CMPT 606

Read Chapter 17

Database Storage

Dr. Abdelkarim Erradi

Department of Computer Science and Engineering

QU

Outline

1. [Database Storage Technologies]
2. [RAID Technology]
3. [Database File Organization]
4. [Buffer Manager]

Acknowledgment
Some slides are based on textbook slides &
CMU DB Course https://15445.courses.cs.cmu.edu/fall2019/

Course Outline – Database Inner Working

Storage

- **Disk Manager**
- **Buffer Pool Manager**
- Access Methods

Query Planning and Execution

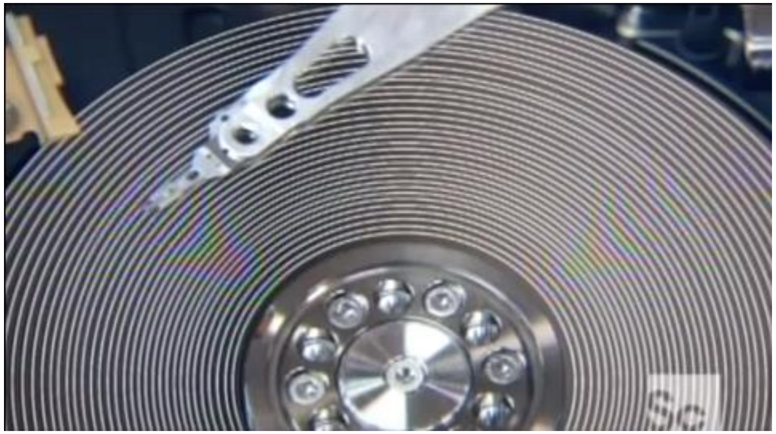
Concurrency Control

Recovery

Disk-Oriented Architecture

- Disk-Oriented Architecture = primary storage location of the database is on non-volatile disk.
  - The DBMS's components manage the movement of data between Disk (non-volatile) and Memory (volatile).
- Storage Engine Design Goals
  - Allow the DBMS to manage **databases that exceed the available amount of memory**
  - Reading/writing to disk is expensive, so **I/O must be managed carefully** to avoid large waits and performance degradation

Database Storage Technologies



< Back

Why study the physical level of DBMS

- Someone has to write the DBMS software and its file manager!
- Some DB systems give the database administrator a range of physical storage options
  - Intelligent use of these options can make a very significant (and user-noticeable) difference in the way the system performs.
  - To "tune" the system properly, the DBA must understand what is happening at the physical level.
- Some of the techniques and algorithms can be used to solve other problems in other contexts

Key components of DBMS performance

- The performance of the DBMS Storage Engine is often the key component of overall performance.
- There are two attributes that can be optimized:

1. **Response time** - defined as the time between the issuance of a command and the time that output for the command begins to be available.

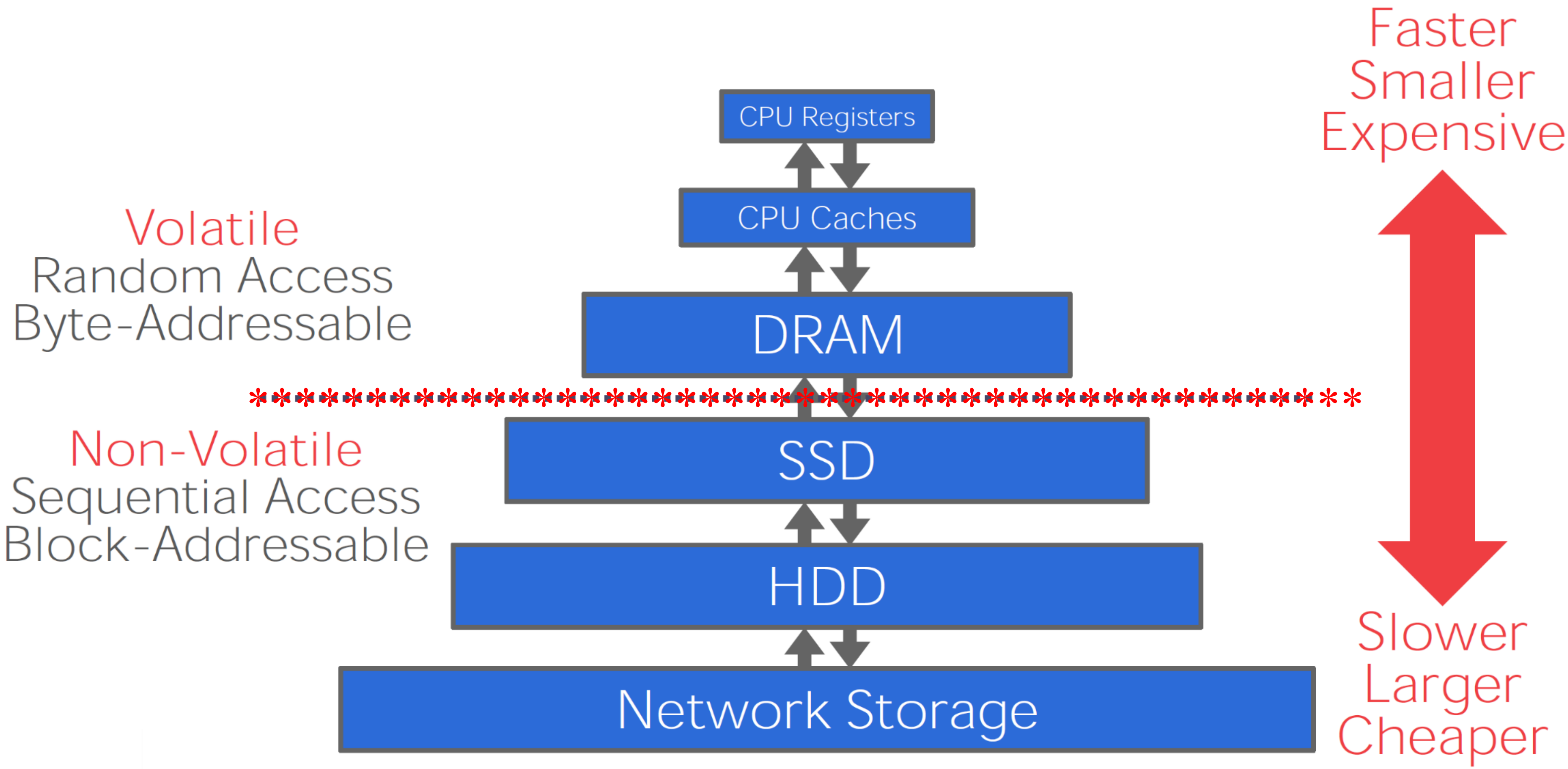
- e.g. if the command is a select statement, the time until the first row of the result appears

=> we want to minimize this

2. Throughput - the number of operations that can be completed per unit time.

=> we want to maximize this

Storage Hierarchy



*** New trend: **Non-volatile memory** (e.g., Intel® Optane™)
https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html

Source: CMU DB Course https://15445.courses.cs.cmu.edu/fall2019/

Storage Hierarchy - Primary storage

- **Primary storage**: Fastest media but volatile
  - Can hold subset of a database used by current transactions.
  - Volatile = data is lost when an application terminates (normally or due to a power failure or crash)
- **Cache**
  - Data and instructions in cache when needed by CPU.
  - On-board (L1) cache on same chip as CPU, L2 cache on separate chip.
  - Capacity couple of MBs, **access time few nanoseconds**
- **Main memory**
  - All active programs and data need to be in main memory.
  - Capacity couple of GBs, access time **10-100 nanoseconds**

Storage Hierarchy - Secondary & Tertiary Storage

- **Secondary storage**: non-volatile, moderately fast access time
  - Also called **online storage**
  - Stores the current version of entire database typically on a magnetic disk.
  - Access time in milliseconds
- **Tertiary storage**: non-volatile, slow access time
  - also called **off-line storage** – often used for archiving older versions of the database
  - Large capacity, access time seconds / minutes.
  - E.g. magnetic tape, optical storage

Large speed gap between Memory and Disk

- The large <u>speed gap</u> between Memory and Disk remains the key issue in DBMS performance.

- Time to access information in disk is the **major determining factor in system performance**.

- The ***number of disk I/Os*** (block accesses) is often used to measure the cost of a database operation.

Relative Daps in Access Time

0.5 ns	L1 Cache Ref	0.5 sec
7 ns	L2 Cache Ref	7 sec
100 ns	DRAM	100 sec
150,000 ns	SSD	1.7 days
10,000,000 ns	HDD	16.5 weeks
~30,000,000 ns	Network Storage	11.4 months
1,000,000,000 ns	Tape Archives	31.7 years

Source: https://gist.github.com/hellerbarde/2843375

- Each level is thousands of times faster than the level below it.
- **Dominance of I/O cost**: A modern CPU can execute millions of instructions while reading a block.

Hard Disks

- Secondary storage device of choice.
- Data is stored and retrieved in units called *disk blocks* or *pages* (typically 4 or 16 kilobytes)
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.

  - Reading several pages in sequence from a disk takes much less time than reading several random pages
- Therefore, **relative placement of pages** on disk has major impact on DBMS performance!

What's Inside A Disk Drive?

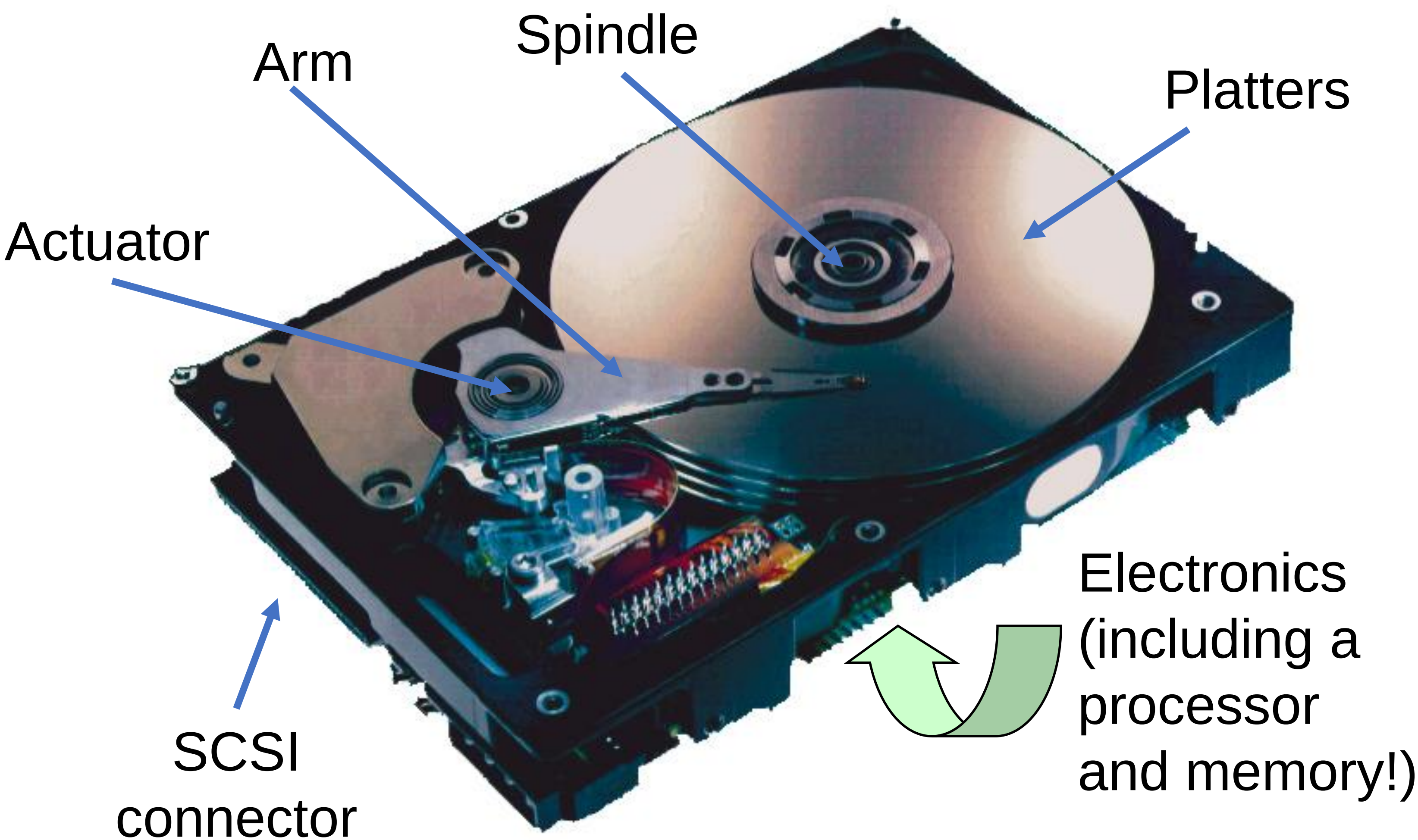
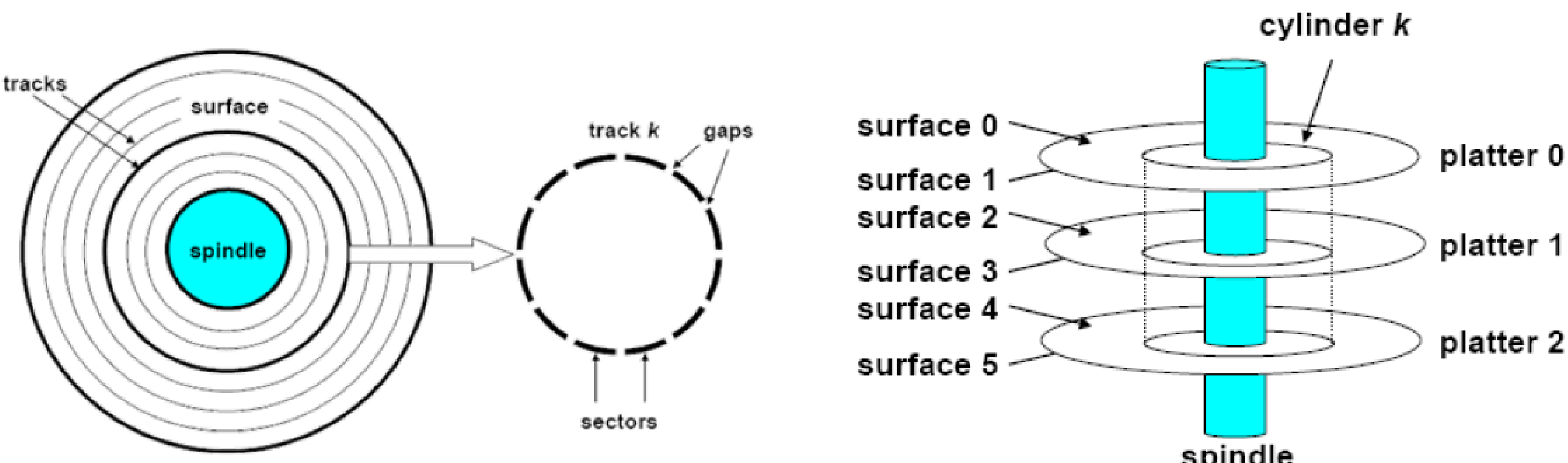


Image courtesy of Seagate Technology

Inside the HD http://www.youtube.com/watch?v=kdmLvl1n82U

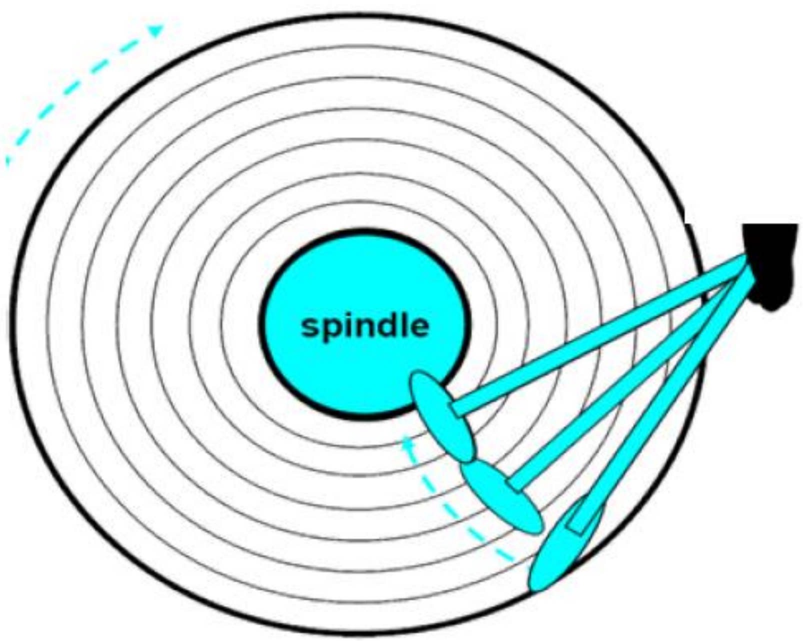
Disk Physical Structure

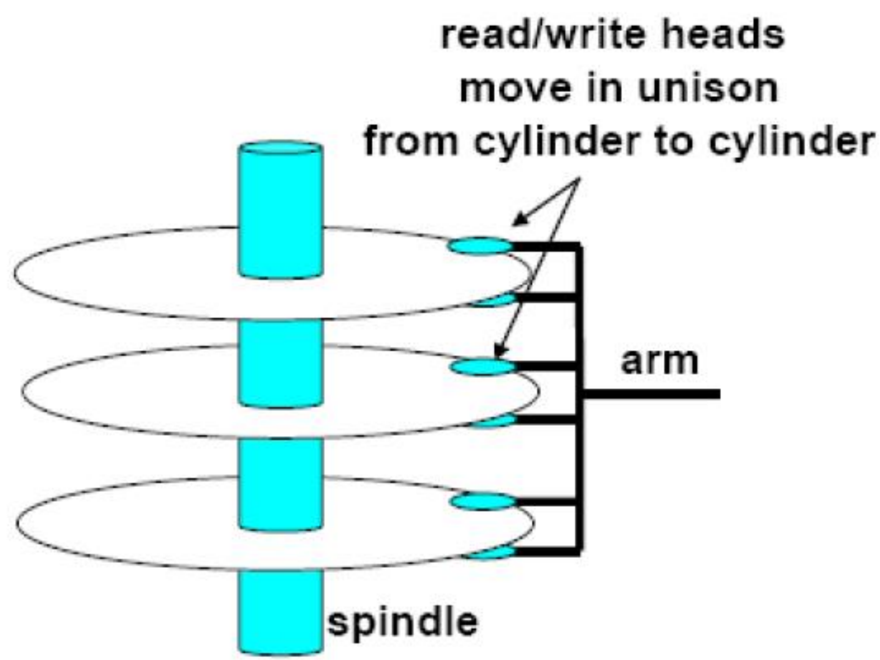
- Disks consist of platters, each with two surfaces
- Each surface consists of concentric rings called tracks
- Each track consists of sectors separated by gaps
  - Track capacities vary typically from 4 to 50 Kbytes or more
- All tracks under heads at the same time make a *cylinder* (imaginary!).
- Only one head reads/writes at any one time.



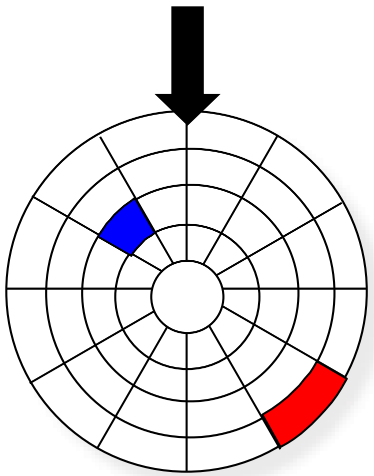
Disk Operation (Single-Platter View)

- The disk surface spins at a fixed rotational rate
- The read/write head is attached to the end of the arm and flies over the disk surface
- By **moving radially**, the arm can position the read/write head over any track



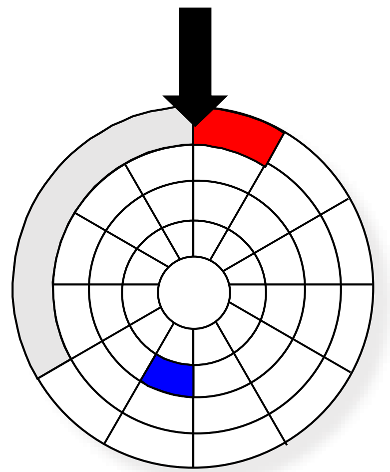


Disk Access – Service Time Components



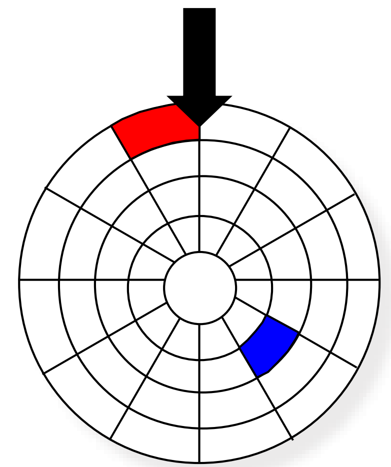
Seek for RED

Seek
moving arms to position disk head on track



Rotational latency

Rotational latency
waiting for block to rotate under head



After RED read

Data transfer
moving data to/ from disk surface

Typically about 1% of the time is actually spent on data transfer, the rest is access time.

Performance Measures of Disks

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins. Consists of:
  - **Seek time** – time it takes to reposition the arm over the correct track.
    - 4 to 10 milliseconds on typical disks
  - **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
    - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
  - **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
    - 25 to 100 MB per second rate, lower for inner tracks
- Access time dominated by **seek time** and **rotational latency**.
  - Disk is about 40,000 times slower than RAM

Disk speeds are dominated by access time

- For this reason, information on disk is always organized in Blocks – blocks are basic units of transfer and storage
  - relatively large chunks (4 or 16 kilobytes) of contiguous information that is read/written as a unit.
  - it always **reads or writes the whole block** containing a desired piece of information.
    - A system never reads or writes a single disk byte.
    - The block size B is fixed for each system.
    - Typical block sizes range from 4 to 16 kilobytes
- Mapping between logical blocks and actual (physical) sectors
  - Maintained by hardware/firmware device called disk controller.
  - Converts requests for logical blocks into (surface,track,sector) triples.

Optimization of Disk Block Access

A major goal of the design of DBMS file systems is to minimize the time spent waiting for disk accesses. 3 ways this is done:

1) Store related information on the same or nearby blocks: read and write of data on ***contiguous disk blocks*** and eliminates seek time and rotational delay time for all but the first block transfer

- Files may get fragmented over time (if data is inserted to/deleted from the file) => reorganize the database files to speed up access

2) Keeping copies of recently-used information in buffers in memory, so that if the same information is needed again if can be accessed without having to go to the disk again

3) Parallelism - spreading information across multiple disks, so that several disks can be going through the physical operations needed to access information at the same time

Example: reading two disk blocks

- Assume

  -- average seek time = 10 ms

  -- average rotational latency = 3 ms

  -- transfer time for 1 block = 0.01875 ms

- **Adjacent** block on same track

  -- access time = 10 + 3 + 2*(0.01875) ms = 13.0375 ms

- **Random** block

  -- access time = 2*(10 + 3 + 0.01875) ms = 26.0375 ms

RAID Technology

< Back

Parallelizing Disk Access using RAID Technology

- **RAID: Redundant Arrays of Independent Disks**
  - an array of independent disks **acting as a single higher-performance logical disk**, providing:
    - high capacity and high speed by using multiple disks in parallel
      - reduce the large speed gap between disks and the memory
    - high reliability by storing data redundantly, so that data can be recovered even if a disk fails

RAID goals

RAID systems seek :

- **to improve throughput** by a technique known as **striping**, in which a single file is spread over multiple disks.
  - **Parallelize** large accesses to reduce response time: multiple accesses to different parts of the same file can often be performed in parallel (assuming that the parts being accessed are on different disks).
- **to improve reliability** by replication of data, so that if a disk fails, the data it contained is available somewhere else.

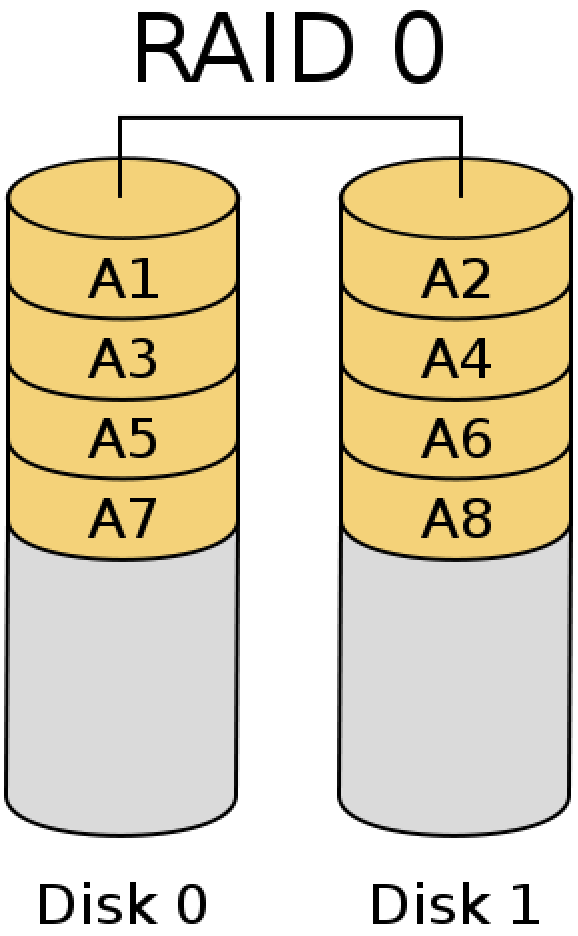
  => improve throughput for reads -if there are multiple copies of an item, then any copy can be read.

  - but creates an issue on write though - since all copies must be updated.

RAID 0 - a.k.a. Striping

- Requires two or more disks
- No lost drive space due to striping
- Fastest read and write performance.
- Raid level 0 has no redundant data and hence has the best write performance at the risk of data loss
  - Offers no data protection.
- The more disks, the more risk.

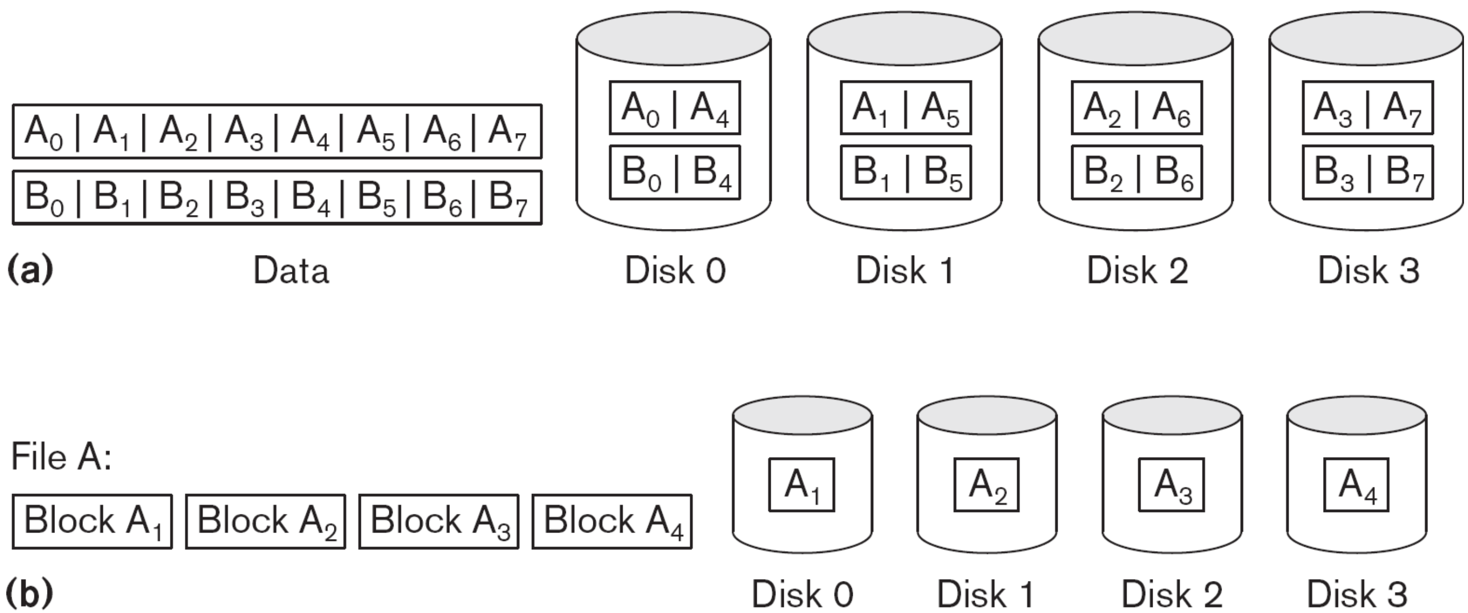
Used in high-performance applications where data loss is not critical



Data striping

- RAID uses a concept called **data striping =** distribute blocks over *n* disks in a round robin fashion.
  - Make disks appear as a single large, fast disk.
  - Requests for different blocks can run in parallel if the blocks reside on different disks

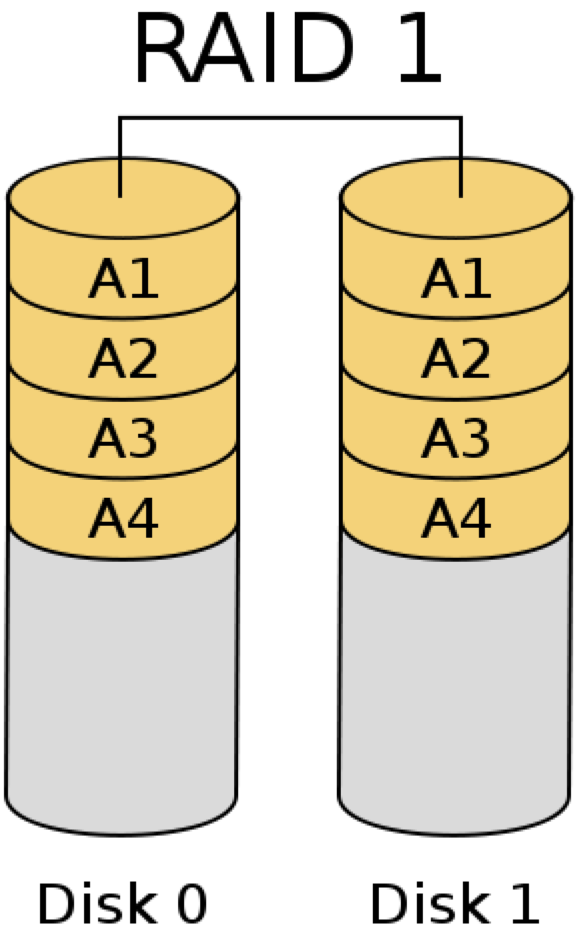
Figure 17.13
Striping of data across multiple disks.
(a) Bit-level striping across four disks.
(b) Block-level striping across four disks.



RAID 1 - a.k.a. Mirroring

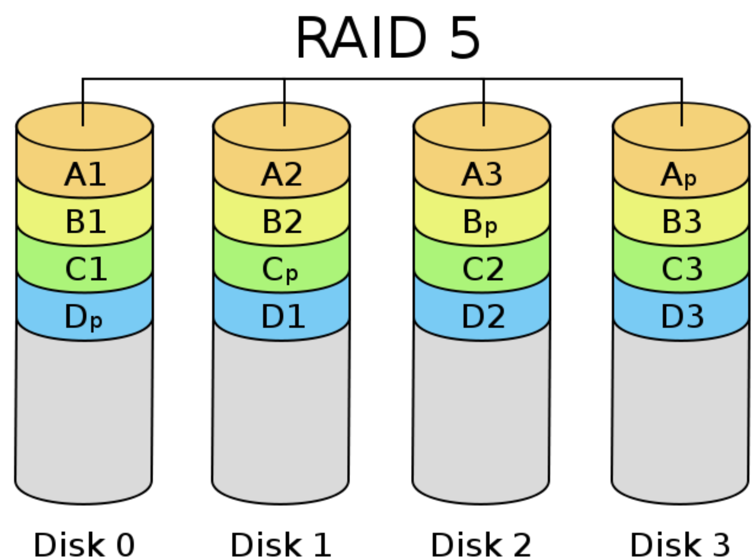
- Raid level 1 uses mirrored disks
- Write speed of one disk
- Read speed of two disks
- Capacity is equal to the size of one

Popular for applications such as storing log files in a database system.



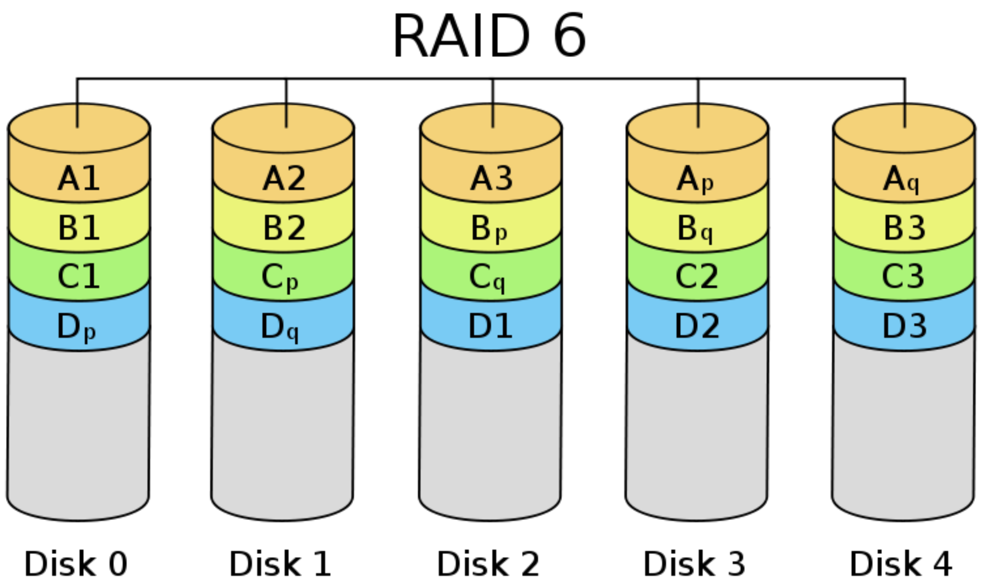
RAID 5 - Striping with Distributed Parity

- Considered best compromise between speed and storage efficiency:
  - Good performance (as blocks are striped) but slower writes due to parity
  - Good redundancy (distributed parity)
- Requires 3 or more drives
- Stripe across all drives with **parity**
- Can loose 1 drive and still function
- Capacity is **n-1** where **n** is number of drives in array

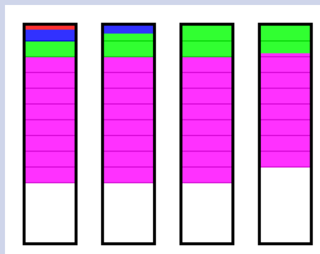
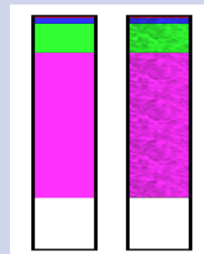
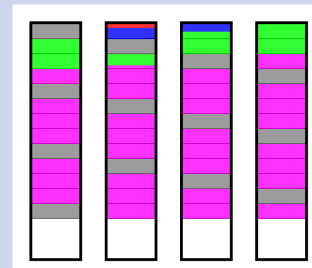
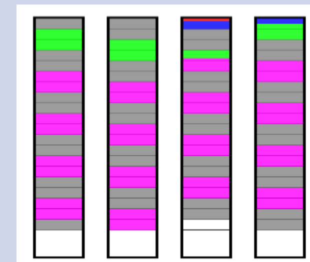


RAID 6 - RAID 5 on Steroids

- 4 or more disk
- Is a stripe with two parity drives
- Can loose two drives and still function
- Capacity is **n-2** where **n** is number of drives in array
- Protect against up to two disk failures by using just two redundant disks



Comparison of Single RAID Levels

	RAID 0	RAID 1	RAID 5	RAID 6
Diagram				
Description	Striping	Mirroring	Striping with Parity	Striping with Dual Parity
Minimum Disks	2	2	3	4
Array Capacity	No. of Drives x Drive Capacity	Drive Capacity	(No. of Drives - 1) x Drive Capacity	(No. of Drives - 2) x Drive Capacity

Comparison of Single RAID Levels

	RAID 0	RAID 1	RAID 5	RAID 6
Storage Efficiency	100%	50%	(Num of drives – 1) / Num of drives	(Num of drives – 2) / Num of drives
Fault Tolerance	None	1 Drive failure	1 Drive failure	2 Drive failures
High Availability	None	Good	Good	Very Good
Degradation during *rebuild*	NA	• Slight degradation • Rebuilds very fast	• High degradation • Slow Rebuild (due to write penalty of parity)	• Very High degradation • Very Slow Rebuild (due to write penalty of dual parity)

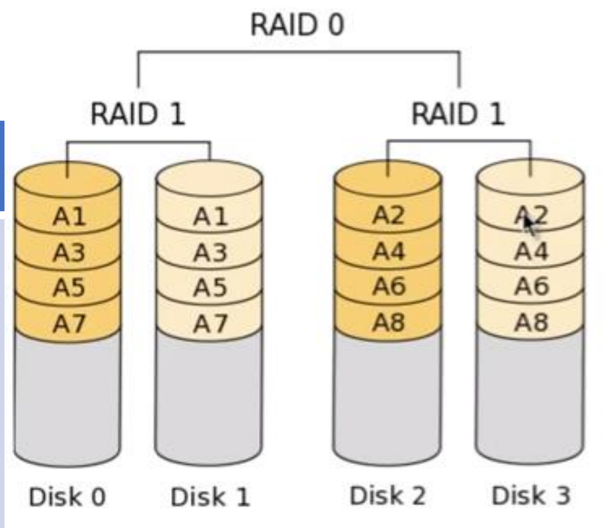
Understanding the Parity

- RAID 5 and RAID 6 store parity information against data for rebuild
- Parity can be calculated using a simple XOR
- eg– “ABCDEFGHIJKL” on a 4 disk RAID 5 array

Disk 1	Disk 2	Disk 3	Disk 4
A (01000001)	B (01000010)	C (01000011)	{P – 01000000}
Parity {P}	D	E	F
G	Parity {P}	H	I
J	K	Parity {P}	L

- If Disk 2 fails then the data “B” can be recalculated as (01000001 XOR 01000011 XOR 01000000) => 01000010 => B
- More info @ http://www.youtube.com/watch?v=LTq4pGZtzho

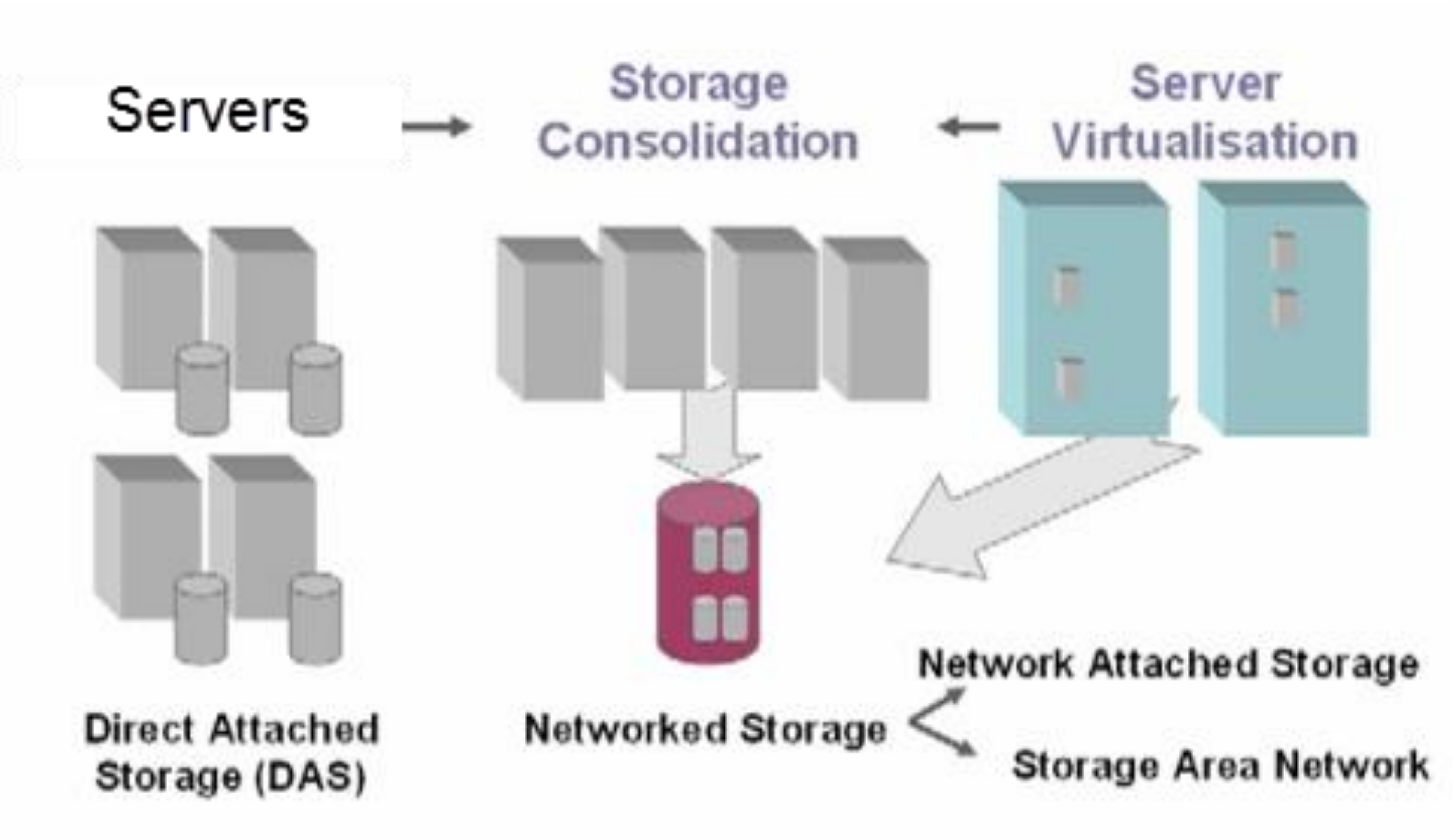
RAID 10 a.k.a. 1+0

Diagram	
Description	Mirroring then Striping
Minimum Disks	Even number > 4
Maximum Disks	Controller Dependant
Array Capacity	(Size of Drive) * (Number of Drives) / 2
Storage Efficiency	50%
Fault Tolerance	Multiple drive failure as long as 2 drives from same RAID 1 set do not fail
High Availability	Excellent

Choice of RAID Level

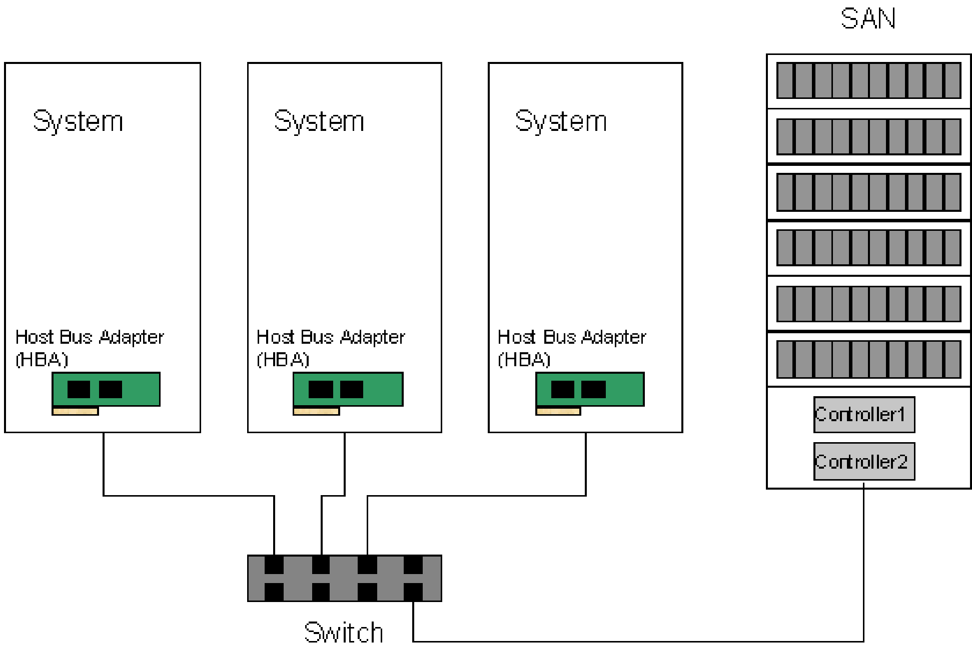
- Factors in choosing RAID level
  - Monetary cost
  - Performance: Number of I/O operations per second, and bandwidth during normal operation
  - Performance during failure
  - Performance during rebuild of failed disk
- RAID 0 is used only when data safety is not important
  - E.g. data can be recovered quickly from other sources
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for most applications

Networked Storage



Source: http://www.youtube.com/watch?v=2T99tW1KEMc

Storage Area Network (SAN)

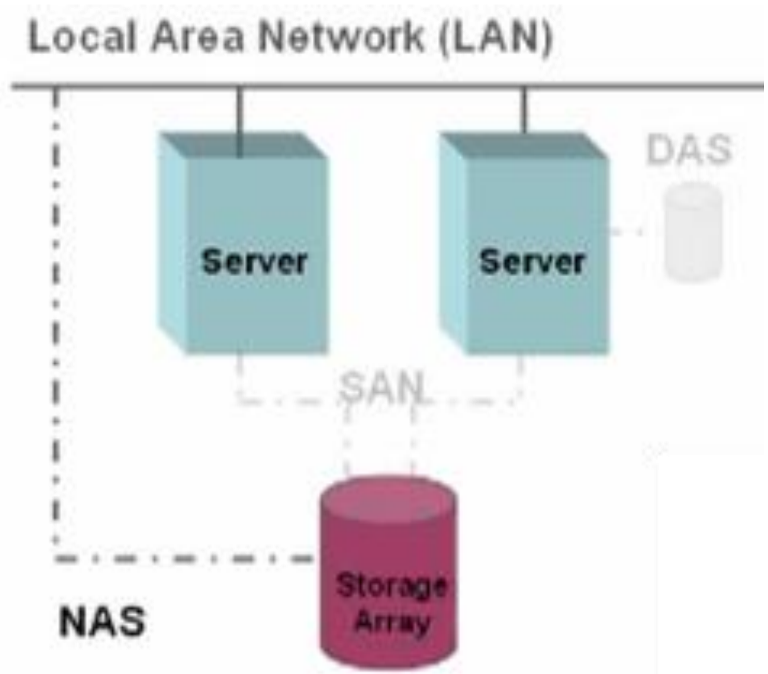


- Online storage peripherals are configured as nodes on a **high-speed network** and can be attached and detached from servers in a very flexible manner
- Servers see SAN as a virtual drives
- Dedicated access - each part of the SAN is dedicated to each server
- Block based storage

- Storage traffic segregating from the rest of the LAN. It typically uses Fiber Channel connectivity

Network Attached Storage (NAS)

- File Server optimized to serve files over the main LAN (OS dedicated to file system)
- File based storage
- Servers see NAS as a Network Share (need to map it to a drive)
- Suitable for sharing files



Database File Organization

< Back

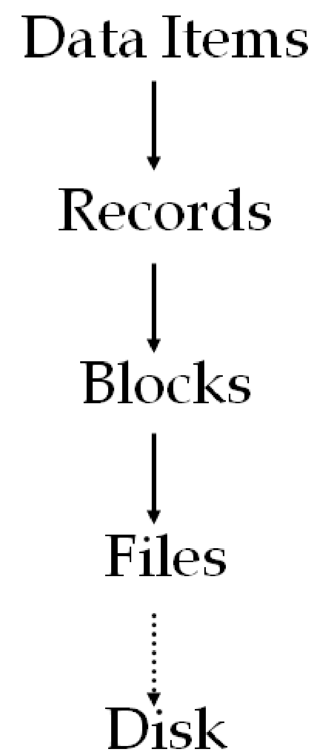
Database Storage

Database Storage addresses 2 problems:

- **Problem #1:**How the DBMS represents the database in files on disk.
  - File Storage
  - Page Layout
  - Record Layout
- **Problem #2:**How the DBMS manages its memory and efficiently move data back-and-forth from disk.

File Organization

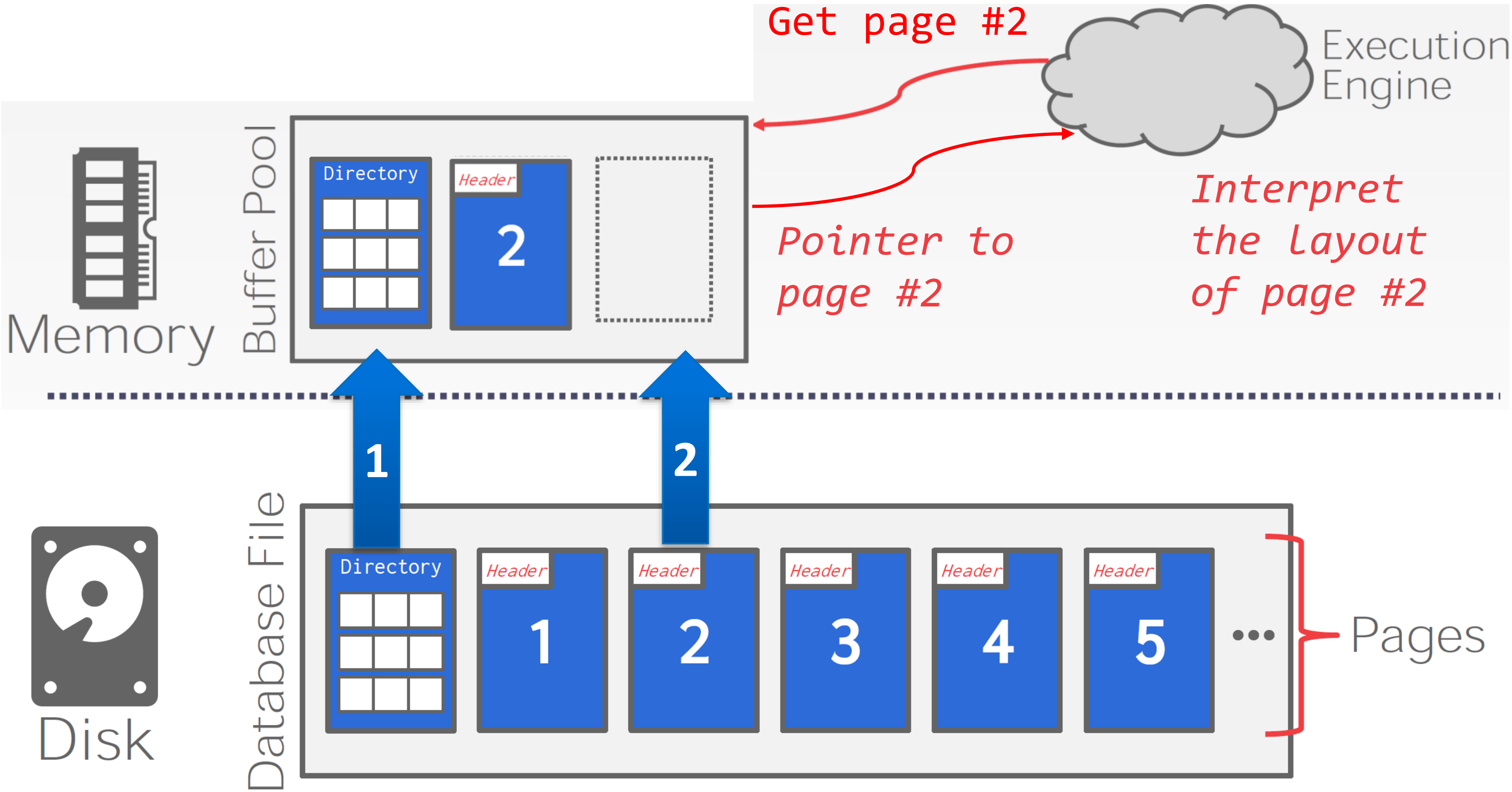
- The DBMS stores a database as one or more ***files*** on disk.
  - The OS doesn't know anything about the contents of these files.
- Each file has a collection of ***pages*** (aka blocks)
- Each ***page*** is a sequence of *records.*
- A record is a sequence of ***fields***.



Storage Manager

- The **storage manager** is responsible for maintaining a database's files.
  - It organizes the files as a collection of pages.
  - Tracks data read/written to pages.
  - Tracks the available space.

How the Storage Engine Works?



Source: CMU DB Course https://15445.courses.cs.cmu.edu/fall2019/

Why NOT use the OS?

- In DBMS the OS is NOT used for moving data pages in and out of memory.
- DBMS (almost) always wants to control things itself and can do a better job at it.
  - Flushing dirty pages to disk in the correct order.
  - Specialized prefetching.
  - Buffer replacement policy.
  - Thread/process scheduling.

Database Pages

- A **page** is a fixed-size block of data (4 to 16KB).
  - It can contain records, meta-data, indexes, log records…
- Each page is given a unique identifier.
- The DBMS uses an indirection layer (i.e., a **dictionary**) to map page ids to physical locations.

4KB	SQLite
	IBM DB2
	ORACLE
8KB	Microsoft SQL Server
	PostgreSQL
16KB	MySQL

Heap File

- DBMSs manage pages in files on disk in different ways:
  - Heap File Organization
  - Sequential / Sorted File Organization
  - Hashing File Organization

- A **heap** file is an unordered collection of pages where records are stored in random order. Supported operations include:

→ Create / Get / Write / Delete Page

→ Iterate over all pages

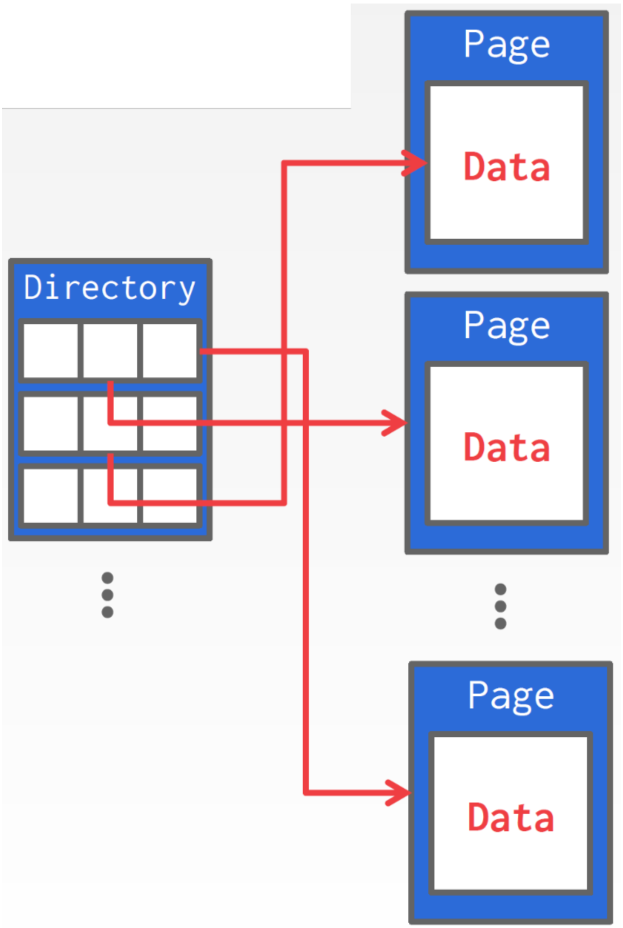
- Need meta-data to keep track of what pages exist and which ones have free space. Typically using a **Page Directory**

Heap File

- A **linear search** through the file records is necessary to search for a record
  - This requires reading and searching half the file blocks on the average, and is hence quite expensive
- Record insertion is efficient as new records are inserted at the end of the file
- Reading the records in order of a particular field requires sorting the file records.
- Deleted rows create gaps in file
  - File must be periodically compacted to recover space

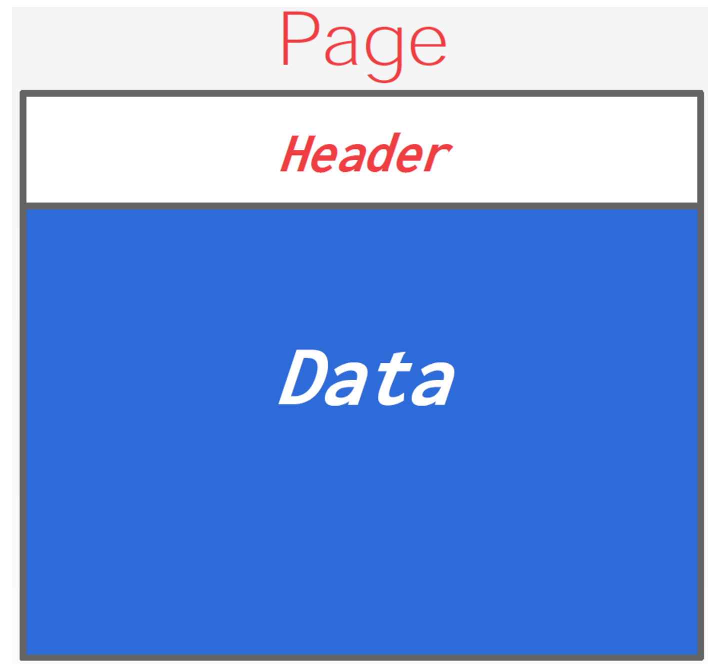
Heap File – Page Directory

- The DBMS maintains special pages that tracks the location of data pages in the database files.
- The directory also records the number of free slots per page.
- The DBMS has to make sure that the directory pages are in sync with the data pages.



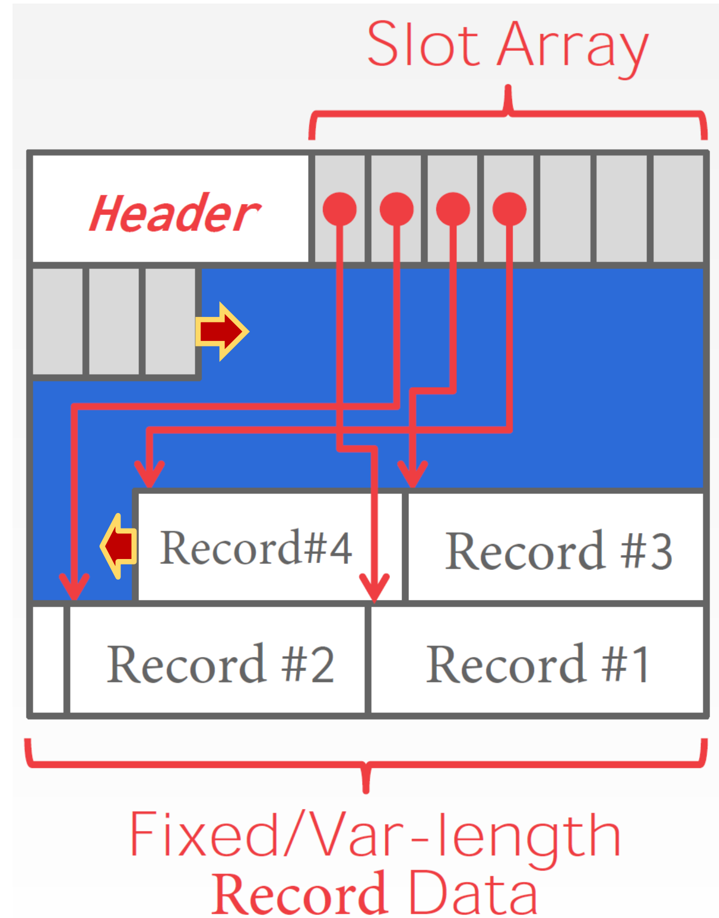
Page Layout

- Every page contains a **header** of meta-data about the page's contents:
  - Page Size
  - Checksum
  - DBMS Version
  - Compression Information
- The page data is organized using 3 approaches:
  - Row-oriented
  - Column-oriented
  - Log-structured



Row-oriented Storage

- A data layout that contiguously stores the values belonging to the columns that make up the entire row.
- The most common layout scheme is called **slotted pages**.
- The slot array maps "slots" to the records' starting position offsets.
- The header keeps track of:
  - The # of used slots
  - The offset of the starting location of the last slot used.

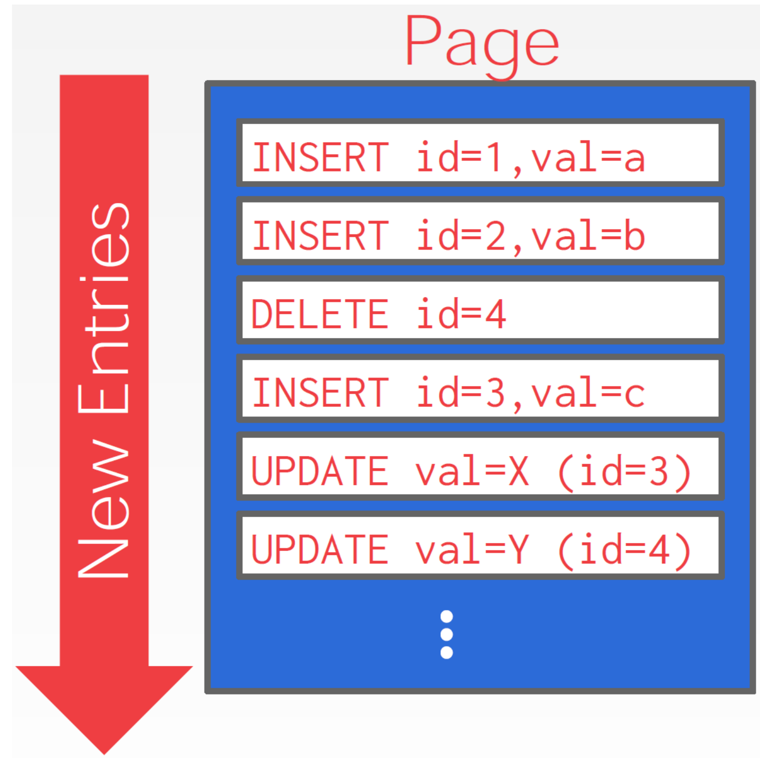


Row-oriented Storage

- Row storage model (aka "n-ary storage model")
- The DBMS stores all attributes for a single record contiguously in a page.
- Ideal for On-line Transaction Processing (OLTP) workloads where queries tend to operate only on an individual entity and insert heavy workloads.
- Advantages
  - Fast inserts, updates, and deletes.
  - Good for queries that need the entire tuple.
- Disadvantages
  - Not good for scanning large portions of the table and/or a subset of the attributes.

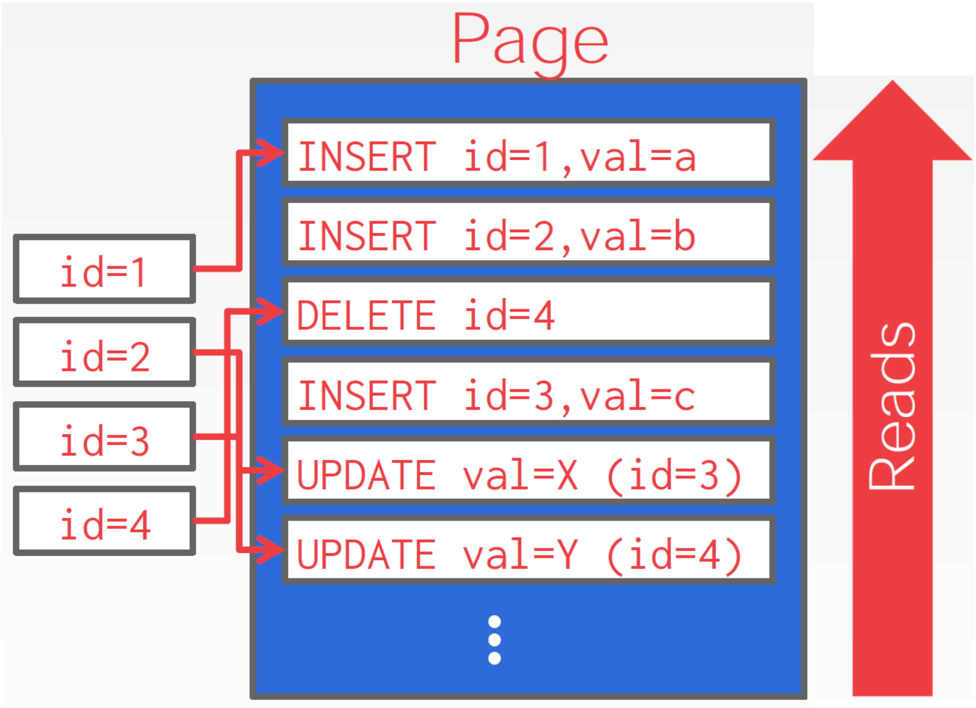
Log-structured Organization

- Instead of storing records in pages, the DBMS only stores **log records**.
- The system appends log records to the file of how the database was modified:
  - Inserts store the entire tuple.
  - Deletes mark the tuple as deleted.
  - Updates contain the **delta** of just the attributes that were modified.

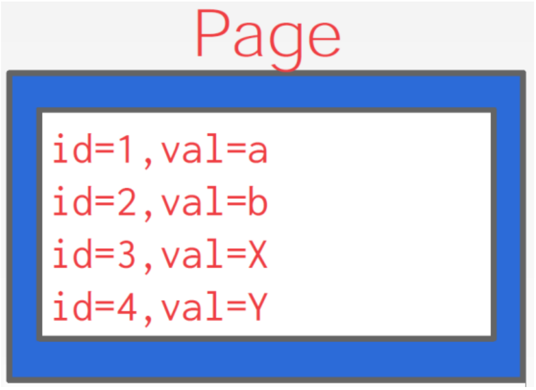


Log-structured Organization

- To read a record, the DBMS scans the log **backwards** and "recreates" the record
- Indexes can be used to allow it to jump to locations in the log.



- Periodically **compact** the log to improve read performance.



Record Layout

- The DBMS assigns each record a **unique record identifier** to keep track of individual records, commonly:

  Record id = **pageId + slot#**

- A record is essentially a sequence of bytes.
- It is the job of the DBMS to interpret those bytes into attribute **types** and **values**.
- DBMS's **catalog** contain the schema information about tables that the system uses to figure out the record's layout.


```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
  c INT,
  d DOUBLE,
  e FLOAT
);
```

Handling String of characters

VARCHAR

NULL-terminated

e.g.,

c	a	t		m	o	u	s	e		

length indicator

e.g.,

3	c	a	t	4	m	o	u	s	e	

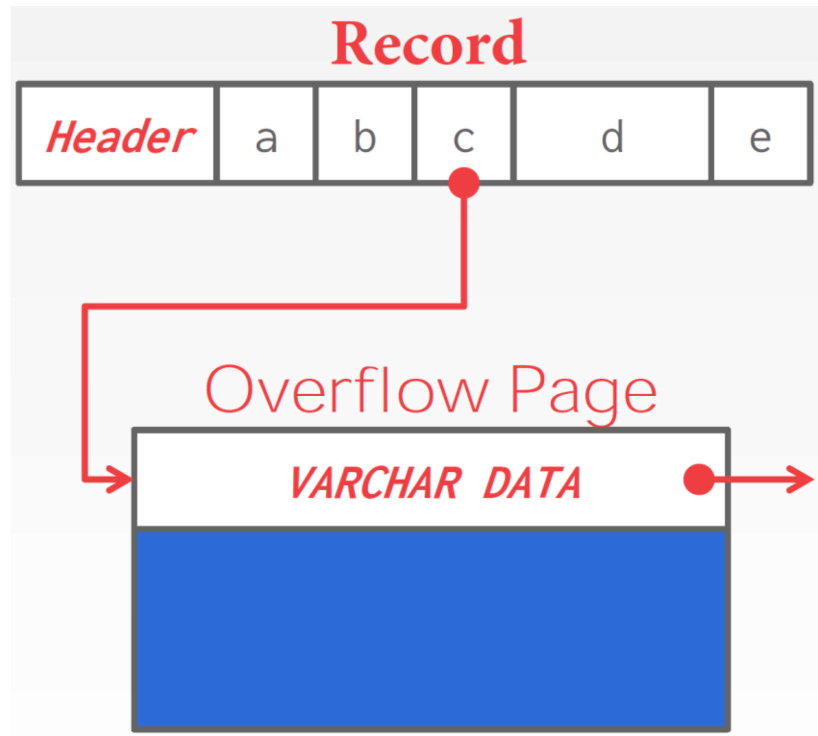
fixed length

CHAR

do not need terminating NULL
or length indicator

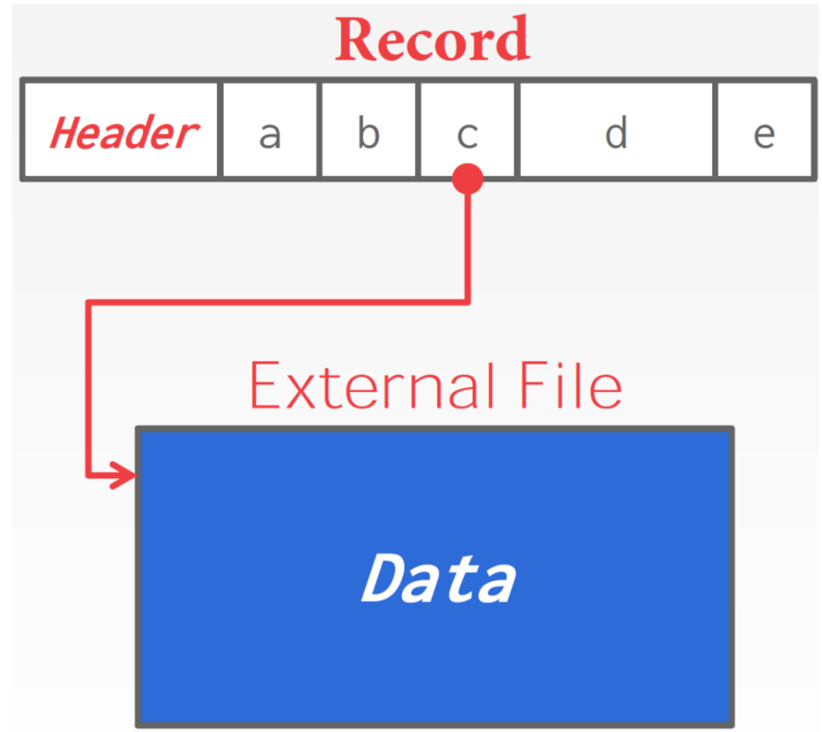
Storing Large Values

- Most DBMSs don't allow a record to exceed the size of a single page.
- To store values that are larger than a page, the DBMS uses separate overflow storage pages.



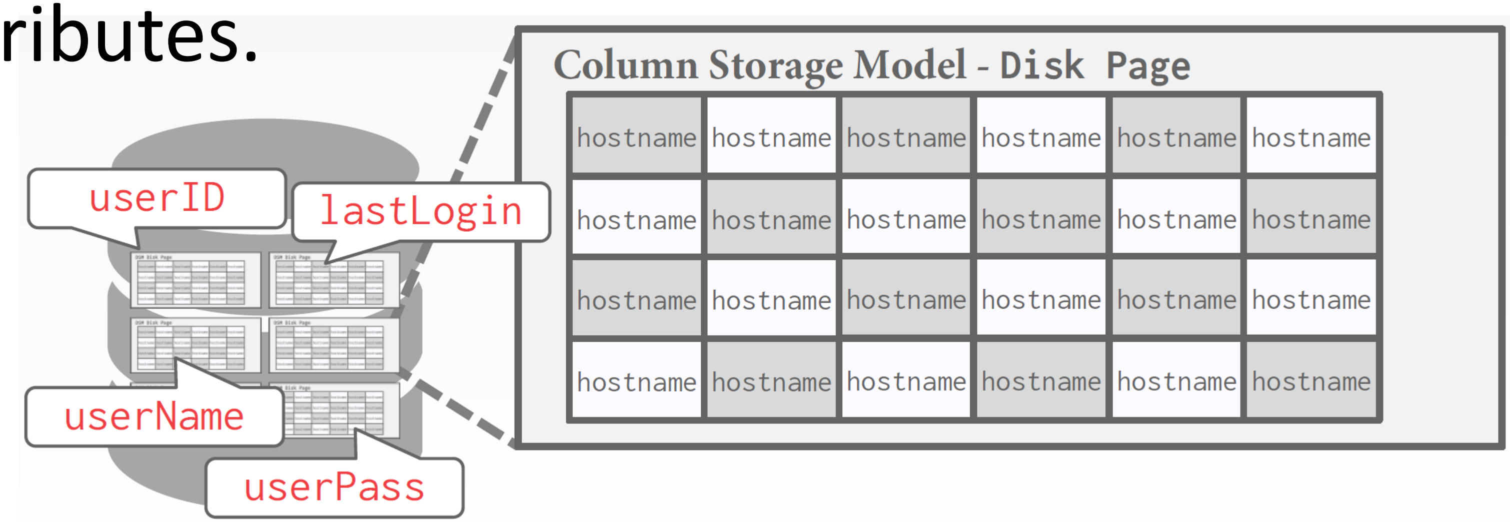
External Value Storage

- Some systems allow you to store a really large value in an external file. Treated as a **BLOB** type.
  - Oracle: **BFILE** data type
  - Microsoft: **FILESTREAM** data type
- The DBMS **cannot** manipulate the contents of an external file
  - No durability protections
  - No transaction protections



Column Storage Model

- The DBMS stores the values of a single column for all records contiguously in a page.
- Ideal for On-line Analytical Processing (OLAP) workloads where read-only queries perform large scans over a subset of the table’s attributes.



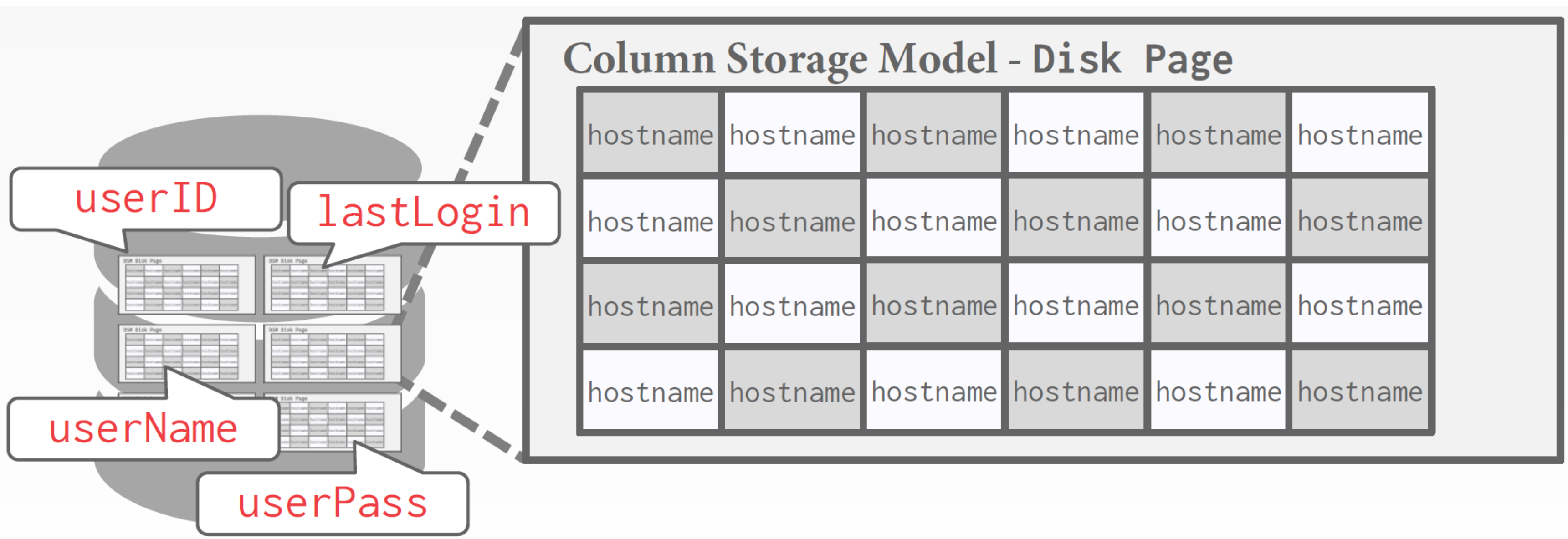
Column Storage Model

- **Advantages**
  - Reduces the amount wasted I/O because the DBMS only reads the data that it needs.
  - Better performing aggregation over large volumes of data. Or for queries that only need a few columns from a wide table.
  - Better query processing and data compression

- **Disadvantages**
  - Slow for point queries returning many columns because of record stitching
  - Slow inserts, updates, and deletes because of record splitting

Example

```sql
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```

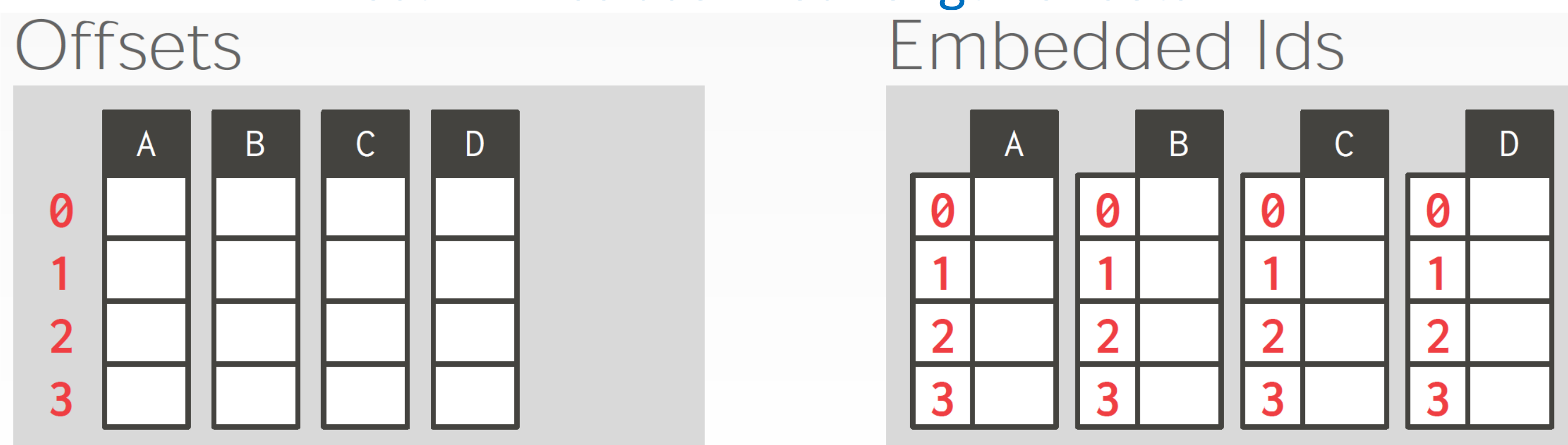


Getting Columns Belonging to a Record

To put the records back together we can use:

- **Choice #1: Fixed-length Offsets**
  - Each value is the same length for an attribute. Then when the system wants the attribute for a specific record, it knows how to jump to that spot in the file.
- **Choice #2: Embedded Tuple Ids**
  - Each value is stored with its record id in a column.

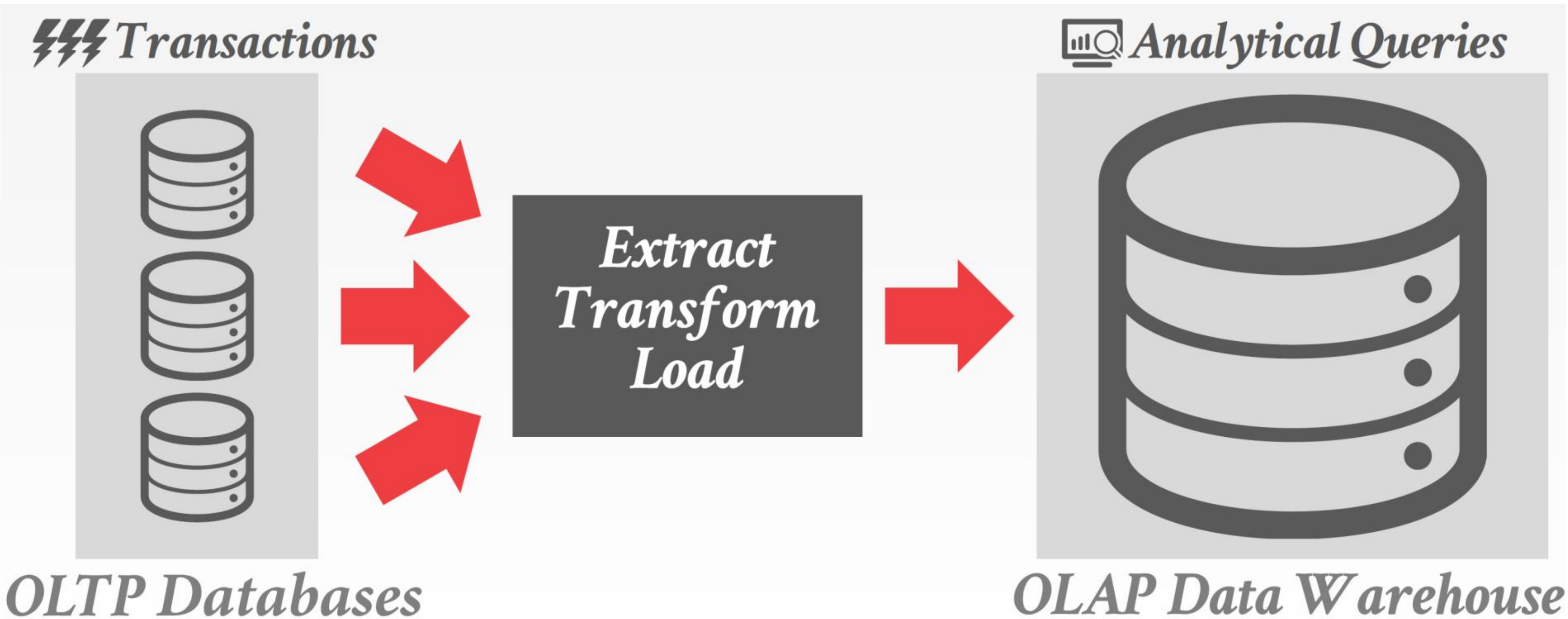
Most DBMSs use fixed-length offsets



OLTP vs. OLAP

- On-line Transaction Processing:
  - Simple queries that read/update a small amount of data that is related to a single/few entities in the database.
- On-line Analytical Processing:
  - Complex read intensive queries that read a lot of data to compute aggregates.
- The DBMS can store records in different ways that are better for either OLTP or OLAP workloads:
  - OLTP = Row Store
  - OLAP = Column Store

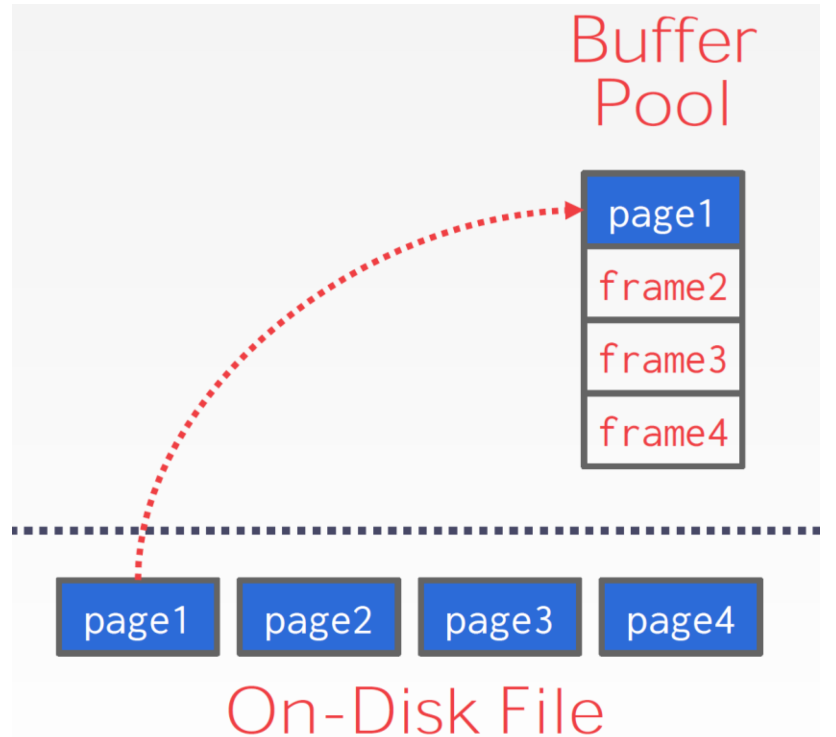
Extract Transform Load (ETL)



Buffer Manager

Buffer Pool

- Data must be in RAM for DBMS to operate on it!
- Buffer pool is an in-memory **cache** of pages read from disk
- Buffer Pool = Memory region organized as an array of fixed-size pages.
  - An array entry is called a **frame**
- When the DBMS requests a page, a copy is placed from disk into one of these frames
- Enables the higher level DBMS components to assume that the needed data is in main memory



Buffer Pool Manager

Why not use the Operating System for the task??

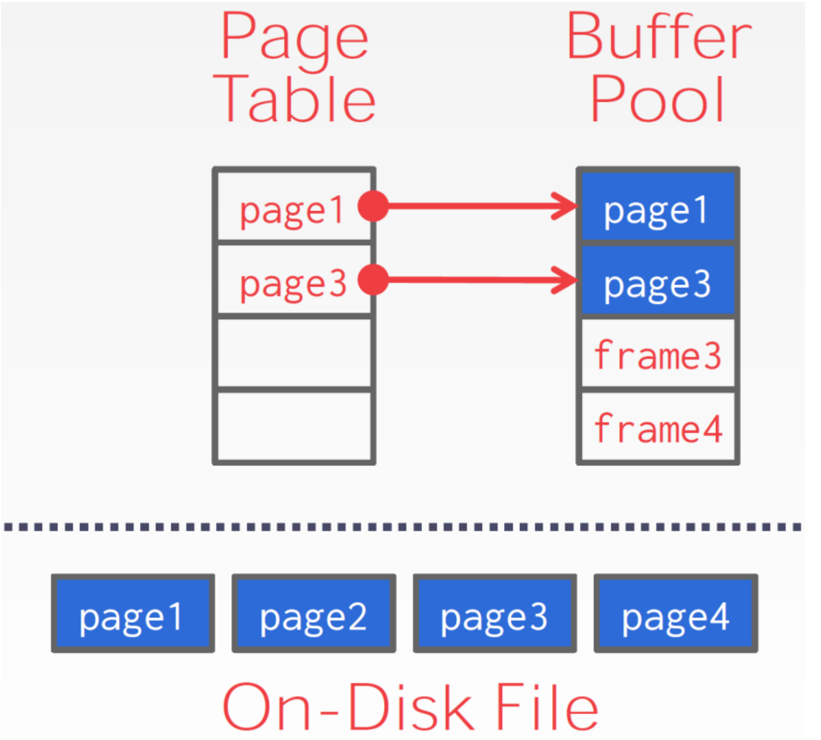
- DBMS may be able to anticipate access patterns

=> Hence, may also be able to perform prefetching

- DBMS needs the ability to force pages to disk

Buffer Pool Metadata

- The **page table** keeps track of database pages that are currently in the buffer pool frames.
- Also maintains additional meta-data per page:
  - **Pin Counter**
  - **Dirty Flag**
- Page table contains:



<frame#, pageid, pin_count, dirty>

Buffer Pool Metadata

- **Pin Counter**: Tracks the number of threads that are currently accessing the frame.
  - A thread has to increment the counter before they access the frame.
  - If Pin Counter > 0, then the storage manager is not allowed to evict that frame from memory.
- **Dirty Flag**: set to 1 when a page is modified
  - This indicates to storage manager that the page must be written back to disk

When a Page is Requested ...

- If requested page is not in pool:
  - If no free frame available, choose a frame for *replacement*
    Only frames with pin_count == 0 are candidates!
  - If frame is not ***dirty***, then simply evict it.
  - If frame is ***dirty***, write it to disk to ensure that its changes are persisted.
  - Read requested page into chosen frame
- *Pin* the page and return its address

More on Buffer Management

- Requestor of page must eventually unpin it (to indicate it is no longer needed) + 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.
  - To pin a page, pin_count++
  - A page is a candidate for replacement if *pin count* == 0 (*"unpinned"*)

Buffer Replacement Policies

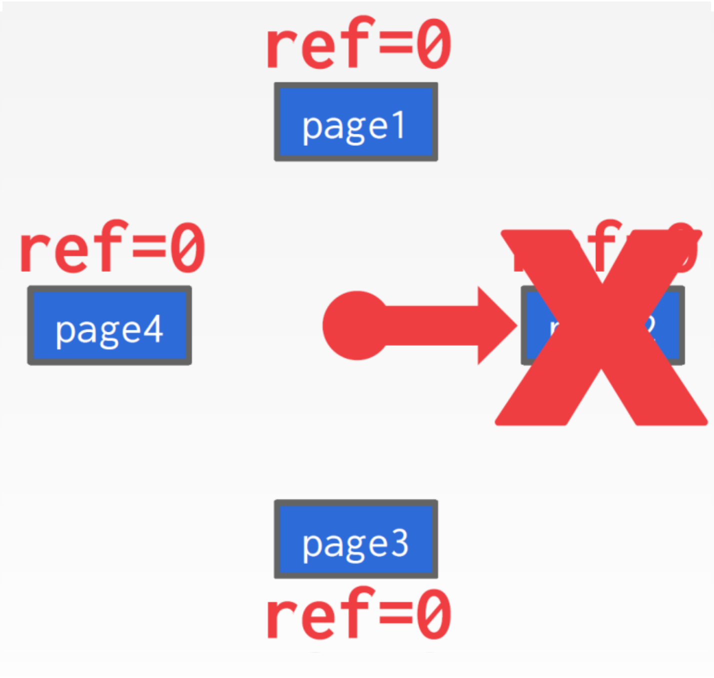
- The buffer pool provides space for a limited number of disk pages
- When the DBMS needs to free up a frame to make room for a new page, it must decide which page to *evict* from the buffer pool
- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU)
  - LRU-K
  - Private Pool Space per Query
  - Priority hints

Least Recently Used (LRU) Policy

- Least Recently Used (LRU) Policy:
  - Maintain a timestamp of when each page was last accessed.
  - When the DBMS needs to evict a page, select the one with the oldest timestamp.

Clock

- Approximation of LRU without needing a separate timestamp per page.
  - Each page has a reference bit
  - When a page is accessed, set to 1
- Organize the pages in a circular buffer with a "*clock hand*":
  - Upon sweeping, check if a page's bit is set to 1.
  - If yes, set to zero. If no, then evict
  - Clock hand remembers position between evictions

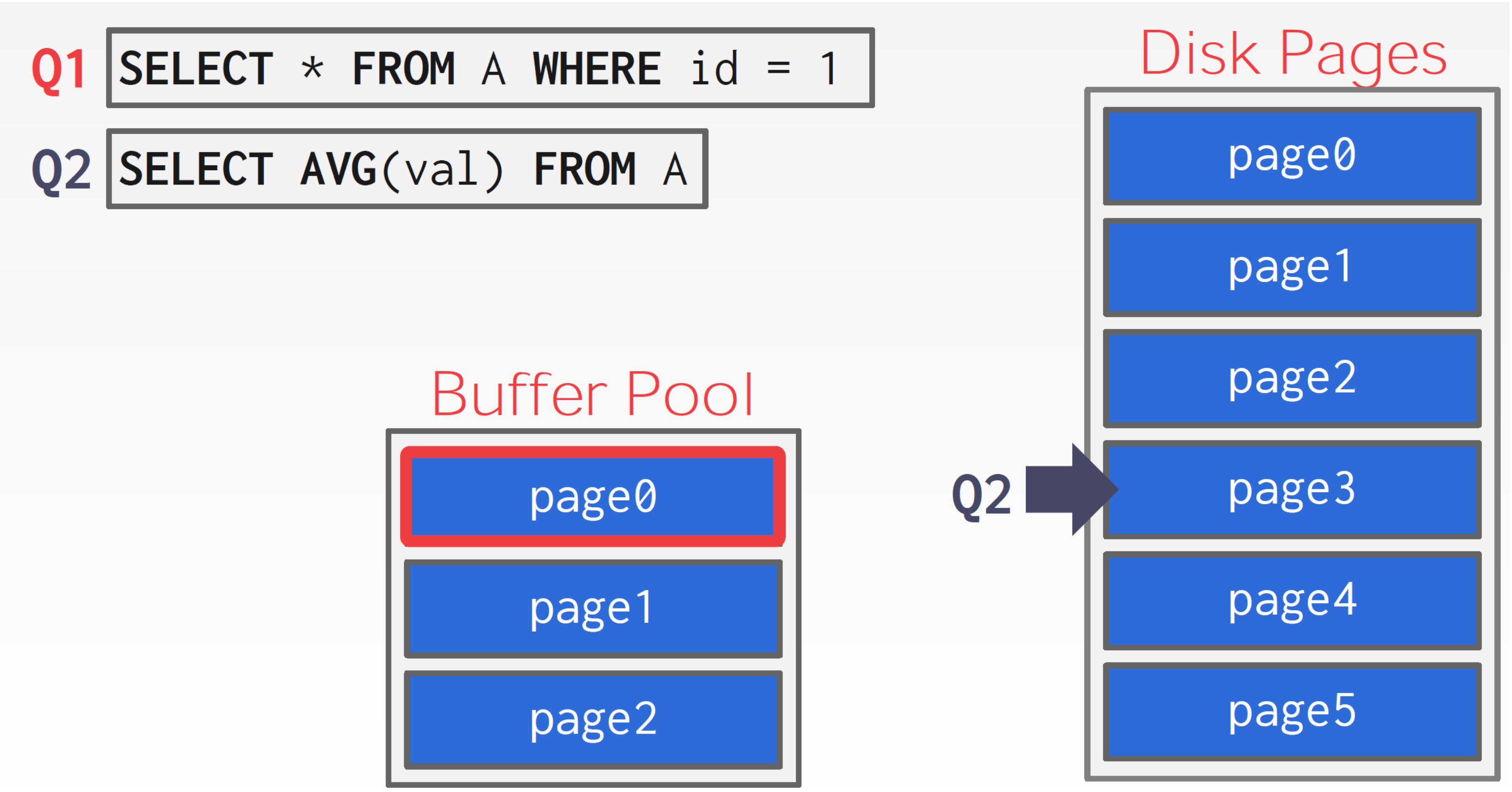


LRU Problems

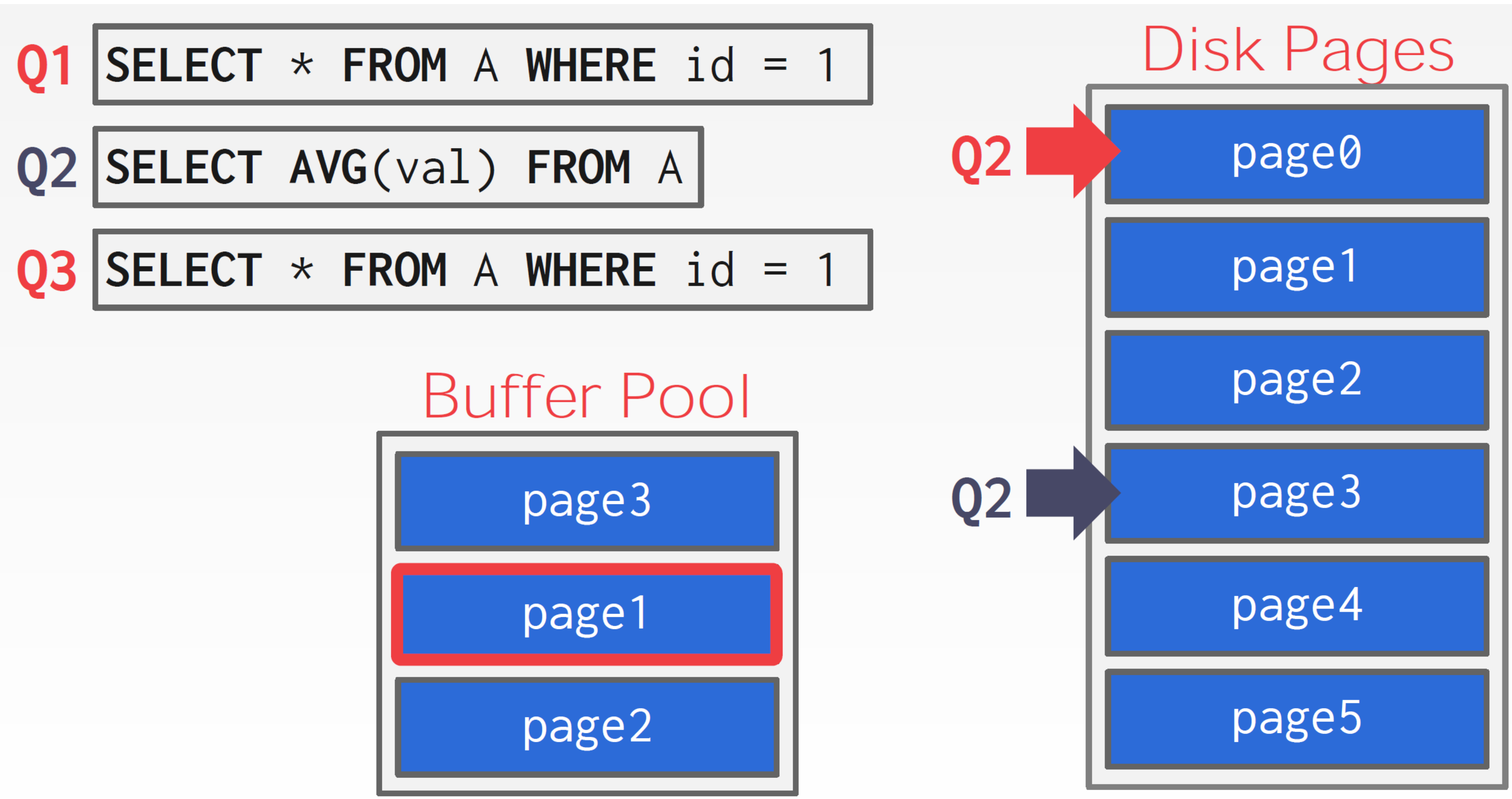
- LRU and CLOCK replacement policies are susceptible to ***sequential flooding***
  - A query performs a sequential scan that reads every page => causing trashing the buffer pool contents due to a sequential scan
  - This pollutes the buffer pool with pages that are read once and then never again

- The most recently used page is actually the most unneeded page.

Sequential Flooding



Sequential Flooding



LRU-K

- Track the history of the last **K** references as timestamps and compute the interval between subsequent accesses
- The DBMS then uses this history to estimate the next time that page is going to be accessed

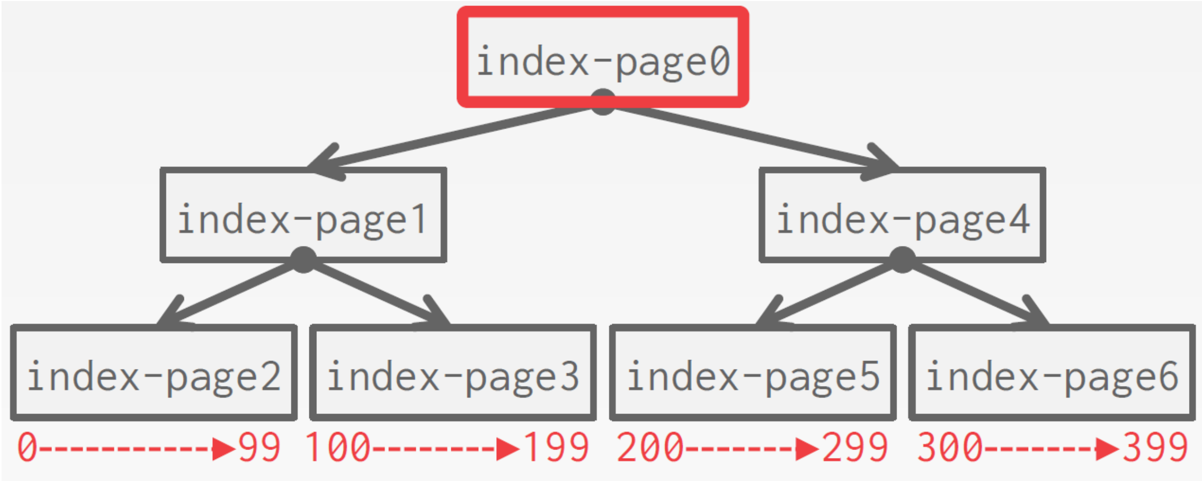
Private Pool Space per Query

- The DBMS chooses which pages to evict on a per transaction/query basis. This minimizes the pollution of the buffer pool from each query.
  - Keep track of the pages that a query has accessed.

Priority hints

- The DBMS knows what the context of each page during query execution
- It can provide hints to the buffer pool on whether a page is important or not

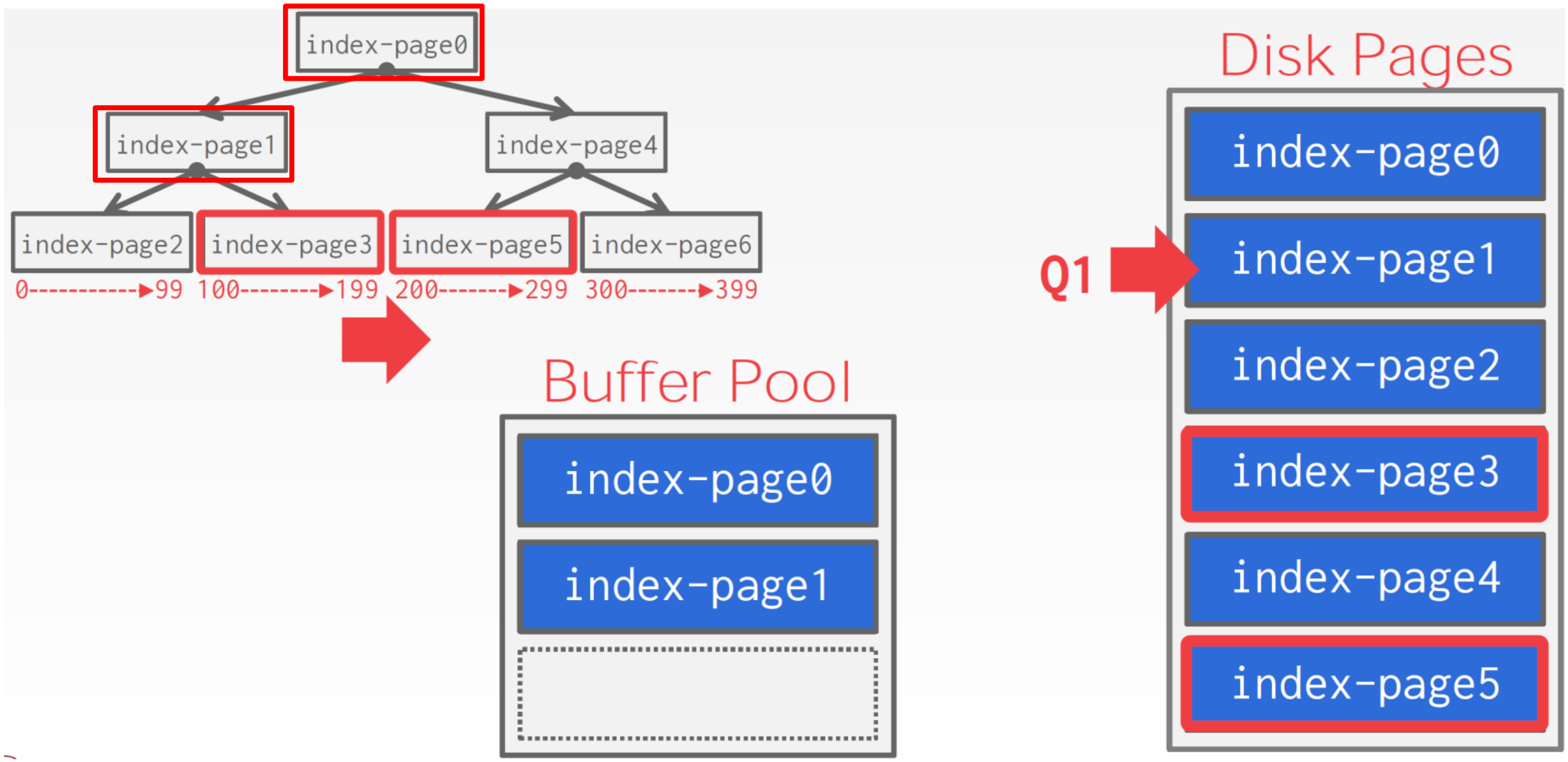
e.g., The **root** of a B+Tree index is always kept on the Buffer Pool



Pre-fetching

- The DBMS can prefetch pages based on the query plan
  - Sequential Scans
  - Index Scans

Q1
```
SELECT * FROM A
 WHERE val BETWEEN 100 AND 250
```



Summary

- I/O times dominate DBMS performance
- Techniques to improve I/O time:
  - Buffer Pool manager: leverages the semantics about the query plan to make better Pre-fetching and Eviction decisions.
  - RAID Technology
  - Store related data on contiguous blocks
  - To keep good performance, the DBMS must occasionally rebuild the database files to merge in the overflow pages and reclaim unused blocks