Chapter 12: Indexing and Hashing

- Basic Concepts
- Ordered Indices
- B+Tree Index Files
- Static Hashing
- Dynamic Hashing

Basic Concepts

- Many queries references small number of records in file. Eg “Find all accounts at Perryridge branch”.
- Inefficient to read each record and check value of branch-name.
- Indexing mechanisms used to speed up access to desired data.
  - E.g., index in book, author catalog in library
- Database system will look index to find on which disk block corresponding record resides and fetch the disk block to get the record

- **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).** Pointer has identifier of disk block and an offset within disk block to identify record within block.
- Index files are typically much smaller than the original file

Index Evaluation Metrics

- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order
  - **Hash indices:** search keys are distributed uniformly across “buckets” using hash function
- Factors on which techniques will be evaluated:
  - **Access types** supported efficiently. E.g.,

    finding particular attribute values or in range values
  - **Access time :** The time it takes to find a particular data item
  - **Insertion time :** Time taken for insert data
  - **Deletion time :** Time taken for delete data
  - **Space overhead :** : The additional space occupied by an index structure.

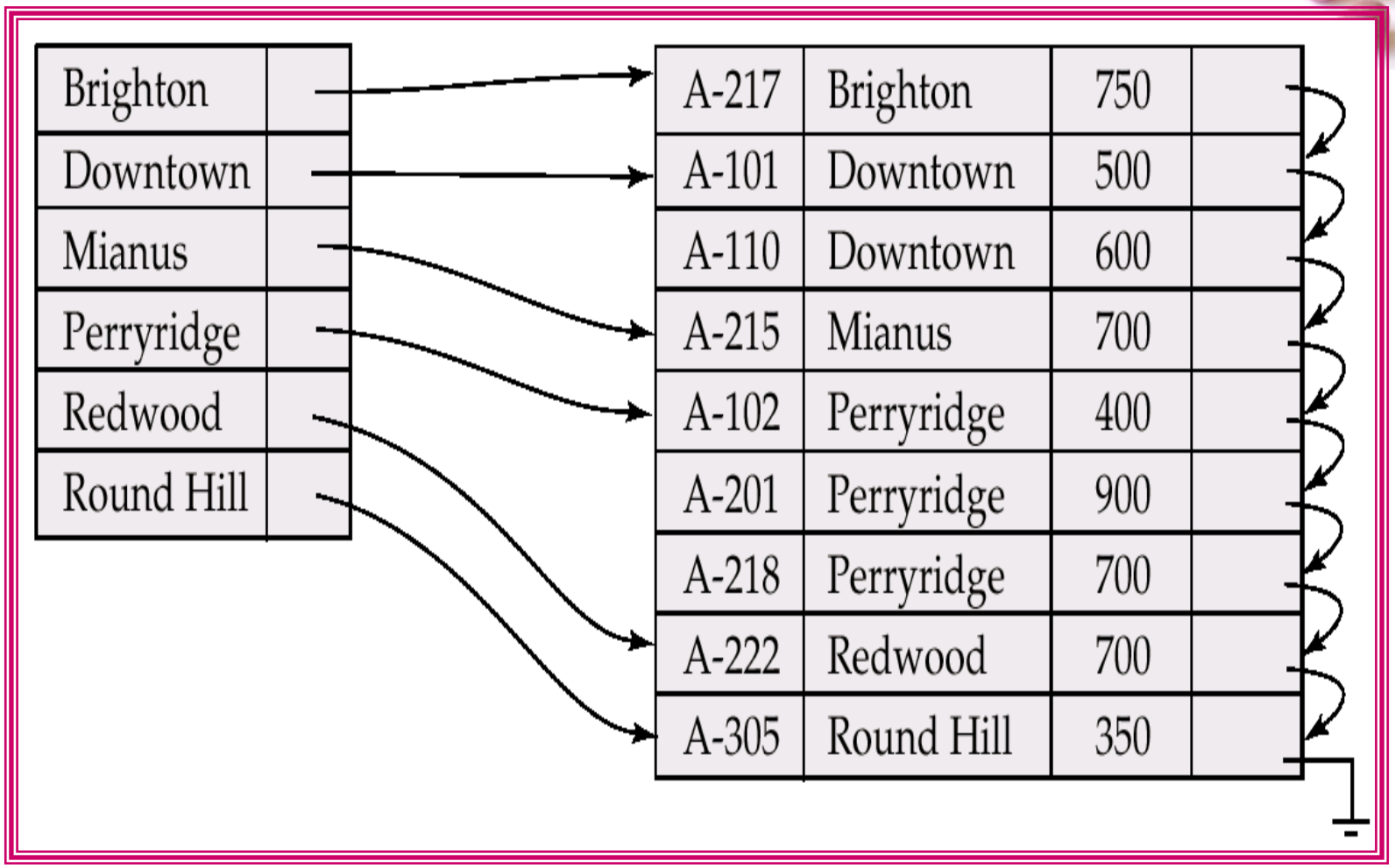
- We often want to have more than one index for a file.
- For example, libraries maintained several card catalogs: for author, for subject, and for title.
- An attribute or set of attributes used to look up records in a file is called a search key.

Ordered Indices

- To gain fast random access to records in a file, we can use an index structure
- Each index entry is associated with particular search key
- An **ordered index,** stores value of search key in sorted order and associates with each search key records that contain it.
- **Primary index:** in a sequentially ordered file, the index whose search key specifies the sequential order of the file.
  - Primary indix is also called **clustering index**
  - The search key of a primary index is usually but not necessarily the primary key.
- Index-sequential file: ordered sequential file with a primary index on search key.

- **Secondary index**: an index whose search key specifies an order different from the sequential order of the file. Also called non-clustering index**.**

Ordered Indices (Contd…)

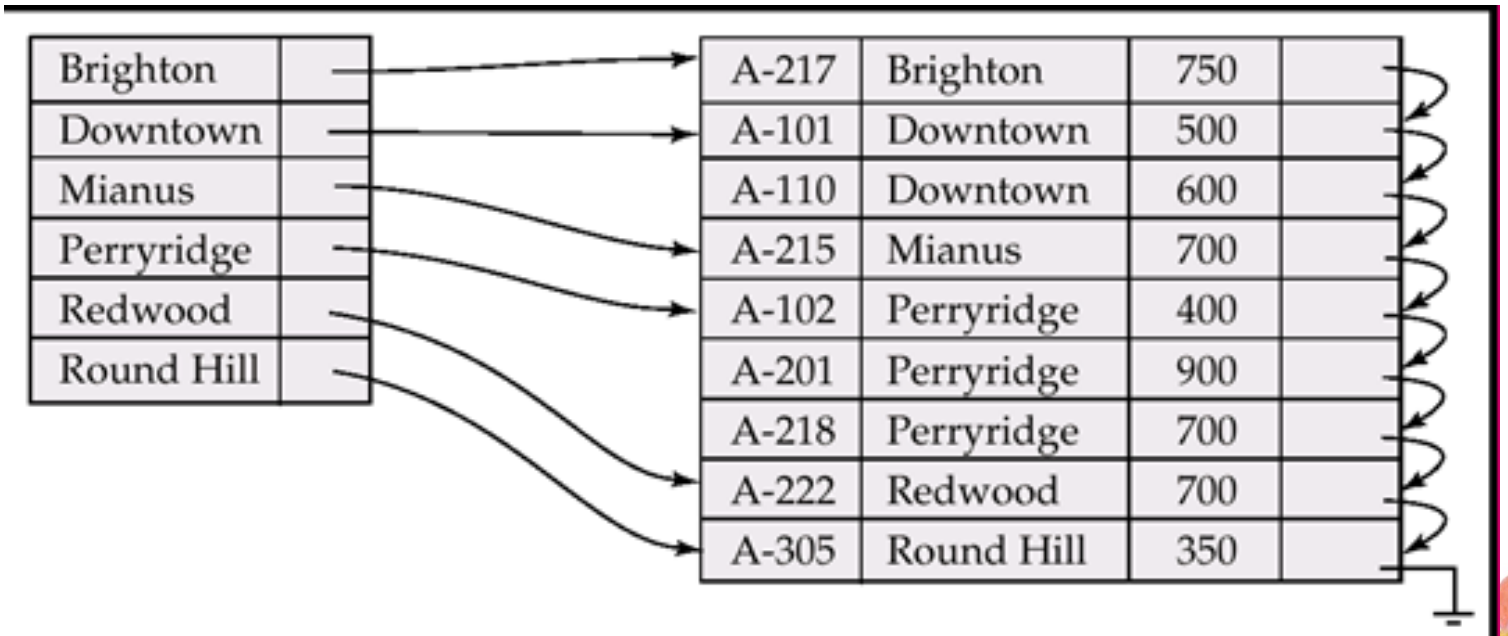


Primary index : , the records are stored in search-key order, with branch-name used as the search key.

Dense Index Files

- Dense index — Index record appears for every search-key value in the file.
- In dense primary index, records are sorted on same search key.
- Index record contains search key value and pointer to first data record with search key value
- Rest of records are stored sequentially
- .

- Suppose that we are looking up records for the Perryridge branch. Using the dense index of Figure we follow the pointer directly to the first Perryridge record.
- continue processing records until we encounter a record for a branch other than Perryridge

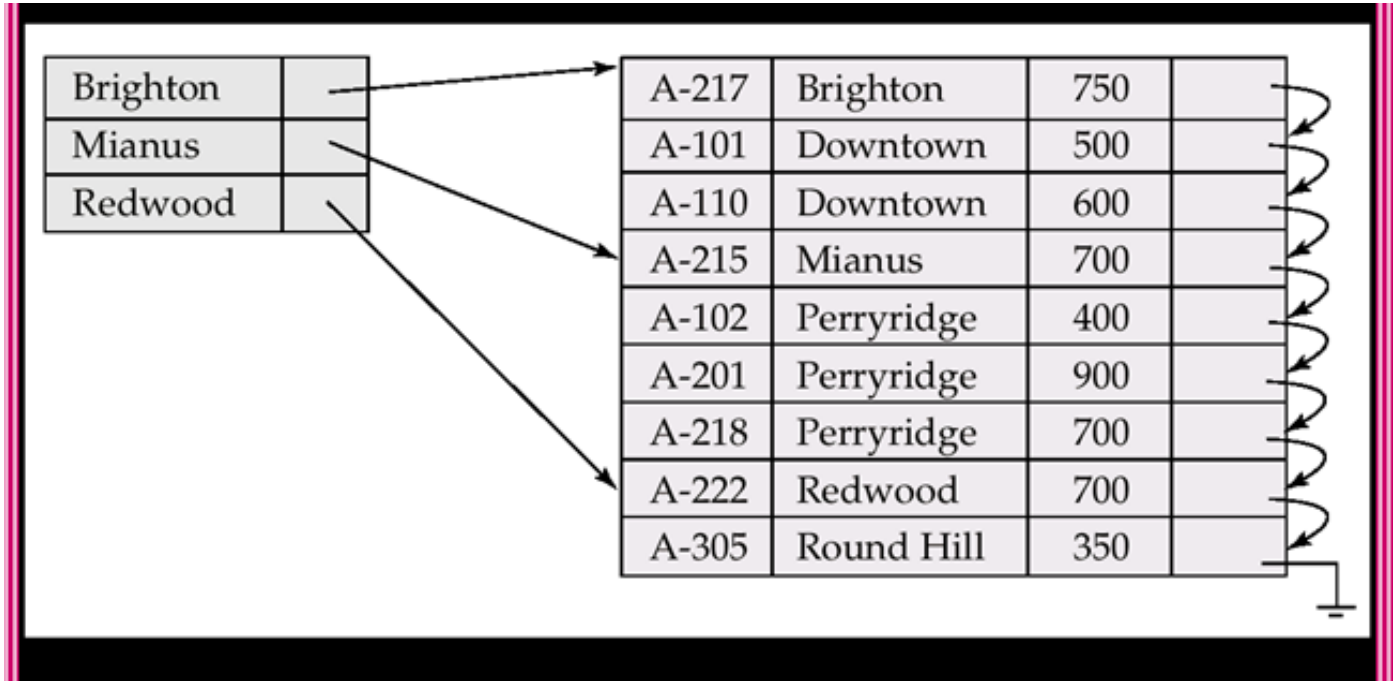


Sparse Index File

- Sparse Index: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate record, find index entry with largest search key value that is less than or equal to search key value we are looking and then search file sequentially.

- For e.g finding a entry for Perryridge branch in
- Given sparse index file.entry for perrryridge branch is not available in sparse index file.
- Since the last entry (in alphabetic order) before “Perryridge” is “Mianus,” , follow that pointer.
- Then read the account file in sequential order until we find the first Perryridge record

- Sparse index file for account table/file



Dense V/S Sparse Index

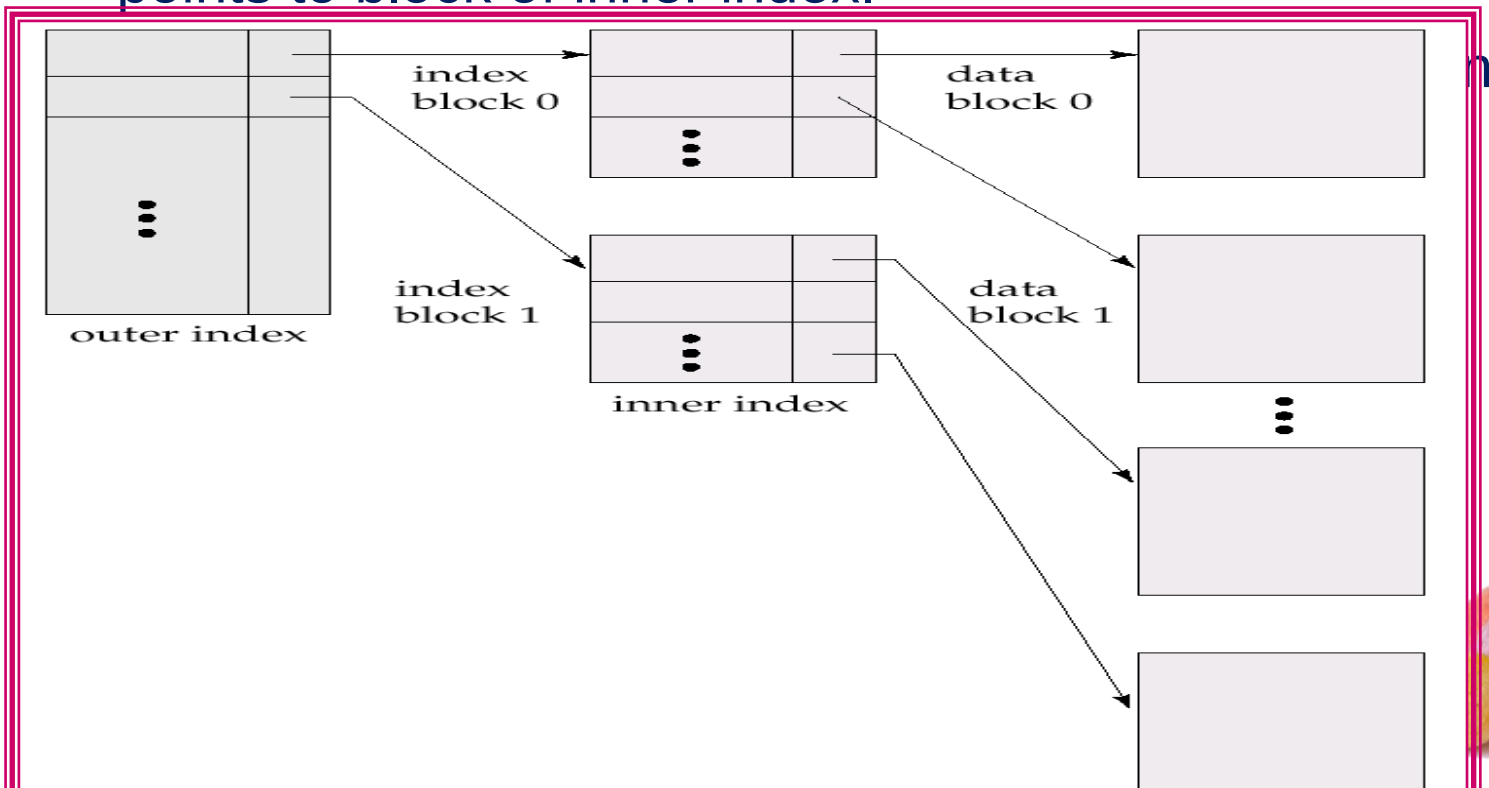
- Dense Index: Faster to locate records but occupy more space Sparse Index: Less space required but not faster
- System designer must make trade off between space and access time
- Good tradeoff: sparse index with an index entry for every block in file. We can locate block containing record that we are looking
- Cost to process a database request is time to bring block from disk to memory.
- Once block is in memory, time required to scan entire block is negligible.

Multilevel Index

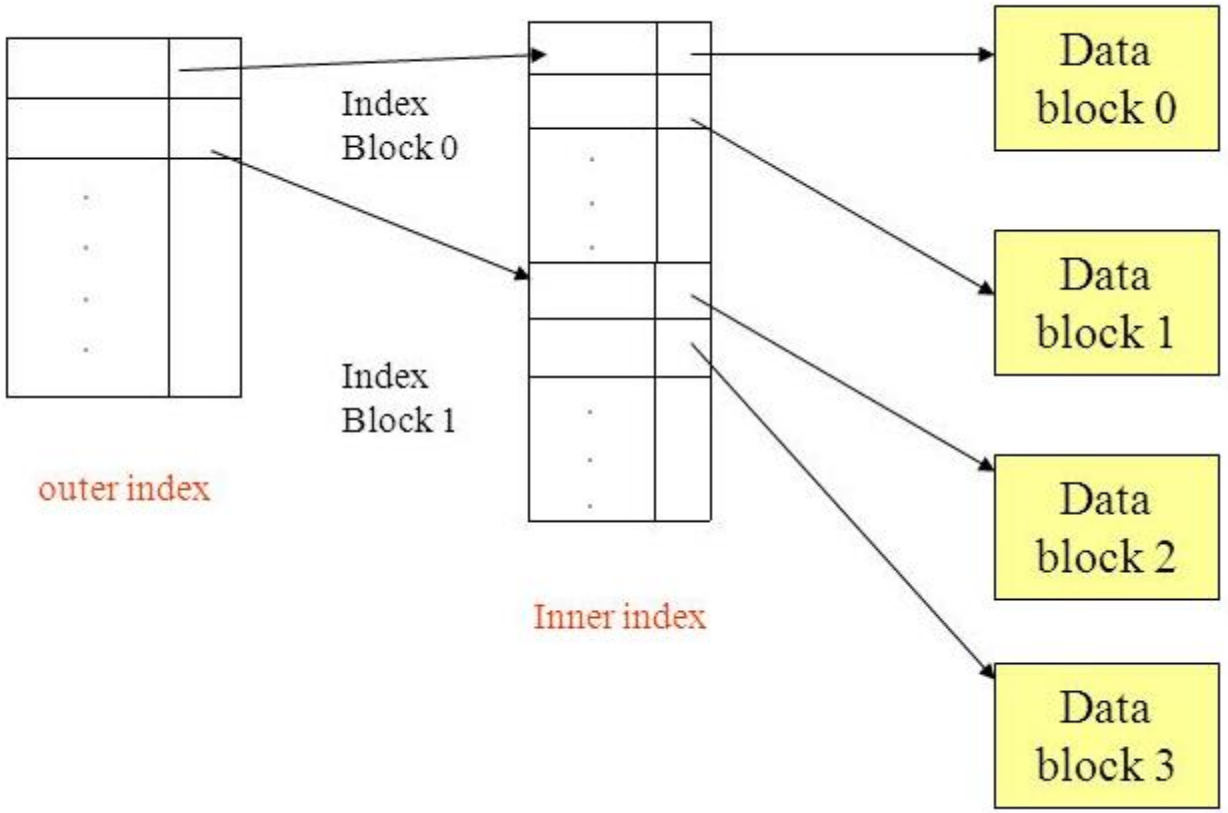
- Suppose file has 100,000 records and 10 records in each block ie 10,000 blocks.
- If one index record per block then index will have 10,000 records.
- Assume that 100 index records fit in 1 block then index file itself will need 100 blocks. Such large index files must be stored on disk.
- To locate a record, binary search is used on index file to locate entry but still it has large cost.
  - If index occupies b blocks, binary search requires $\log_2(b)$ blocks to read
  - E.g if 100 index blocks, binary search will require 7 blocks read will be required to read single record, which is very time consuming

Multilevel Index

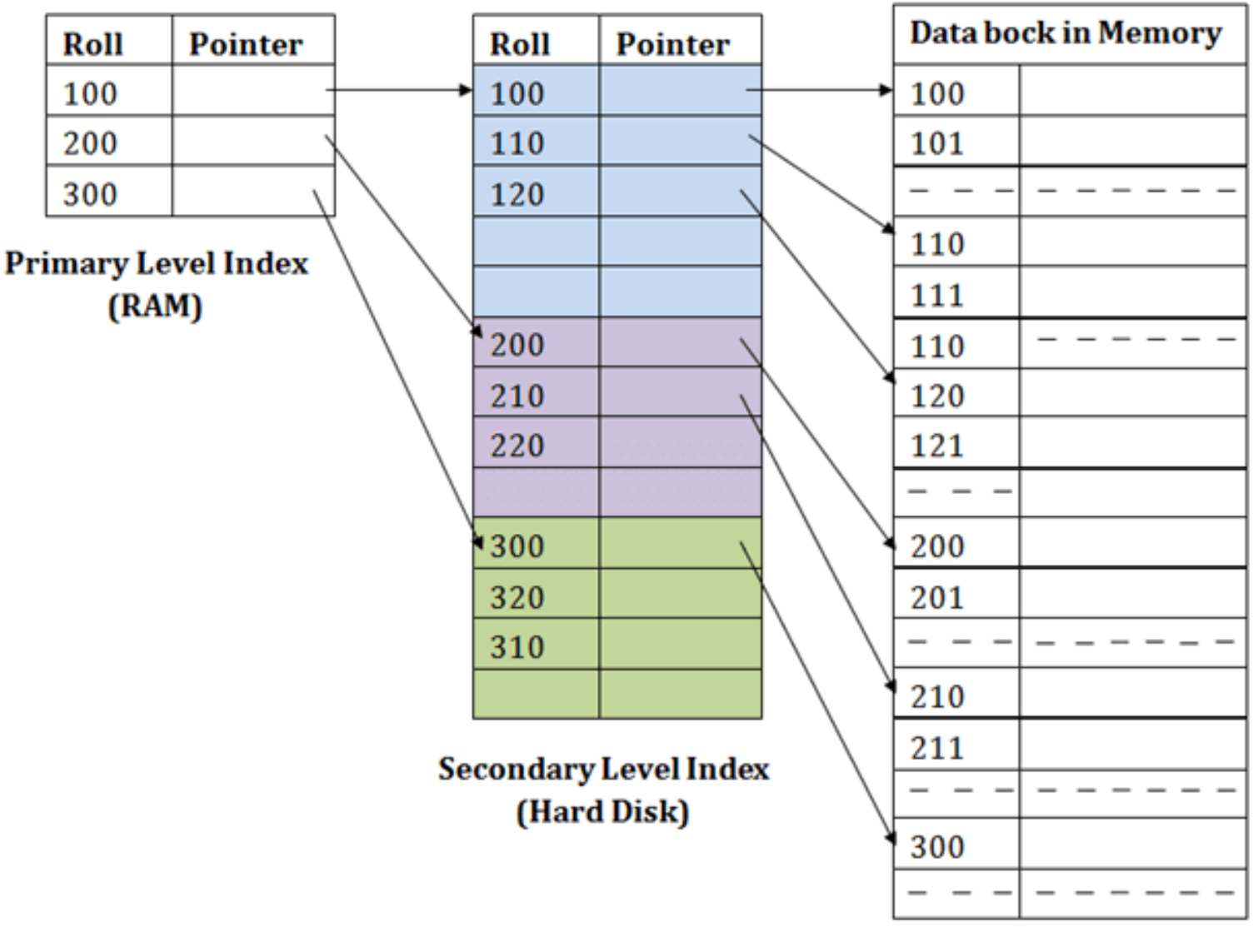
- To deal with this problem,
- Treat index file as any other sequential file and construct sparse index on primary index.
- To locate record, first use binary search on outer index to find record for largest search key value. Pointer points to block of inner index.

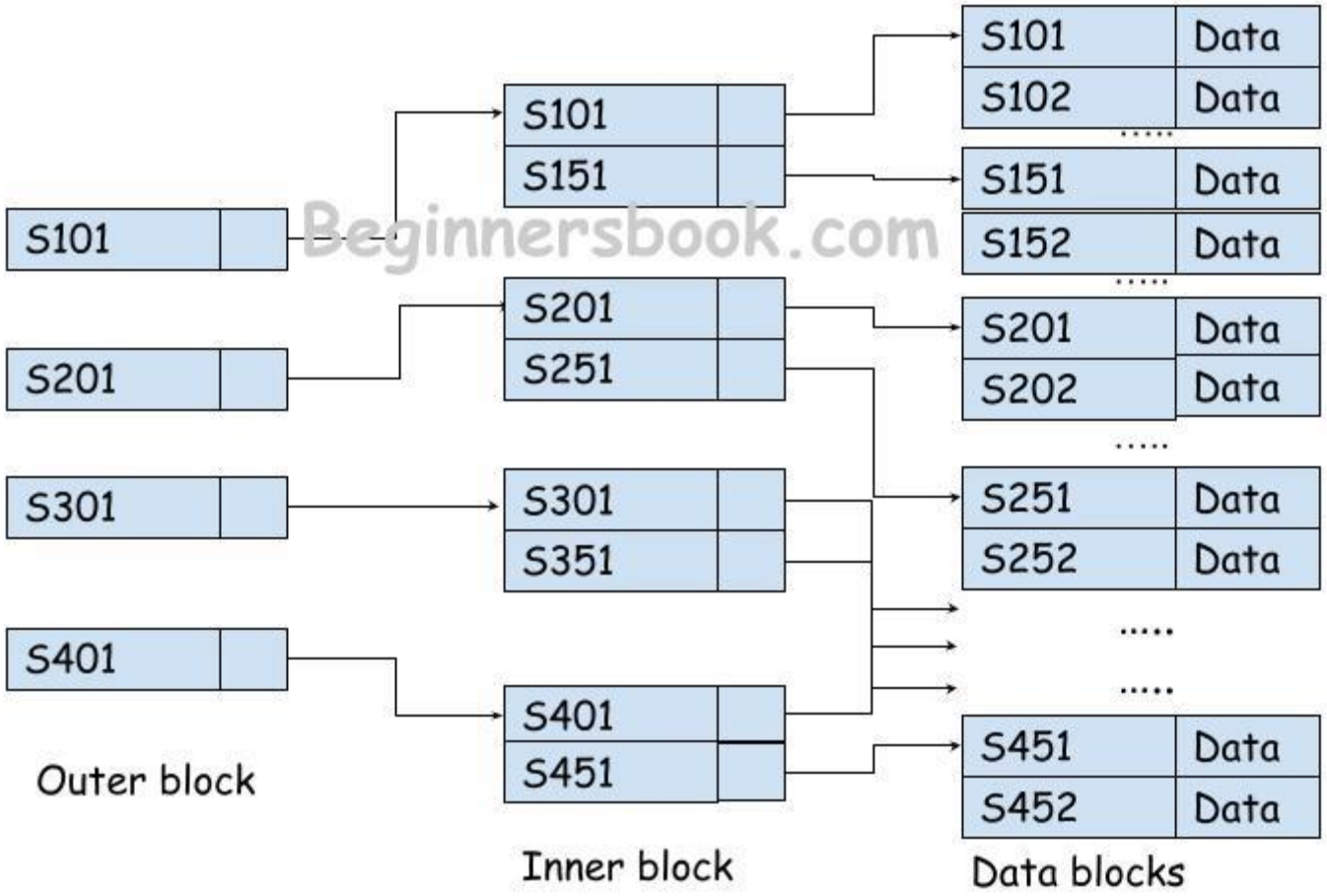


Multilevel index cont



Multilevel index




S101
S201
S301
S401
Outer block
S101
S151
S201
S251
S301
S351
S401
S451
Inner block
S101 Data
S102 Data
S151 Data
S152 Data
S201 Data
S202 Data
S251 Data
S252 Data
S451 Data
S452 Data
Data blocks
Beginnersbook.com

Multilevel Index

- If our file is extremely large even though outer index may grow too large to fit in memory we can create another level of index.
- Indices with two or more levels are called as multi level indices.
- Searching of record with multilevel index requires significantly fewer IO operations.