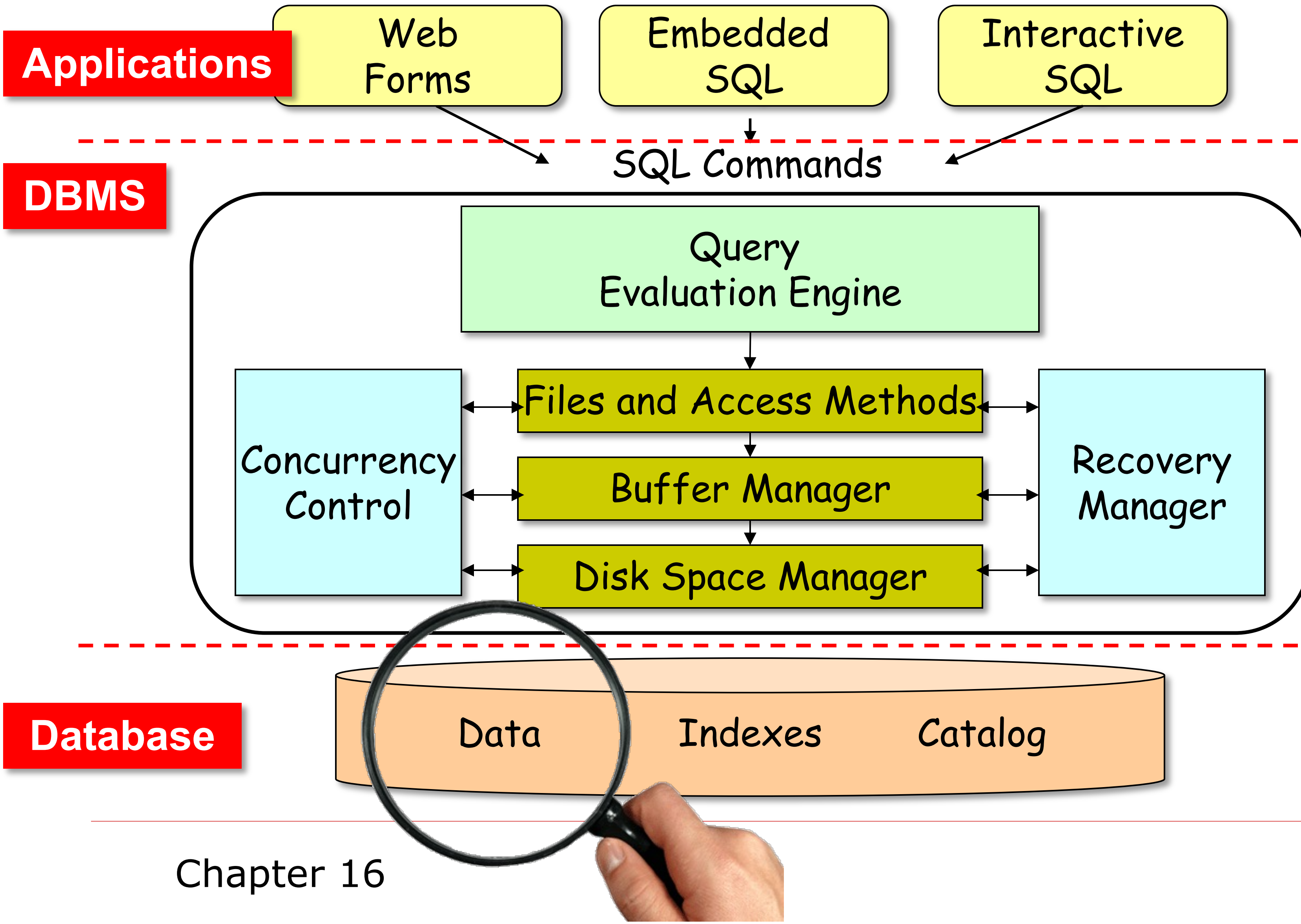
Database Management System (DBMS)



Queries and Transactions

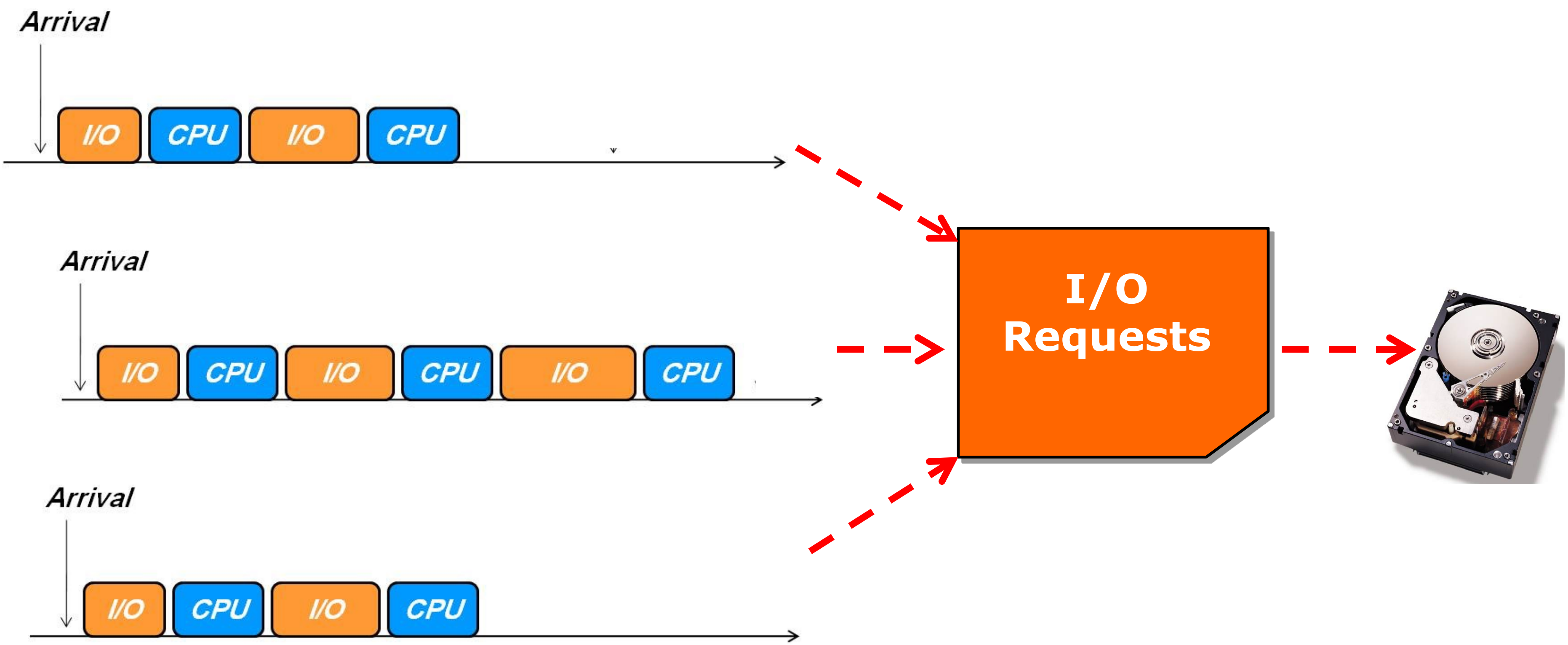
- Queries: requests to the DBMS to retrieve data from the database (=Reads)
- Updates: requests to the DMBS to insert, delete or modify/update existing data (=Reads+Writes)

- **Transactions**: logical grouping of query and update requests to perform a task
  - Logical unit of work (like a function/subroutine)
  - Example:
    - ATM withdraw:

    x=read(balance), x=x-$200, write(balance, x)

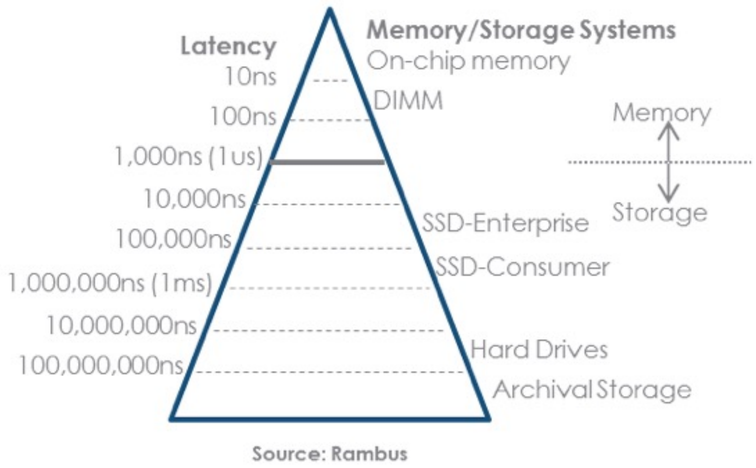
Transaction I/Os

- I/O requests: read or write



Storage Hierarchy

1. **Primary**
   - Random Access Memory (RAM)
2. **Secondary**
   - Disks
3. **Tertiary**
   - Tapes



Storage Options

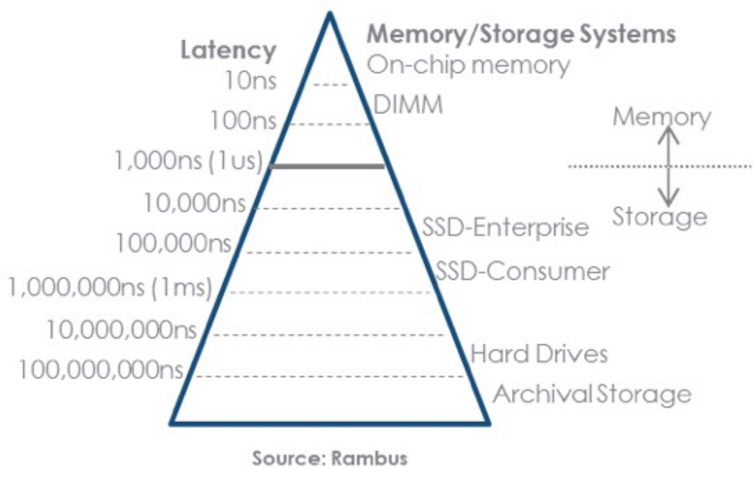
- Disk is slow (order of millisecond) ,

but RAM is fast (order of nanosecond)

- Why not store everything in RAM?
    1. Expensive cost
    2. Small capacity
    3. Volatile

Storage Hierarchy

1. **Primary**
   - Random Access Memory (RAM)
   - *For currently used data*
2. **Secondary**
   - Disks
   - *For the main database*
3. **Tertiary**
   - Tapes
   - *For archiving older data*



Disks and Files

- DBMS stores information on **hard disks**
- This has major implications on DBMS design:
  - **READ:** transfer data from disk to RAM
  - **WRITE:** transfer data from RAM to disk
  - Both are high-cost (I/O) operations (i.e., slow)
    - Must be planned carefully!

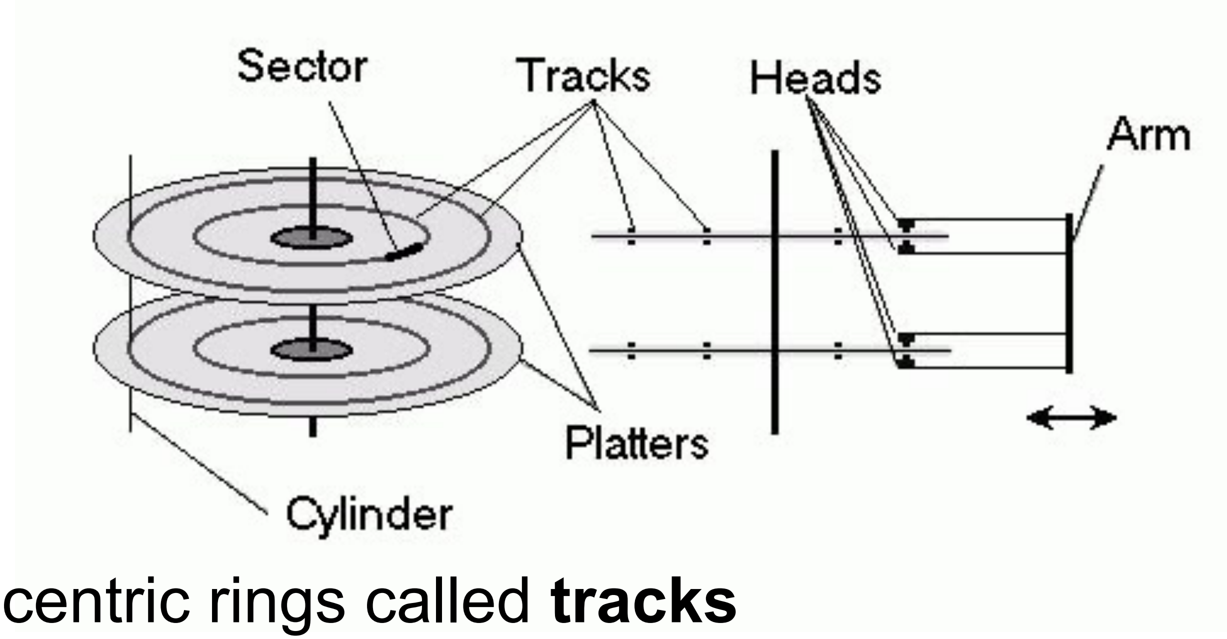
Magnetic Disks

- Data is stored and retrieved in units called disk **blocks** or **pages**
- Main advantage over tapes:

  random access (vs. *sequential*)

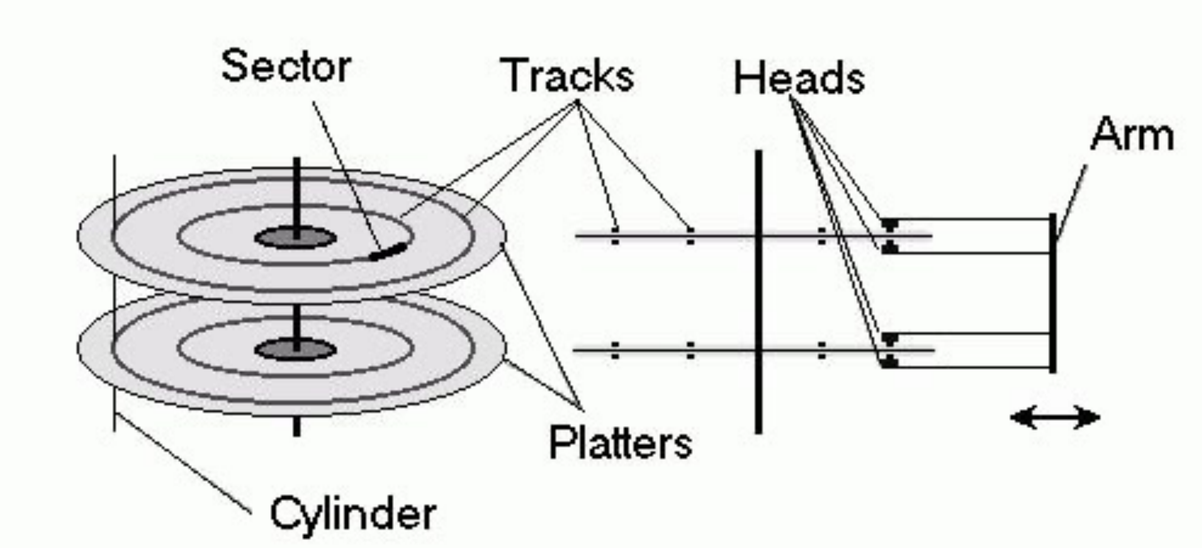
- Unlike RAM, time to retrieve a page **varies** depending upon location on disk
- *See next slides…*

Disk Components 1



- Concentric rings called **tracks**
- One or more **platter**
  - Single-sided or double-sided platters
- All tracks with same diameter = **cylinder**
- Each track is divided into **sectors**

Disk Components 2



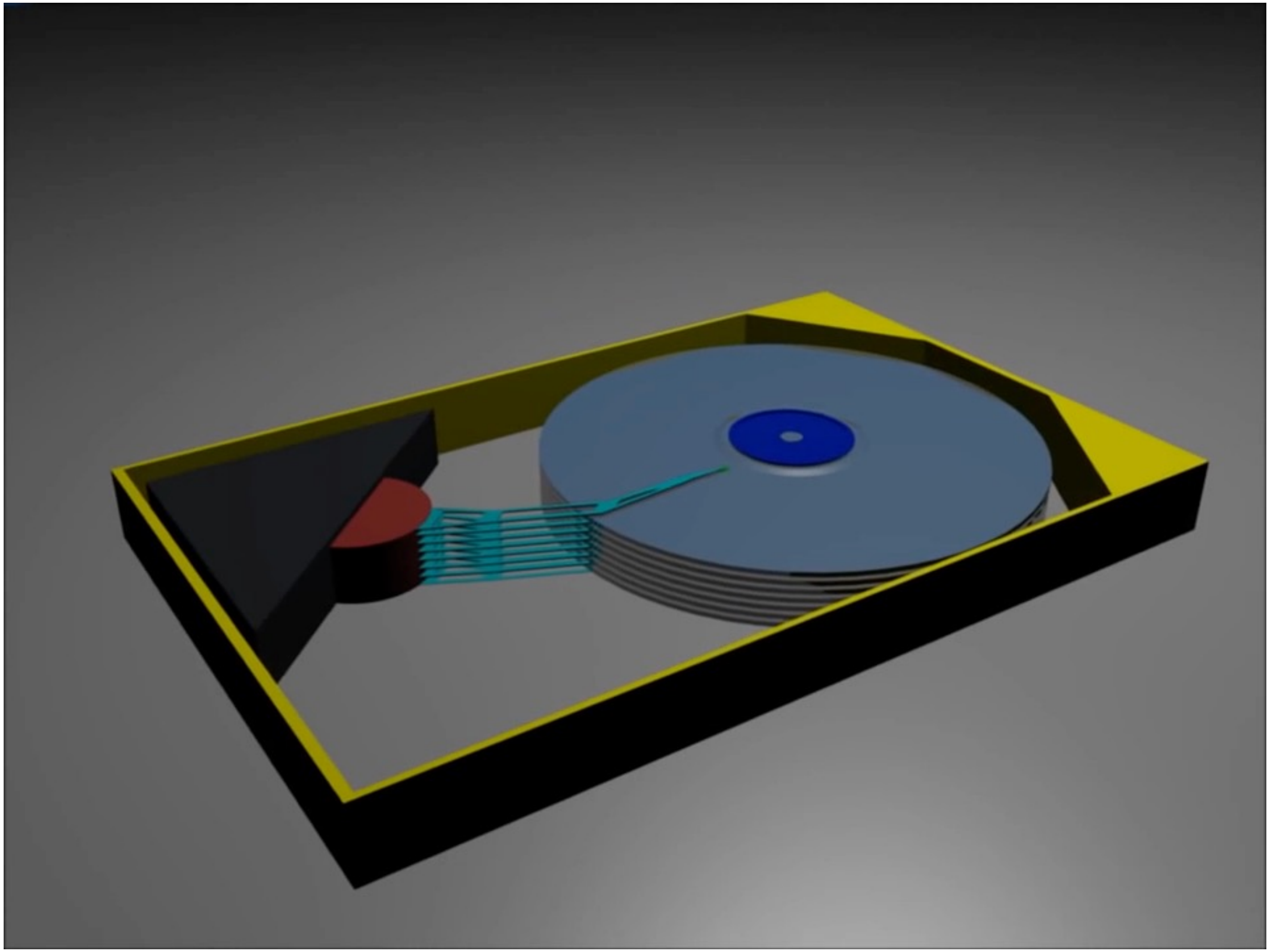
- A **Block** is multiple contiguous sectors
- Only one **head** reads/writes at any one time
- The **arm** assembly is moved in or out to position the head on a desired track

Accessing a Disk Block

- Time to access (read/write) a disk block:

1. **Seek time:**
   - Moving arms to position disk head on track
2. **Rotational delay:**
   - Waiting for block to rotate under head
3. **Transfer time:**
   - Actually moving data to/from disk surface

Hard Disk Example 1



Hard Disk Example 2

Questions

Parameter	Value
Sector size	512 bytes
# of sectors per track	50 sectors
# of tracks per surface	2000
# of platters	5 double-sided
Rotational speed	5400 rpm
Average Seek time	10 msec

1. What is the capacity of a track?
2. What is the capacity of a surface?
3. What is the disk capacity?

Given rotational speed is 5400 rpm (revolution per minutes), find:

1. the maximum **rotational delay**,
2. the average **rotational delay**

Assume one track can be transferred per rotation, what is the **transfer rate**?

What is the average time to read an entire track from the beginning?

Accessing a Disk Block

For any block of data,

Average block access time =

seek time + rotational delay + transfer time

- Seek time and rotational delay dominate!
- Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?


LIAR! He just upgraded the processor!
Just tuned the execution time of a query from 5.6 sec to 5.2 sec !
www.SQLServerGeeks.com
www.facebook.com/SQLServerGeeks
www.facebook.com/groups/TheSQLGeeks

But First, How is Data Stored on Disk?

Data Elements

- **Field**: a database attribute (sequence of bytes)
- **Record**: sequence of fields

```
CREATE TABLE Student (
        Sid INTEGER,
        Name CHAR(24),
        Age INTEGER);
```

0 – 3	4 – 27	28 – 31
Sid	Name	Age

- **Block**: sequence of records
- **File**: sequence of blocks

Blocks & Files

```
CREATE TABLE Student (
        Sid INTEGER,
        Name CHAR(24),
        Age INTEGER);
```

SID	Name	Age
546007	Peter	18
546100	Bob	19
546107	Ann	21
546207	Jane	20
546240	John	24
546350	Ben	18
546420	Suzy	27
546500	Peter	20

Blocks & Files

```
CREATE TABLE Student (
        Sid INTEGER,
        Name CHAR(24),
        Age INTEGER);
```

		SID	Name	Age
Block 0	Record 0	546007	Peter	18
	Record 1	546100	Bob	19
	Record 2	546107	Ann	21
	Record 3	546207	Jane	20
Block 1	Record 4	546240	John	24
	Record 5	546350	Ben	18
	Record 6	546420	Suzy	27
		546500	Peter	20

Block Header	Record 0	Record 1	Record 2	Record 3

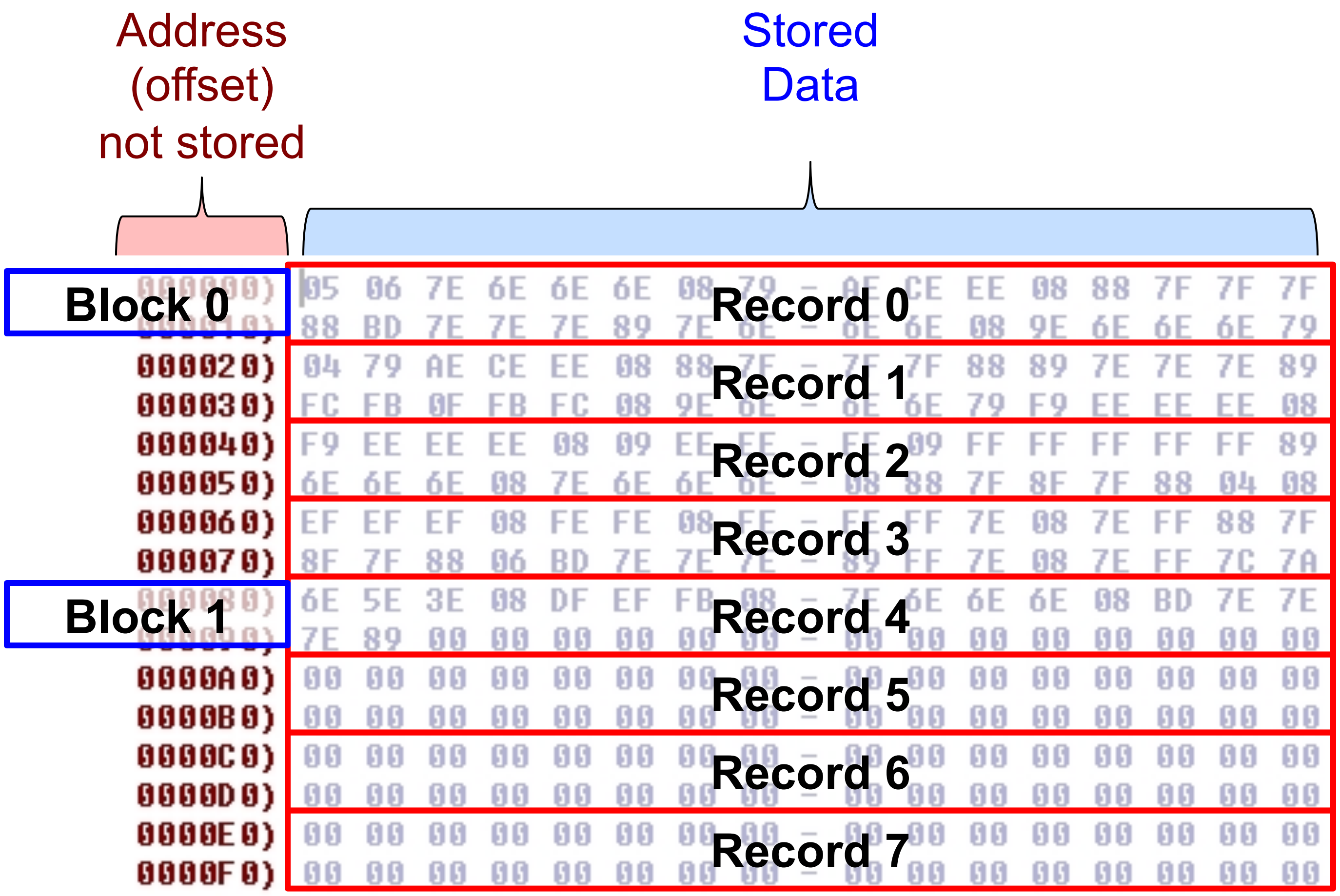
Data on Disk

Address (offset) not stored

Stored Data

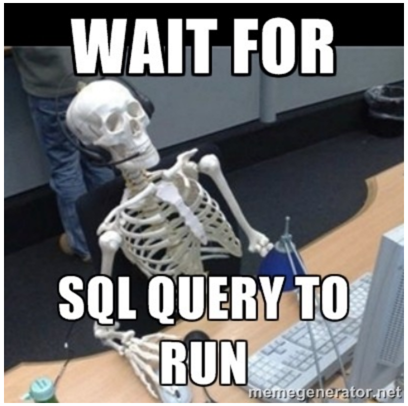
```
000000) 05 06 7E 6E 6E 6E 08 79 - AE CE EE 08 88 7F 7F 7F
000010) 88 BD 7E 7E 7E 89 7E 6E - 6E 6E 08 9E 6E 6E 6E 79
000020) 04 79 AE CE EE 08 88 7F - 7F 7F 88 89 7E 7E 7E 89
000030) FC FB 0F FB FC 08 9E 6E - 6E 6E 79 F9 EE EE EE 08
000040) F9 EE EE EE 08 09 EE EE - EE 09 FF FF FF FF FF 89
000050) 6E 6E 6E 08 7E 6E 6E 6E - 08 88 7F 8F 7F 88 04 08
000060) EF EF EF 08 FE FE 08 FE - FE FF 7E 08 7E FF 88 7F
000070) 8F 7F 88 06 BD 7E 7E 7E - 89 FF 7E 08 7E FF 7C 7A
000080) 6E 5E 3E 08 DF EF FB 08 - 7E 6E 6E 6E 08 BD 7E 7E
000090) 7E 89 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0000A0) 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0000B0) 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0000C0) 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0000D0) 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0000E0) 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0000F0) 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
```

Records on Disk



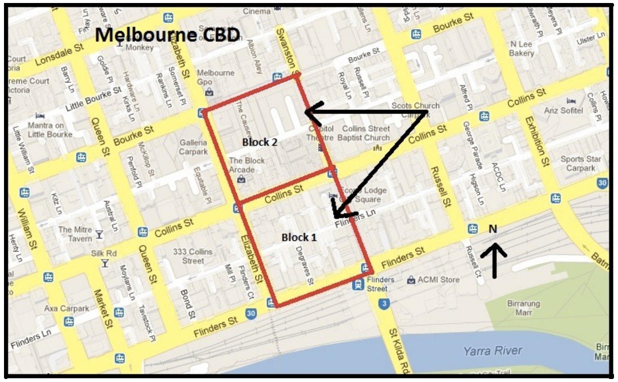
Improving Access Time

- Data access time could be improved by:
  1. Block Transfer
  2. Cylinder-based Organization
  3. Buffering and Prefetching
  4. Multiple Disks
  5. Record Placement



Block Transfer 1

- To minimize access time: data is always transferred from/to disks by **blocks** of bytes
- A disk block is an adjacent sequence of sectors from a single track of one platter
- Block size typically ranges from 512 bytes to 4KBytes.



Block Transfer 2

Block Header	Record 0	Record 1	Record 2	Record 3

- Block header includes:
  - Block ID
  - Role info such as data block, index block, etc.
  - Links to other blocks of the same table/file
  - Free-list
  - Timestamp

- **Blocking factor** bfr = $\lfloor B/R \rfloor$ *records per block*
  - B=block size and R=Record size

Block Transfer 3

- **Unspanned**
  - Records are not allowed to cross block boundaries

Record 0	Record 1	Record 2	Record 3	**Empty**

Record 4	Record 5	Record 6	Record 7	**Empty**

- **Spanned**
  - A record can span more than one block

Record 0	Record 1	Record 2	Record 3	**Record 4**

Record 4	Record 5	Record 6	Record 7	…

Queries Meet I/O

```
SELECT *
FROM Student
WHERE Age = 18
```

The DBMS could execute this query as:

For each **block**$_i$ in Student
    Read block$_i$ from disk /* I/O access */
    For each **record**$_j$ in **block**$_i$
        Check the *Age* field in **record**$_j$
        if *Age* == 18
            Output **record**$_j$
    End For
End For

Block	Record	*SID*	*Name*	*Age*
Block 1	Record 1	546007	Peter	18
	Record 2	546100	Bob	19
Block 2	Record 3	546107	Ann	21
	Record 4	546207	Jane	20
Block 3	Record 5	546240	John	24
	Record 6	546350	Ben	18
Block 4	Record 7	546420	Suzy	27
	Record 8	546500	Peter	20

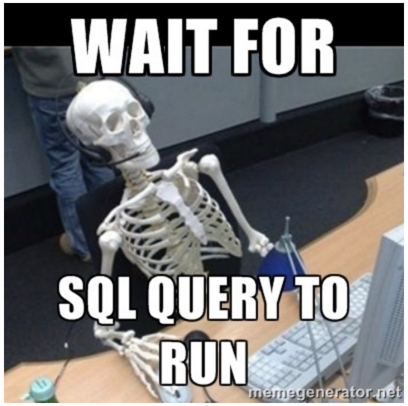
Files of Records

- Each **Read block$_i$** from disk /* I/O access */ is an example of an I/O access that might experience delay due to:
  1. Seek Latency
  2. Rotational Delay
  3. Transfer time

- To reduce delay, the DBMS uses more techniques to improve access time…

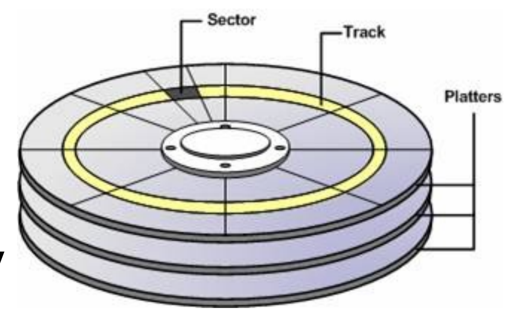
Improving Access Time

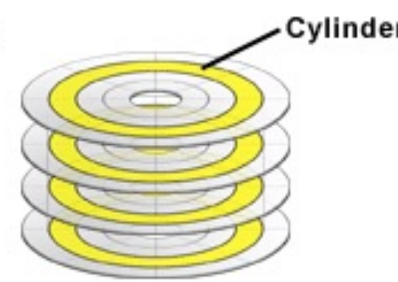
- Data access time could be improved by:
  1. Block Transfer
  2. Cylinder-based Organization
  3. Buffering and Prefetching
  4. Multiple Disks
  5. Record Placement



Cylinder-based Organization 1

- **Observation:** Data blocks in a relation are likely to be accessed together
- **Idea:** Store such blocks <u>next</u> to each other
- *"Next"* block concept:
  - Blocks on same track, followed by
  - Blocks on same cylinder, followed by
  - Blocks on adjacent cylinder





Cylinder-based Organization 2

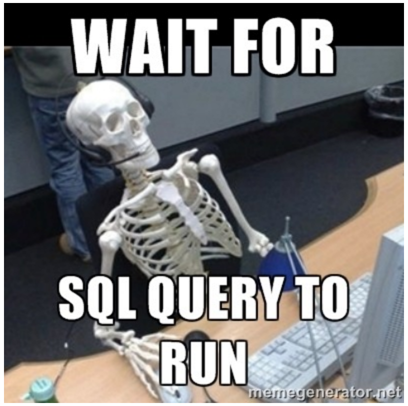
- To minimize seek and rotational delay
  - Blocks on the same track or cylinder effectively involve only:

    the first seek time and the first rotational latency

- Advantages:
  - Excellent if access pattern matches storage
    - "Next" block on disk is next block to read
  - Only one process/transaction is using the disk

Improving Access Time

- Data access time could be improved by:
  1. Block Transfer
  2. Cylinder-based Organization
  3. Buffering and Prefetching
  4. Multiple Disks
  5. Record Placement



Buffering & Prefetching 1

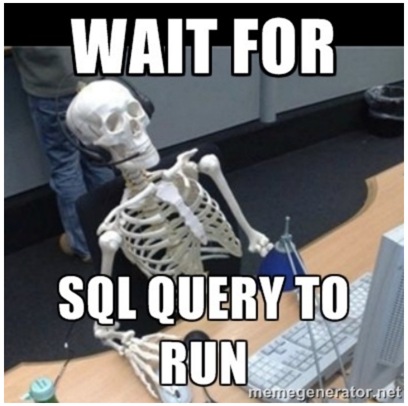
- Buffering (caching):
  - **Idea:** Keep as many blocks in memory as possible to reduce disk accesses
  - Might run out of memory space → replace blocks!
  - **Cache Replacement Policies:**
    1. LRU (Least Recently Used)
    2. LFU (Least Frequently Used)
    3. …

Buffering & Prefetching 2

- Prefetching (double buffering)
  - **Situation:** Needed data is known, but the timing of the request is unknown
  - **Idea:** speed-up access by pre-loading needed data
  - **Cons:**
    - Requires extra main memory
    - No help if requests are random (unpredictable)

Improving Access Time

- Data access time could be improved by:
  1. Block Transfer
  2. Cylinder-based Organization
  3. Buffering and Prefetching
  4. Multiple Disks
  5. Record Placement



Files of Records

- Each **Read block$_i$** from disk /* I/O access */ is an example of an I/O access that might experience delay due to:
  1. Seek Latency
  2. Rotational Delay
  3. Transfer time

- It might also experience **queuing** delay!

Queuing Delay

Queuing Delay

- **Queuing Delay** occurs when:
  - Multiple queries are executed at the same time, and
  - Those queries access the same disk at the same time (i.e., send I/O requests)
- The disk head can only serve one request at a time!
- **Solution:** Use multiple disks organization

Multiple Disks

- Disk drives continue to become:
  - smaller and cheaper

- Use multiple disks to support **parallel** access
  - 2X20 GB drives is faster than a single 40GB drive
    - But more expensive…
  - More efficient if they are kept busy!

Multiple Disks: Organizations

- Data **partitioning** over several disks
  - Pros: increases access rate
  - Problem: if popular data is stored on same disk ➔ request collisions (hot spots)
  - Cons: the cost of several small disks is greater than a single one with the same capacity
- **Mirror** disks: Disks hold identical copies
  - Pros: Doubles read rate (disk 1 **OR** 2)
  - Does not have the problem of request collisions
  - Does not slow down write requests (disk 1 **AND** 2)
  - Cons: Pay cost for two disks to get the storage of one

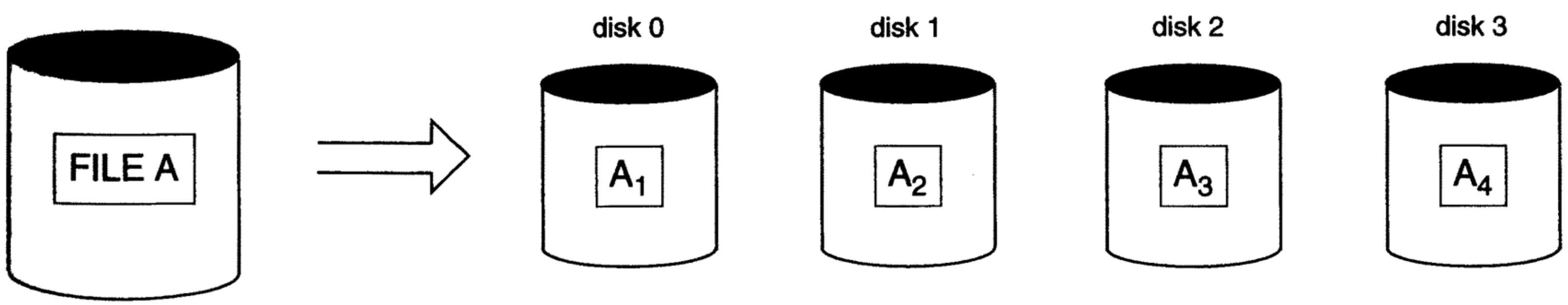
RAID Technology

RAID Technology

- A major advance in secondary storage technology is represented by the development of **RAID**
  - Redundant Arrays of Inexpensive (Independent) Disks

- Acts as a single high-performance large disk

- **Data Striping:** distributes data transparently over multiple disks to make them appear as a single large, fast disk



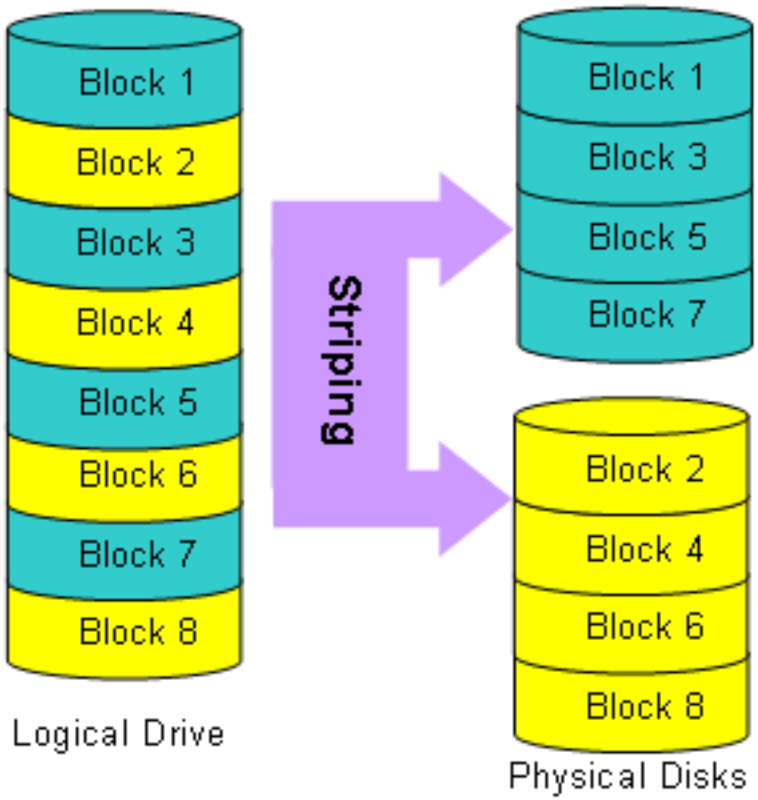
Data Striping

- **Bit-level striping:** Split groups of bits over different disks
  - Example: split each bit of a byte over 8 disks
  - Bit number *x* is written on disk number *x*
  - Every disk participates in every access
    - Every access reads 8 times as many data

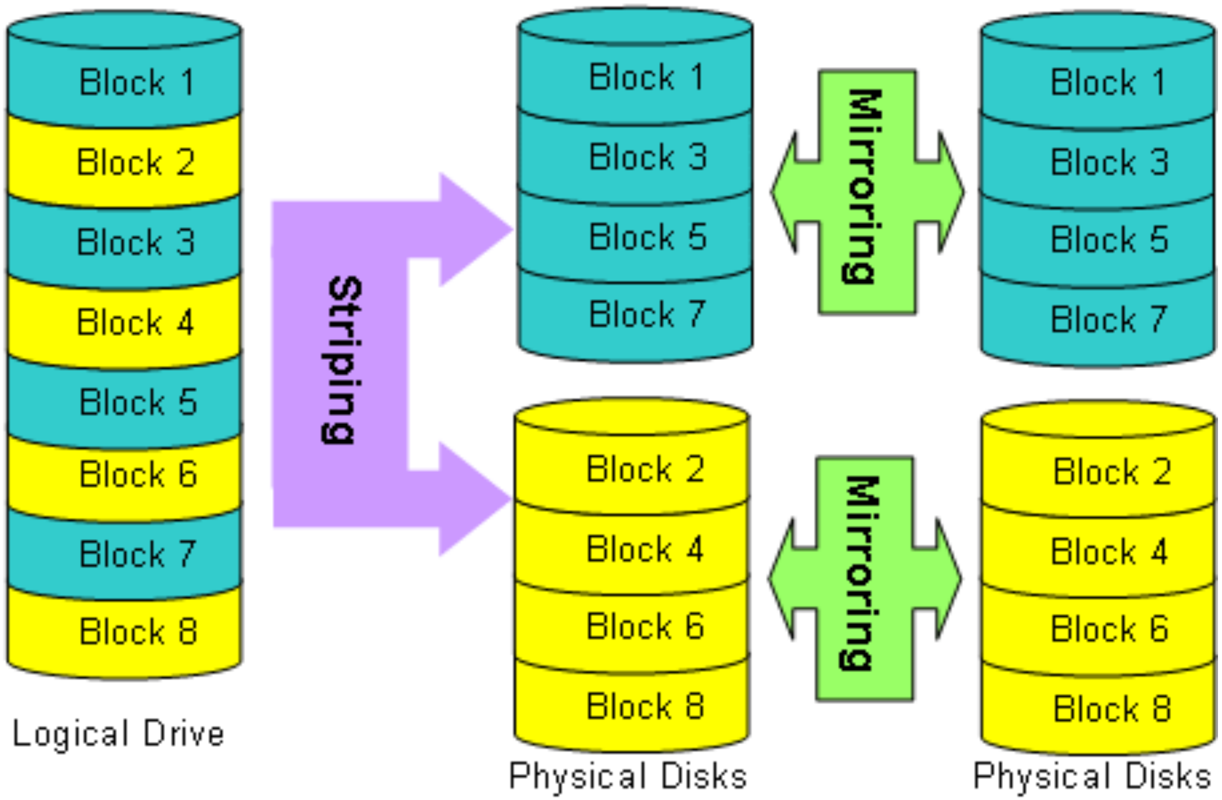
- **Block-level** striping for blocks of a file
- **Sector-level** striping for sectors of a block

- **RAID Goals:**
  - Parallelize accesses so the **access time is reduced**
  - Combining Striping, Mirroring and Reliability ➔ levels of RAID

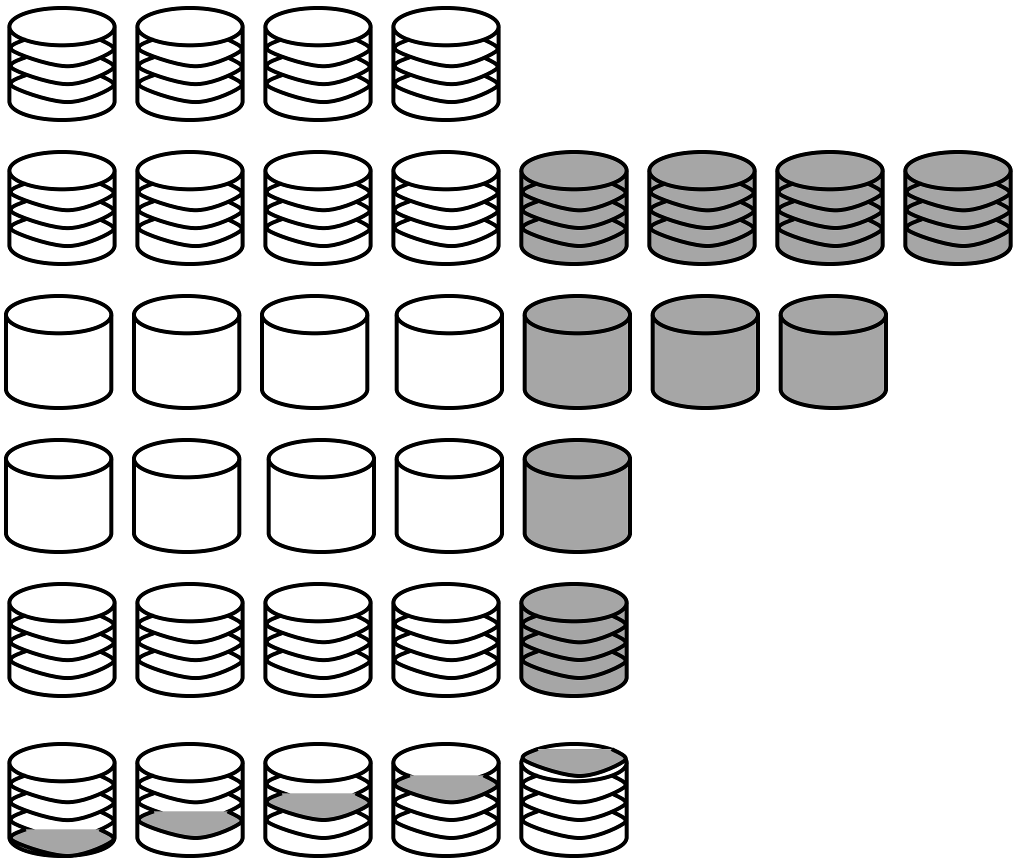
RAID 0


Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
Logical Drive
Striping
Block 1
Block 3
Block 5
Block 7
Block 2
Block 4
Block 6
Block 8
Physical Disks

RAID 1


Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
Logical Drive
Striping
Block 1
Block 3
Block 5
Block 7
Block 2
Block 4
Block 6
Block 8
Physical Disks
Mirroring
Mirroring
Block 1
Block 3
Block 5
Block 7
Block 2
Block 4
Block 6
Block 8
Physical Disks

Levels of RAID



0: Non-Redundant

1: Mirrored

2: Memory Style ECC

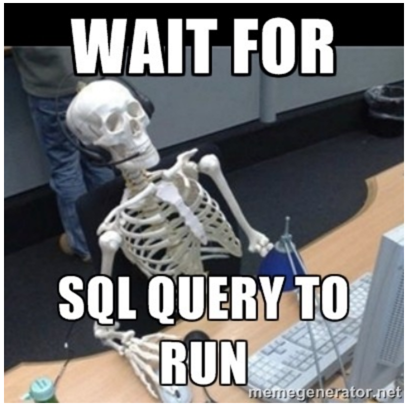
3: Bit-Interleaved Parity

4: Block-Interleaved Parity

5: Block-Interleaved
Distribution-Parity

Improving Access Time

- Data access time could be improved by:
  1. Block Transfer
  2. Cylinder-based Organization
  3. Buffering and Prefetching
  4. Multiple Disks
  5. Record Placement



Record Placement

- Records
  1. Fixed-length Records
  2. Variable-length Records

- Files
  1. File of Unordered Records (Heap)
  2. File of Ordered Records (Sorted)

Files of Records

- A **file** is a sequence of records, where each record is a collection of data values
- Records are stored on disk **blocks**
- The **blocking factor bfr** for a file is the (average) number of file records stored in a disk block
- A file can have **fixed-length** records or **variable-length** records
- A **file descriptor** (or **file header**) includes information that describes the file

File Descriptor

File Descriptor

- field names and their data types,
- the addresses of the file blocks on disk

  <block$_i$ (in file) @ address (on disk)>
- Example:
  - <block 1 @ track 2, block 4>

    (first block in file is @ the

    4th block of track 2 on disk)
  - <block 2 @ track 5, block 0>
  - … Cylinder-based organization?

		SID	Name	Age
Block 1	Record 1	546007	Peter	18
	Record 2	546100	Bob	19
Block 2	Record 3	546107	Ann	21
	Record 4	546207	Jane	20
Block 3	Record 5	546240	John	24
	Record 6	546350	Ben	18
Block 4	Record 7	546420	Suzy	27
	Record 8	546500	Peter	20

Record Placement

- Records
  1. Fixed-length Records
  2. Variable-length Records

- Files
  1. File of Unordered Records (Heap)
  2. File of Ordered Records (Sorted)

Fixed-Length Records

```
CREATE TABLE Student (
        Sid INTEGER,
        Name CHAR(24),
        Age INTEGER);
```

0 Sid 3	4 Name 27	28 Age 31

		SID	Name	Age
Block 1	Record 1	546007	Peter	18
	Record 2	546100	Bob	19
Block 2	Record 3	546107	Ann	21
	Record 4	546207	Jane	20
Block 3	Record 5	546240	John	24
	Record 6	546350	Ben	18
Block 4	Record 7	546420	Suzy	27
	Record 8	546500	Peter	20

Fixed-Length Records

"*Age*" of Record 1 at:
offset (byte): 28

"*Age*" of Record 2 at:
offset (byte): 1*Record Length + 28

Can easily identify the first-byte position of each field (**relative to** the beginning of the record or block)

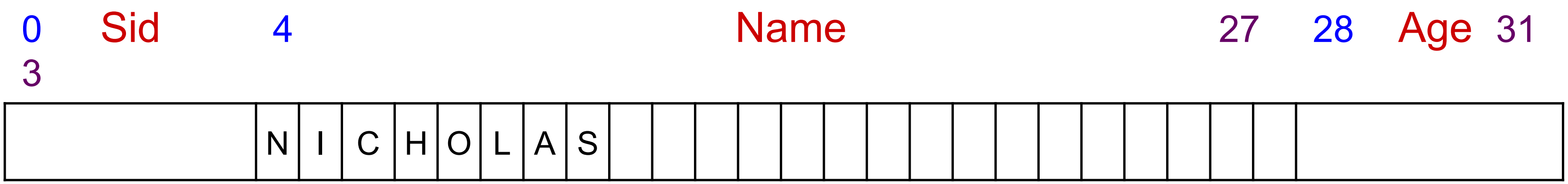
Fixed-length records cannot span separate blocks

		SID	Name	Age
Block 1	Record 1	546007	Peter	18
	Record 2	546100	Bob	19
Block 2	Record 3	546107	Ann	21
	Record 4	546207	Jane	20
Block 3	Record 5	546240	John	24
	Record 6	546350	Ben	18
Block 4	Record 7	546420	Suzy	27
	Record 8	546500	Peter	20

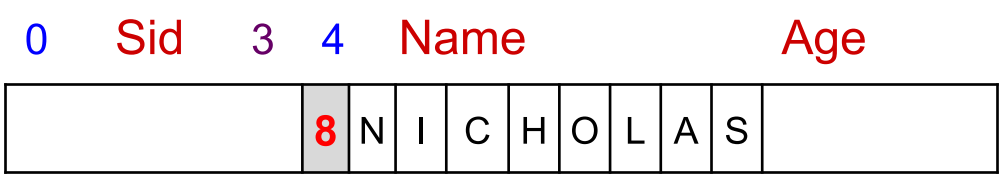
0 Sid 3 4 Name 27 28 Age 31

Variable-Length Records

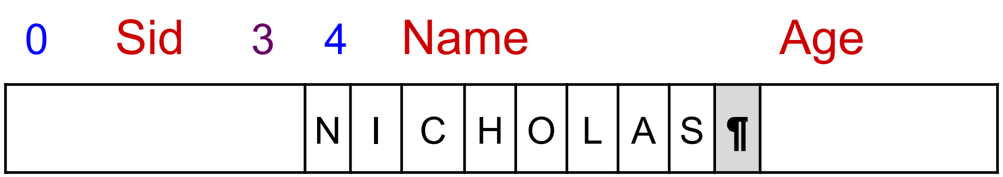
Maximum Length:



Length + Data:



Data + Separator:



Variable-Length Records

1. Store **field length** before the field value
   - E.g., "8" is the length of "Nicholas"
2. Attach a **special end-of-field** symbol to terminate each field
   - Any special character that does not appear in any field value
   - E.g., ¶, § , ■

- **Pros:**
  - No record internal fragmentation (saves space)

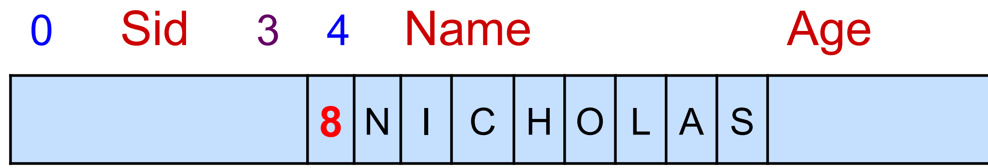
Comparisons 1

- **Pros** of fixed-length records:
  - No space is needed for storing **extra** info for the fields within record
    - No length or separator (vs. variable-length)
  - **Equally** fast access to all fields
    - *Name* is always at position 4,

      Age is always at position 28, …
  - Fixed field length **simplifies** insert, delete, update

Comparisons 2

- **Cons** of fixed-length records:
  - **Block** internal fragmentation
    - Due to unspanned organization
  - **Record** internal fragmentation
    - Due to using maximum length for every field
  - More **disk accesses** for reading a given number of records
    - Due to taking more disk space



Comparisons 3

- **Cons** of variable-length records:
  - Access time for a field is **proportional** to the distance from the beginning of record
    - To access "*Age*", "Name" must be accessed first
      - To know where Name ends and *Age* starts
    - More costly if there are more fields before *age*
  - Not easy to **reuse** space which was occupied by a deleted record
  - No space for record to **grow** longer (must move record that needs to grow)
    - If "Nicholas" is updated to "Nicholas Smith"!

Record Placement

- Records
  1. Fixed-length Records
  2. Variable-length Records

- Files
  1. File of Unordered Records (Heap)
  2. File of Ordered Records (Sorted)

  - Method of arranging a file of records on external storage -- many alternatives exist!

Unordered Files

- The simplest file structure: records are stored in no particular order
- Also called: **Heap**, Pile, or Random File
- New records are inserted at the **end** of file
  1. The last disk block is copied into buffer (i.e., memory)
  2. New record is added
  3. Block is rewritten back to disk

- Record insertion is quite efficient

Unordered Files – Delete

- To delete a record:
  1. Find its block
  2. Copy the block into a buffer
  3. Delete the record from the buffer
     - An extra bit is stored for each record (**deletion marker**)
     - Set the deletion marker to 1 (**invalid record**)
  4. Rewrite the block back to disk

- What to do with the unused space?
  1. **Periodic reorganization:** records are packed by removing deleted records, or
  2. Insert new records in the empty space

Unordered Files – Insert & Delete

Block	Record	SID	Name	Age
Block 1	Record 1	746007	Peter	18
	Record 2	546100	Bob	19
Block 2	Record 3	546107	Ann	21
	Record 4	646207	Jane	20
Block 3	Invalid			
	Record 6	246350	Ben	18
Block 4	Record 7	946420	Suzy	27
	Record 8	546500	Peter	20

Delete Record 5

Block	Record	SID	Name	Age
Block 1	Record 1	746007	Peter	18
	Record 2	546100	Bob	19
Block 2	Record 3	546107	Ann	21
	Record 4	646207	Jane	20
Block 3	Record 9	999111	Sam	24
	Record 6	246350	Ben	18
Block 4	Record 7	946420	Suzy	27
	Record 8	546500	Peter	20

Insert Record 9

Unordered Files – Insert & Delete

		SID	Name	Age
Block 1	Record 1	746007	Peter	18
	Record 2	546100	Bob	19
Block 2	Rec			
	Rec			
Block 3	Inva			
	Record 6	246350	Ben	18
Block 4	Record 7	946420	Suzy	27
	Record 8	546500	Peter	20

Delete Record 5

		SID	Name	Age
Block 1	Record 1	746007	Peter	18
	Record 2	546100	Bob	19
Block 2			n	21
			ne	20
Block 3			m	24
	Record 6	246350	Ben	18
Block 4	Record 7	946420	Suzy	27
	Record 8	546500	Peter	20

Insert Record 9

File records are inserted at the end of the file **or in any file block with free space**

Unordered Files – Search

- Examples of Search:
  - `WHERE` age = 20,
  - `WHERE` sid=999111, …

- To search in a heap file:
  - A **linear search** through the file records is necessary
    - Search records one by one!

- Linear search is **expensive**… but how expensive?

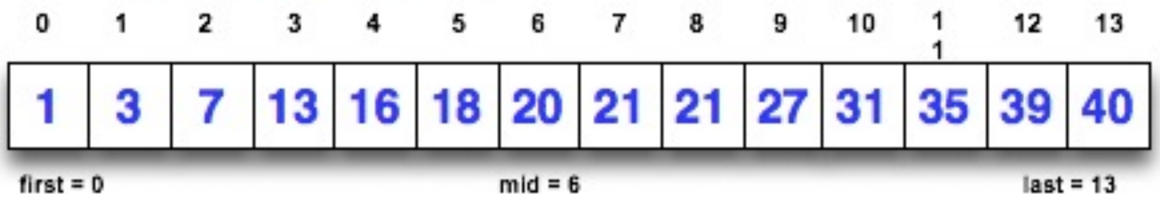
Unordered Files – Search

- Search on a **unique** field (e.g., sid=999111)
  - Read from beginning of the file and stop when record with sid=999111 is found
  - **Half** the file records are read from disk into memory (on average)
- Search on a **non-unique** field (e.g., age = 20)
  - **All** the file records are read from disk into memory
- **Range** Search on **any** field (e.g., ann < name < peter)
  - **All** the file records are read from disk into memory

Ordered Files

- File records are kept sorted by the values of an **ordering** field
- Also called: **Sorted** or Sequential File
- If the ordering field is a <u>key field</u> of the file/table, then:
  - The ordering field has unique values
  - The ordering field is called **ordering key**
- <u>Search</u> for a record on its ordering field value is quite efficient (**binary search algorithm**)

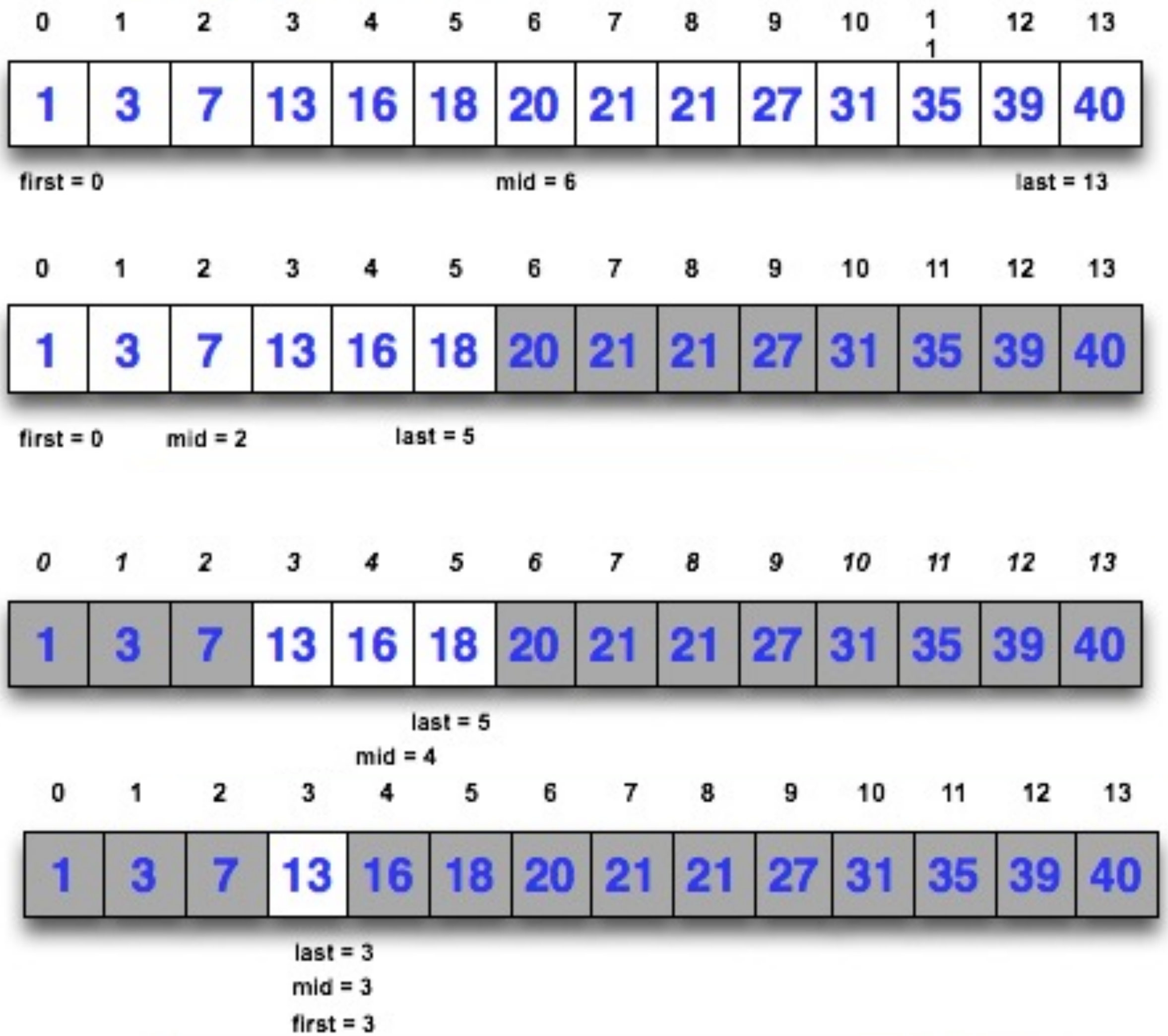
Binary Search



- Assume a file of 14 records and bfr = 1
- At each step, binary search
  - Reads the middle record (mid = ⌊(first+last)/2⌋)
  - Compares the input value V_{search} with the value in the middle record of the file V_{mid}
    - If $\mathbf{V_{search} = V_{mid}}$ then search finishes
    - if $\mathbf{V_{search} < V_{mid}}$
      then the algorithm is repeated on the left (i.e., top) sub-file
    - if $\mathbf{V_{search} > V_{mid}}$
      then the algorithm is repeated on the right (i.e., bottom) sub-file
- If length of sub-file is zero, then V_{search} is not found

Binary Search – Example

$V_{search} = 13$



Binary Search – How Fast?

- In each iteration, Binary search **halves** the number of records to check

- Example: assume a file of 256 records and bfr = 1
  - After 1^{st} iteration, 128 records remain to search
  - After 2^{nd} iteration, 64 records remain to search
  - …
  - Until 1 record remains
  - $256 = 2^8$, $128 = 2^7$, …, until $1=2^0$
  - Number of iterations = 8 = $\mathbf{\log_2}$ 256

- Number of I/O reads = **$\log_2$ (number of blocks in file)**

Ordered Files – Search

- Search on **ordering key** (e.g., sid=999111)
  - Binary Search
- Search on **non-ordering key** (e.g., email = s1@uq)
  - Linear Search
- Range search on **ordering key** (e.g., 10 < sid < 20)
  - Binary search to find the first record
  - Followed by sequential access until the end of the range

Ordered Files – Insert

- Insertion is an **expensive** operation: records must remain in the correct order
- To insert a record:
  - Find its correct position in file (based on its ordering field)
  - Make space in the file to insert the record in that position
  - On average, half the blocks of the file must be moved
    1. Read those blocks
    2. Move records among them
    3. Rewrite them back to disk

Ordered Files – Insert

		SID	Name	Age
Block 1	Record 1	10	Peter	18
	Record 2	20	Bob	19
Block 2	Record 3	30	Ann	21
	Record 4	50	Jane	20
Block 3	Record 5	60	Sam	24
	Record 6	70	Ben	18
Block 4	Record 7	80	Suzy	27

Insert Record (40, Kevin, 22)

		SID	Name	Age
Block 1	Record 1	10	Peter	18
	Record 2	20	Bob	19
Block 2	Record 3	30	Ann	21
Block 3	Record 4	50	Jane	20
	Record 5	60	Sam	24
Block 4	Record 6	70	Ben	18
	Record 7	80	Suzy	27

Make "space"!

Ordered Files – Insert

		SID	Name	Age
Block 1	Record 1	10	Peter	18
	Record 2	20	Bob	19
Block 2	Record 3	30	Ann	21
	Record 4	50	Jane	20
Block 3	Record 5	60	Sam	24
	Record 6	70	Ben	18
Block 4	Record 7	80	Suzy	27

		SID	Name	Age
Block 1	Record 1	10	Peter	18
	Record 2	20	Bob	19
Block 2	Record 3	30	Ann	21
	Record 8	**40**	**Kevin**	**22**
Block 3	Record 4	50	Jane	20
	Record 5	60	Sam	24
Block 4	Record 6	70	Ben	18
	Record 7	80	Suzy	27

Insert Record (40, Kevin, 22)

Comparison of I/O Costs

File Type	Scan File	Equality Search	Range Search	Insert
Heap	?	?	?	?
Sorted	?	?	?	?

- Assume:
  - B: The number of blocks in file
  - In heap file, search is on unique attribute
  - In sorted file, search is on ordering key