groups)
  - REDO disks are good places to invest in RAID10

# Other Typical Bottlenecks

- Temporary tablespace – used especially for large SORTS
  - And sorting is everywhere – sort-merge join, group by, order by, distinct, ...
  - Receives many concurrent accesses from many processes
  - Hot spot – fast reads, fast writes, but failure is not critical
  - RAID0
- System tablespace
  - Holds data dictionary – important for everything
  - Required all the time – logs, latches, system log data, ...
  - RAID1
    - Better: SSD or mem-cached