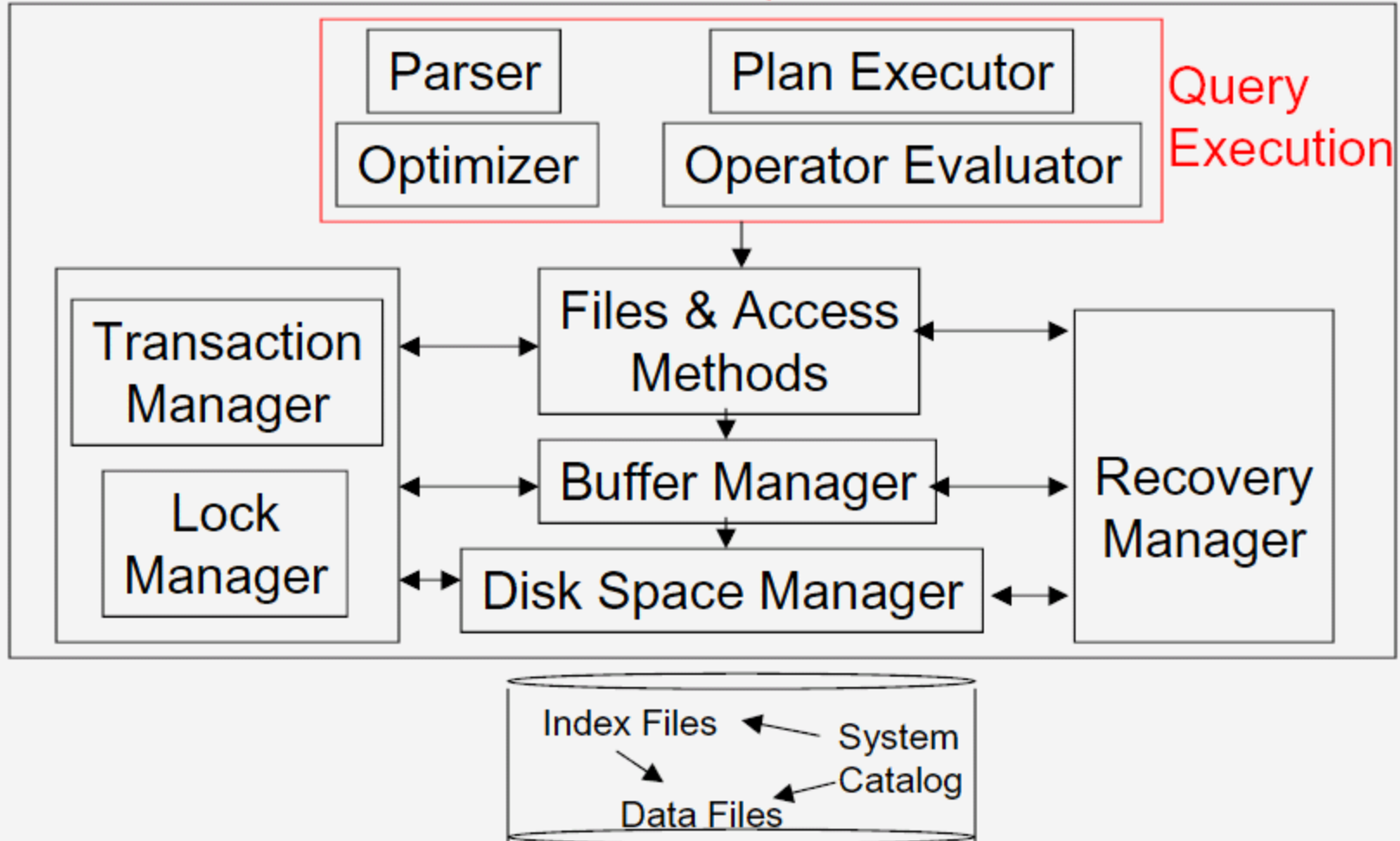


Physical Structure of Databases

7

Detailed Structure of a DBMS



Physical Structure of DB files

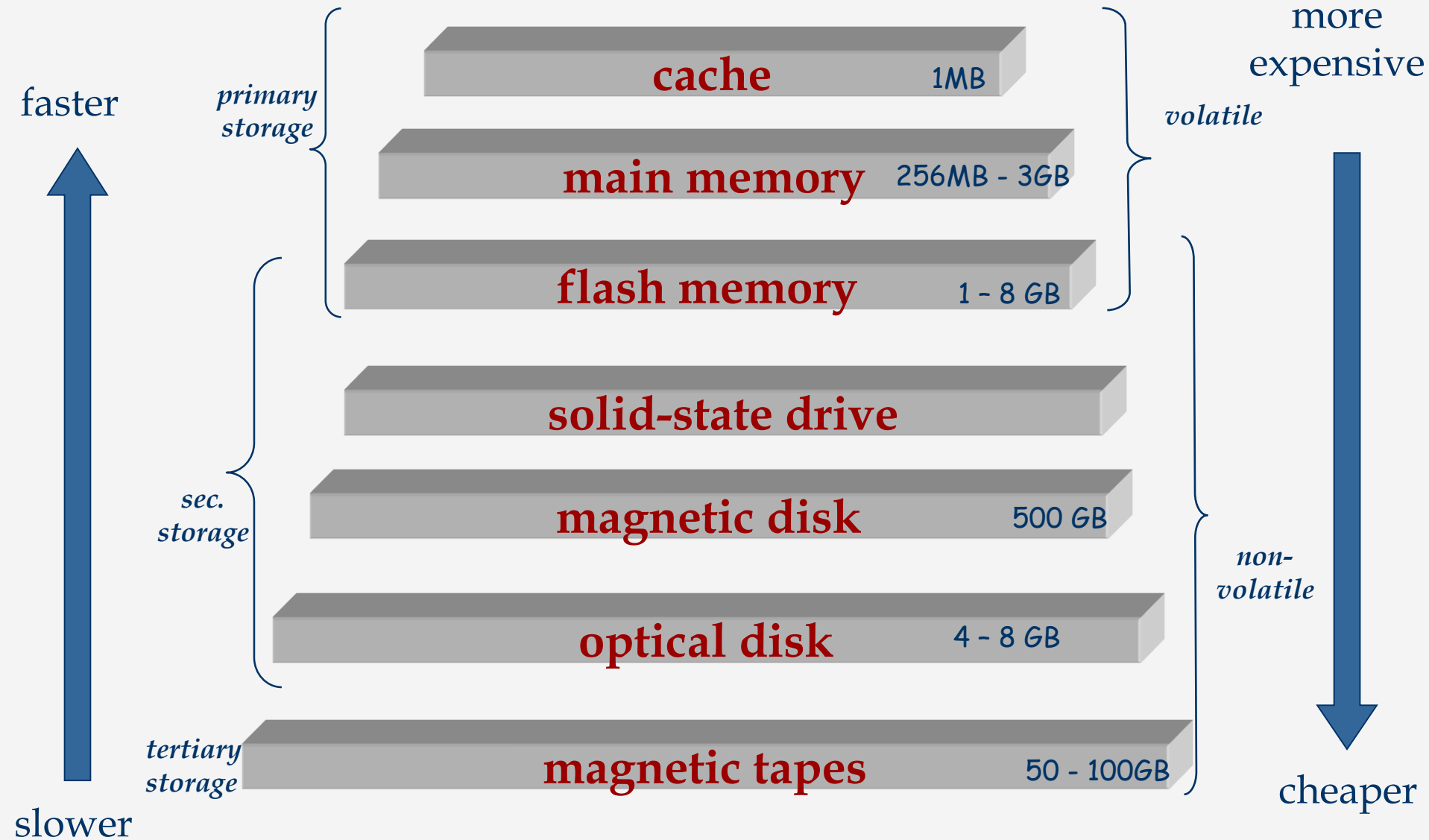
- DBMSs store information on disks.
- This has major implications for DBMS design!
 - **READ**: transfer data from disk to main memory.
 - **WRITE**: transfer data from main memory to disk.

Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

Why not store everything in main memory?

- (Typical) Answers:
 - Costs too much
 - Main memory is volatile (we need persistent data)
- Typical procedure (“storage hierarchy”)
 - MM – for currently used data (primary storage)
 - Hard-disks – for the main database (secondary storage)
 - Tapes – for archiving old versions of the data (tertiary storage)

Storage Device Hierarchy

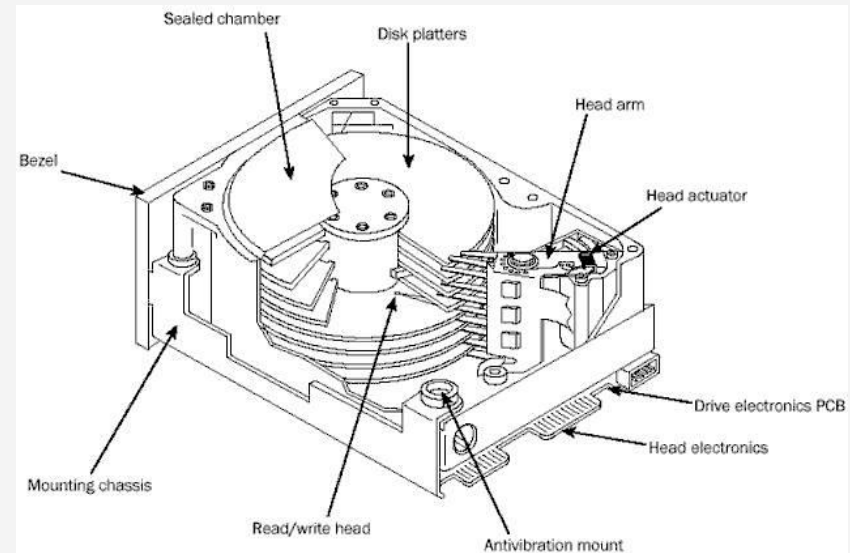


Moore's Law

- Gordon Moore: “Integrated circuits are improving in many ways, following an exponential curve that doubles every 18 months”
 - Speed of processors
 - Number of bits that can be put on a chip
 - Number of bytes that disk can hold
 - Parameters that DO NOT follow Moore's law:
 - Speed of accessing data in main memory
 - Speed at which disks rotate
- ⇒ Latency becomes progressively larger
- Time to move data between levels of hierarchy appears to take longer compared with time it takes to compute

Disks

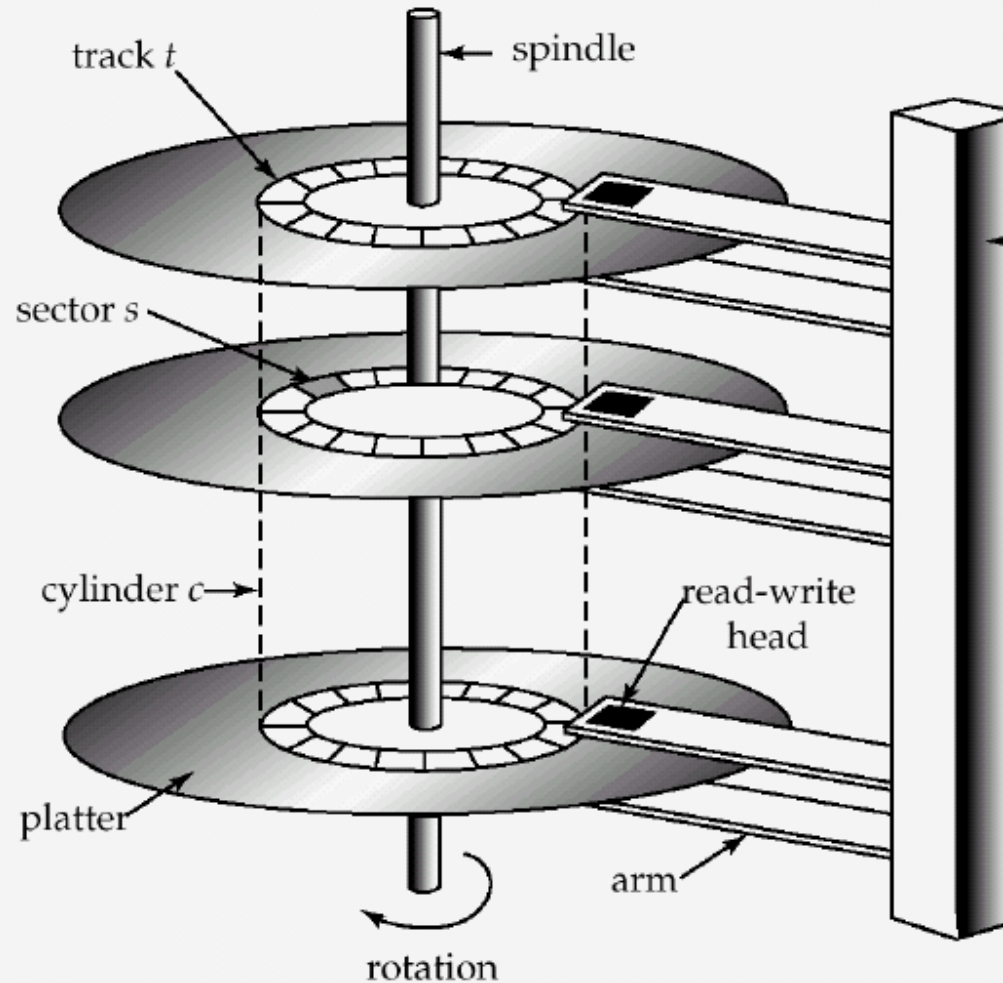
- The choice for the secondary storage device
- Main advantage over tapes: *random access*
- Data is stored and retrieved in units called **disk blocks** or **pages**.
- Unlike main memory, time to retrieve a disk page varies depending upon location on disk.



! Relative placement of pages on disk
has major impact on DBMS performance !

Components of a Disk

- The platters spin (90rps)
- The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a **cylinder** (imaginary!).
- Only one head reads / writes at any one time.
- **Block size** is a multiple of **sector size** (which is fixed).



Accessing a Disk Page

- Time to access (read/write) a disk block:
 - **seek time** (moving arms to position disk head on track)
 - **rotational delay** (waiting for block to rotate under head)
 - **transfer time** (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
 - Seek time varies from about 1 to 20msec
 - Rotational delay varies from 0 to 10msec
 - Transfer rate is about 1msec per 4KB page
- Key to lower I/O cost: **reduce seek/rotation delays!**
- Hardware vs. software solutions?

Arranging Pages on Disk

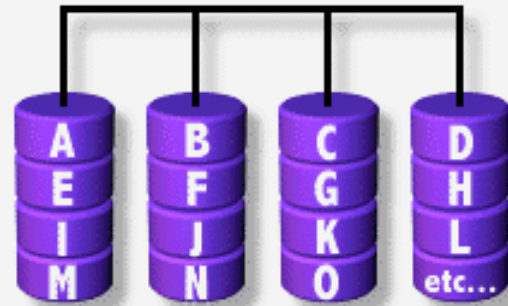
- **`Next'** block concept:
 - blocks on same track, followed by
 - blocks on same cylinder, followed by
 - blocks on adjacent cylinder
- Blocks in a file should be arranged sequentially on disk (by **`next'**), to minimize seek and rotational delay.
- For a **sequential scan**, pre-fetching several pages at a time is a big win!

RAID

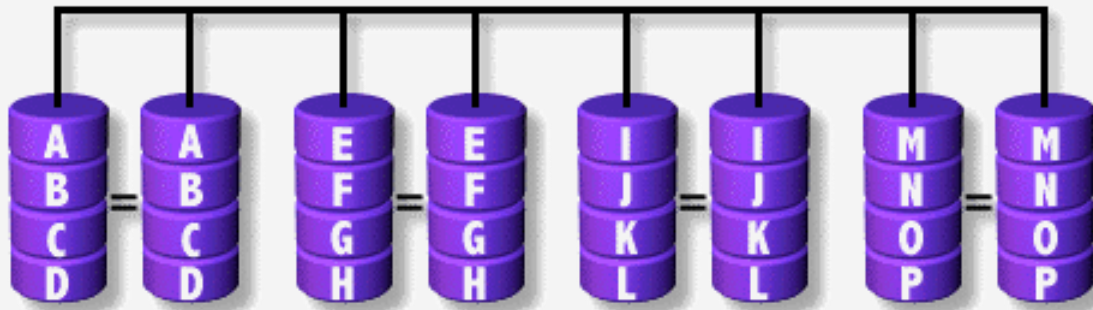
- Disk Array: Arrangement of several disks that gives abstraction of a single, large disk.
 - More cost effective to use a number of cheap, small disks than few large ones
- Goals: Increase performance and reliability.
- Two main techniques:
 - Data striping*: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
 - Redundancy*: More disks -> more failures. Redundant information allows reconstruction of data if a disk fails.

RAID Levels

- Level 0: No redundancy



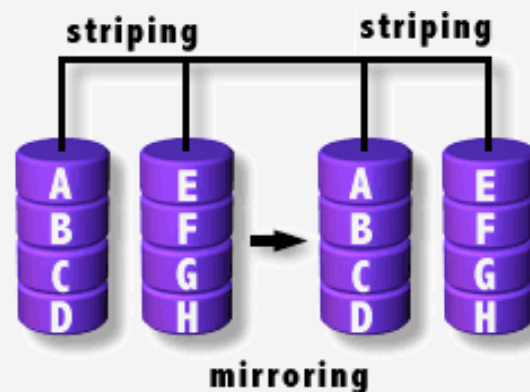
- Level 1: Mirrored (two identical copies)
 - Each disk has a mirror image (check disk)
 - Parallel reads, a write involves two disks.
 - Maximum transfer rate = transfer rate of one disk



RAID Levels

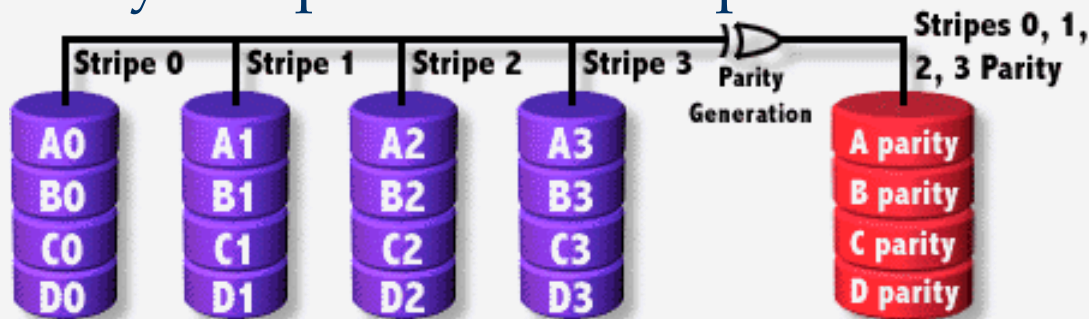
■ Level 0+1: Striping and Mirroring

- Parallel reads
- A write involves two disks.
- Maximum transfer rate = aggregate bandwidth



■ Level 3: Bit-Interleaved Parity

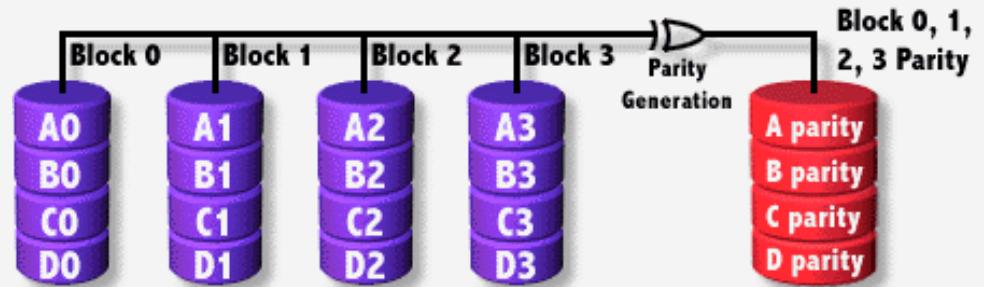
- Striping Unit: One bit. One check disk.
- Each read and write request involves all disks
- Disk array can process one request at a time.



RAID Levels

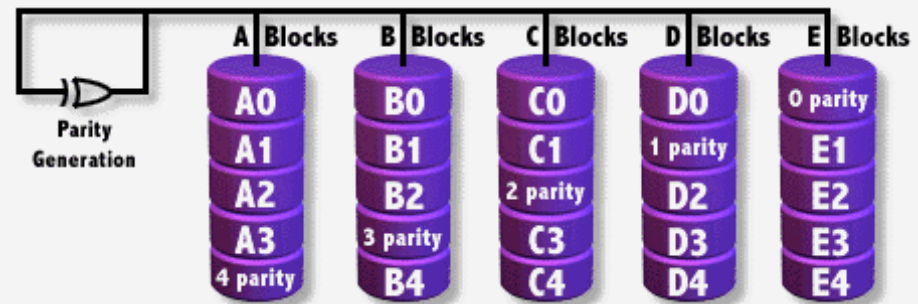
- Level 4: Block-Interleaved Parity

- Striping Unit: One disk block. One check disk.
- Parallel reads possible for small requests, large requests can utilize full bandwidth
- Writes involve modified block and check disk



- Level 5: Block-Interleaved Distributed Parity

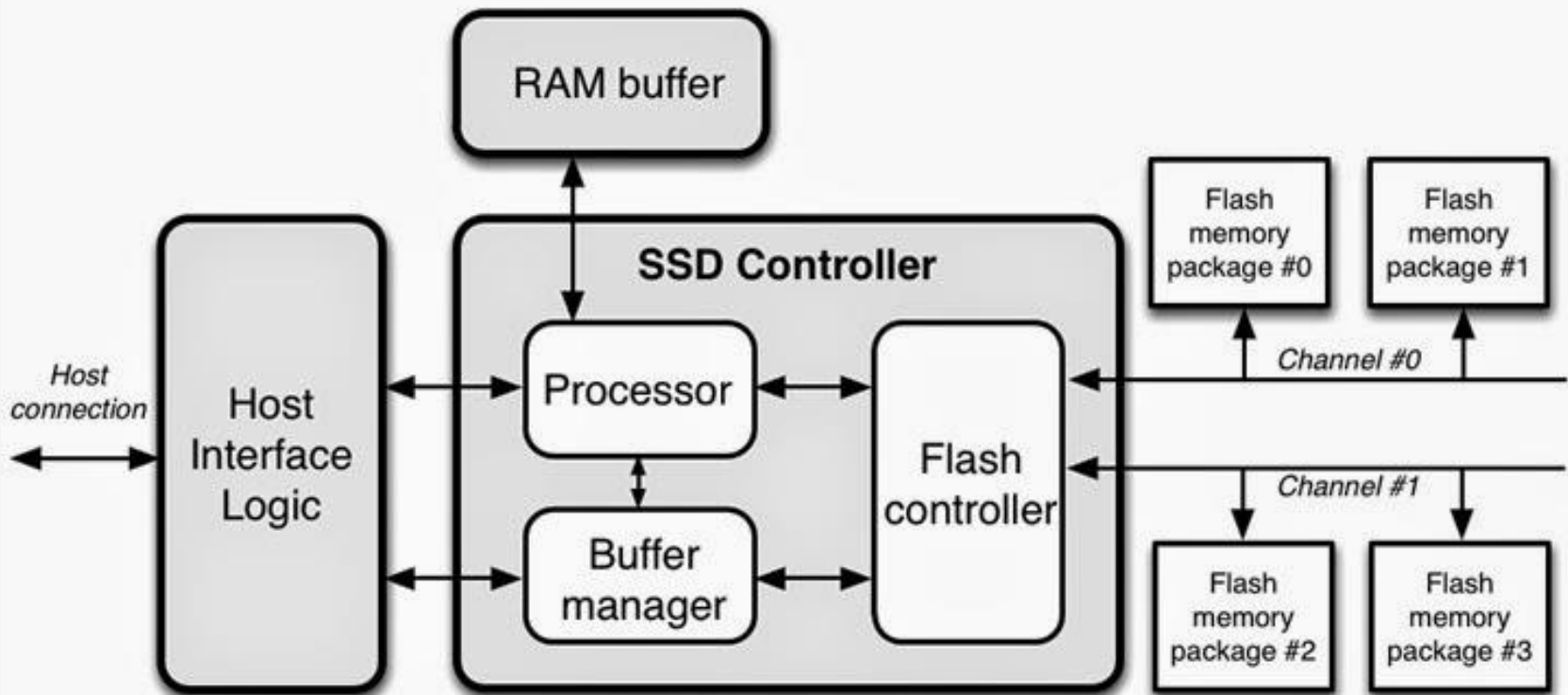
- Similar to RAID Level 4, but parity blocks are distributed over all disks



Solid State Drive

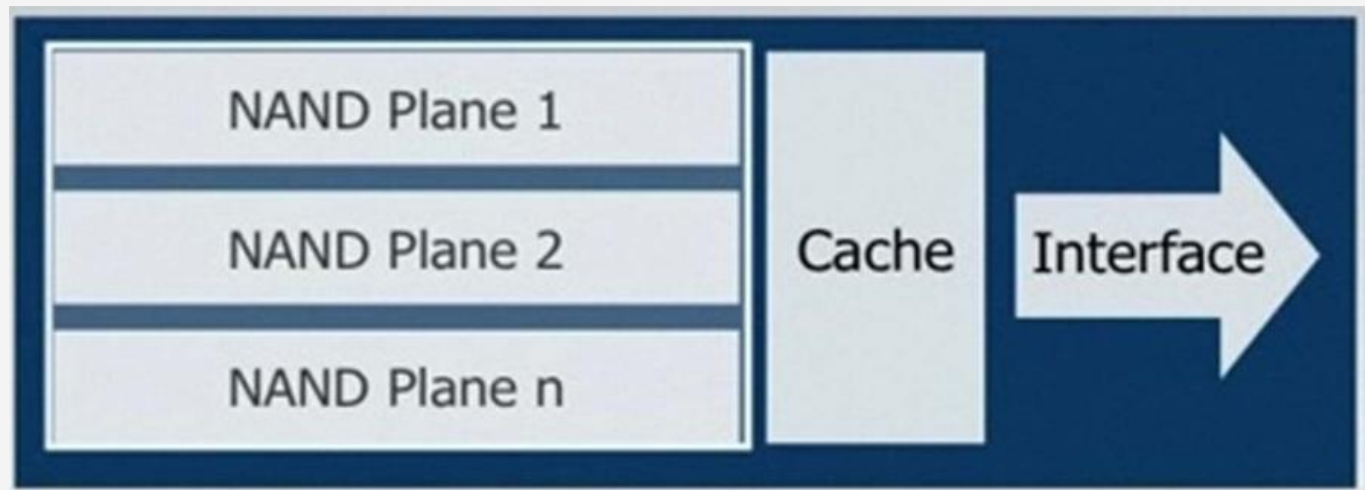
- SSDs contain a number of NAND flash components (16/32 GB)

Architecture of a solid-state drive



Solid State Drive

- NAND Flash Media contains NAND cells arranged in multiple planes:
 - the planes allow for parallel access to the NAND
 - they also allow for interleaving
- The data moves in and out through a cache element



Solid State Drive - Advantages

- Super low latency
 - zero seek time
- Very fast read and write speeds
- Physically more robust
 - Shock resistance
 - Zero moving parts
 - Completely silent
 - Low power consumption
- Excel at small/short reads and writes
- Immune to data fragmentation

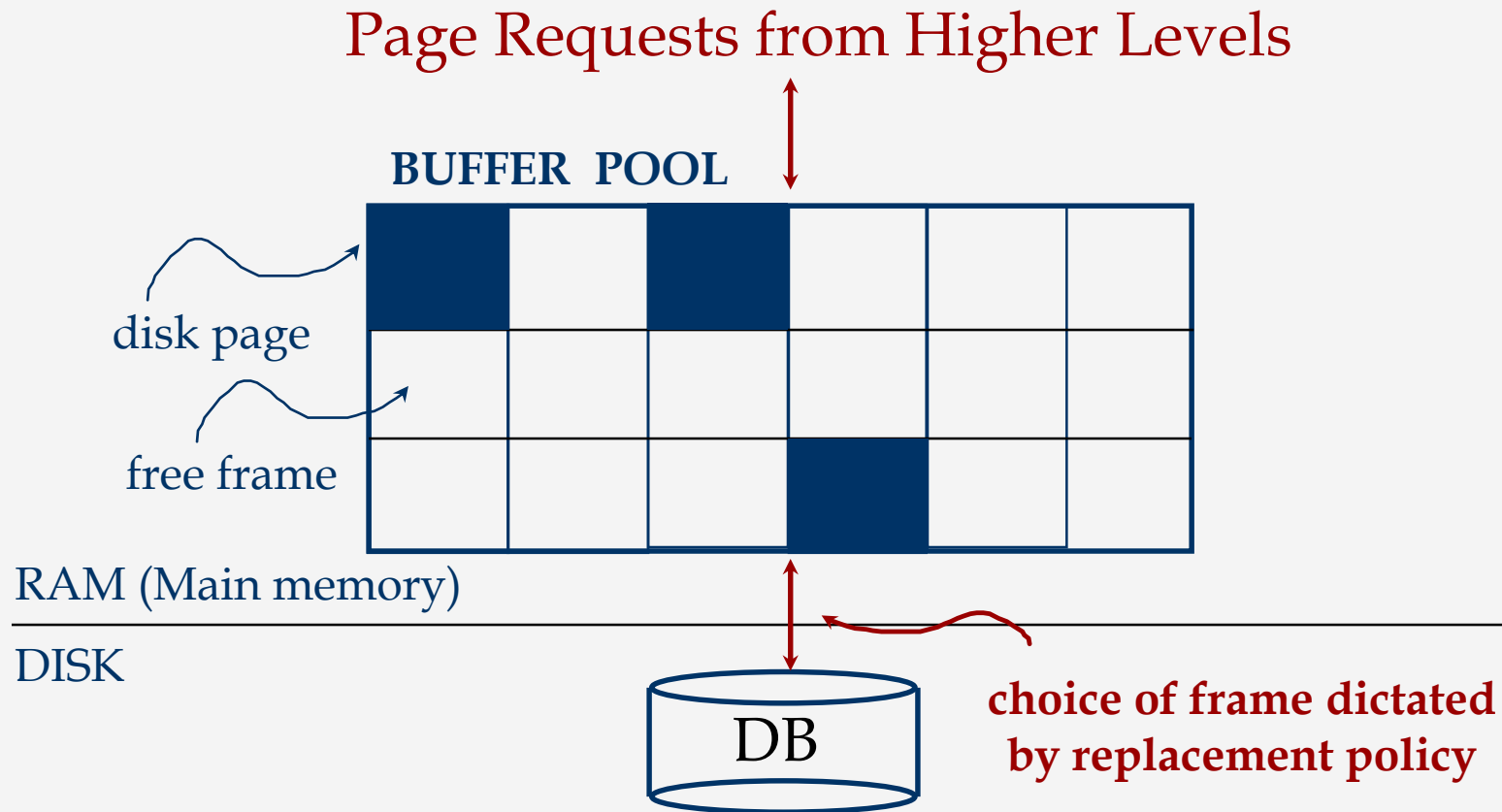
Solid State Drive - Disadvantages

- Cost per GB is much higher than HDDs
- General Size
 - 3TB consumer 3.5'' HDDs are available (relatively common)
 - 1TB consumer 3.5'' SSDs are available (very rare and expensive)
- Limited write cycles
 - 1 to 2 million write cycles before wear out for MLC (multi-level cell)
 - Up to 5 million write cycles before wear out for SLC

Buffer Management in a DBMS

- **Buffer** – portion of main memory available to store copies of disk blocks.
- **Buffer manager** – subsystem responsible for allocating buffer space in main memory.
- Programs call on the **buffer manager** when they need a block from disk.
 - Data must be in main memory for DBMS to operate on it!

Buffer Management in a DBMS



- Table of $\langle \text{frame_no}, \text{page_id} \rangle$ pairs is maintained.

When a Page is Requested...

- If requested page is not in pool:
 - Choose a frame for **replacement**
 - If frame is dirty, write it to disk
 - Read requested page into chosen frame
- *Pin* the page and return its address.

*! If requests can be predicted (e.g., sequential scans)
pages can be pre-fetched several pages at a time!*

More on Buffer Management

- Requestor of page must unpin it, and indicate whether page has been modified:
 - **dirty** bit is used for this.
- Page in pool may be requested many times,
 - a **pin count** is used. A page is a candidate for replacement iff **pin count** = 0.
- CC & recovery may entail additional I/O when a frame is chosen for replacement. (e.g. *Write-Ahead Log* protocol)

Buffer Replacement Policy

- Frame is chosen for replacement by a *replacement policy*
 - **Least Recently Used (LRU)**: use past pattern of block references as a predictor of future references. Queries have well-defined access patterns (e.g. *sequential scans*), and a database system can use the information in a user's query to predict future references.
 - **Toss-immediate**: frees the space occupied by a block as soon as the final tuple of that block has been processed
 - **Most recently used (MRU)**: after the final tuple of a block has been processed, the block is unpinned and it becomes the most recently used block.

Buffer Replacement Policy

- Buffer manager can use **statistical information** regarding the probability that a request will reference a particular relation
- The replacement policy can have big impact on number of I/O's - depends on the **access pattern**.
- *Sequential flooding*: Nasty situation caused by LRU + repeated sequential scans.
 - **No buffer frames < No pages in file** means each page request causes an I/O. MRU much better in this situation (but not in all situations, of course).

DBMS vs. OS File System

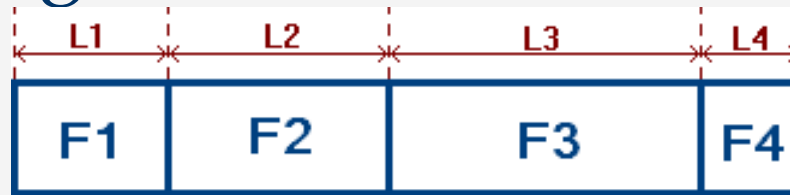
- OS does disk space & buffer management: why not let OS manage these tasks?
- Differences in OS support: portability issues
- Some limitations, e.g., files can't span disks.
- Buffer management in DBMS requires ability to:
 - pin a page in buffer pool, force a page to disk (important for implementing CC & recovery),
 - adjust *replacement policy*, and pre-fetch pages based on access patterns in typical DB operations.

Files of Records

- Higher levels of DBMS operate on **records** and **files of records**, not on pages or blocks
- **File** = collection of pages; each page contains a collection of records; must support:
 - insert/delete/modify record
 - read a particular record (using a **record id**)
 - scan all records (possibly filtered)
- A page containing a record can be identified using the record's id (**rid**)

Record Formats

- Fixed length

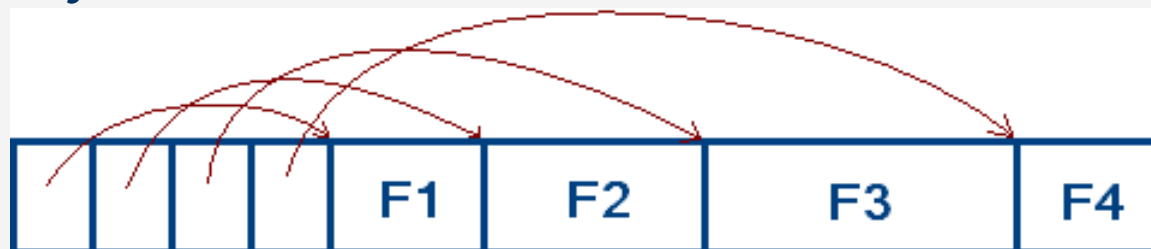


- Variable length

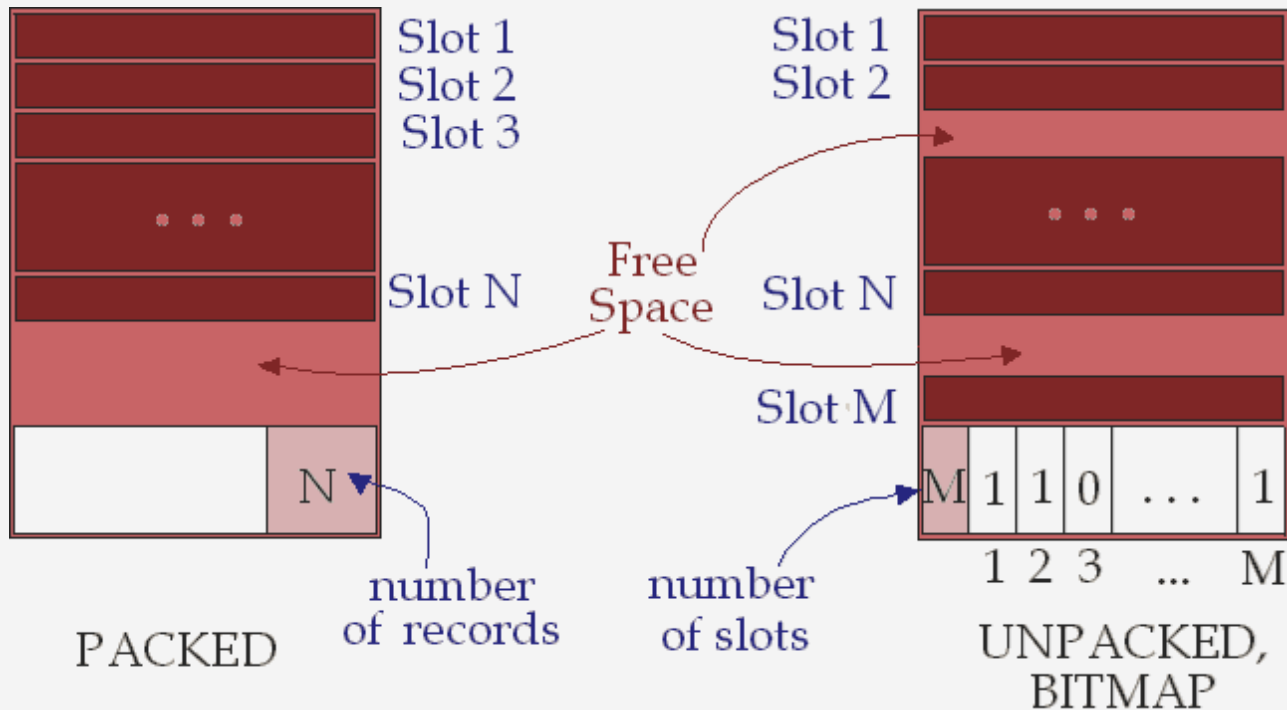
- Fields delimited by special symbols



- Array of field offsets

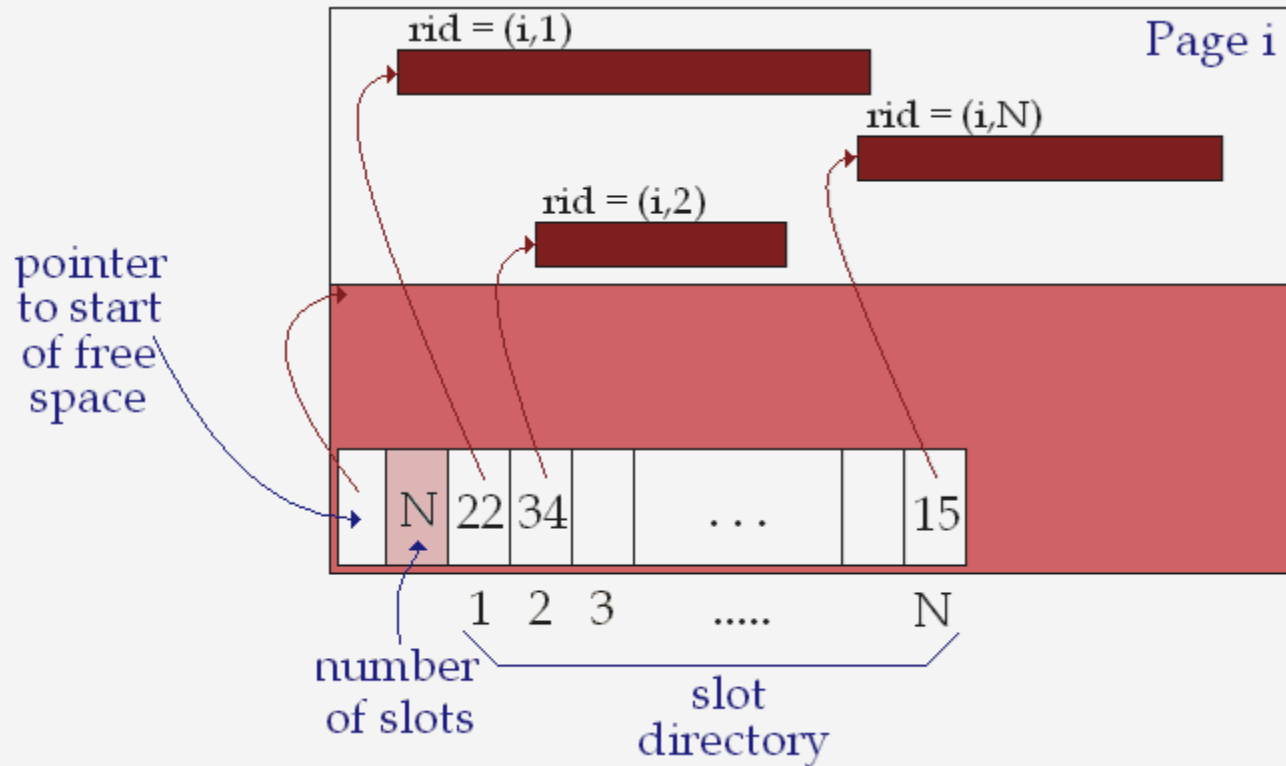


Page Formats: Fixed Length Records



- Record id = <page id, slot no>. In first alternative, moving records for free space management changes rid; may not be acceptable.

Page Formats: Variable Length Records



- Can move records on page without changing *rid*; so, attractive for fixed-length records too.