





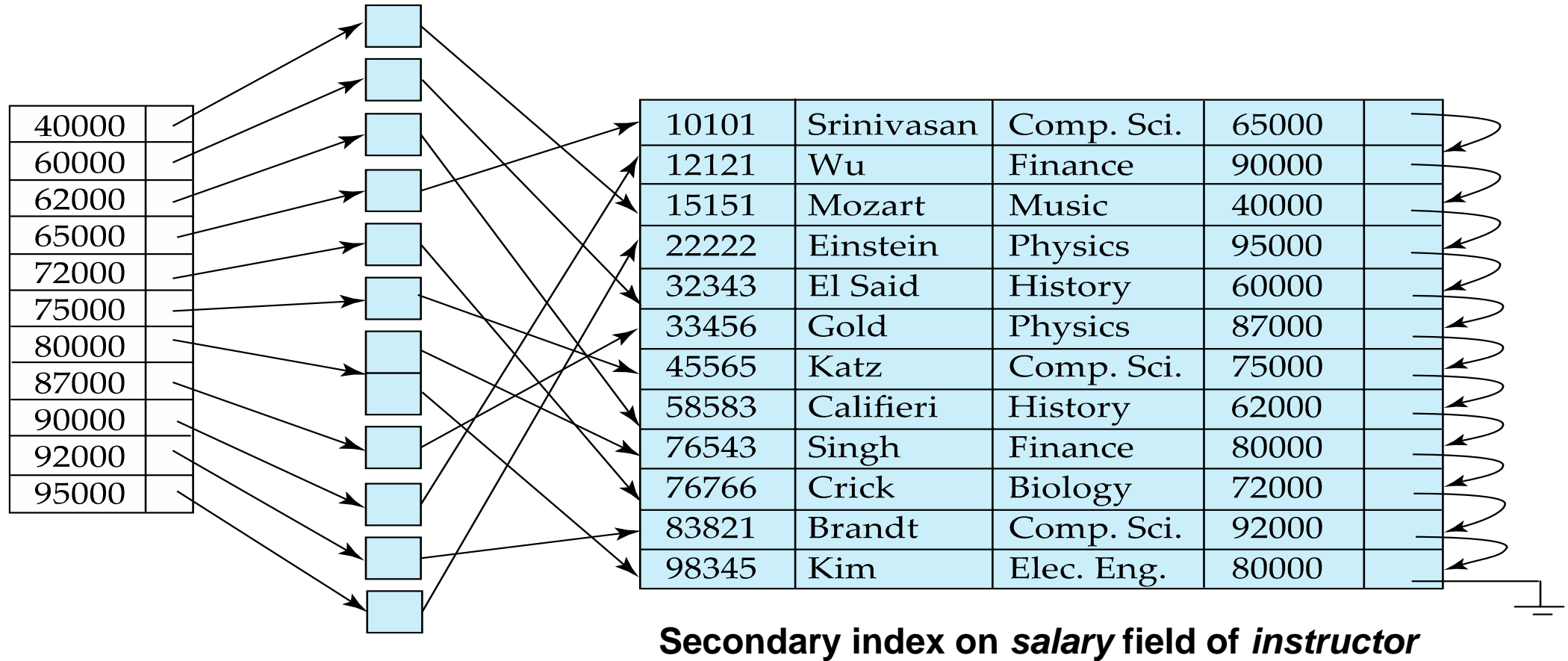
# CSE3103 : Database FALL 2020

Nazmus Sakib  
Assistant Professor  
Department of Computer Science and Engineering  
Ahsanullah University of Science and Technology

# Secondary Indices

- Frequently, one wants to find all the records whose values in a certain field (which is not the search-key of the primary index) satisfy some condition.
  - Example 1: In the *instructor* relation stored sequentially by ID, we may want to find all instructors in a particular department
  - Example 2: as above, but where we want to find all instructors with a specified salary or with salary in a specified range of values
- We can have a secondary index with an index record for each search-key value

# Secondary Indices Example



- 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.

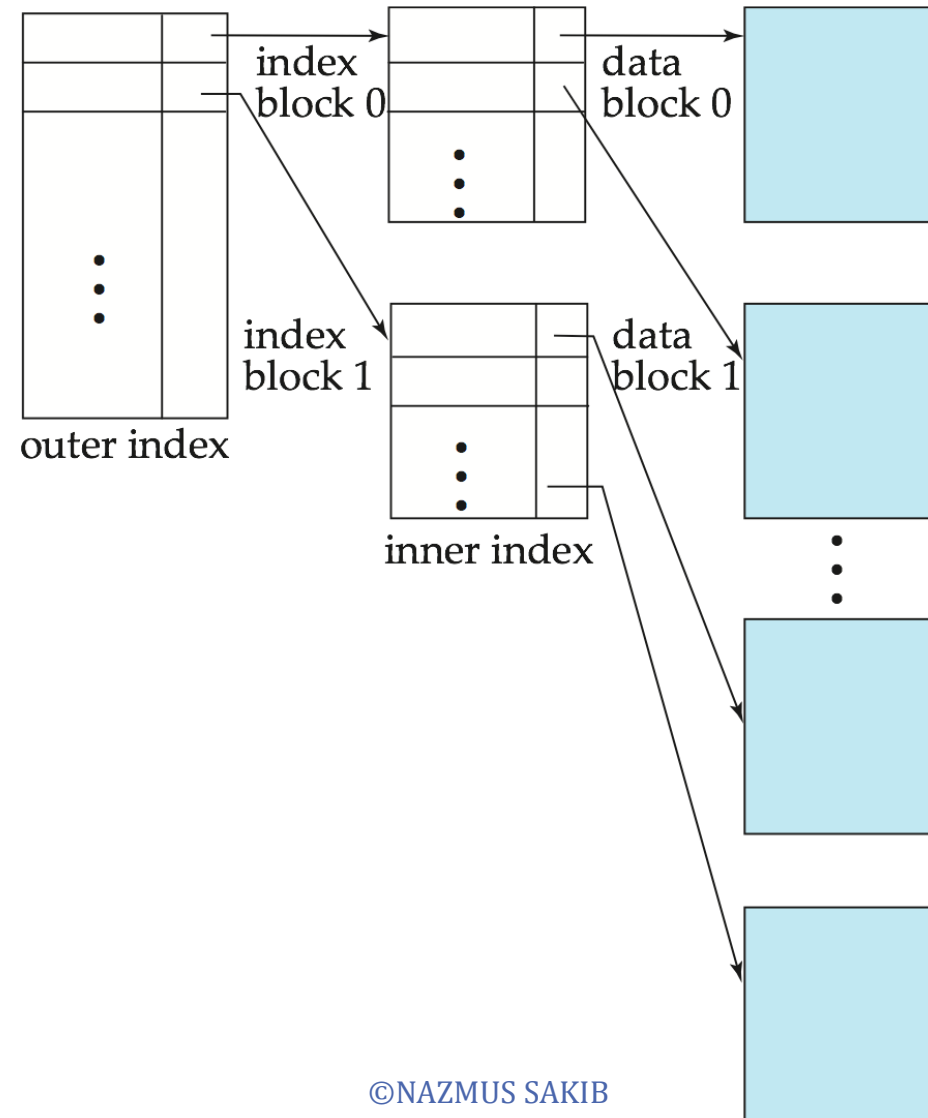
# Primary and Secondary Indices

- Indices offer substantial benefits when searching for records.
- BUT: Updating indices imposes overhead on database modification  
--when a file is modified, every index on the file must be updated,
- Sequential scan using primary index is efficient, but a sequential scan using a secondary index is expensive
  - Each record access may fetch a new block from disk
  - Block fetch requires about 5 to 10 milliseconds, versus about 100 nanoseconds for memory access

# 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
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.

# Multilevel Index (Cont.)



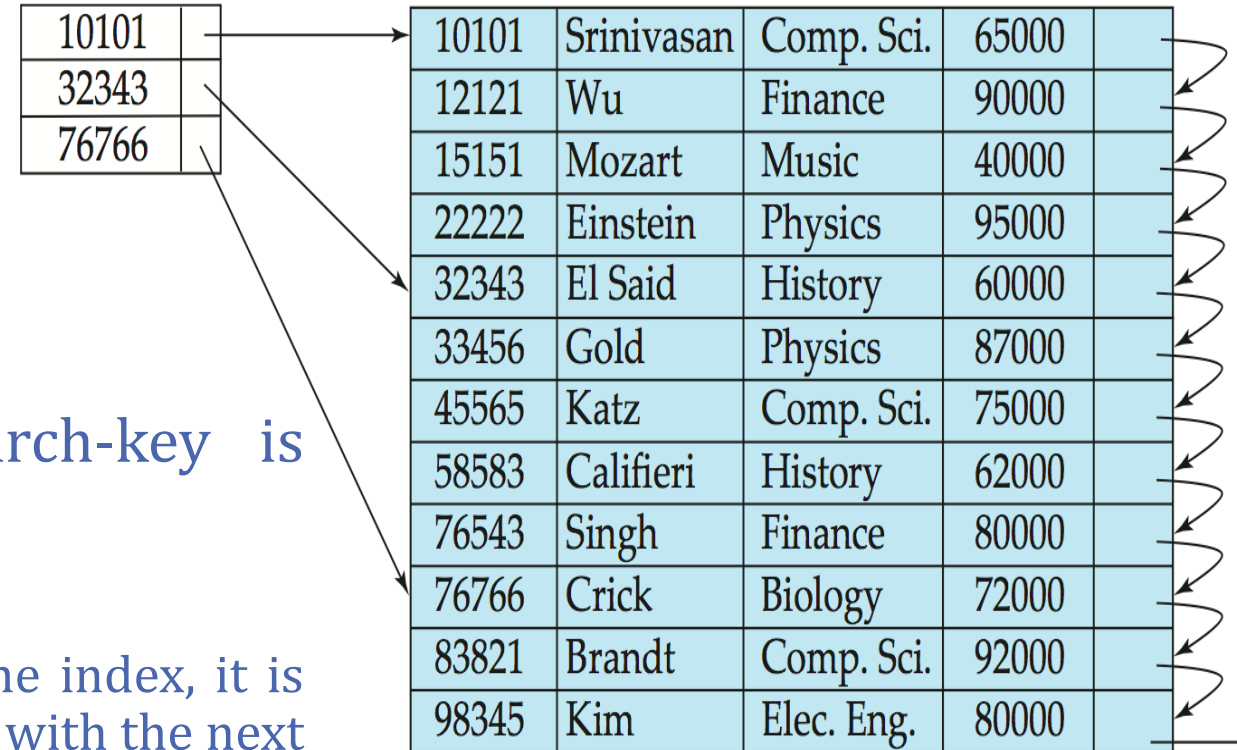


# Index Update: Deletion

- If deleted record was the only record in the file with its particular search-key value, the search-key is deleted from the index also.

- **Single-level index entry deletion:**

- **Dense indices** – deletion of search-key is similar to file record deletion.
- **Sparse indices** –
  - if an entry for the search key exists in the index, it is deleted by replacing the entry in the index with the next search-key value in the file (in search-key order).
  - If the next search-key value already has an index entry, the entry is deleted instead of being replaced.





# Index Update: Insertion

- **Single-level index insertion:**
  - Perform a lookup using the search-key value appearing in the record to be inserted.
  - **Dense indices** – if the search-key value does not appear in the index, insert it.
  - **Sparse indices** – if index stores an entry for each block of the file, no change needs to be made to the index unless a new block is created.
    - If a new block is created, the first search-key value appearing in the new block is inserted into the index.
- **Multilevel insertion and deletion:** algorithms are simple extensions of the single-level algorithms

