Database and Information Systems

Course Roadmap

Chapter 1	Introduction to Databases
Chapter 2	Integrity Constraints and ER Model
Chapter 3	Relational Databases and Schema Refinement
Chapter 4	Query Language
Chapter 5	Transaction and Concurrency Control
Chapter 6	Indexing

Architecture of Computer System

- Three main components of Computer System: CPU, Main memory (RAM), Secondary memory (Hard Disk)
- Example Query: Select * From student Where rollno = 50;

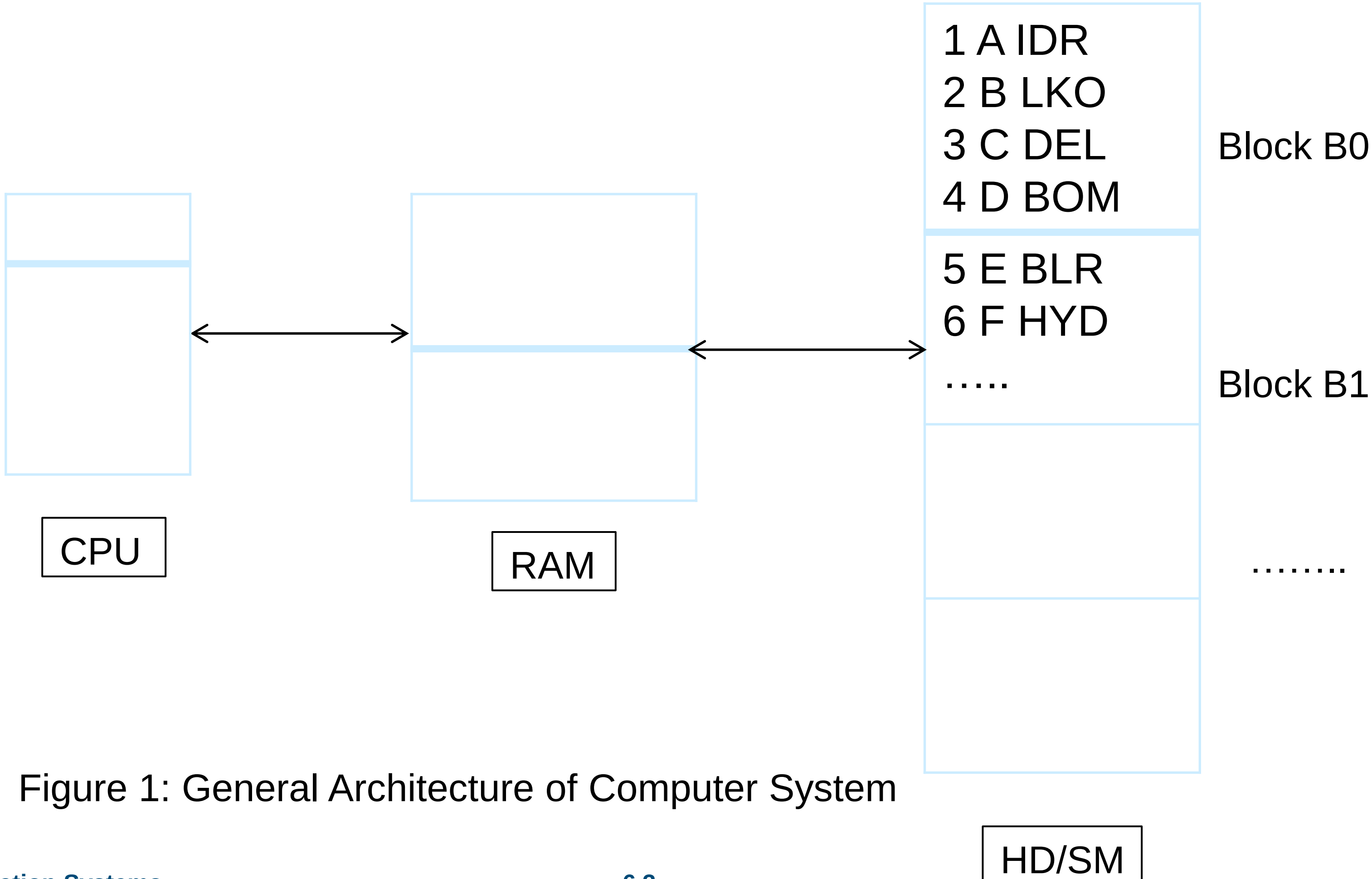


Figure 1: General Architecture of Computer System

Architecture of Computer System

- CPU is very fast compare to Secondary Memory (SM) or Hard Disk (HD)
- CPU is directly not connected with SM
- CPU is connected to SM through Primary Memory (PM) or RAM
  - RAM is a volatile memory
- Data is permanently stored in SM
- When CPU execute a query, data is brought into PM then CPU perform the operation and changes are stored back into SM
  - This is called I/O cost
- Secondary Memory (SM) is divided into same size of blocks or pages
- Primary memory (PM) is also divided into same size of blocks
- Block size in SM = Block size in PM
- Data (or records of a table) in SM can be ordered or unordered
  - A block usually can contain multiple records

Architecture of Computer System

- Example Query: Select * From student Where rollno = 50;
- I/O cost if data is unordered = O(n), where n is no of blocks
- I/O cost if data is unordered = O(log(n))

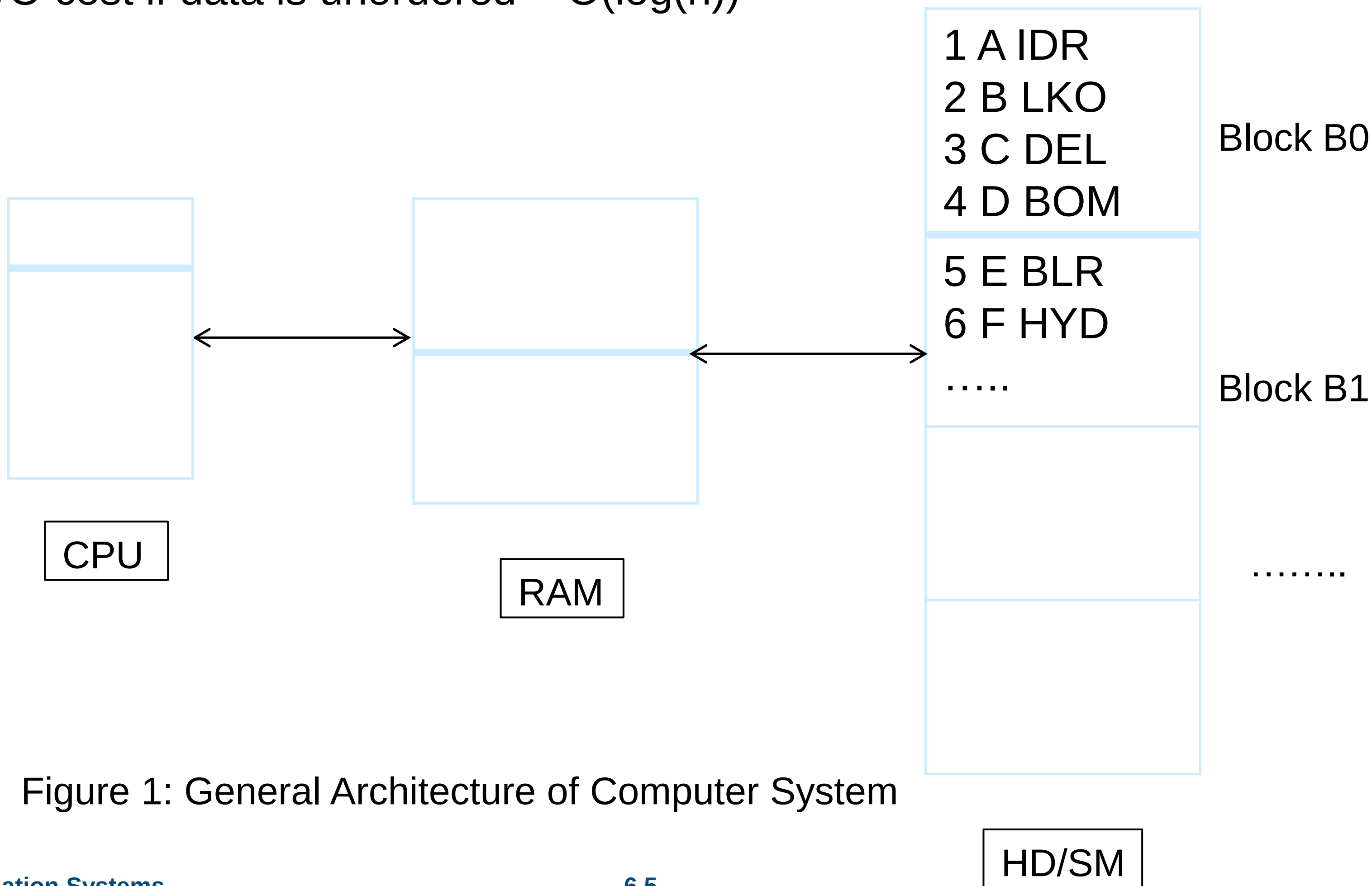


Figure 1: General Architecture of Computer System

Example on I/O Cast

- Example
  - Consider a Hard Disk (HD) with Block Size 1000 bytes, each record size is 250 bytes, total records are 10000 and the data entered in HD without any order (unordered).
    - Find average time complexity (I/O cost) to search a record from HD
  - Solution
    - No of records in each block = 1000/250= 4
    - Total no of block = 10000/4 = 2500
    - I/O cost = 1 (best case), 2500 (worst case)
    - Average cost = 2500/2 =1250
    - We can say, it has linear time complexity O(n)
    - If data is sorted, time complexity is O(log(n))

Why Indexing

- Indexing is used to reduce I/O cost
- It is used to speed up access to desired data
- Our aim in indexing is to transfer/call less no of blocks from SM to PM
- Example: Book indexing reduce search time
  - If no index table is available in the book, we may need to search all the pages for a search topic (let us assume, search topic is Indexing). The worst case time complexity can be in order of no of pages in book
  - Index table size is less and search will be fast. After looking the page number from index table for a search topic (let us assume, search topic is Indexing), we can directly go to desired page

Basic Concepts of Indexing

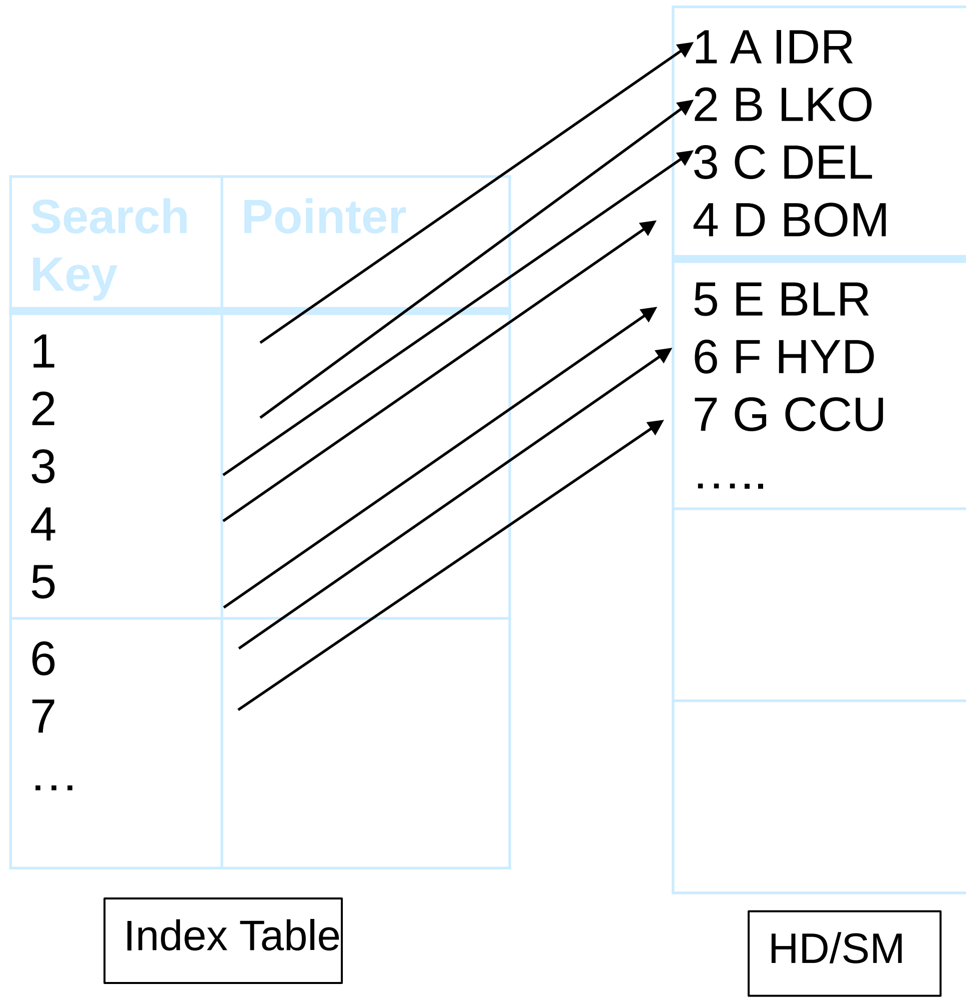
- Indexing mechanisms used to speed up access to desired data.
- **Search Key** - attribute to set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form

search-key	pointer

- Index files are typically much smaller than the original file
- **Index table block size same as block size of SM or PM**
- Index Table is also stored in SM permanently and once execution starts, the index table is first brought back to RAM and searched for desired key
- **Key in Index Table is always sorted and unique**

Basic Concepts of Indexing

- Search key is an attribute that is present in the table
- Pointer refers to the actual location of record in the SM

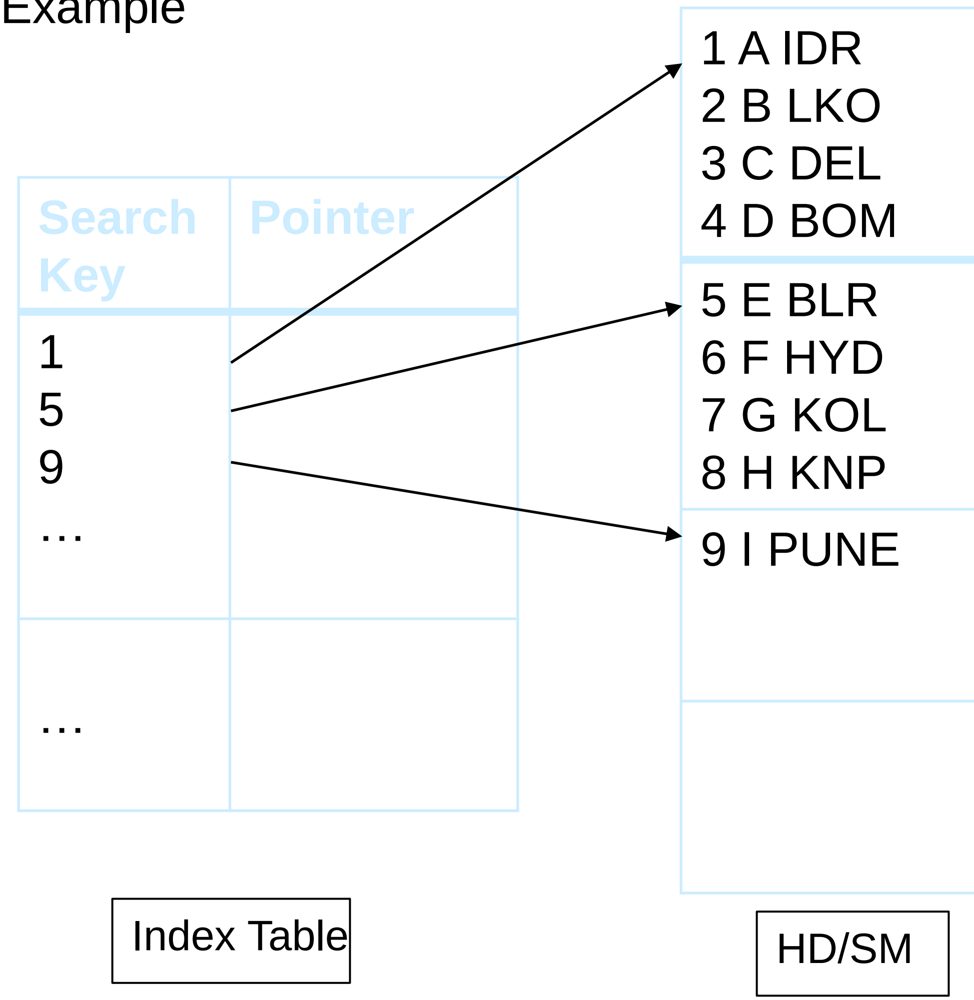


Types of Indexing

- Types of Indexing: Dense and Sparse
- **Dense index** — Index record appears for every search-key value in the file
  - Example: Figure that is shown in the last slide (slide 6.9) is an example of Dense Index
- **Sparse Index**: contains index records for only some search-key
  - Applicable when records are sequentially **ordered on search-key**
  - To locate a record with search-key value *K* we:
    - Find index record with largest search-key value < *K*
    - Search file sequentially starting at the record to which the index record points

Sparse Index

- Sparse Index Example



Basic Concepts of Indexing

- Questions
  - Why index table size is less than size of table in hard disk?
  - Why index table block can hold more number of records compare to block of table in hard disk?

I/O Cost with Indexing

- Example
  - Consider a Hard Disk (HD) with Block Size 1000 bytes, each record size is 250 bytes, total records are 10000 and the data entered in HD without any order (unordered).
    - Find average time complexity (I/O cost) to search a record from Index Table where index table size is 20 Byte (10B for key + 10B for pointer)
  - Solution
    - Index table block size = block size in HD = 1000B
    - No of entries in Index table = 1000B/20B = 50
    - If Sparse indexing, total no of entries in Index table = no of blocks in HD = 2500
    - No of blocks in Index table = 2500/50 = 50
    - Index is sorted so search time in index = log(50)
    - Total search time = index table search time + 1 = log(50)+1 = 7
    - If Dense Indexing, search time = log (10000/50) + 1 = 9

I/O Cost with and without Indexing

	Without Indexing	With Indexing
Ordered Data	12	7
Unordered Data	2500	9

Table: Comparison of worst case complexity with and without indexing for the example mentioned in above slides

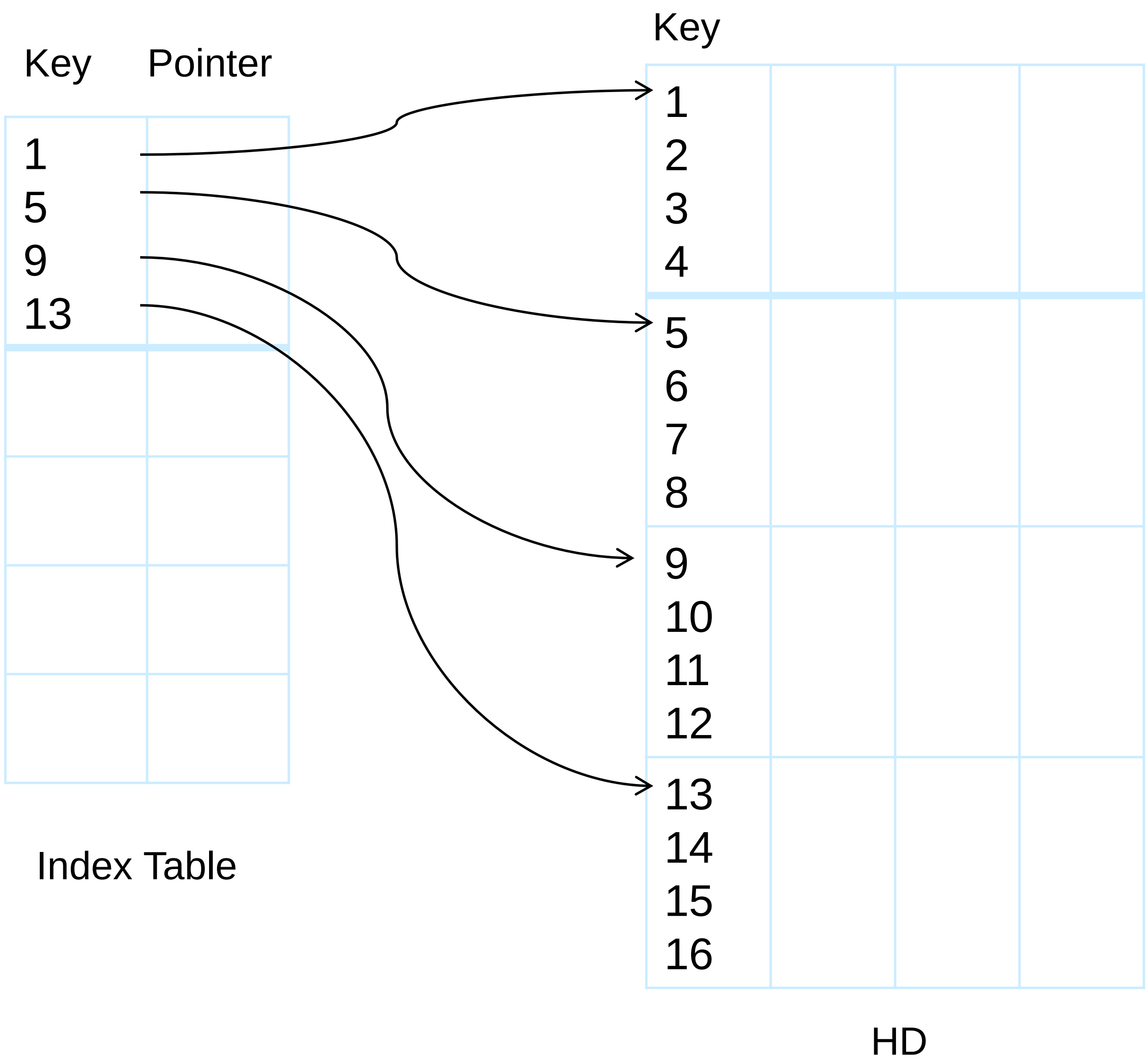
Types of Indexes

- Types of Indexes
  - Primary Index
  - Clustered Index
  - Secondary Index

	Key	**Non Key**
Ordered File	Primary Index	Clustered Index
Unordered File	Secondary Index	Secondary Index

  - Ordered File indicates data is in sorted order
  - Key indicates search key is a primary key

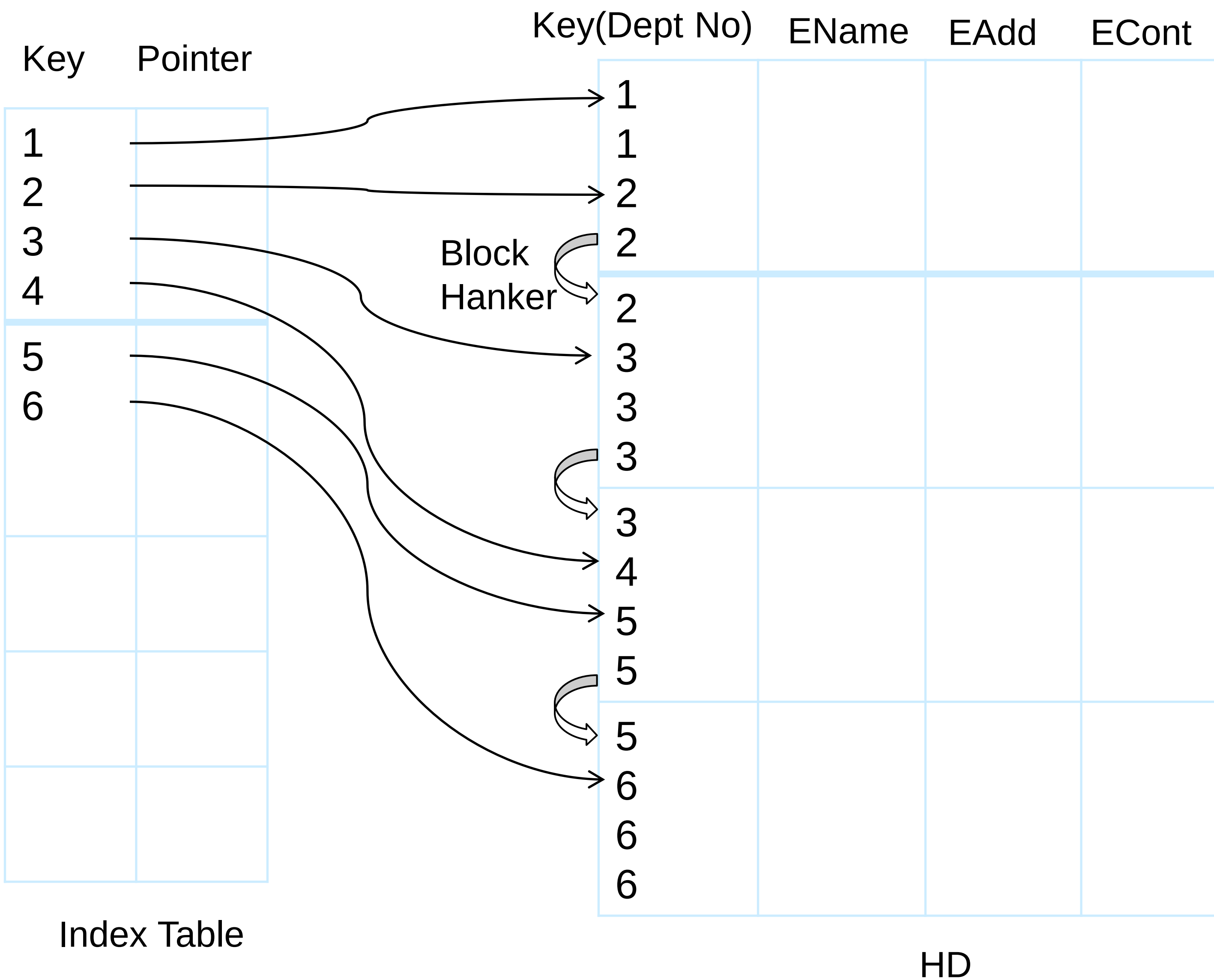
Primary Index



Primary Index

- When we can apply Primary Index
  - Data is **sorted**
  - Search key is **unique or primary key**
- Primary Index follows **sparse indexing**
- Number of entries in Index Table = Number of blocks in HD
- To locate a record with search-key value *K* we:
  - Find index record with largest search-key value < *K*
  - Search file sequentially starting at the record to which the index record points
- Search Time in Primary Index = log(N) + 1; where N is the number of blocks in index table
- Atmost one primary index for database table

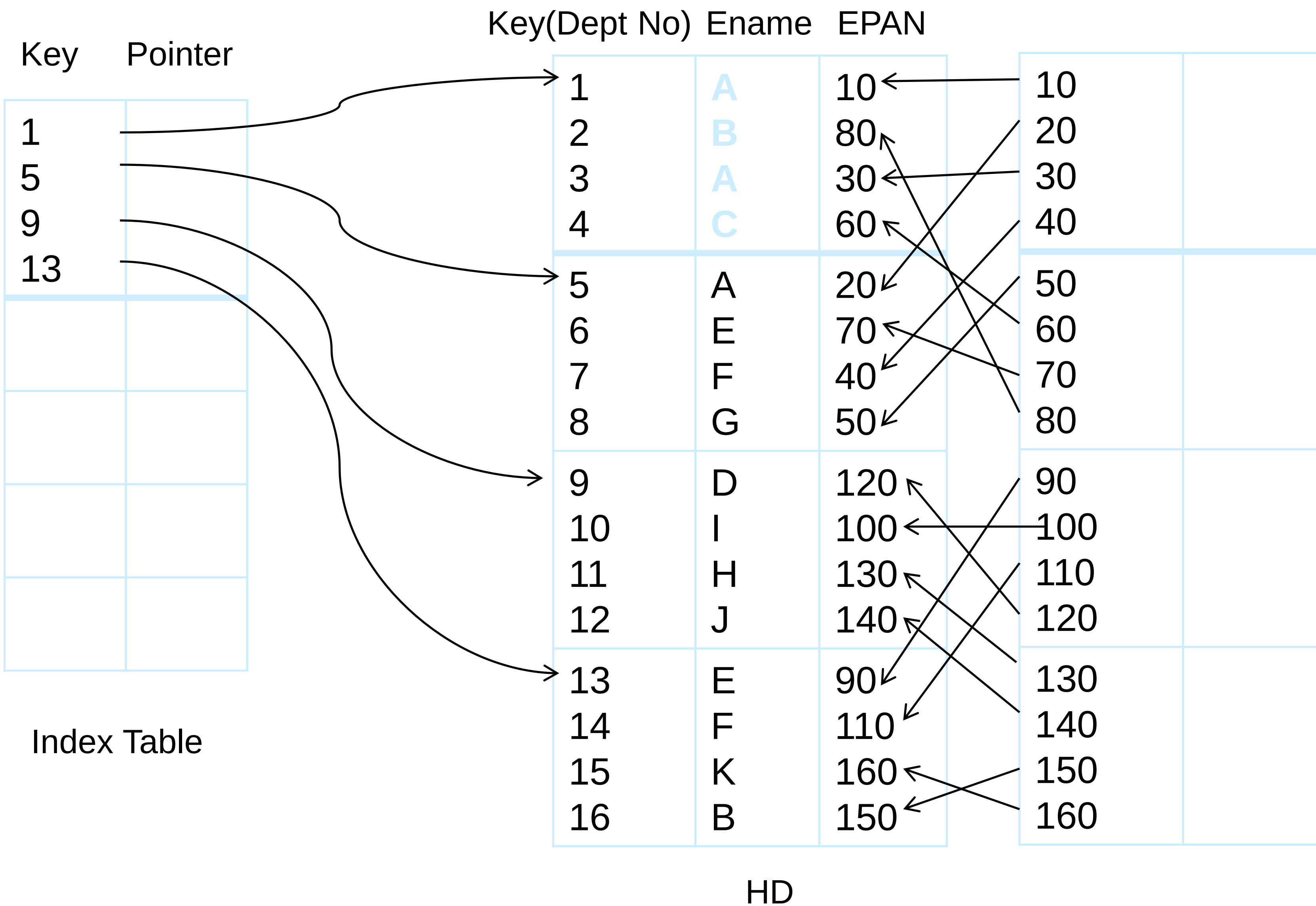
Clustered Index



Clustered Index

- Clustered index is used when there is a **ordered** file and search key is a **non key**
- Clustered Index follows **sparse indexing**
- Best case search time = log(N) + 1; where N is the number of blocks in index table
- Worst case search time = log(N) + 1 + *1* (*this can be more than 1, worst case no of blocks in HD*)
- Atmost one clustering index for database table

Secondary Index



Secondary Index

- Secondary indexed is used when data is **unordered**
- No of records in Index Table = No of records in HD
- Secondary indices have to be **dense**
- Search complexity = log(N) + 1, where N is number of blocks in Index table (where secondary search based on KEY)
- If secondary search is based on non key then we need to maintain intermediate layer that is block of record pointers
  - Search complexity = log(N) + 1 + 1

Dynamic Multilevel Index

- When Index size increases, we create Index table for original Index table
  - There can be index table for index table until last table is a single block
  - It creates problem while inserting or deleting a record in a table
    - Tree Data Structure is used to avoid the problem

Key	Ename

HD

Key	Ptr

Index Table

Key	Ptr

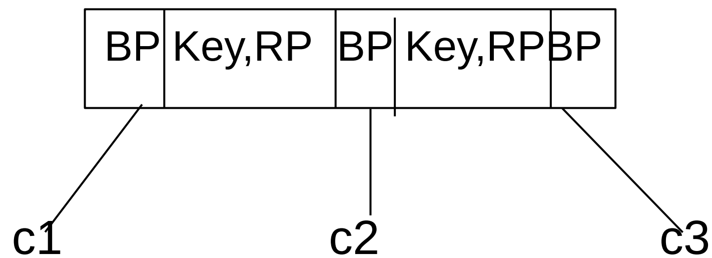
Index Table

Key	Ptr

Index Table

B-Tree

- Tree?. B-Tree is a balanced tree (leaf at same level). B-Tree properties:
  - Block pointer or tree pointer
  - Keys
  - Data pointer or record pointer



  - Order of a B-Tree (p): Max number of children a node can have

Children	Root	Intermediate or Leaf Node
Max	p	p
Min	2	$\lceil p/2 \rceil$

  - Keys and Record pointer always have a 1 less than p (or order no of children in a node)
  - Data is inserted in sorted order, like binary search tree

Order of B-Tree

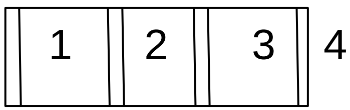
- Example
  - B-Tree with key size = 10 bytes, block size =512 bytes, data pointer size = 8 byte, block pointer size = 5 byte. Find order of B-Tree

BP	Key,RP	BP	Key,RP	BP

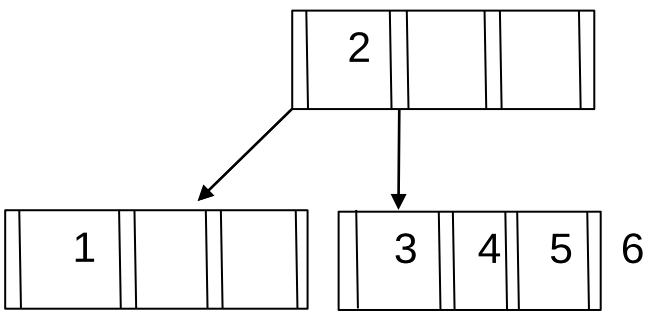
  - Let's assume we have n BP
    - n*BP + (n-1)*key + (n-1)*RP ≤ Block Size
    - n*5 + (n-1)*10 + (n-1)*8 ≤ 512
    - n ≤ 23.04
- In the above example, how many max and min keys possible in root node and non-root node(intermediate and leaf node)?
- We usually keep more than one keys in a block (node) of B-Tree
  - Note: In normal tree, we usually keep one key in a node

Insertion in B-Tree

- Order of a B-Tree is 4, insert the following keys in B-Tree: 1,2,3,4,5,6
  - B-Tree follows the properties of Binary Search Tree



Find median of (1,2,3,4), can follow left ordering



Insertion in B-Tree

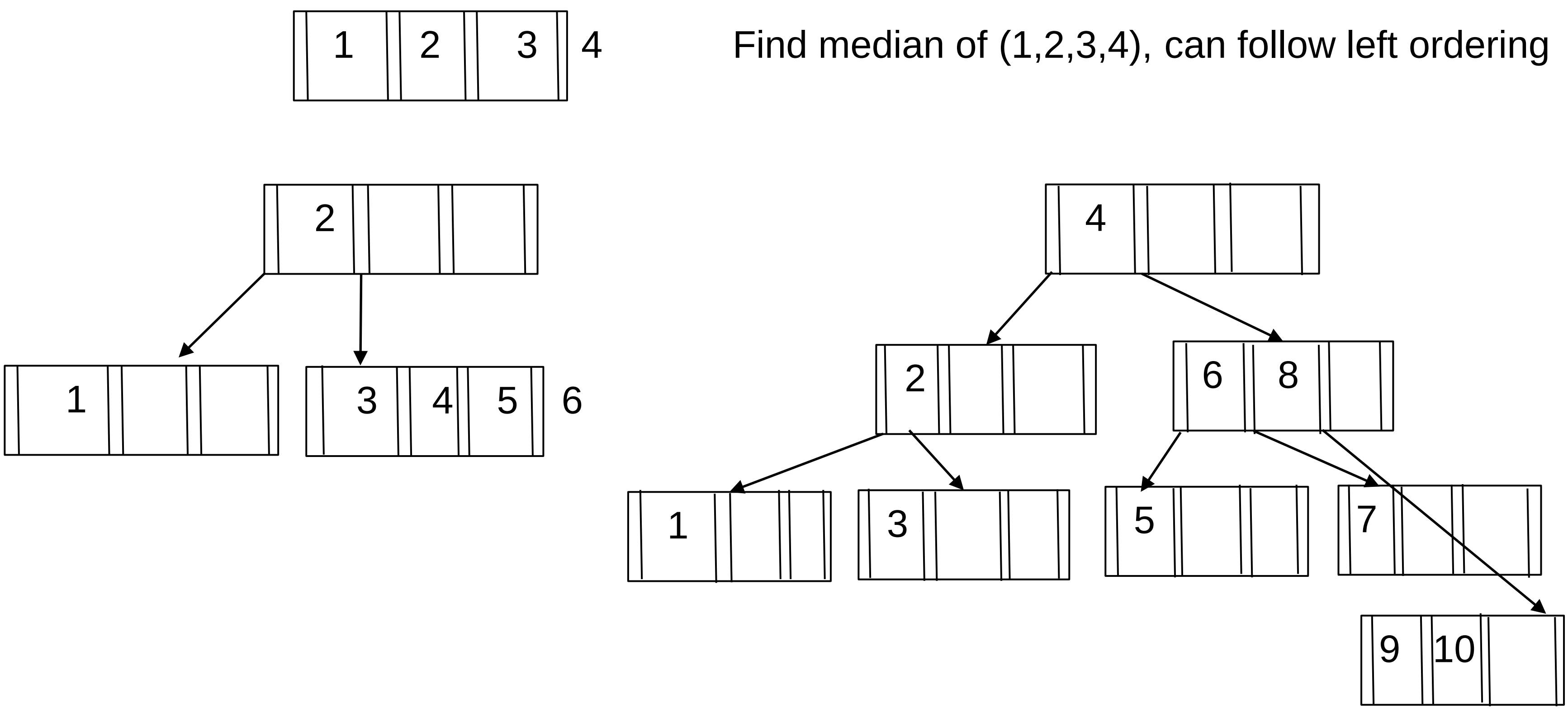
- Question (+1 Reward Question)- The last reward question of the course!!
  - Insert the following keys in the B-Tree that you created:7,8,9,10
    - What is height of B-Tree?
    - How many levels are there in B-Tree?
    - What are key values present in the root node?

Insertion in B-Tree

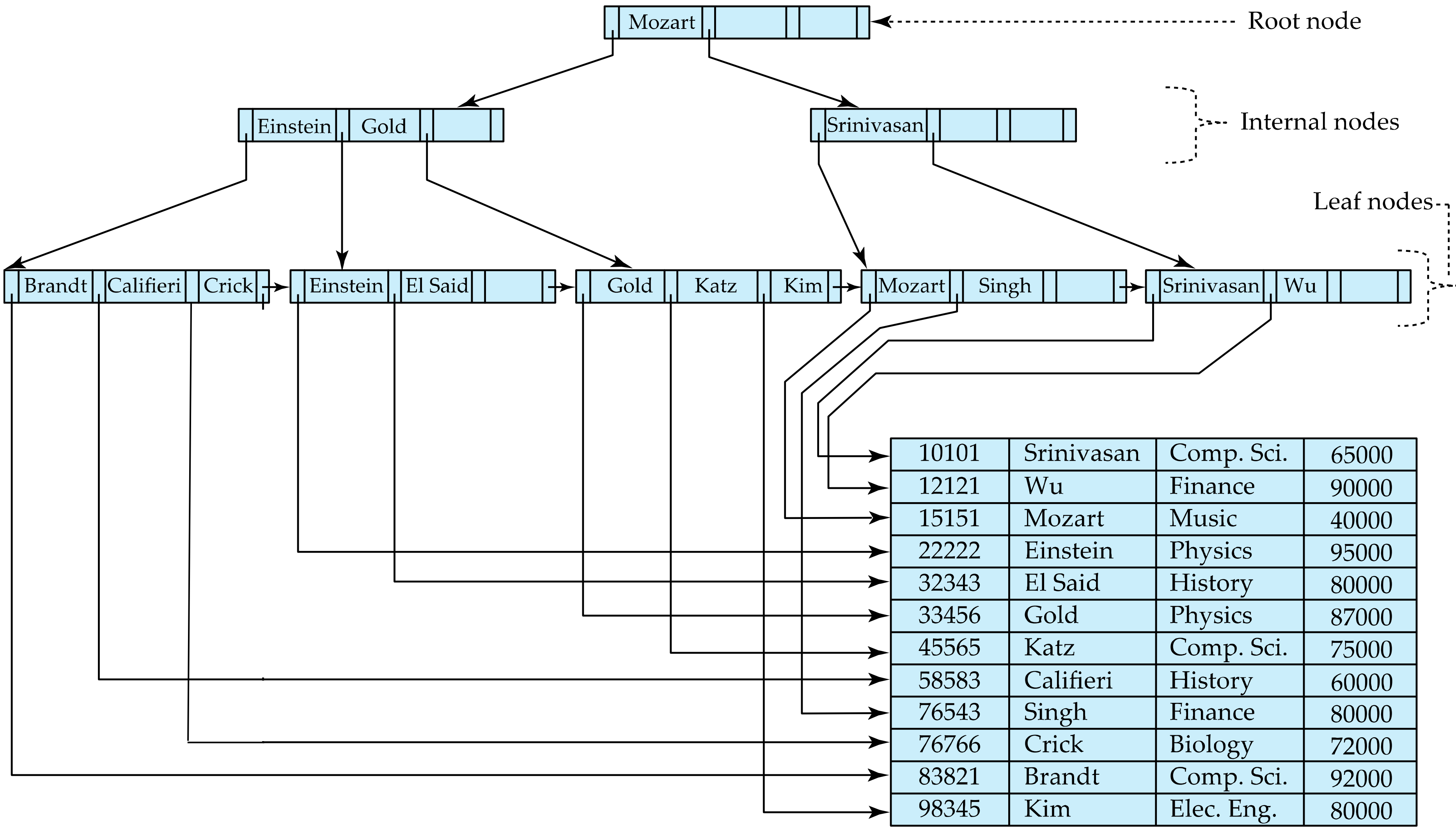
- Question in B-Tree?
  - What is height of B-Tree?
  - How many levels are there in B-Tree?
  - What are key values present in the root node?

Insertion in B-Tree

- Order of a B-Tree is 4, insert the following key in the B-Tree that you created: 1,2,3,4,5,6,7,8,9,10
  - B-Tree follows the properties of Binary Search Tree



B⁺-Tree



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

B-Tree vs. B^+-Tree

B-Tree	**B^+ - Tree**
Data is stored in leaf as well as internal nodes	Data is stored only in leaf nodes
Searching is slower	Searching is faster
No redundant search key present	Redundant keys would present
Leaf nodes are not linked together	Leaf nodes are linked together

References

- Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. Vol. 6. New York: McGraw-Hill, 1997.
- Ramez Elmasri, Shamkant B. Navathe. *Fundamentals of Database Systems.* Edition 6. Pearson, 2010.