

IF3140 – Sistem Basis Data Indexing

SEMESTER II TAHUN AJARAN 2024/2025



Modified from [Silberschatz's slides](#),
"Database System Concepts", 7th ed.

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Summer

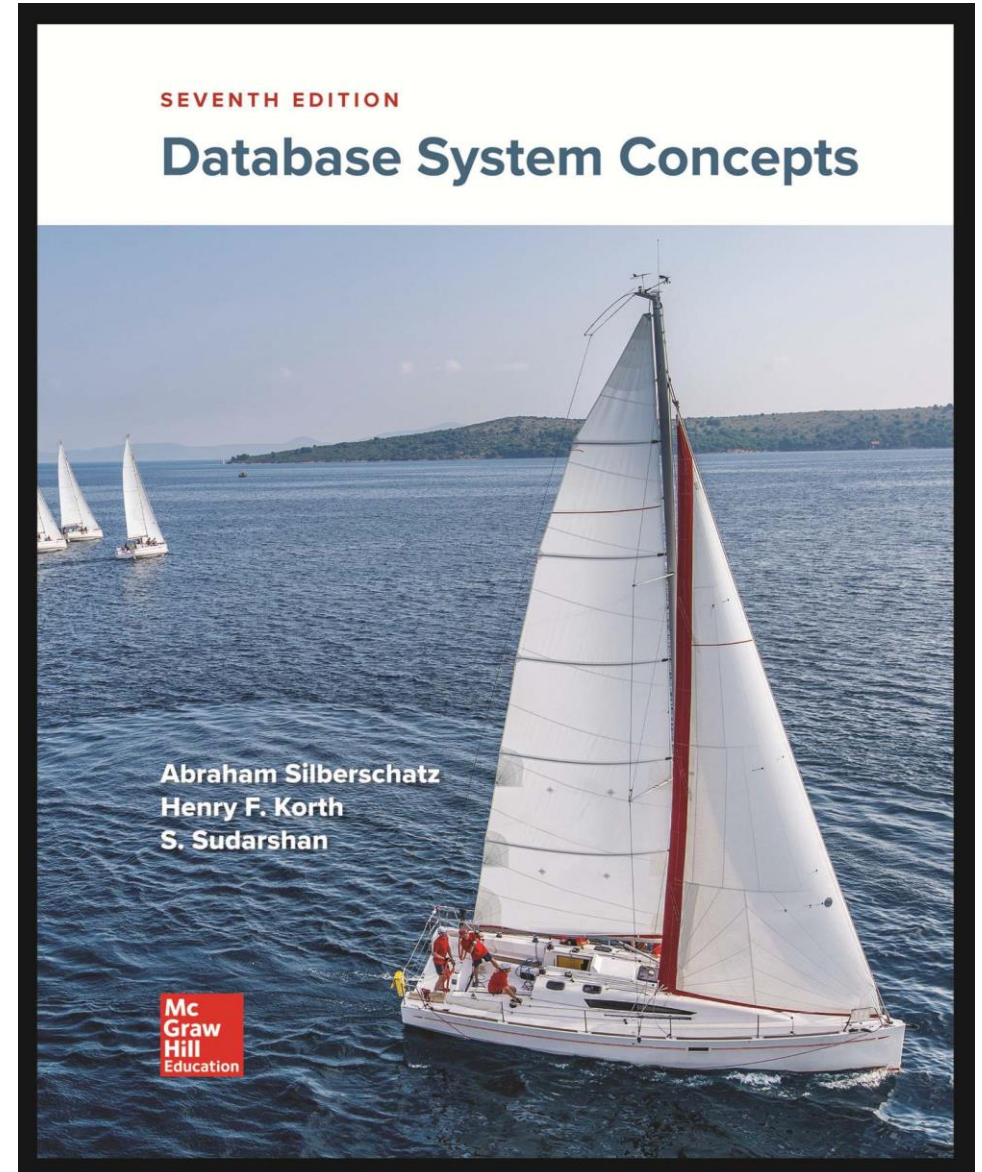
Silberschatz, Korth, Sudarshan:
“Database System
Concepts”, 7th Edition

- **Chapter 14:** Indexing



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
“Database System Concepts”, 7th ed.



Index

- Index → used to speed up access to desired data.
 - E.g., author catalog in library
- **Search Key** - attribute or set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form

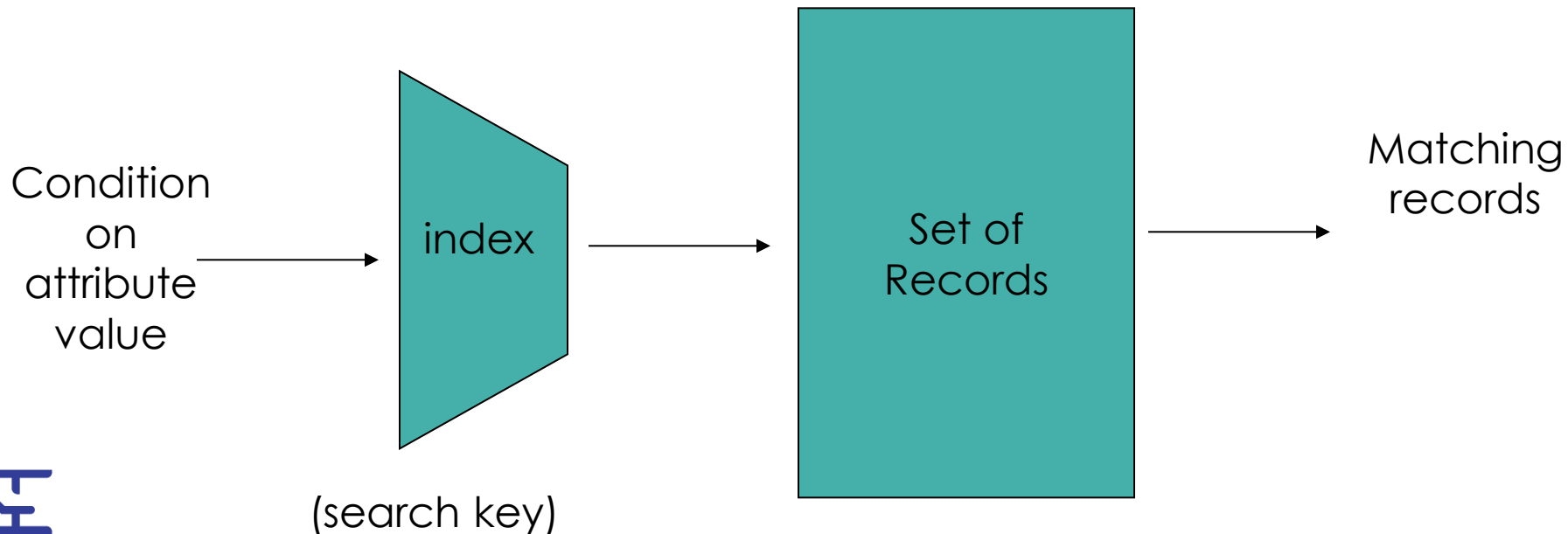
search-key	pointer
------------	---------

- Two basic kinds of indices:
 - **Ordered indices:** search keys are stored in sorted order
 - **Hash indices:** search keys are distributed uniformly across “buckets” using a “hash function”.



Index

- An index is a data structure that **supports efficient access** to data
- It facilitates searching, sorting, join operation, etc., efficiently instead of scanning all table rows
- Index files are typically **much smaller** than the original file



Index Evaluation Metrics

- Access types supported efficiently. E.g.,
 - records with a **specified value** in the attribute
 - or records with an attribute value falling in a specified **range of values**.
- Access time
- Insertion time
- Deletion time
- Space overhead

Several Index Methods

- Several index methods we will discuss:
 - Ordered indices ([index-sequential file](#))
 - B⁺-tree
 - Hash indexes
 - Bitmap indexes

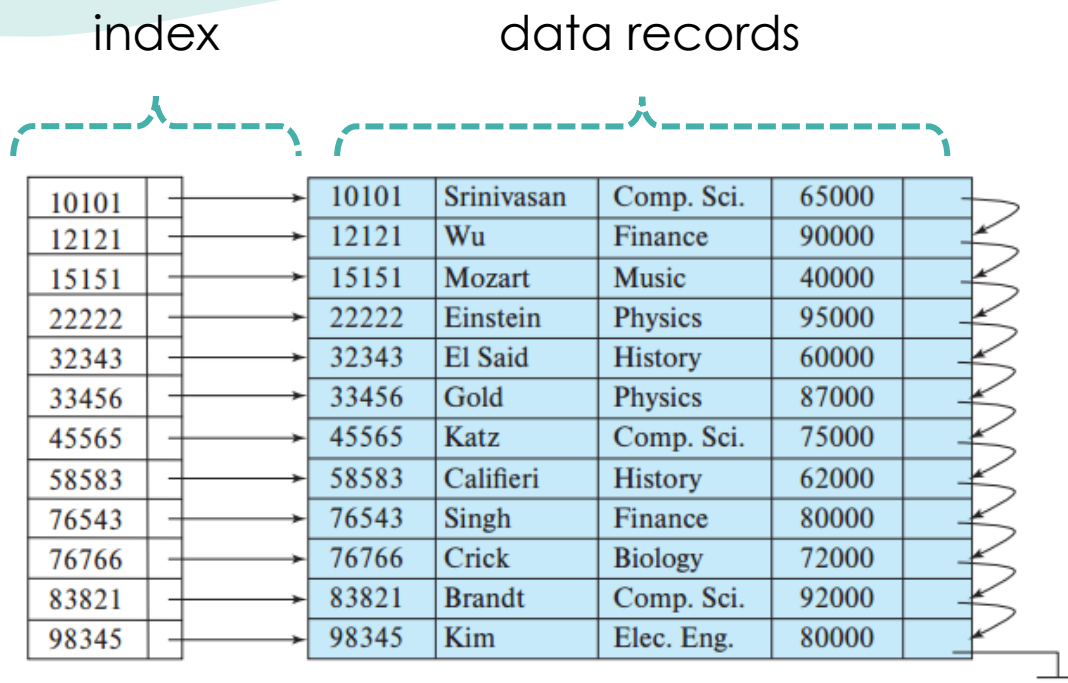


Ordered Indices

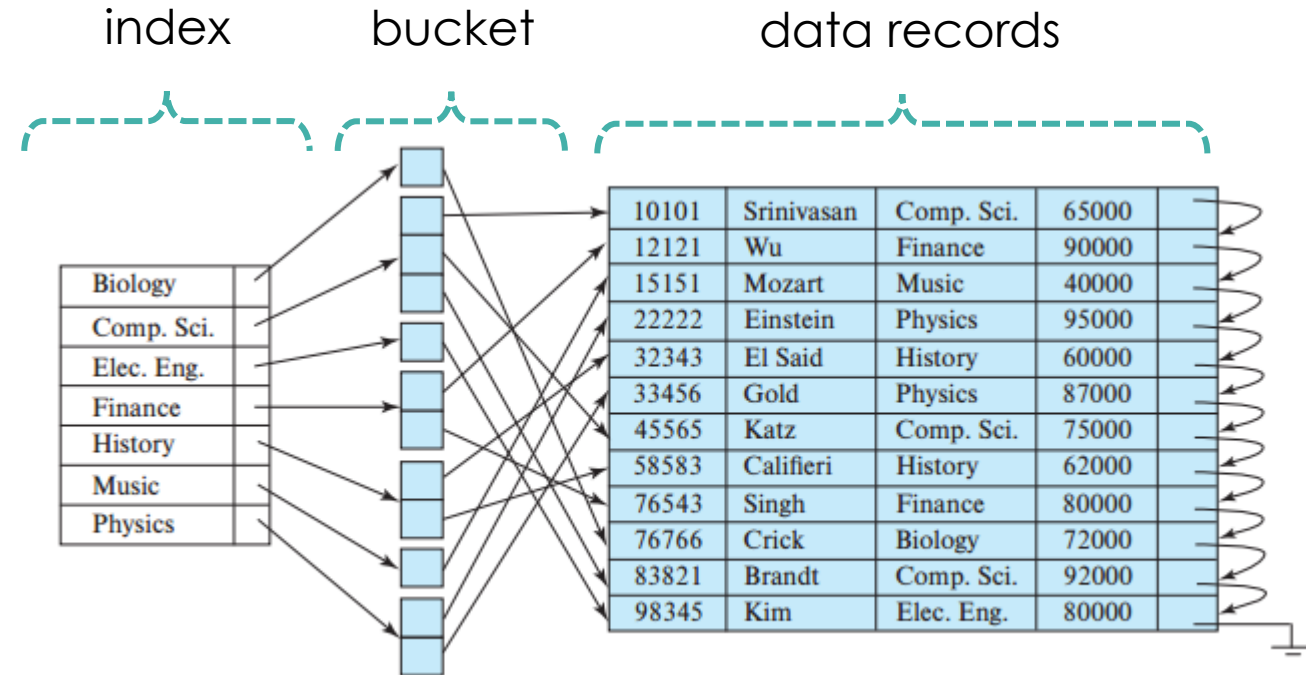
- In an **ordered index**, index entries are stored sorted on the search key value. E.g., author catalog in library.
- **Primary index**: an index whose search key specifies the sequential order of the file. Also called **clustering index**.
 - A table can **have only one clustered index**
 - The search key of a primary index is usually but not necessarily the primary key.
- **Secondary index**: an index whose search key specifies an order different from the sequential order of the file. Also called **non-clustering index**.
- **Index-sequential file**: ordered sequential file with a primary index.



Primary (clustering) vs. Secondary (non-clustering) Index



Primary index (clustering index) on **ID**



Secondary index (non-clustering index) on **dept_name**

- Thus, a table can **have only one clustered index**
- E.g., SQL Server 2005 supports up to 249 non-clustering indices on a table

Ordered indices - Dense Index

- **Dense index** — Every search-key appears in the index records.
- E.g. index with search key on **ID** attribute of **instructor** relation

10101	→	10101	Srinivasan	Comp. Sci.	65000	↙
12121	→	12121	Wu	Finance	90000	↙
15151	→	15151	Mozart	Music	40000	↙
22222	→	22222	Einstein	Physics	95000	↙
32343	→	32343	El Said	History	60000	↙
33456	→	33456	Gold	Physics	87000	↙
45565	→	45565	Katz	Comp. Sci.	75000	↙
58583	→	58583	Califieri	History	62000	↙
76543	→	76543	Singh	Finance	80000	↙
76766	→	76766	Crick	Biology	72000	↙
83821	→	83821	Brandt	Comp. Sci.	92000	↙
98345	→	98345	Kim	Elec. Eng.	80000	↙

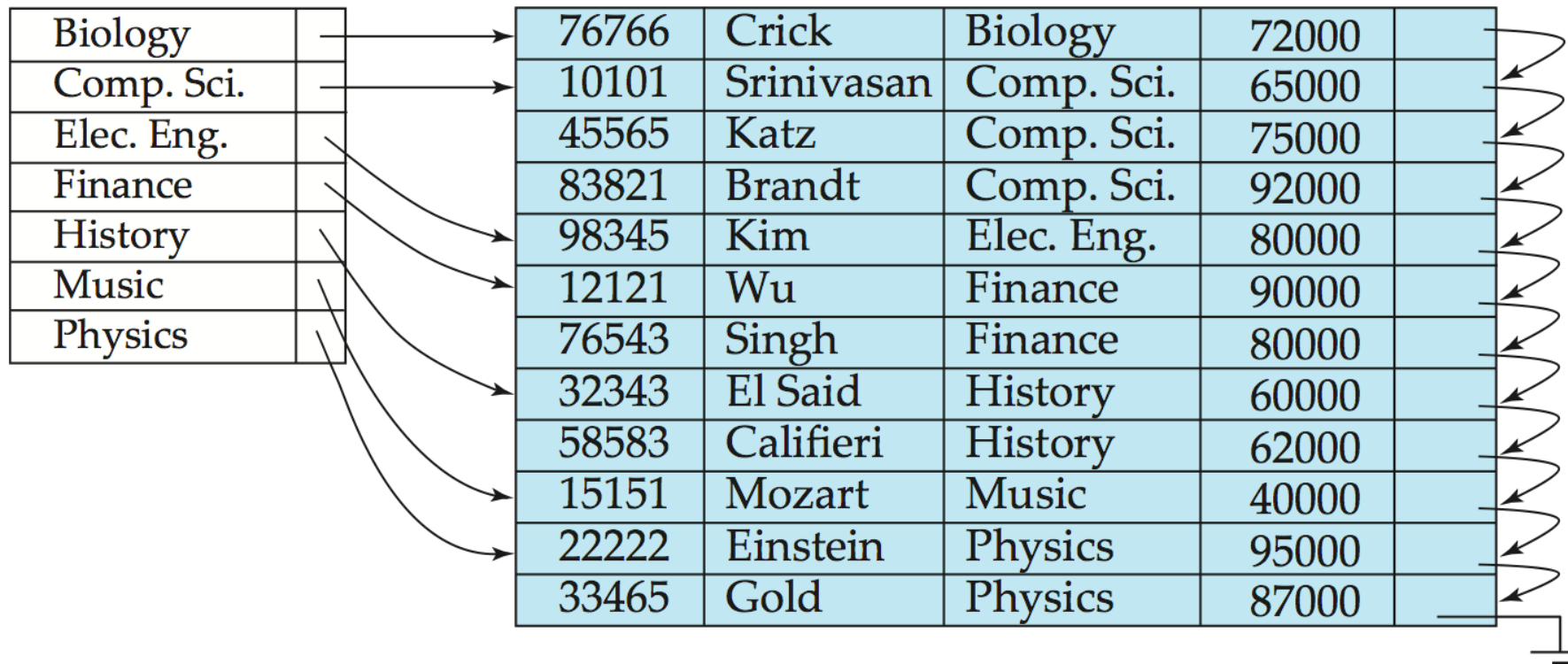
Note:

- Record is sorted based on the search key in a linked-list structure
- Locating index record is done by binary search (complexity = $\log_2 n$)

Ordered indices - Dense Index (Cont.)

- Dense index on **dept_name**, with *instructor* file sorted on **dept_name**

Biology		76766	Crick	Biology	72000	
Comp. Sci.		10101	Srinivasan	Comp. Sci.	65000	
Elec. Eng.		45565	Katz	Comp. Sci.	75000	
Finance		83821	Brandt	Comp. Sci.	92000	
History		98345	Kim	Elec. Eng.	80000	
Music		12121	Wu	Finance	90000	
Physics		76543	Singh	Finance	80000	
		32343	El Said	History	60000	
		58583	Califieri	History	62000	
		15151	Mozart	Music	40000	
		22222	Einstein	Physics	95000	
		33465	Gold	Physics	87000	



See! All the search-keys appear in the index file

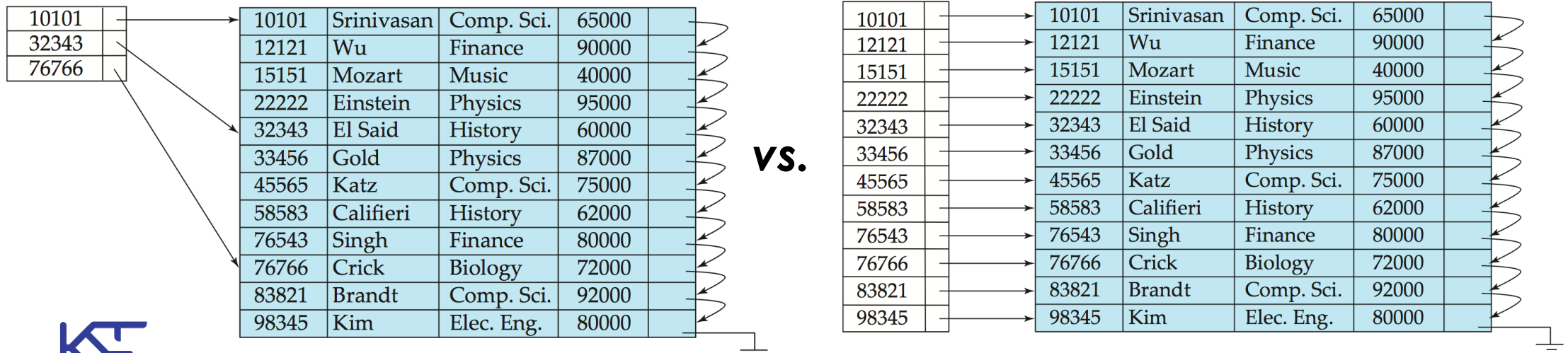
Ordered indices – Sparse Index

- **Sparse Index**: only some search-key values appear in the index records.
 - 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 that is $\leq K$**
 - **Search file sequentially** starting at the record to which the index record points

10101		10101	Srinivasan	Comp. Sci.	65000	
32343		12121	Wu	Finance	90000	
76766		15151	Mozart	Music	40000	
		22222	Einstein	Physics	95000	
		32343	El Said	History	60000	
		33456	Gold	Physics	87000	
		45565	Katz	Comp. Sci.	75000	
		58583	Califieri	History	62000	
		76543	Singh	Finance	80000	
		76766	Crick	Biology	72000	
		83821	Brandt	Comp. Sci.	92000	
		98345	Kim	Elec. Eng.	80000	

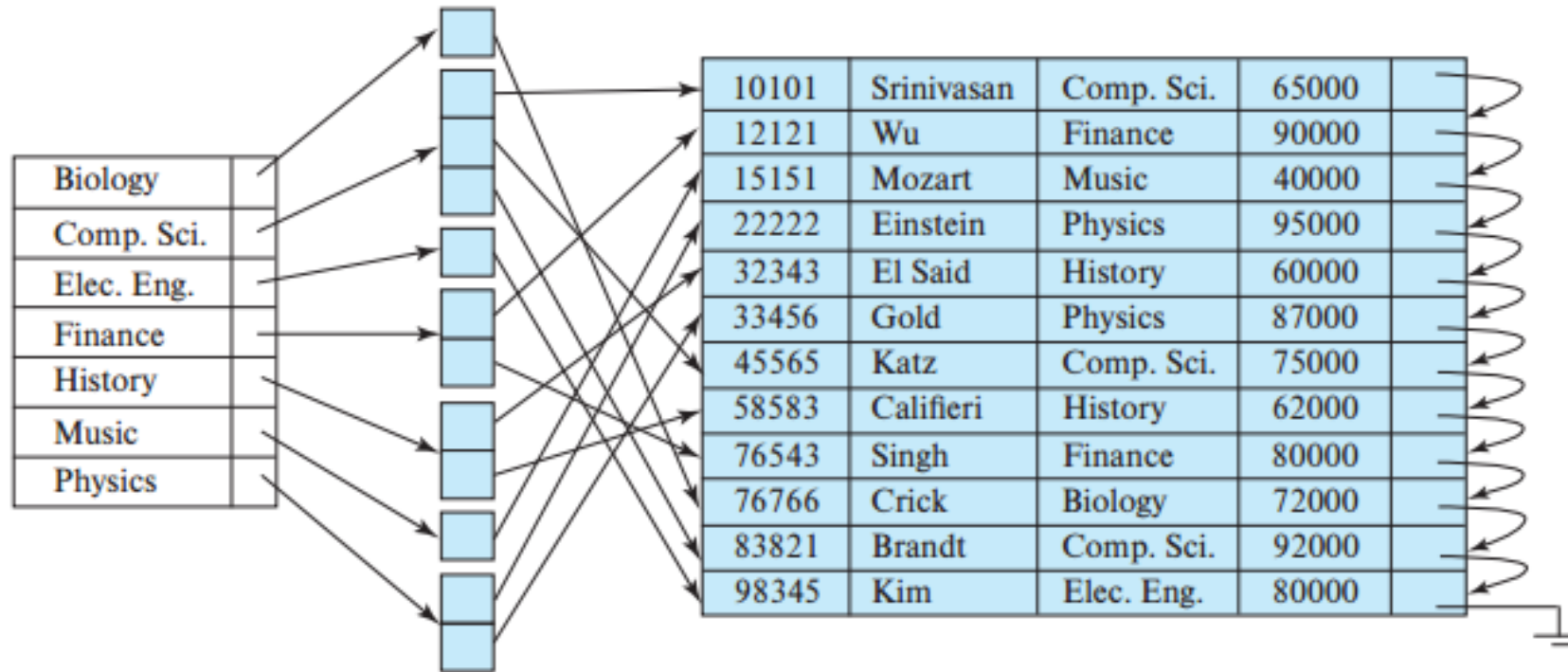
Ordered indices – Sparse Index (Cont.)

- Compared to dense indices:
 - Less space and less maintenance overhead for insertions and deletions.
 - Generally slower than dense index for locating records.
- **Good tradeoff:** sparse index with an index entry for every block in file, corresponding to least search-key value in the block.



Ordered indices – Secondary Index (revisit)

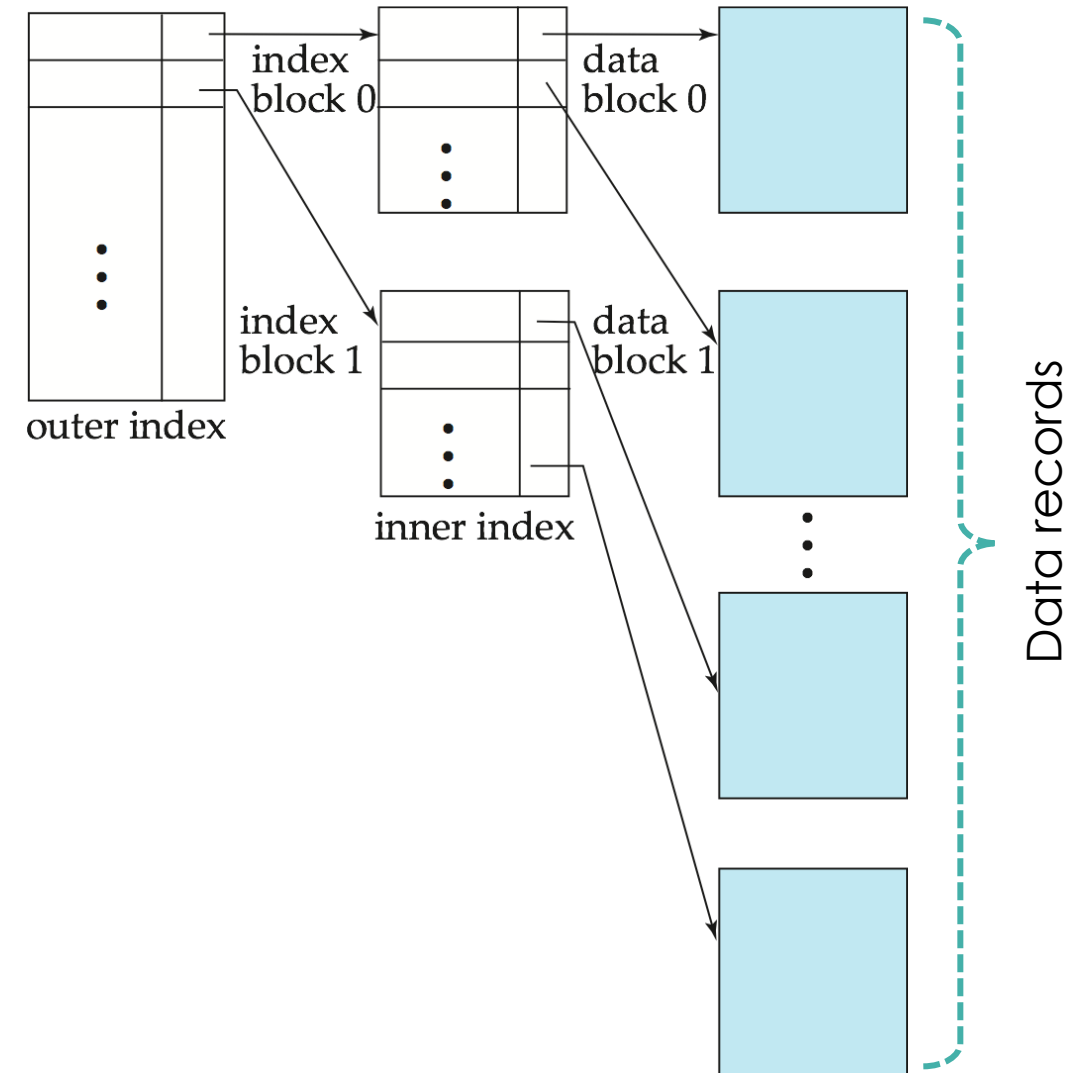
- Index record **points to a bucket** that contains pointers to all the actual records with that particular search-key value.
- Secondary indices **have to be dense**



Secondary index on **dept_name** attribute of *instructor*

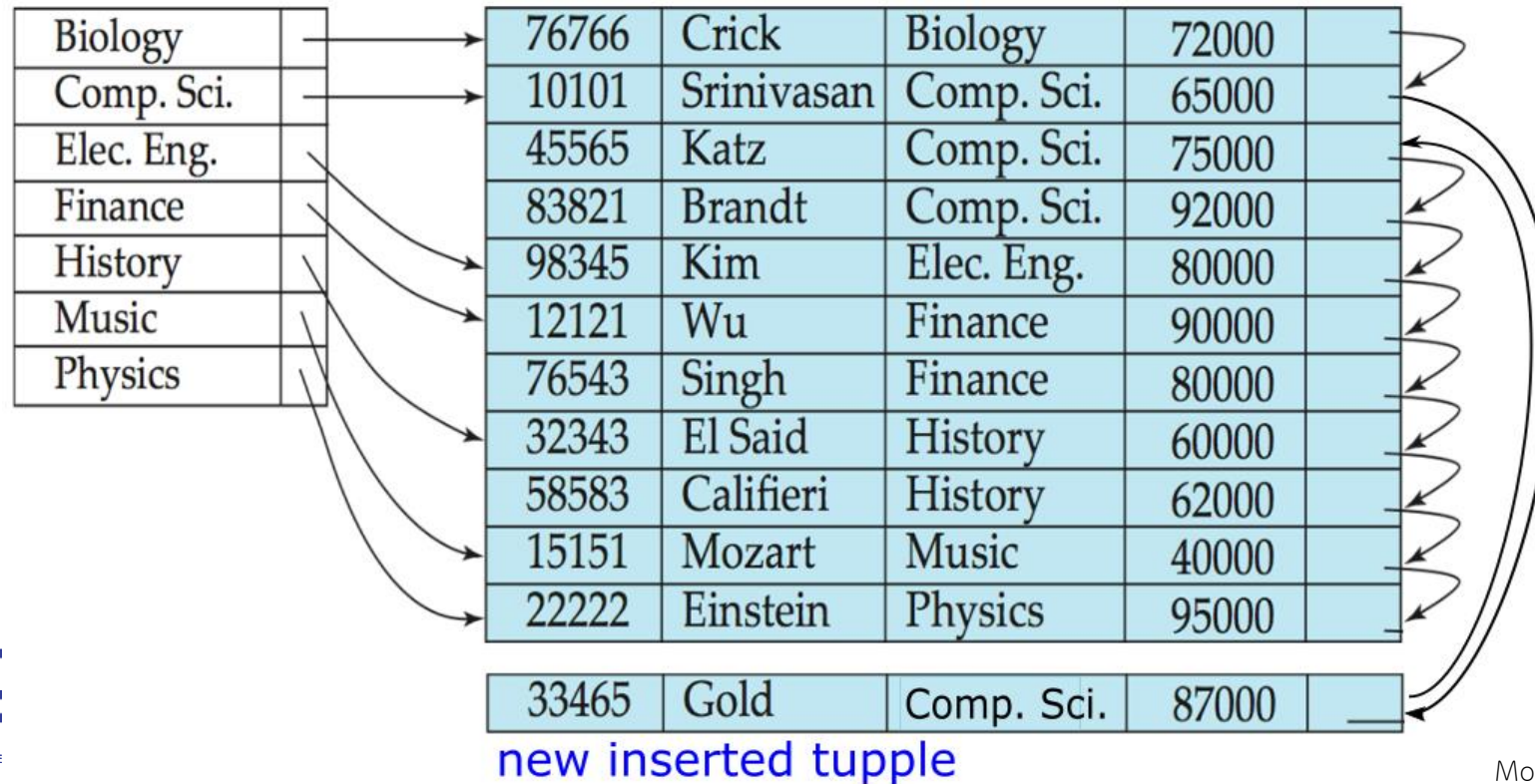
Ordered indices – Multilevel Index

- If primary index does not fit in memory, access becomes expensive.
- **Solution:** treat primary index kept on disk as a sequential file and construct a sparse index on it.
 - **outer index** – a sparse index of primary index
 - **inner index** – the primary index file



Ordered indices on Insert, Delete, Update

- In **index-sequential file**, as file grows with insert, delete and update, many **overflow blocks get created**.
- **Periodic reorganization** of entire file is required, otherwise, **more random disk I/O is needed**.



E.g., querying records whose **dept_name** is "Comp. Sci." now needs more disk I/O

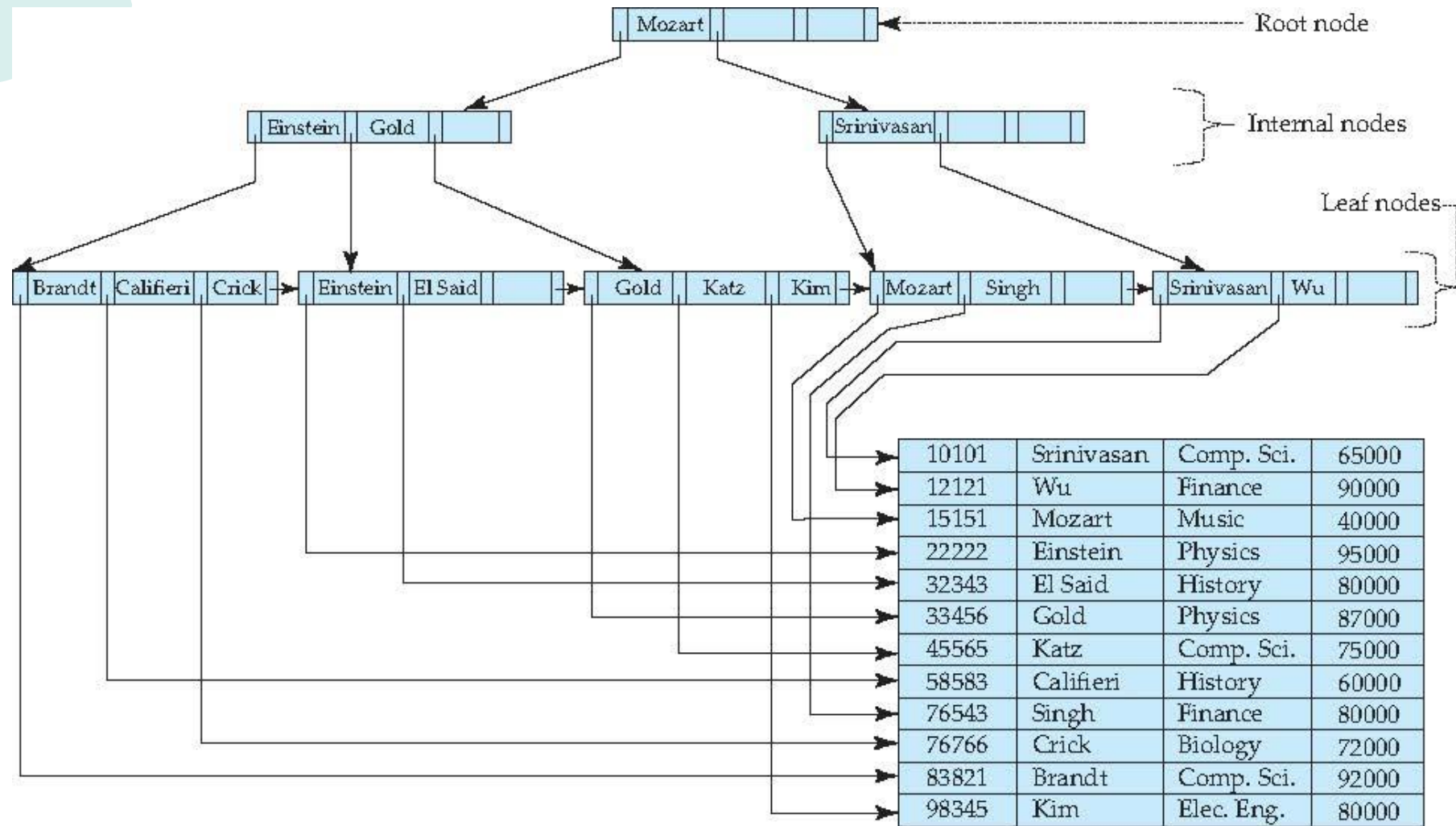
B⁺-Tree Index

B⁺-tree indices are **an alternative to indexed-sequential files**.

- Disadvantage of indexed-sequential files
 - **performance degrades** as file grows, since **many overflow blocks get created**.
 - **Periodic reorganization** of entire file is required.
 - Searching using binary search has **(log₂ n) cost**, later, we will know a cheaper cost using B⁺-tree
- Advantage of B⁺-tree index files:
 - automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
 - **Reorganization of entire file is not required** to maintain performance.
- (Minor) disadvantage of B⁺-trees:
 - extra insertion and deletion overhead, space overhead.
- Advantages of B⁺-trees **outweigh** disadvantages
 - **B⁺-trees are used extensively**



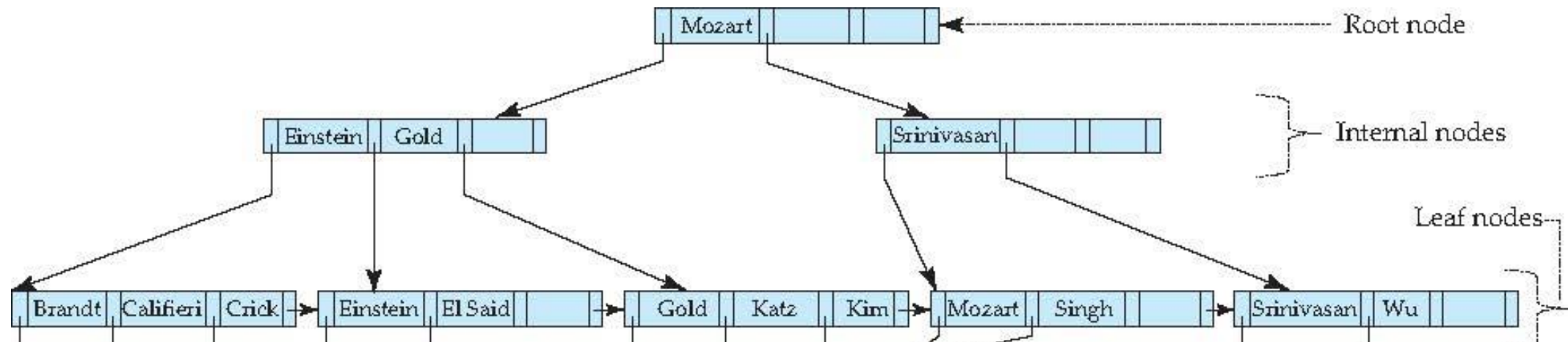
Example of B⁺-Tree



B+-Tree Index (Cont.)

- A B+-tree with max degree n is a rooted tree satisfying the following properties:
 - All paths from root to leaf are of the same length
 - Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children
 - A leaf node has between $\lceil (n-1)/2 \rceil$ and $n-1$ values
 - Special cases:
 - If the root is not a leaf, it has at least 2 children.
 - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and $(n-1)$ values.

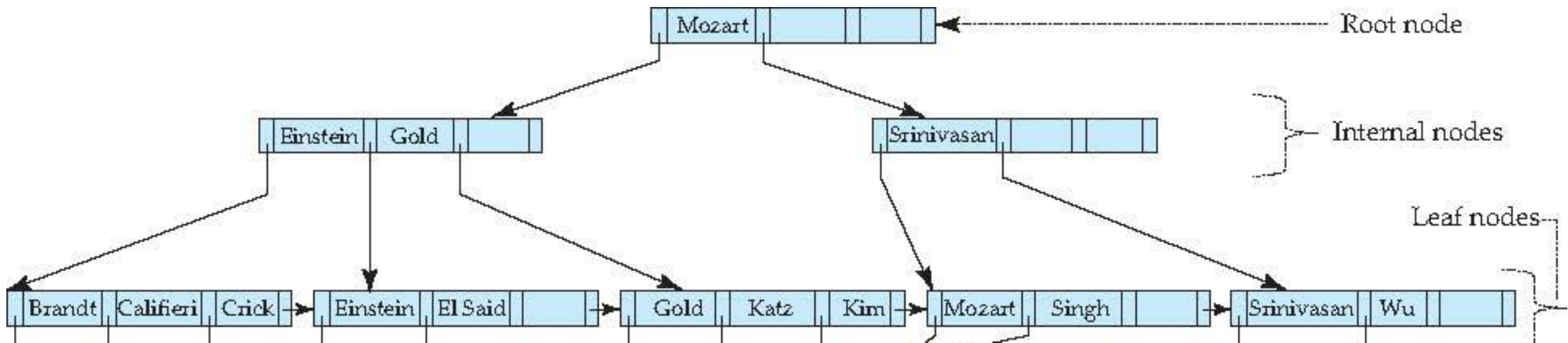
B+-Tree Index (Cont.)



A B⁺-tree example with n = 4

B⁺-Tree Index (Cont.)

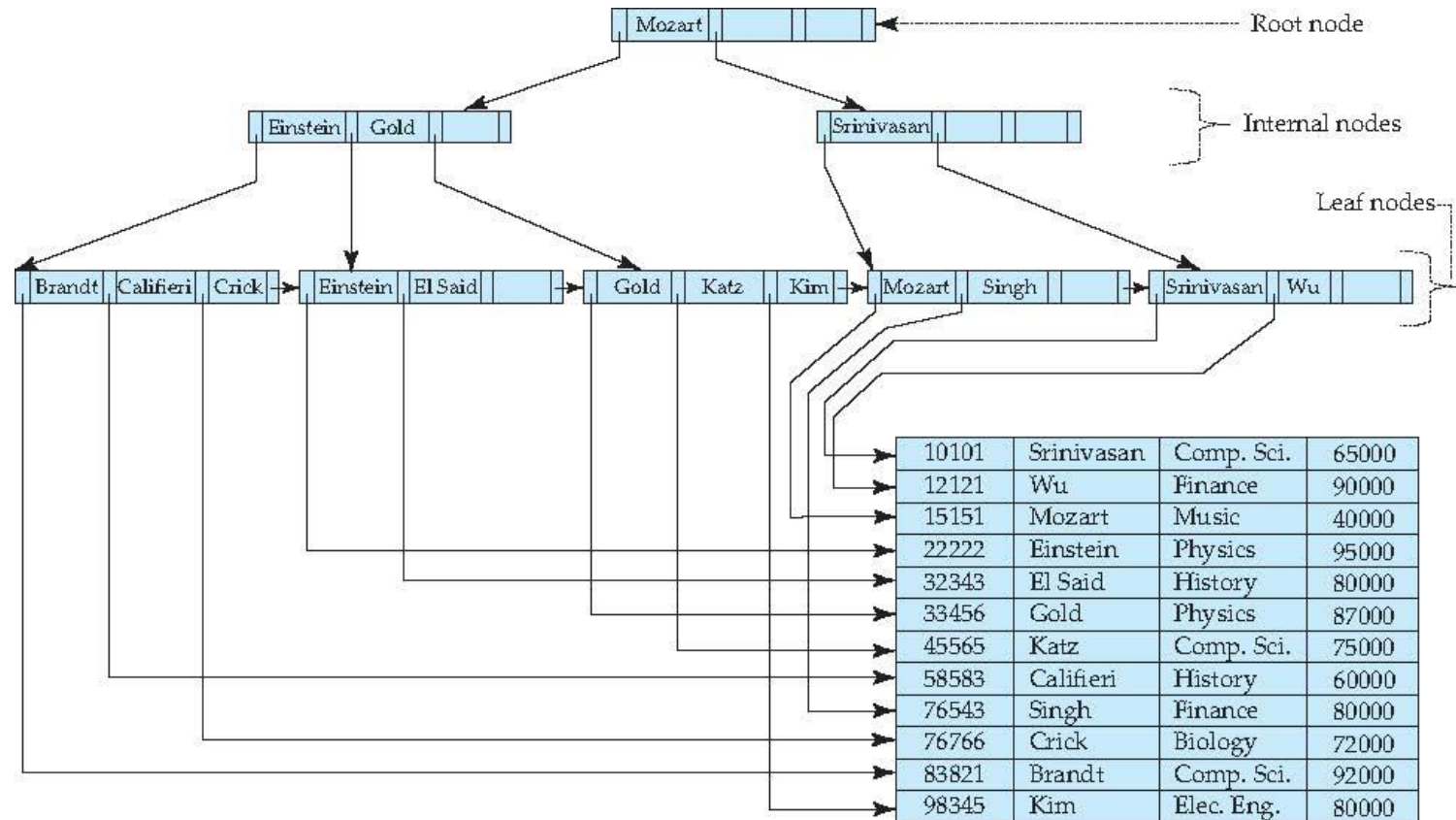
- Search **key is in sorted order**, having properties of n-ary search tree
- There is a **pointer** in the end of leaf pointing to the **next leaf**
- The non-leaf levels of the B⁺-tree form a hierarchy of sparse indices.



A B⁺-tree example with n = 4

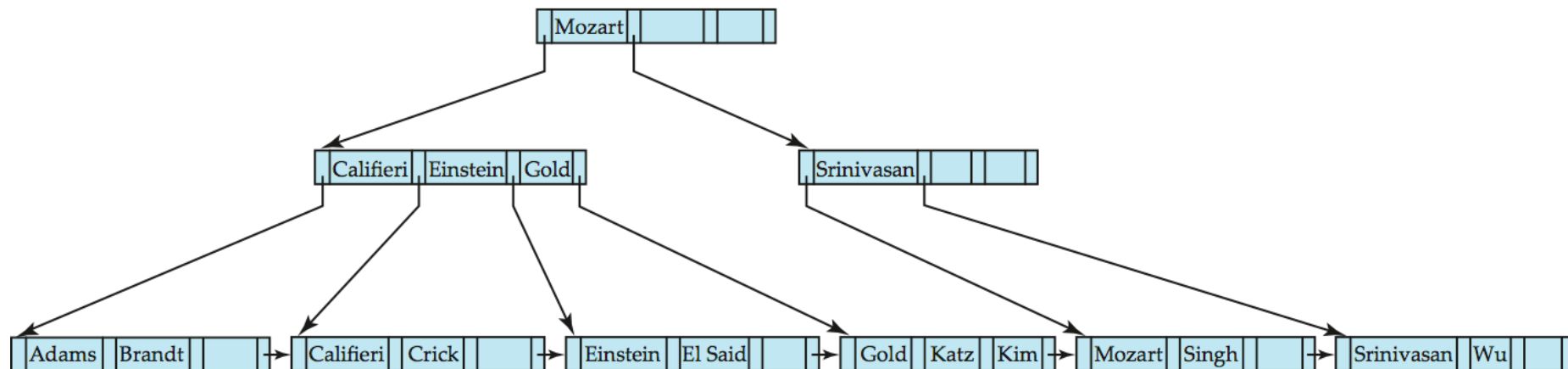
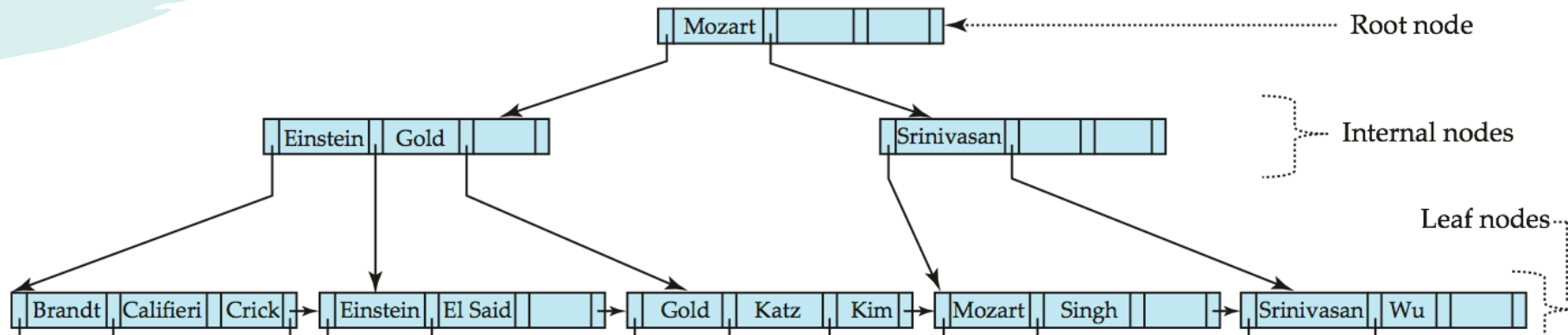
Queries on B^+ -Trees

- With K search-key values, the tree height is no more than $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$.
- A node is generally the same size as a disk block, typically 4 KB
 - and n is typically around 100 (40 bytes per index entry).
- With 1 million search key values and $n = 100$
 - at most $\log_{50}(1,000,000) = 4$ nodes are accessed in a lookup.



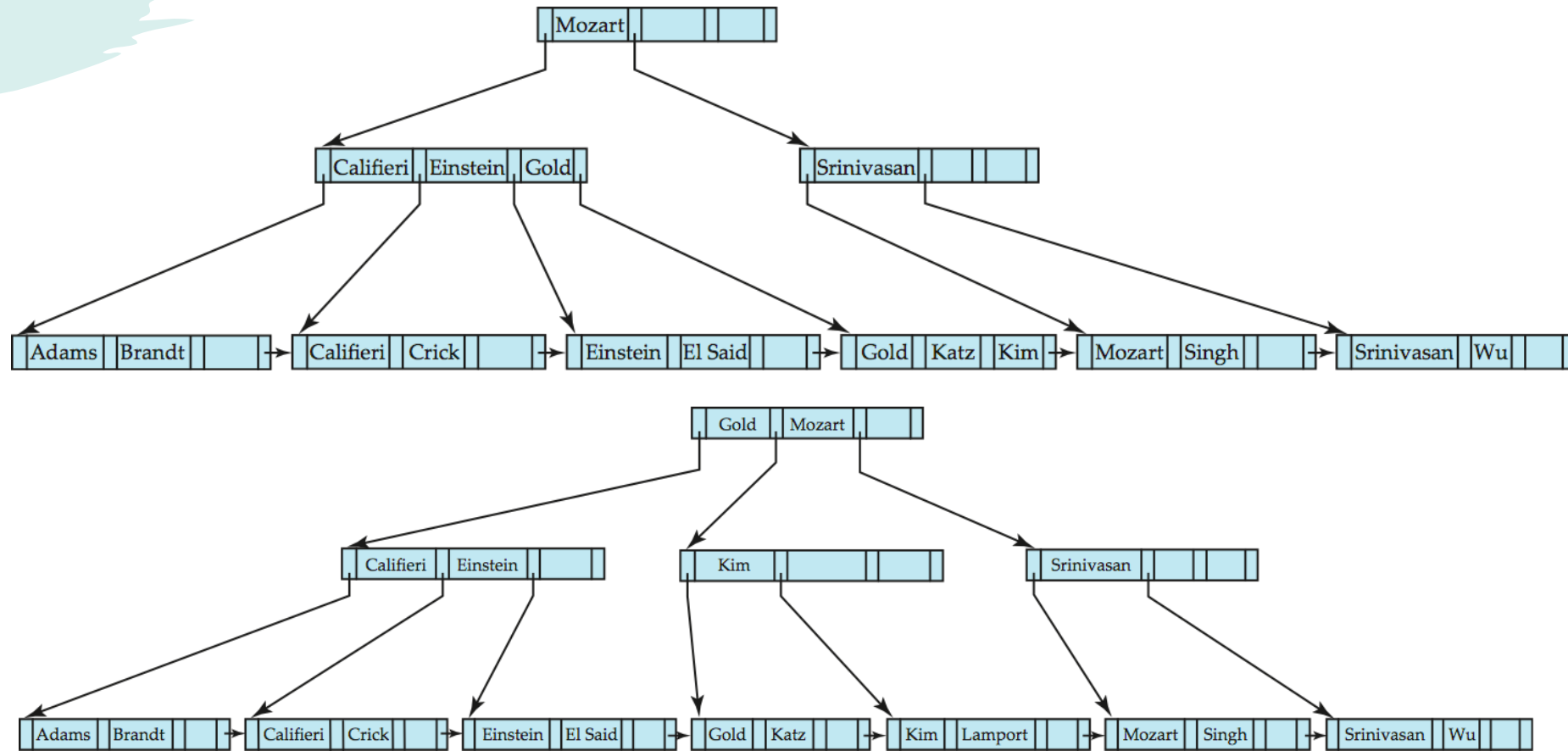
- It handles efficiently **range query**!

B⁺-Tree Insertion



B⁺-Tree before and after insertion of "Adams"

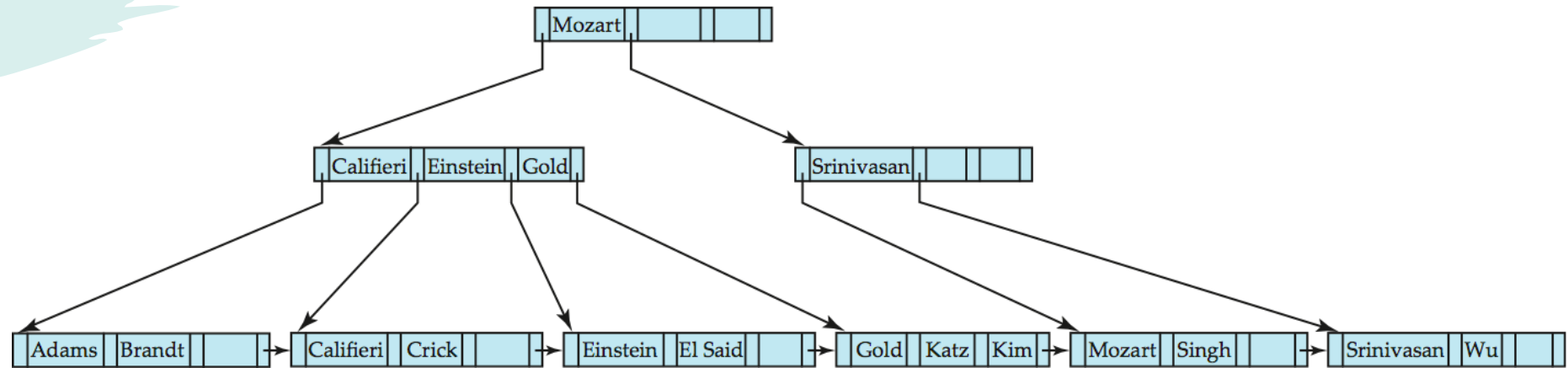
B⁺-Tree Insertion (Cont.)



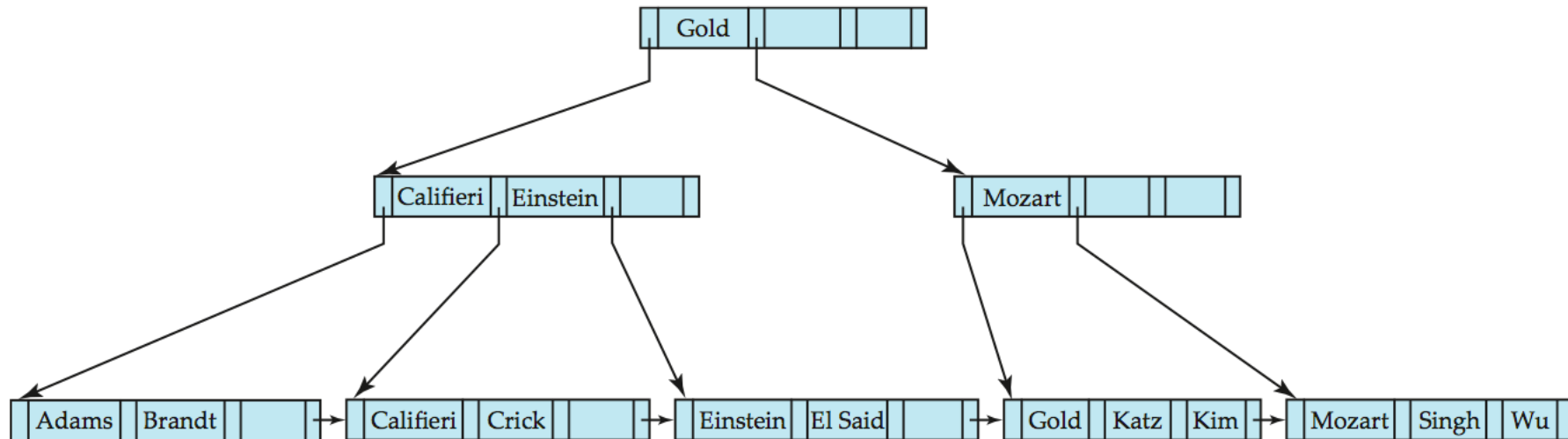
B⁺-Tree before and after insertion of “Lampport”

- There is **split** and **merge** mechanisms to maximize the number of search-keys in each node. Read the detail in the textbook!

Examples of B^+ -Tree Deletion

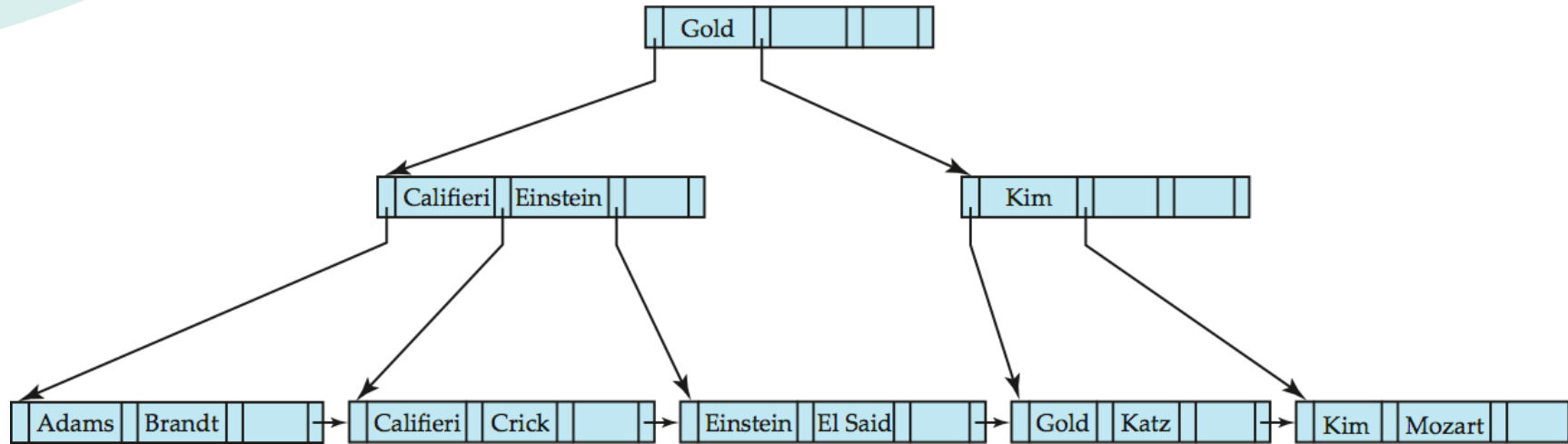


Before and after deleting "Srinivasan"



- Deleting "Srinivasan" causes merging of under-full leaves

Examples of B+-Tree Deletion (Cont.)



Deletion of “Singh” and “Wu” from result of previous example

- Leaf containing Singh and Wu became underfull, and borrowed a value Kim from its left sibling
- Search-key value in the parent changes as a result

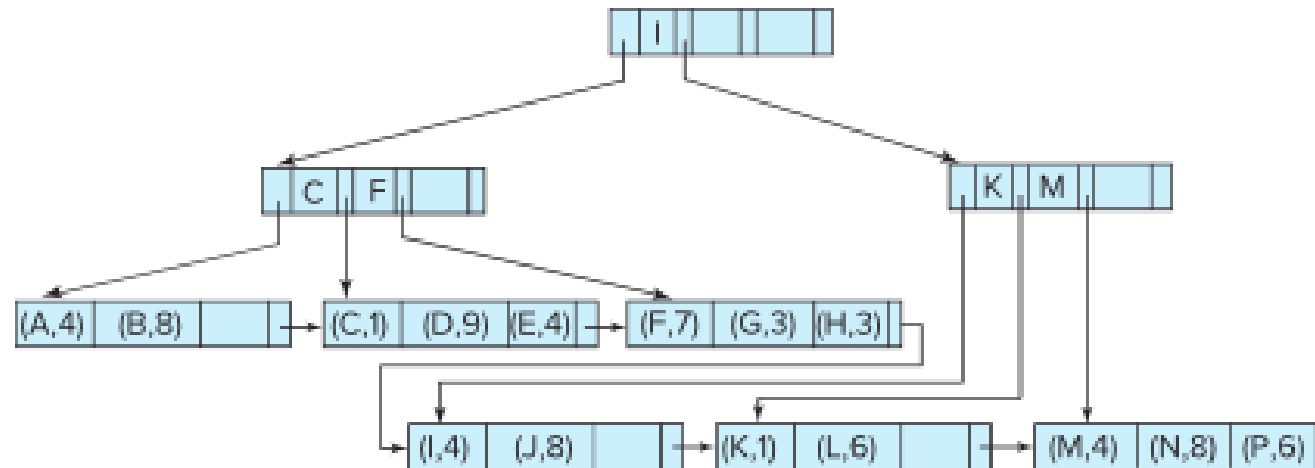
B+-Tree File Organization

- B+-Tree File Organization:
 - Leaf nodes in a B+-tree file organization store records, instead of pointers
 - Helps keep data records clustered even when there are insertions/deletions/updates
- Leaf nodes are still required to be half full
 - Since records are larger than pointers, the maximum number of records that can be stored in a leaf node is less than the number of pointers in a nonleaf node.
- Insertion and deletion are handled in the same way as insertion and deletion of entries in a B+-tree index.



B+-Tree File Organization (Cont.)

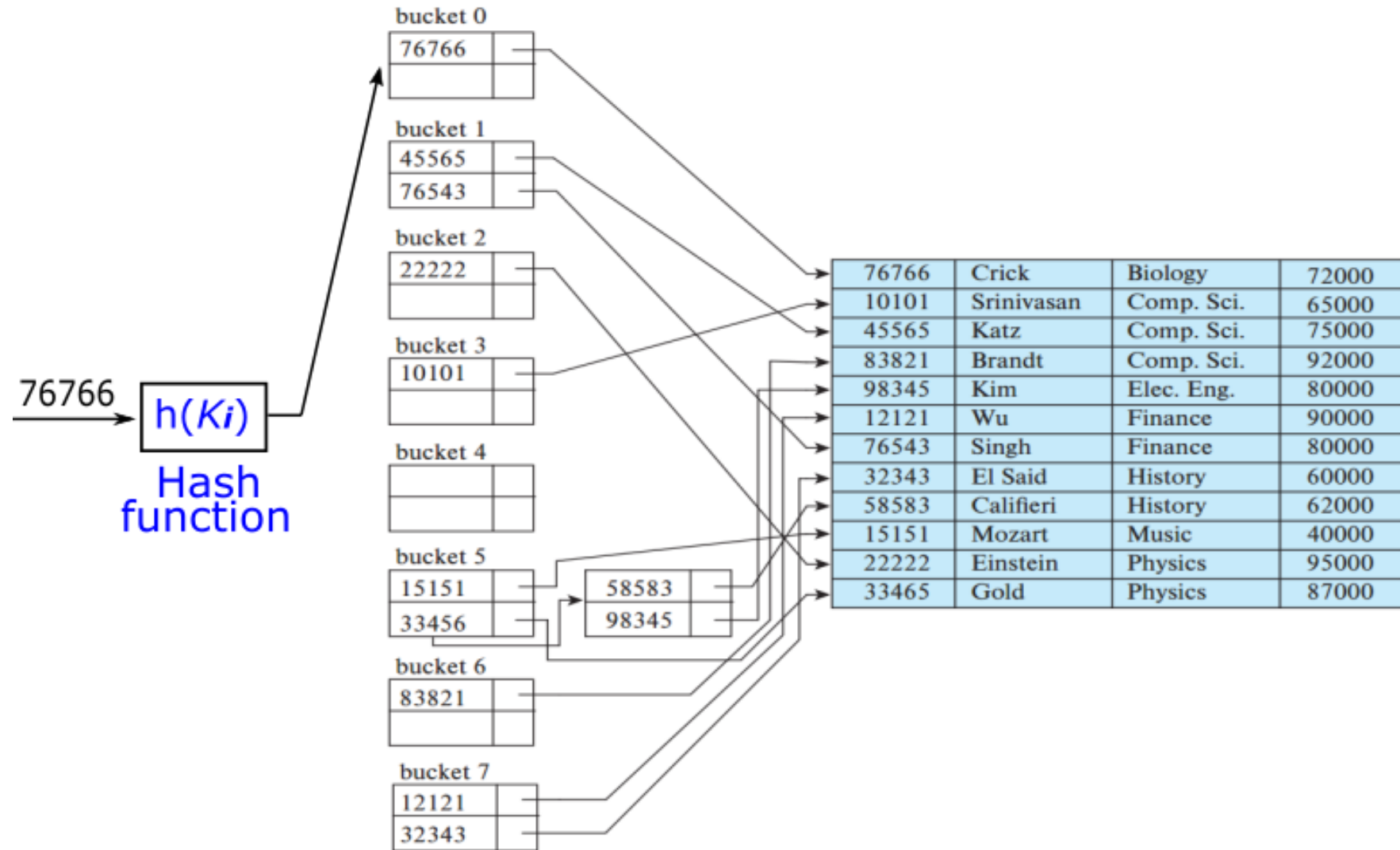
- Example of B+-tree File Organization



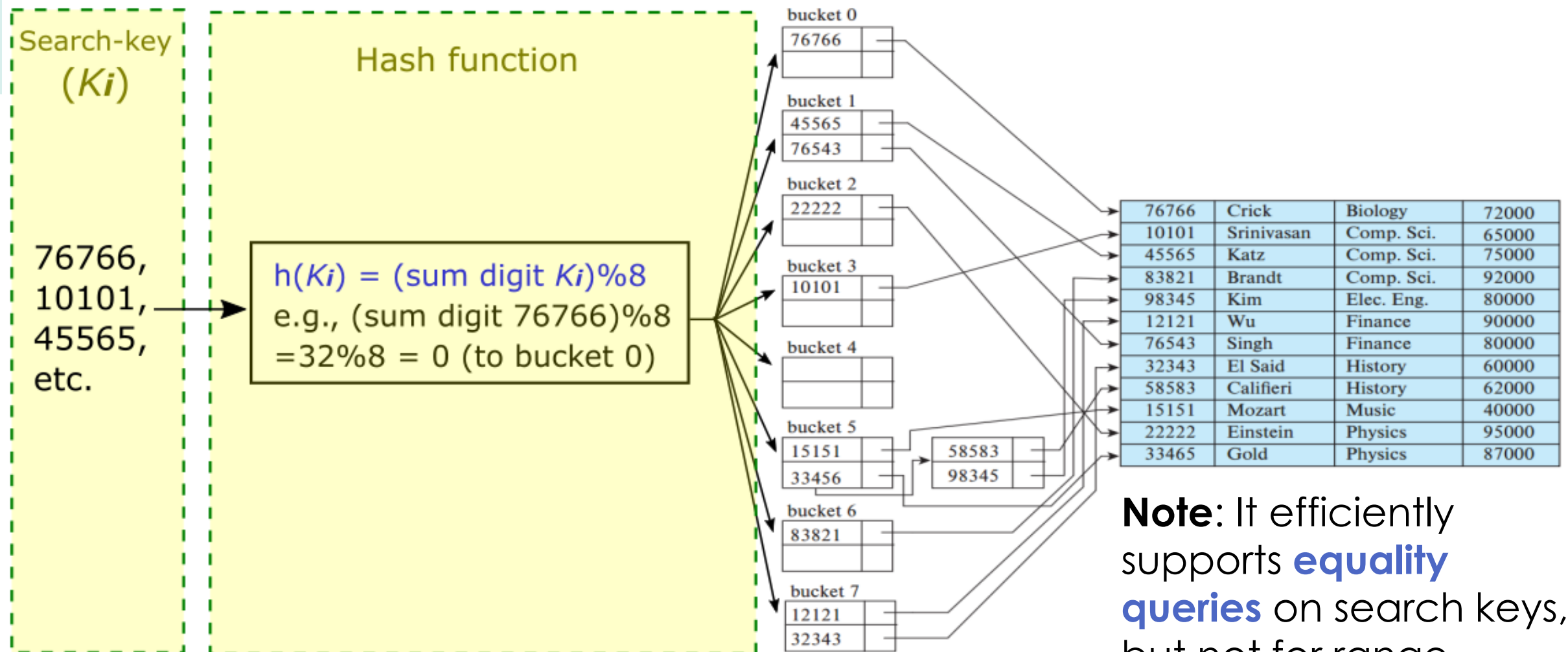
- Good space utilization important since records use more space than pointers.
- To improve space utilization, involve more sibling nodes in redistribution during splits and merges
 - Involving 2 siblings in redistribution (to avoid split / merge where possible) results in each node having at least $\lfloor 2n/3 \rfloor$ entries

Hash Index

- Hash index uses function $h(k_i)$ mapping search-key value K_i to bucket.
- Bucket** is a unit of storage containing one or more index records, typically is one block
- Bucket **contains search-key and pointer to record**



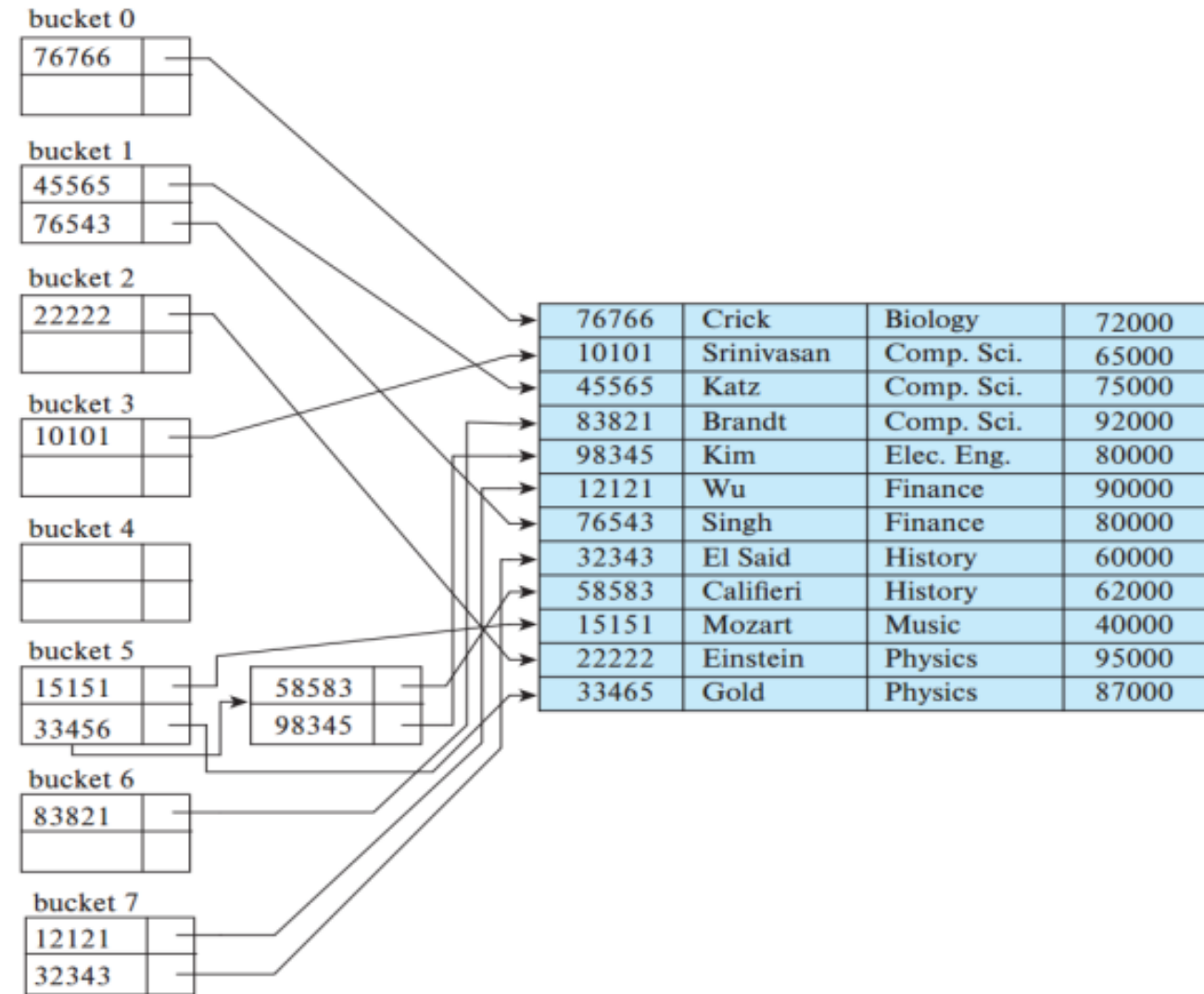
Hash Index Example



Note: It efficiently supports **equality queries** on search keys, but not for range queries.

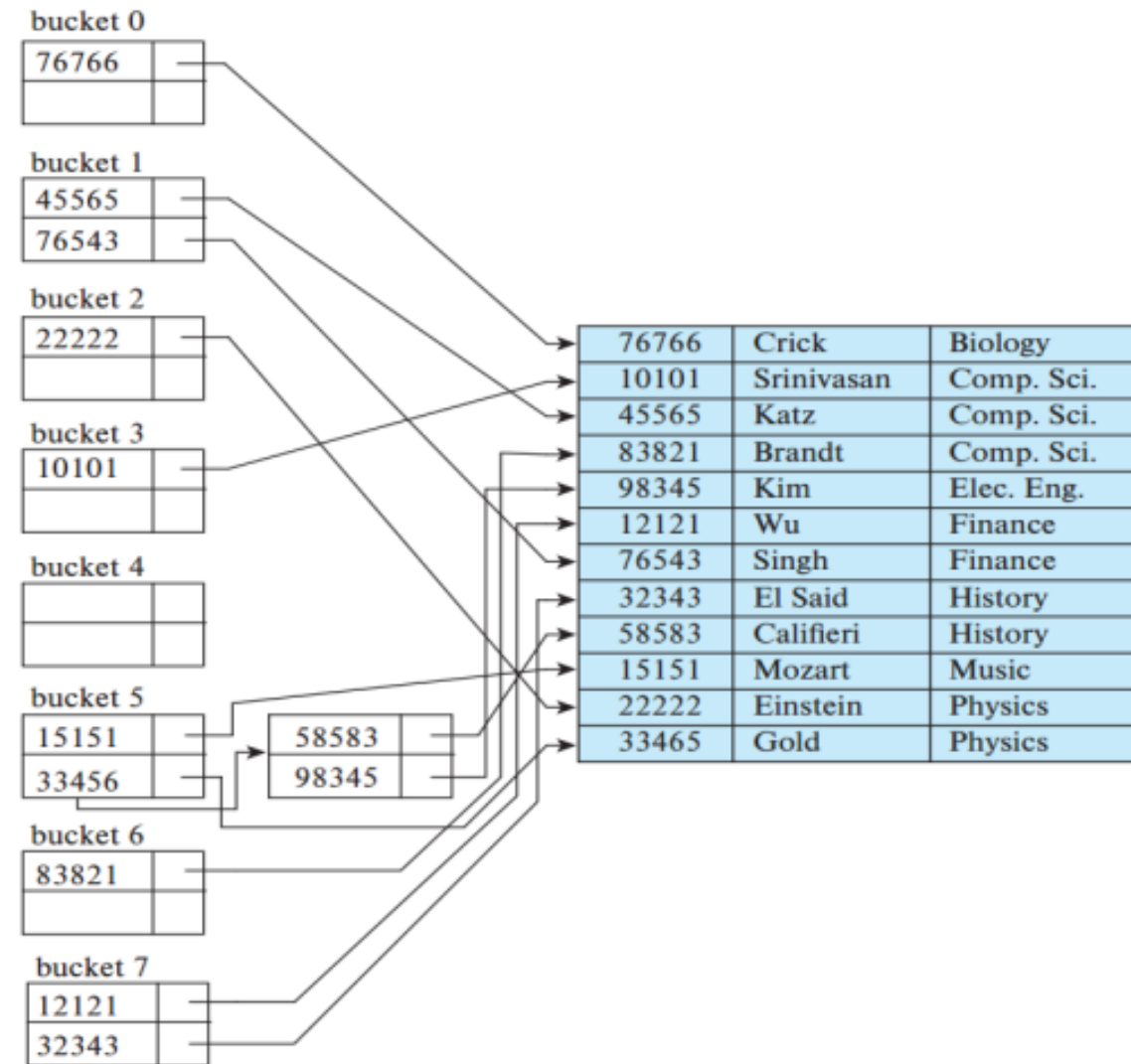
Hash Index (Cont.)

- Records with **different search-key** values may be **mapped to the same** bucket causing **overflow** bucket (see bucket 5)
- Overflow chaining** – the overflow buckets of a given bucket are chained together in a linked list.
- In such case, entire bucket has to be searched sequentially to locate a record.



Hash Index (Cont.)

- Ideal hash function has these properties to minimize overflow bucket
 - **Uniform:** the hash function assigns each bucket the **same number** of index records
 - **Random:** each bucket will have nearly the same number of index records, **regardless of the actual distribution** of search-key values



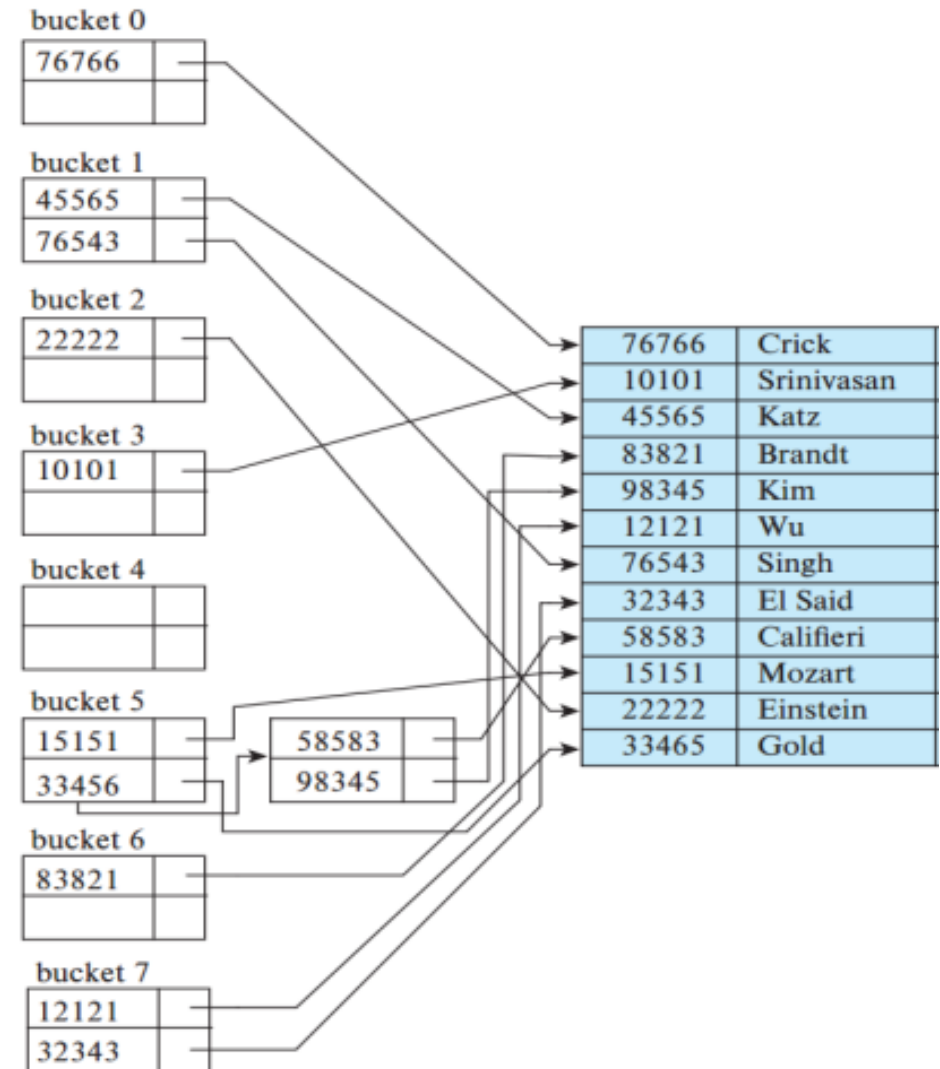
Hash Index (Cont.)

- Type of hashing methods:
 - **Static hashing:** the set of buckets is fixed at the time the index is created → need to know how many records in advance
 - **Dynamic hashing:** the hash index can be rebuilt with an increased number of buckets

More details, read in the book



KNOWLEDGE & SOFTWARE ENGINEERING



Hash File Organization

- A **bucket** is a unit of storage containing one or more records (a bucket is typically a disk block).
- In a **hash file organization** we obtain the bucket of a record directly from its search-key value using a **hash function**.
- Hash function h is a function from the set of all search-key values K to the set of all bucket addresses B .
- Hash function is used to locate records for access, insertion as well as deletion.
- Records with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate a record.

Example of Hash File Organization

Hash file organization of *instructor* file, using *dept_name* as key
(See figure in next slide.)

- There are 10 buckets,
- The binary representation of the *i*th character is assumed to be the integer *i*.
- The hash function returns the sum of the binary representations of the characters modulo 10
 - E.g. $h(\text{Music}) = 1$ $h(\text{History}) = 2$
 $h(\text{Physics}) = 3$ $h(\text{Elec. Eng.}) = 3$



Example of Hash File Organization (cont.)

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Hash file organization of *instructor* file, using *dept_name* as key (see previous slide for details).



Bitmap Indices

- Special type of index designed for **efficient querying on multiple keys**
- Applicable on attributes having relatively **small number of distinct values**
 - **E.g.**, gender, country, state, etc.
 - **E.g.**, income-level (broken up into a small number of levels, such as 0-9999, 10000-19999, 20000-50000, 50000-infinity)

record number	ID	gender	income_level
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Bitmap Indices (Cont.)

- Bitmap indices are useful for queries on **multiple attributes**
 - not particularly useful for single attribute queries
- Queries are answered using bitmap operations
 - Intersection (and)
 - Union (or)
 - Complementation (not)
 - **E.g.**, query for **males** with income **level L1**: $10010 \text{ AND } 10100 = 10000$
 - Can then retrieve required tuples.
 - Counting number of matching tuples is even faster



Index Definition in SQL

- Create an index:

create index <index-name> **on** <relation-name>(<attribute-list>)

E.g.: **create index** *b-index* **on** *branch(branch_name)*

- Use **create unique index** to indirectly specify and enforce the condition that the search key is a candidate key.
 - Not really required if SQL **unique** integrity constraint is supported
- To drop an index
drop index <index-name>
- Most database systems allow specification of type of index, and clustering.



End of Topic



KNOWLEDGE & SOFTWARE ENGINEERING