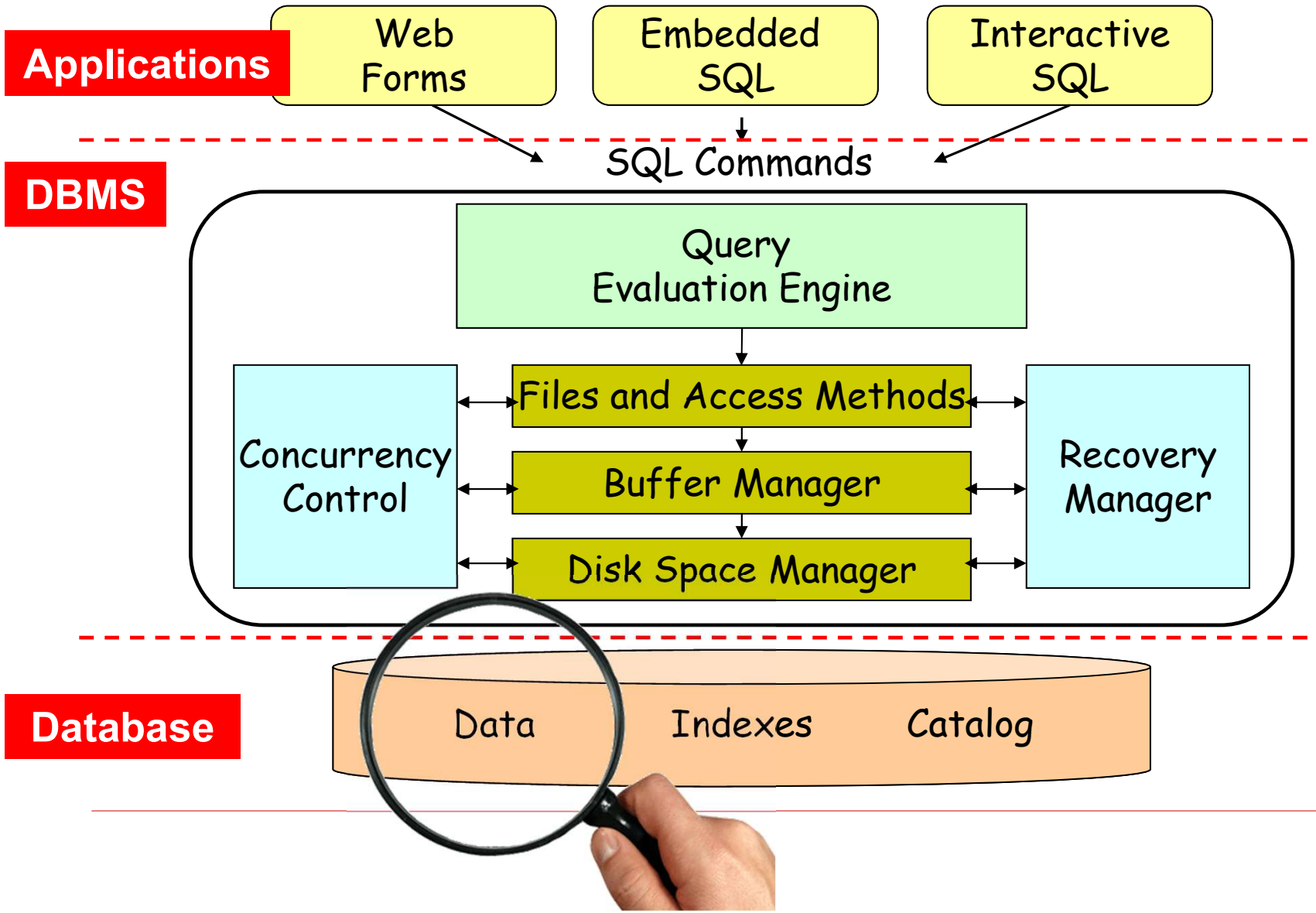
Comp2411 Database Systems

File Organization and Indexing

What is a “Good” Database System?

- **User-friendly**
  - Data & Execution Abstraction
- **Correct & Reliable**
- **Efficient & High-Performance**

Database Management System (DBMS)



Storage Hierarchy

1. **Primary**
   - Random Access Memory (RAM) - fast
   - *For currently used data*
2. **Secondary**
   - Disks - slow
   - *For the main database*
3. **Tertiary**
   - Tapes – very slow
   - *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
    - Disk is slow (order of millisecond)
    - RAM is fast (order of nanosecond)

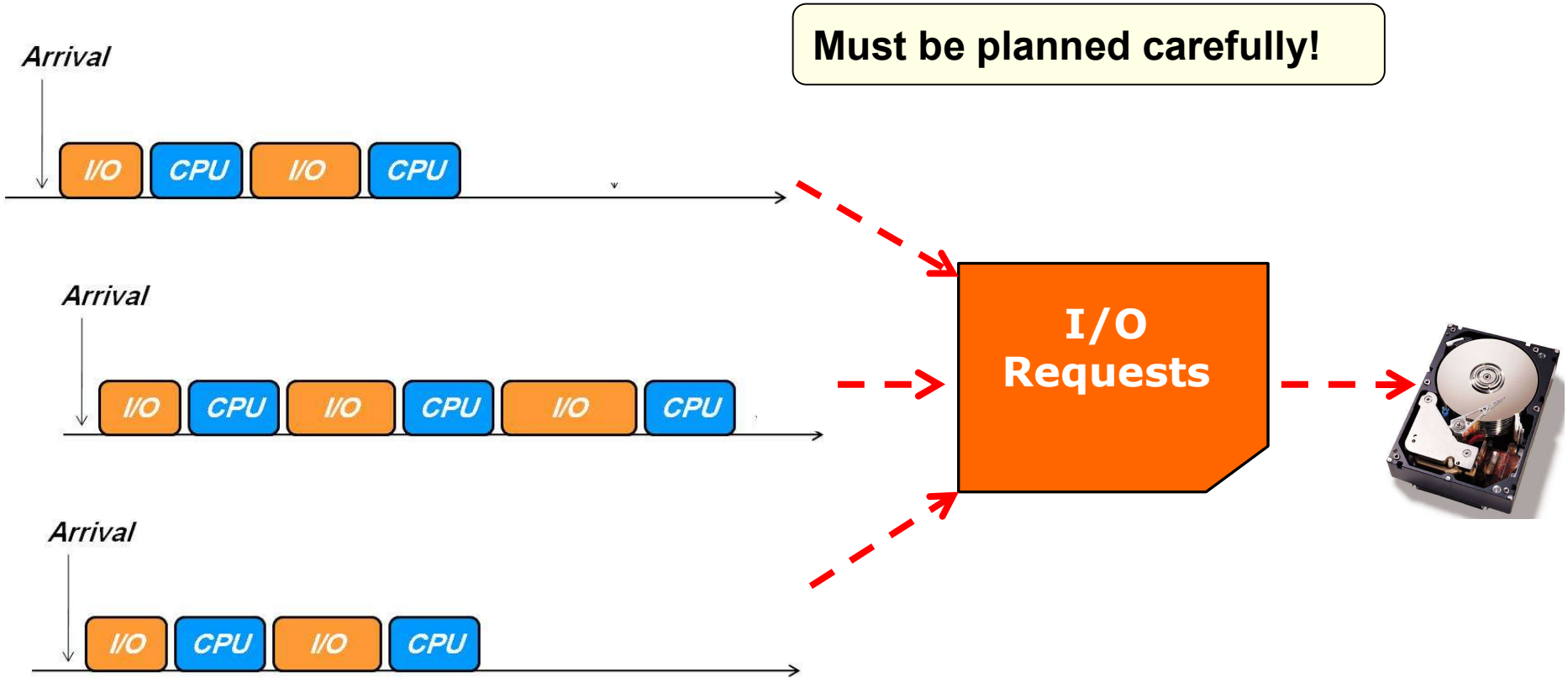
Queries and Transactions

- 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:

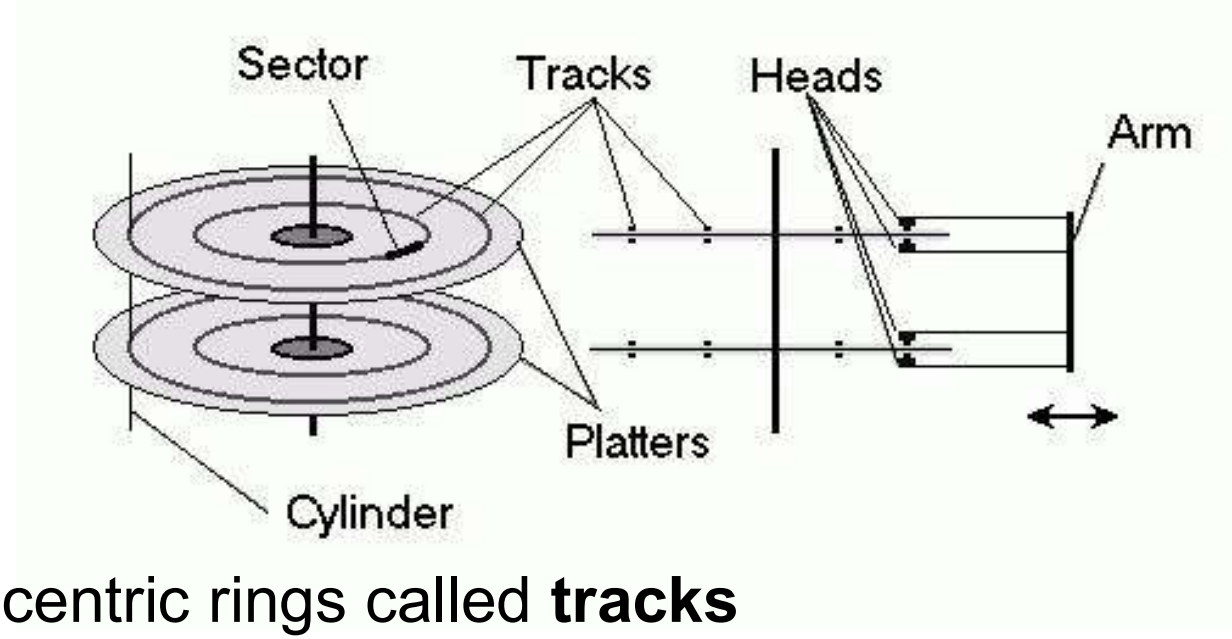
    x=read(balance), x=x-$200, write(balance, x)

Transaction I/Os

- I/O requests: read or write

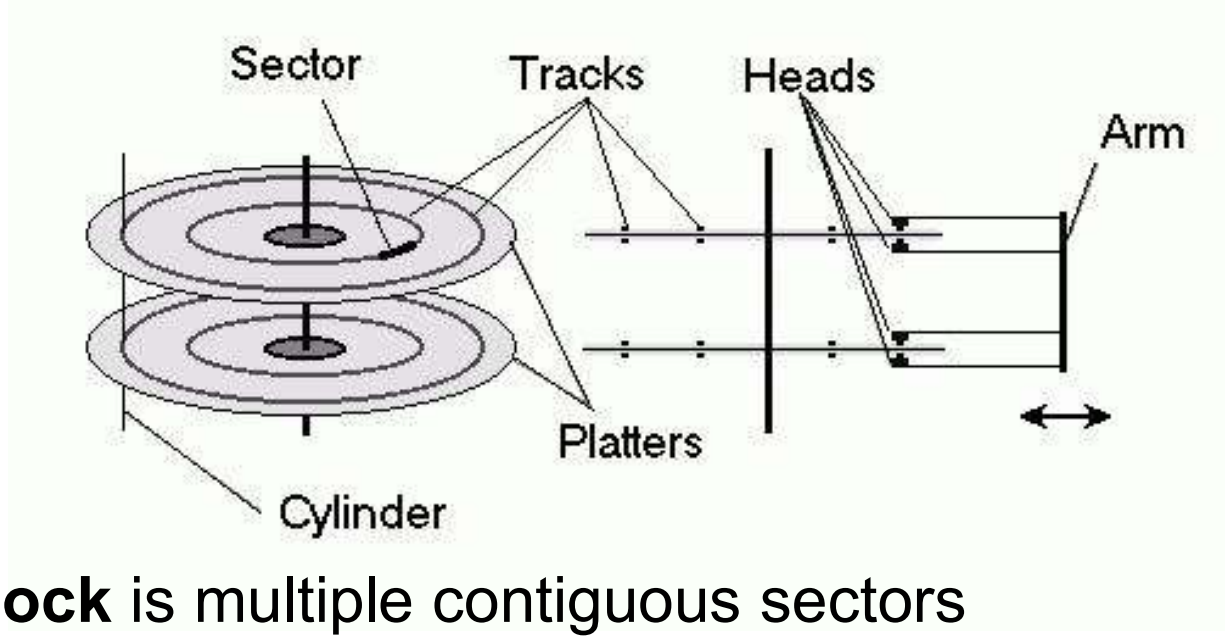


Disk Components



- 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

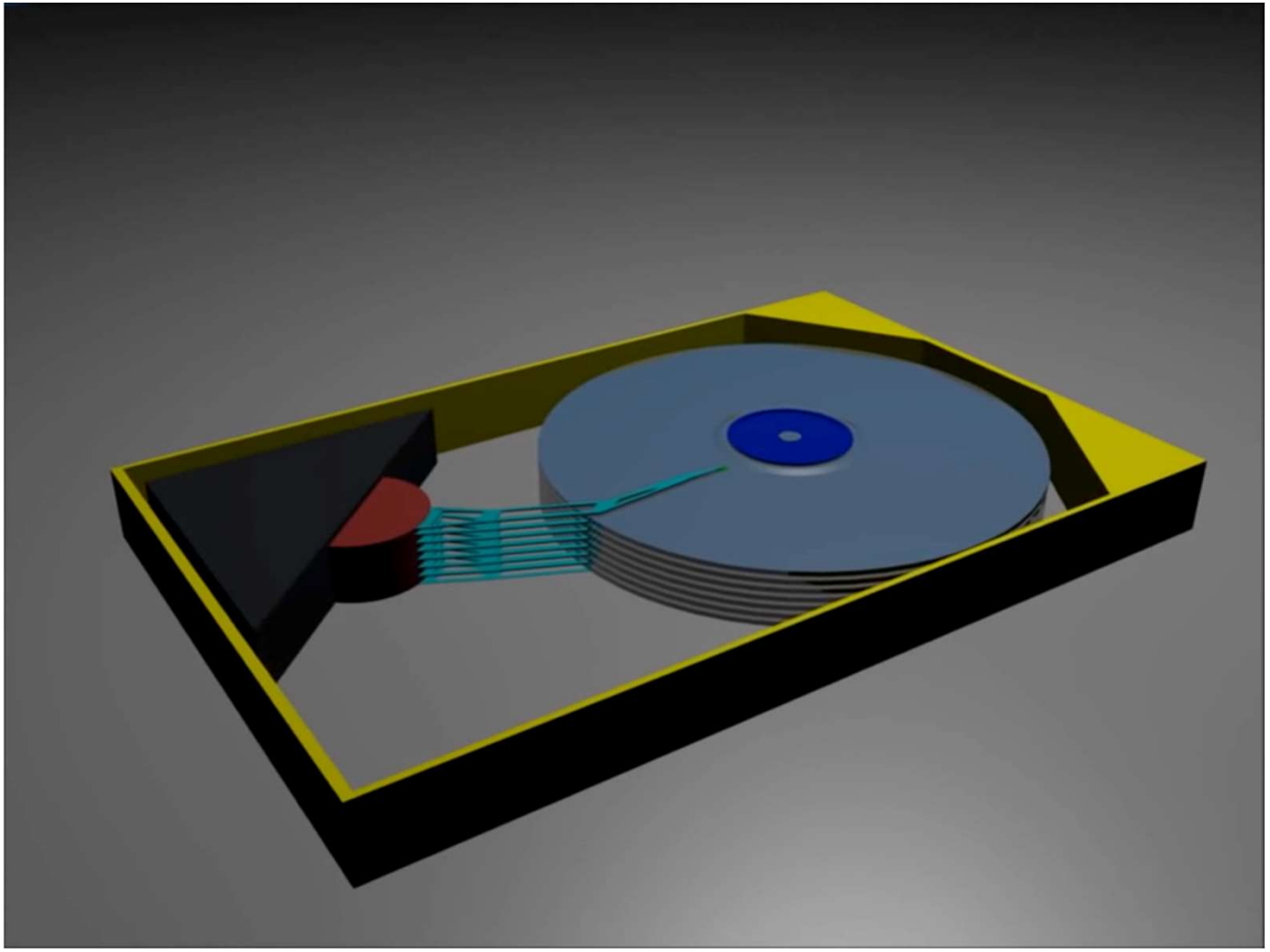


- A **Block** is multiple contiguous sectors
  - The basic unit for data access
- Only one **head** reads/writes at any one time
- The **arm** assembly is moved in or out to position the head on a desired track

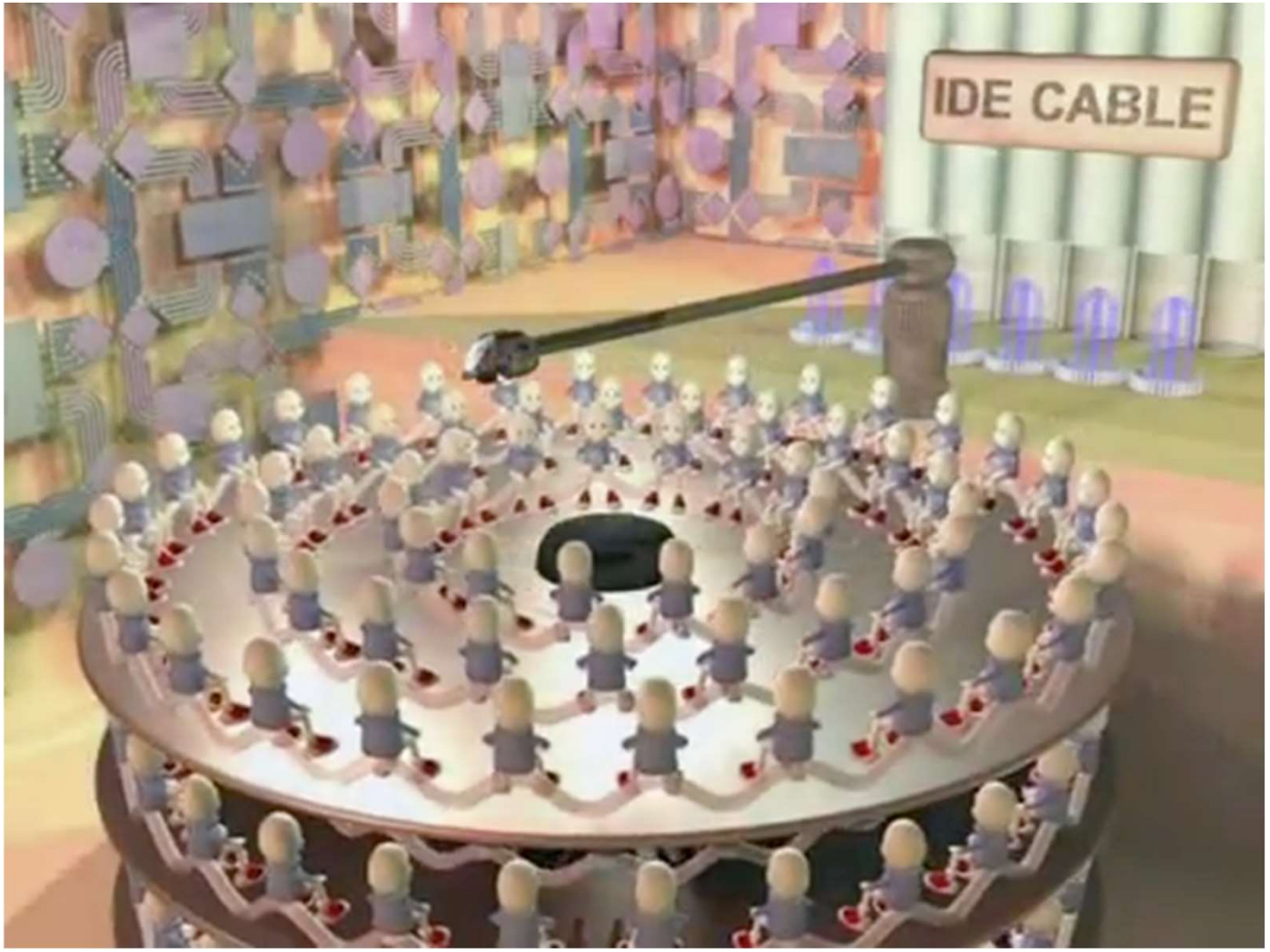
Accessing a Disk Block

- 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



Question

Parameter	Value
Sector size	512 bytes
# of sectors per track	50 sectors
# of tracks per surface	2000
# of platters	5 double-sided

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

Track capacity = # of sectors * sector size
50 * 512 = 25,600 bytes

Surface Capacity = # of tracks * track capacity
2000 * 25,600 = 51,200,000 bytes

Disk Capacity = # of surfaces * surface capacity
5 * 2 * 51, 200,000 = 512,000,000 bytes

Question

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

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

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

Maximum rotational delay is: *the time required for one complete rotation*

Maximum rotational delay = 1/5400 minutes = 60/5400 = 0.011 seconds

Average rotational delay = 60 / (5400 * 2) = 0.0055 seconds

Question

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

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

Transfer Rate = *the amount of transferred data in a given amount of time*

= track capacity / rotation time

= 25,600 / 0.011 = 2,304 Kbytes/second

Question

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

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

Average track access time

= average *seek time* + *average rotational delay* + *track transfer time*

= 0.01 + 0.0055 + 0.011 seconds

Accessing a Disk Block

For any block of data,

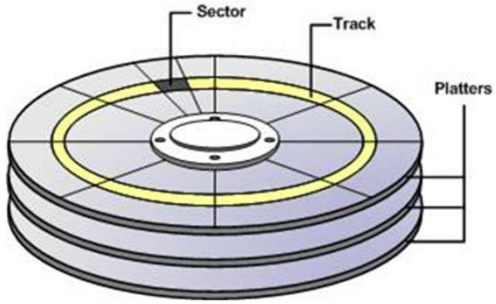
Average block access time =

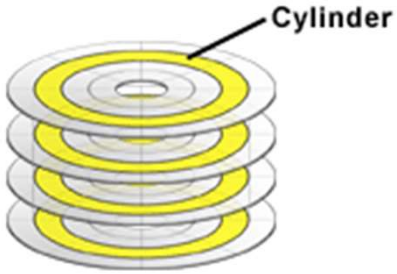
seek time + rotational delay + transfer time

- Seek time and rotational delay dominate!
- Key to lower I/O cost: reduce seek/rotation delay!

Cylinder-based Organization

- **Observation:** Data blocks in a relation are likely to be accessed together
- **Idea:** Store such blocks next 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

- 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

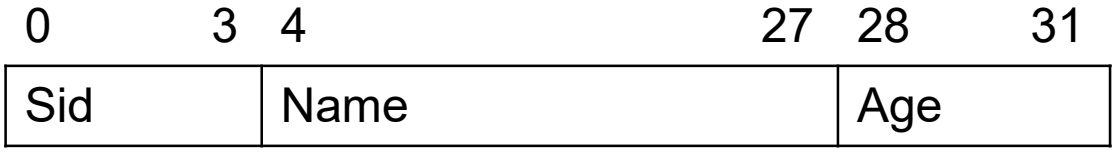
- Advantages:
  - Excellent if access pattern matches storage
    - "Next" block on disk is next block to read
  - Only one process/transaction is using the disk

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);
```



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

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

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

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
- Blocks
  1. Unspanned Blocks
  2. Spanned Blocks
- Files
  1. Unordered Files (Heap, Random)
  2. Ordered Files (Sequential)

Record Placement

- Records
  1. Fixed-length Records
  2. Variable-length Records
- Blocks
  1. Unspanned Blocks
  2. Spanned Blocks
- Files
  1. Unordered Files (Heap, Random)
  2. Ordered Files (Sequential)

Fixed-Length Records

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

		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

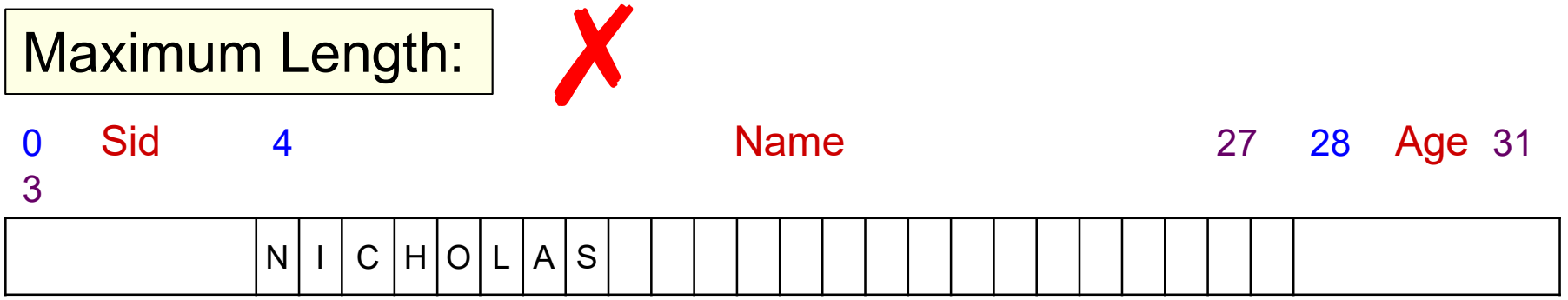
Can easily identify the first-byte position of each field (**relative to** the beginning of the record or block)

Fixed-length records cannot span separate blocks

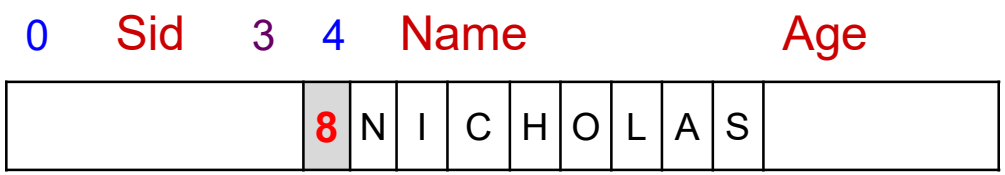
		SID	Name	Age
Block 1	Record 1	546007	Peter	18
	Record 2	546100	Bob	19
Block 2	Record 3	546107	Ann	21
	Record 4	546207	Jane	20
Block 3	Record 5	546240	John	24
	Record 6	546350	Ben	18
Block 4	Record 7	546420	Suzy	27
	Record 8	546500	Peter	20

Variable-Length Records

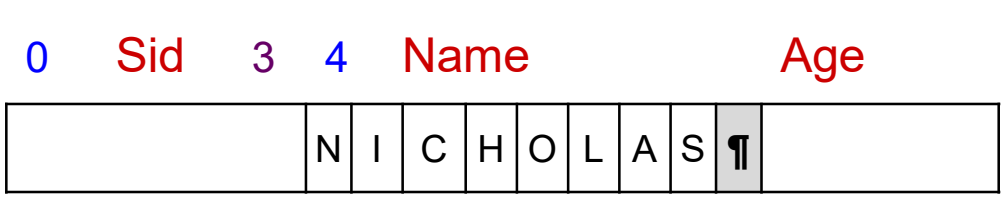
Maximum Length:



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 & Cons?**

Comparisons

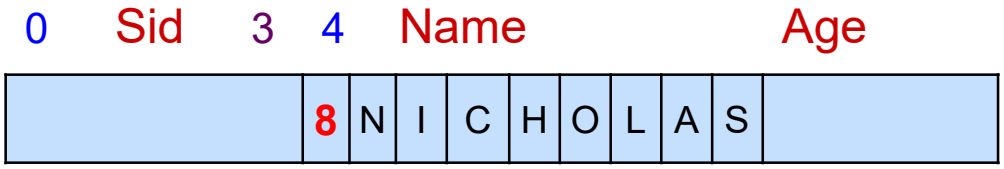
- **Pros** of fixed-length records:
  - No space is needed for storing **extra** info for the fields within record
    - No length or separator (vs. variable-length)
  - **Equally** fast access to all fields
    - *Name* is always at position 4,

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

Comparisons

- **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



- **Pros** of variable-length records:
  - No record internal fragmentation (saves space)
- **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
  - 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
- Blocks
  1. Unspanned Blocks
  2. Spanned Blocks
- Files
  1. Unordered Files (Heap, Random)
  2. Ordered Files (Sequential)

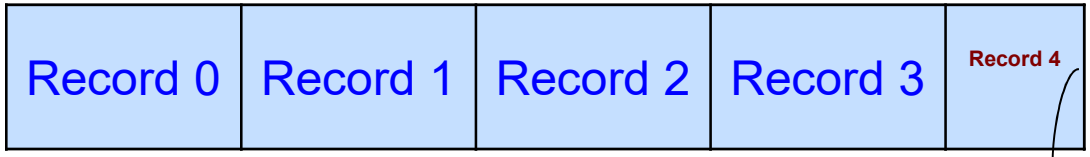
Block Organization

Unspanned

- Records are not allowed to cross block boundaries

Spanned

- A record can span more than one block



Pros & Cons?

Record Placement

- Records
  1. Fixed-length Records
  2. Variable-length Records
- Blocks
  1. Unspanned Blocks
  2. Spanned Blocks
- Files
  1. Unordered Files (Heap, Random)
  2. Ordered Files (Sequential)

Unordered Files

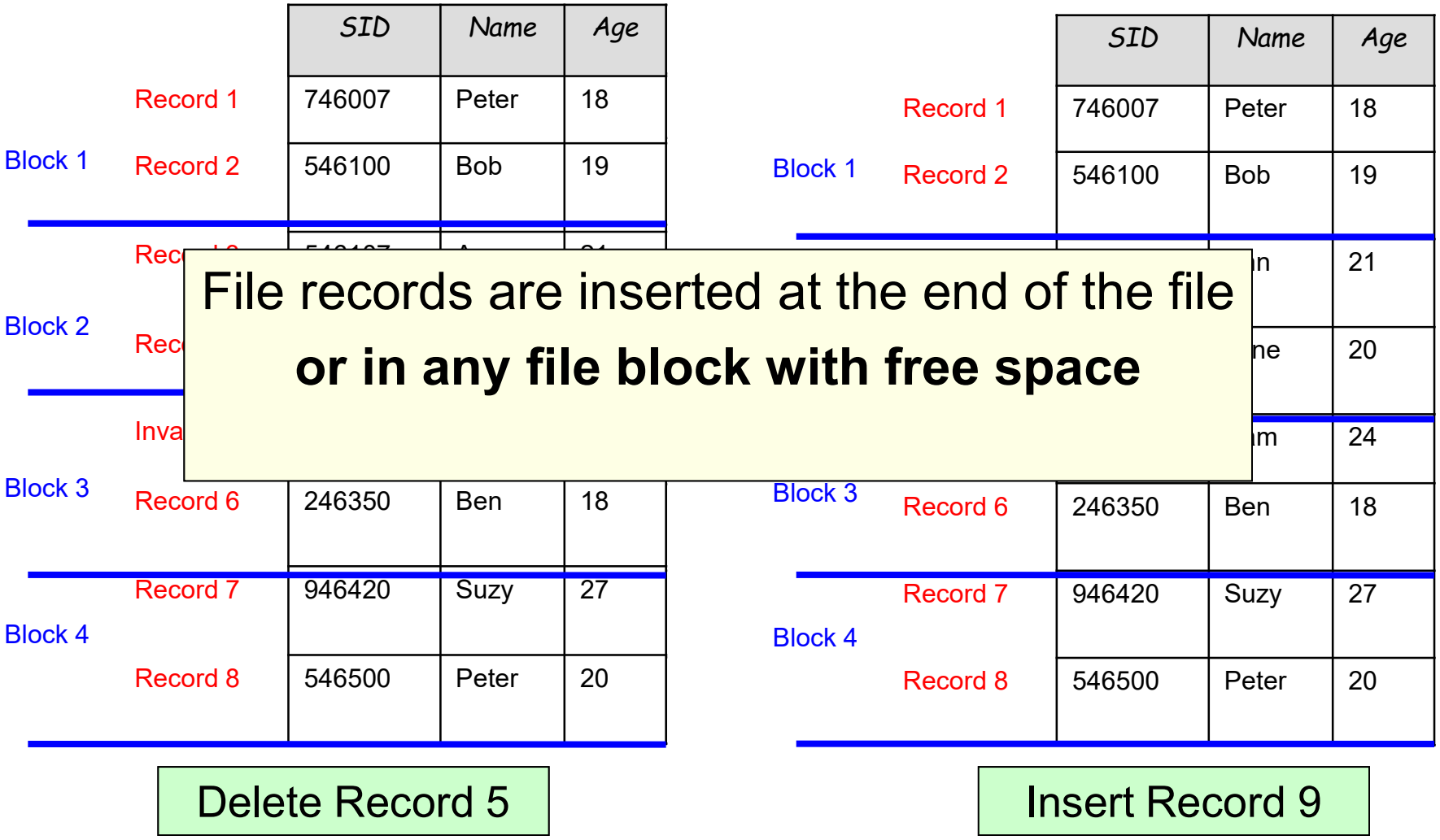
- The simplest file structure: records are stored in no particular order
- Also called: **Heap**, Pile, or Random File
- New records are inserted at the **end** of file
  1. The last disk block is copied into buffer (i.e., memory)
  2. New record is added
  3. Block is rewritten back to disk
- Record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



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?

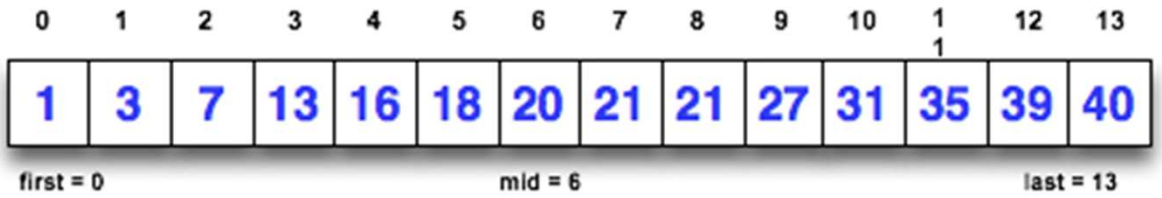
Unordered Files – Search

- 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 key field of the file/table, then:
  - The ordering field has unique values
  - The ordering field is called **ordering key**
- Search for a record on its ordering field value is quite efficient (**binary search algorithm**)

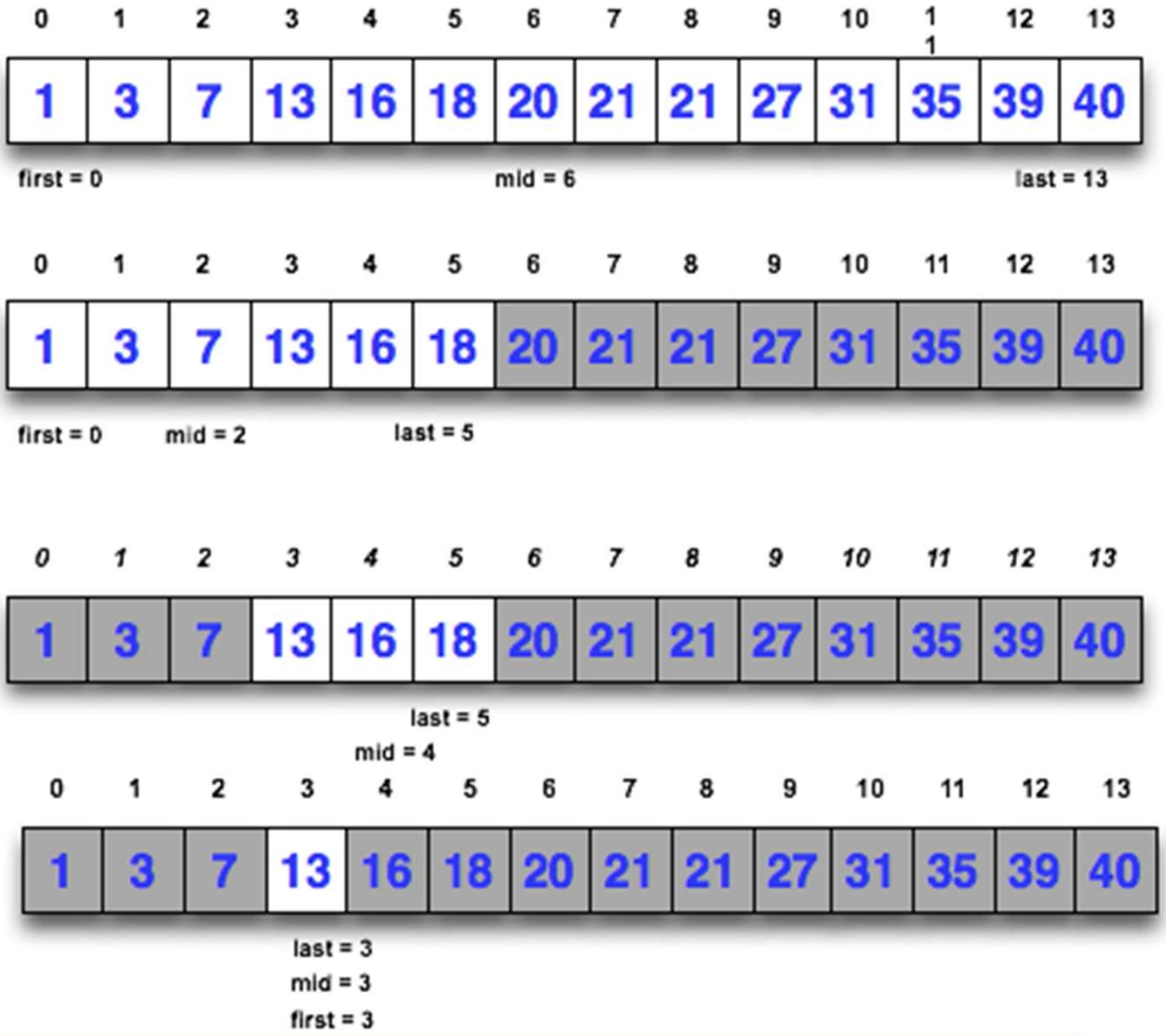
Binary Search



- 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 1st iteration, 128 records remain to search
  - After 2nd 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 = $\mathbf{\log_2}$ **(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@gmail)
  - 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

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
	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

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

Comparison of I/O Costs

File Type	Scan File	Equality Search	Range Search	Insert
Heap	B	0.5B	B	2
Sorted	B	$\log_2 B$	$\log_2 B$+ # matching blocks	$\log_2 B$+B

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