MC 302 – DBMS: Indexing

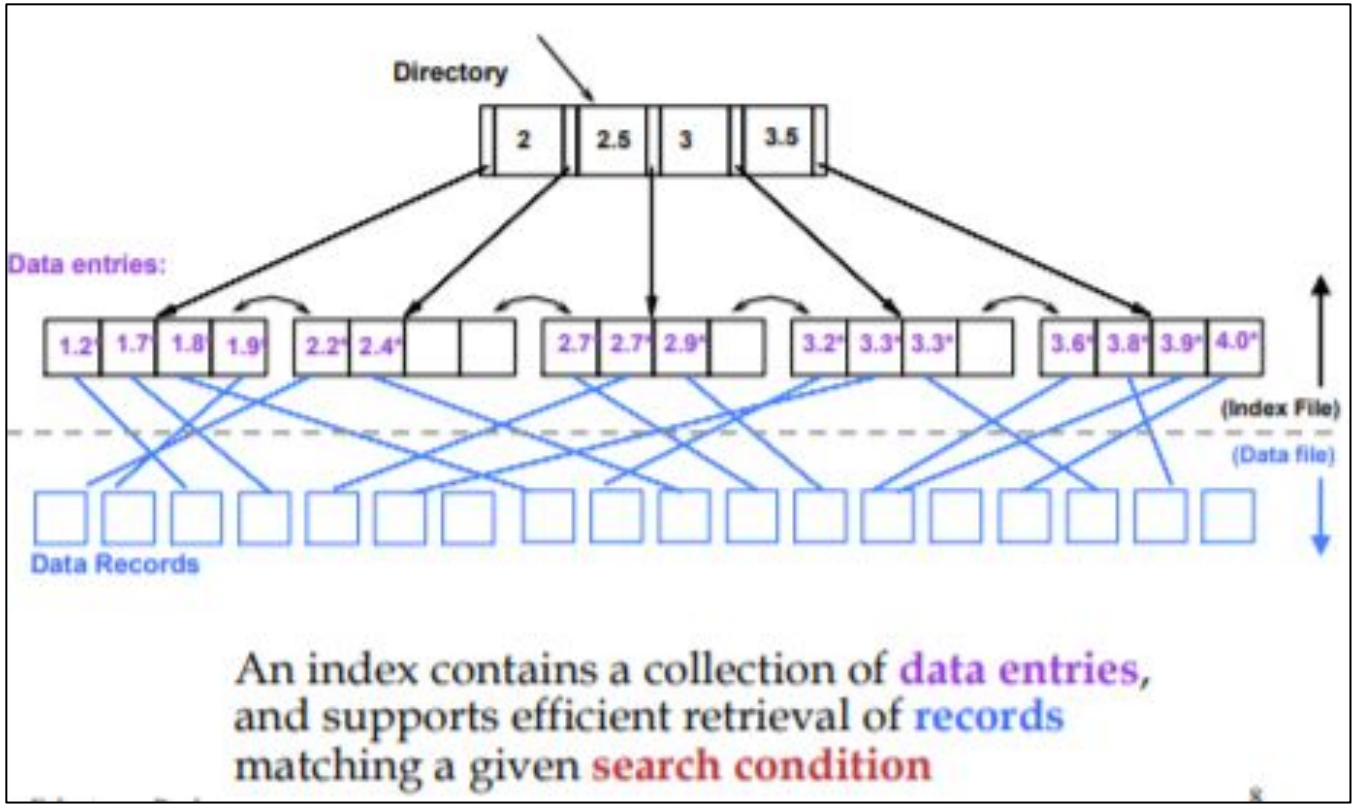
Goonjan Jain
Department of Applied Mathematics
Delhi Technological University

How to find records quickly?

- Use indexes.
- index contains auxiliary info that directs searches to the desired data entries
- Alternative way of accessing the records without affecting physical placement of records
- Search Key – an attribute or a set of attributes used to look up records in a file
- Can have multiple (different) indexes per file. – E.g. file sorted on age, with a hash index on name and a B+tree index on salary

Indexes

- ‘data entries’ == what we store at the bottom of the index pages
- what would you use as data entries?
- (3 alternatives here)



Alternatives for Data Entry **k** in Index

1. Actual data record (with key value k)

126	Aman	New Delhi	110042	999-888-777-0

2. <k, relation_id of matching data record>

1000 INR	Relation_id 1
1000 INR	Relation_id 2

3. <k, list of relation_ids of matching data records>

1000 INR	Relation_id 1	Relation_id 2	Relation_id 3

Dense Index vs Sparse Index

- Dense Index – an entry for each search key value
- Sparse Index – index entry for some of the values

Types of Indexes

- Ordered Indices –
  - Based on sorted ordering of values
- Hash Indices-
  - Based on uniform distribution of values across a range of buckets
  - Values are assigned to buckets
  - The bucket is determined by hash function
- Factors-
  - Access Type
  - Access Time
  - Insertion Time
  - Deletion Time
  - Space overhead

Types of ordered Indices

- Single Level Order Indexes
- Multilevel Indexes

Single Level Ordered Indexes

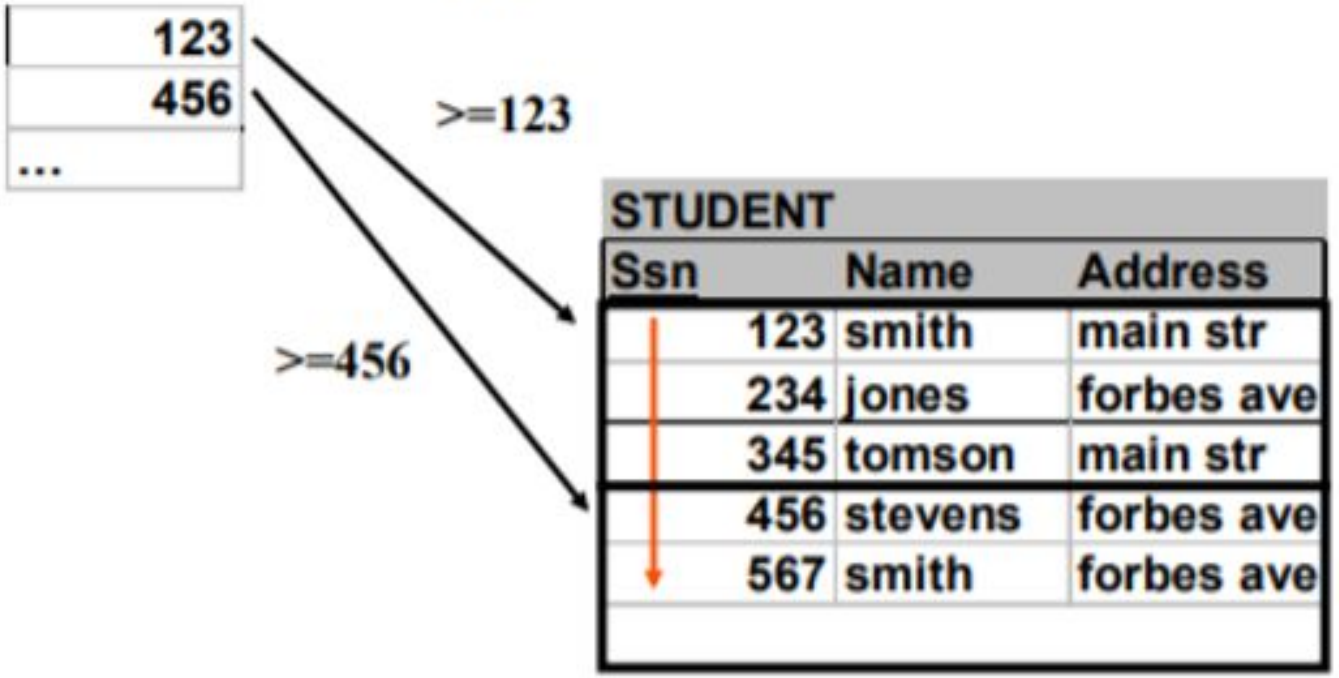
- Values in index are sorted so that can do a binary search
- Types of Ordered Indexes
  - Primary Index - index key includes the file's primary key
  - Clustering Index
  - Secondary Index – provides secondary access where primary already exists

Primary Index

- Specified on the ordering key field of a ordered file of records
- Is a sparse index
- Ordering Key Field-
  - Used to physically arrange/order the file records on the disk
  - Every record has unique value for the field
- PI is an ordered file
  - Records with 2 fields-
    - Same data type as ordering key field – called the PK of data file
    - Pointer to a disk block
  - One index entry per block of data file
  - First record of each block of data file is called anchor record of the block
  - Record whose PK value is k, lies in block P(i)

$$k(i) < k < k(i+1)$$

Primary Index



Primary Index

- Advantages – index file needs fewer blocks than data file
  - Fewer index entries
  - Small sized index entries
- Disadvantages –
  - To insert a record at its correct position
    - Move records to make space for new record
    - Anchor records of blocks might have changed, change index entries
  - To delete a record
    - Move records to fill space of the deleted record
    - Anchor records of blocks might have changed, change index entries

Primary Index - Numerical

- Ordered file with
  - r = 30,000 records
  - Block size, B = 1024 bytes
  - File Records size, R = 100 bytes
  - Blocking factor (bfr) for the file = B/R = 1024/100 = 10 records per block
  - Number of blocks needed for file, b = r/bfr = 3,000 blocks
  - Binary Search needs $\log_2 b = 12$ block accesses

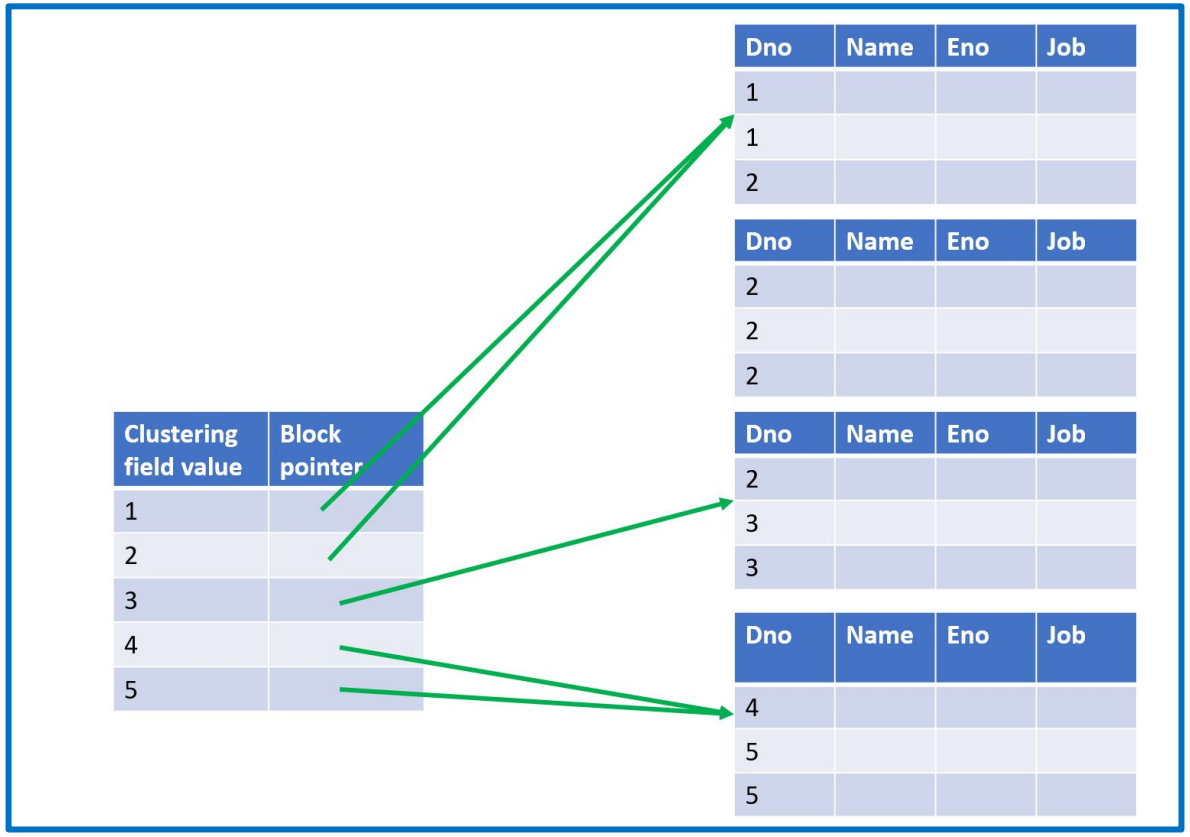
Numerical contd…

- Ordering field value size, V = 9 bytes
- Block pointer, P = 6 bytes
- Size of each index entry R_i = 9 + 6 = 15 bytes
- Blocking factor (bfr_i) for index = B/R_i = 68 entries per block
- Total number of index entries r_i = no. of blocks in data file = 3,000
- The number of index blocks, $b_i = r_i/bfr_i$ = 45 blocks
- Binary Search needs $\log_2 b_i$ = 6 block accesses
- Search a record using index = 6 + 1 = 7 block accesses

Clustering Index

- Records are physically ordered on Clustering Field
  - A non-key field
  - Field does not have distinct value for each record
- Clustering index
  - An ordered file with 2 fields -
    - Same data type as clustering field
    - Block pointer
  - One entry for each distinct value
  - Block pointer points to the first block in data file with a record with that value

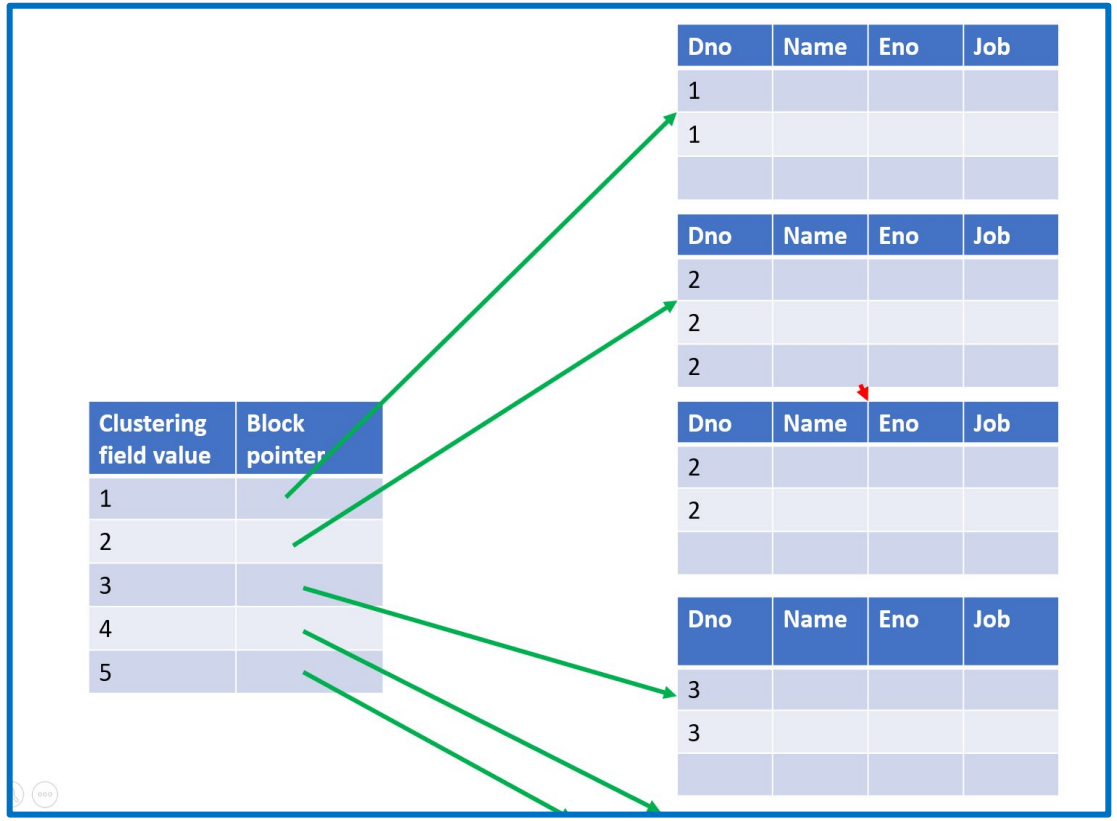
Clustering Index



Clustering Index

- Is a sparse index
- Record insertion and deletion is still a problem
- To alleviate the problem –
  - Reserve whole block for each value of clustering index

Clustering Index



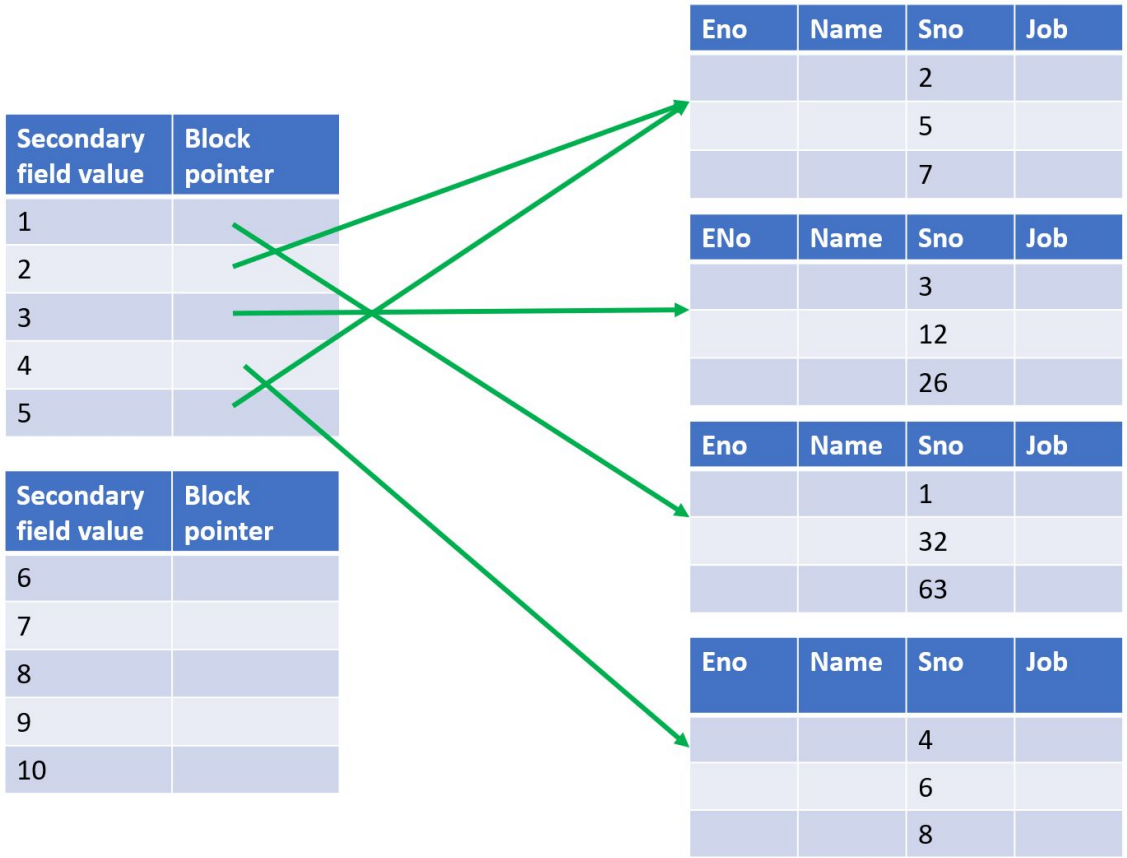
Secondary Index

- Index can be on
  - A candidate key
  - Non – key field with duplicate values
- Index entries –
  - Data type of ordering field
  - Either block pointer or record pointer
- Can be many secondary indexes

Secondary Index – on Key Field

- Field also called secondary key
- 1 index entry for each record
- Index is dense
- Data records are not ordered, cannot use block anchors
- Once the appropriate block is transferred to main memory, search the desired record
- Needs more storage space than a primary index

Secondary Index – on Key Field



Secondary Index - Numerical

- File with r = 30,000 records
- Record size, R = 100 bytes
- Block size, B = 1024 bytes
- File has r/B blocks, b = 3000 blocks
- For linear search, number of block accesses = b/2 = 1500 block accesses

Secondary Index - Numerical

- Non-ordering key field size, V = 9 bytes
- Block pointer, P = 6 bytes
- Each index entry R_i = 9 + 6 = 15 bytes
- Blocking factor bfr_i for index = B/R_i = 1024/15 = 68 entries per block
- Total no. of index entries r_i = no. of records = 30,000
- No. of blocks for index files $b_i = r_i/bfr_i$ = 442 blocks
- Binary search on secondary index = $\log_2 b_i$ = 9 block accesses
- Total number of block accesses = 9+1 =10

Secondary Index- Non Key field

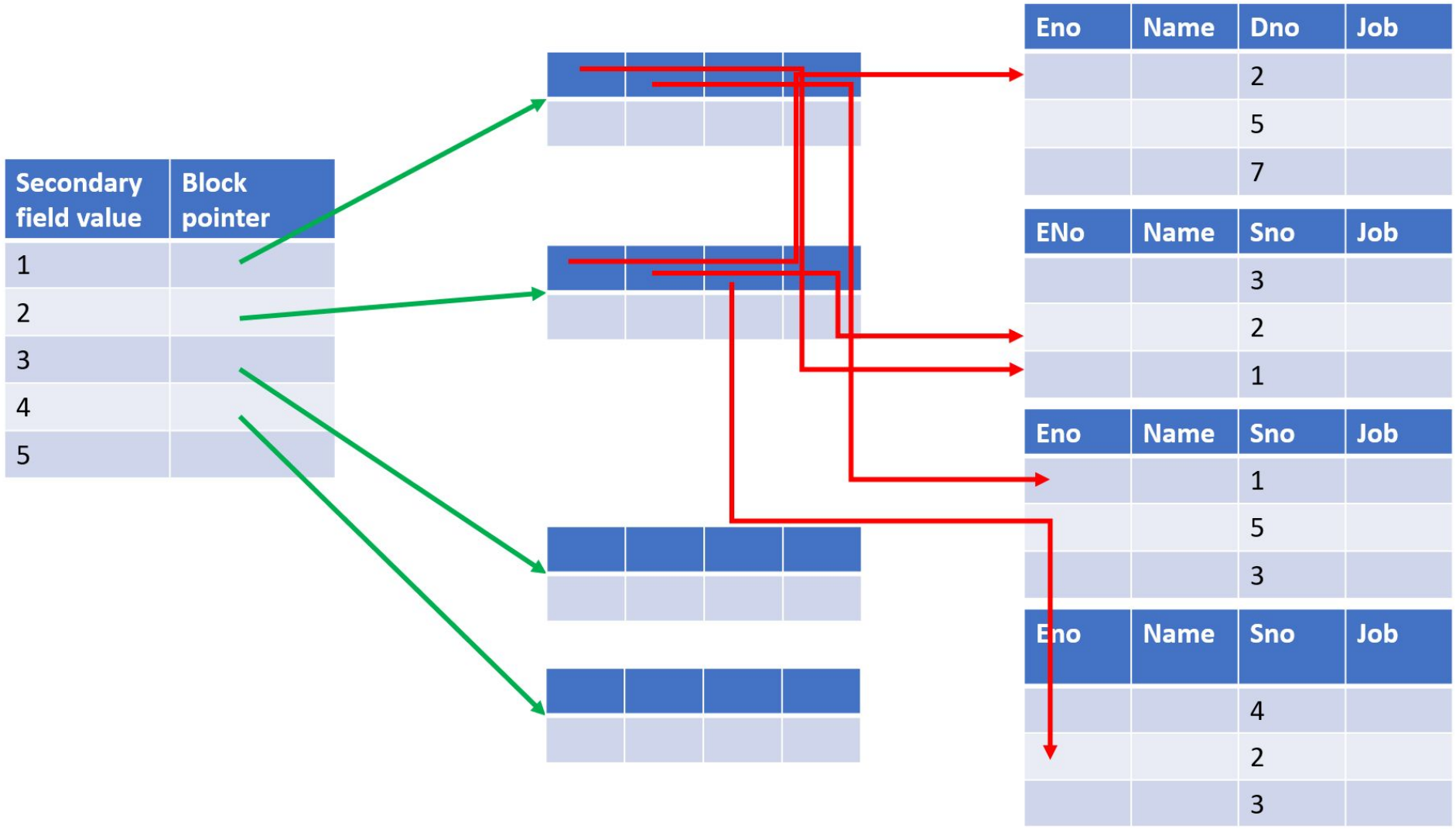
- 3 implementation options:
  - Several entries with same value – one for each record
  - Variable length records for index entry

    <p(i,1), p(i,2), p(i,3), …, p(i,k)>

    one pointer for each block containing record
  - Create extra level of indirection

Secondary Index - indirection

- Non-dense scheme
- Pointer in index points to a disk block.
- Disk block contains record pointers.
- Record pointers point to the data file record
- In case of overflow of record pointers from a disk block, linked list of blocks is used


Secondary field value
Block pointer
1
2
3
4
5
Eno
Name
Dno
Job
2
5
7
ENo
Name
Sno
Job
3
2
1
Eno
Name
Sno
Job
1
5
3
Eno
Name
Sno
Job
4
2
3

Multilevel Indexes

- Motivation-
  - Reduce the part of index
- Search space is reduced faster
- Bfr_i is called fanout (f_o) of multilevel index
- Fanout depends on how many index entries fit within a block
- Searching requires approx. $\log_{fo} b_i$ block accesses
- Suppose,
  - block size, B = 4096
  - Block pointer, P = 4 bytes
  - Key field value size, V = 9 bytes
  - Fanout, $f_o = B/(P+V) = 4096/13 = 315$

Multilevel Index – index creation

- Create a primary index for all levels
  - Ordered file with distinct values k(i)
- Fanout for all levels is same
- Needs another level if the previous levels needs more than 1 block of disk
- Number of entries in $i+1^{th}$ level is given by

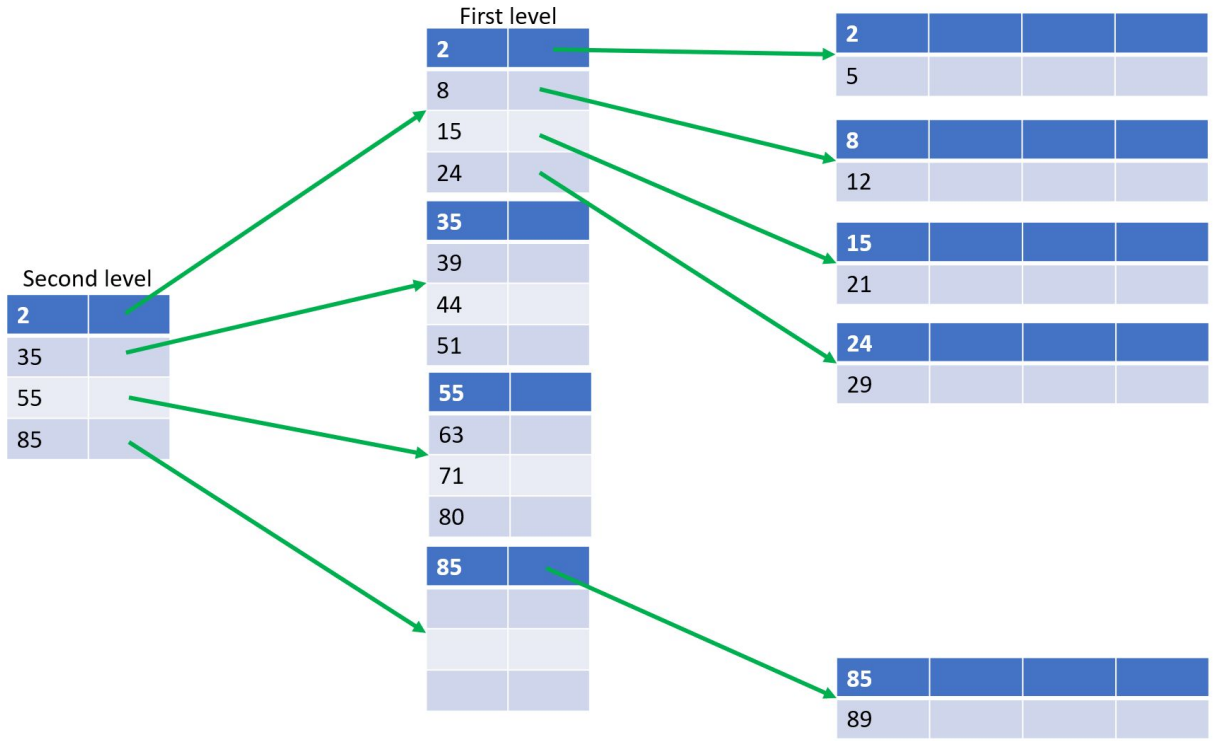
$$r_{i+1} = \lceil r_i / f_o \rceil$$

- The block at the t^{th} level – top index level
- A multilevel index with r_i first level index entries will have approx. t levels

$$t = \lceil \log_{fo} r_i \rceil$$

- Can be used on any index – primary, clustering, or secondary, given
  - Fixed length entries
  - First level index has distinct values

Multilevel Index



Multilevel Index on secondary index(key)- Numerical

- Number of records, r = 30,000
- Record size, R = 100 bytes
- Block size, B = 1024 bytes
- Number of blocks to hold the file, b = r*R/B = 3000 blocks
- Key field value size, V = 9 bytes
- Block pointer, P = 6 bytes
- Each index entry = P+V = 15 bytes
- Blocking factor of index, bfr_i = B/(P+V) = 68 entries/block = f_o

Multilevel Index-Numerical

- Number of first level blocks, $b_1 = \lceil b/f_o \rceil = 30000/68 = 442$ blocks
- Number of second level blocks, $b_2 = \lceil b_1/f_o \rceil = 442/68 = 7$ blocks
- Number of third level blocks, $b_3 = \lceil b_2/f_o \rceil = 7/68 = 1$ block

- To access a record,
  - Access 1 block at each level
  - Access 1 block from data file
- Total number of block accesses = 3+1 = 4

Dynamic Multilevel Indexes using B-Trees & B^+-Trees

- Search tree of order(ρ) p
  - each node contains at most p-1 search values
  - p pointers

$$<p_1, k_1, p_2, k_2, \ldots, p_{q-1}, k_{q-1}, p_q>$$

  - All search values are assumed to be unique
- 2 constraints:
  - Within each node k1 < k2 < k3 < k4 <… < k_{q-1}
  - For all values of x in the subtree pointed at by P_i , $K_{i-1} < x < K_i$ for 1 < I < q.

Dynamic Multilevel Indexes

- Values in tree can be the values of one of the fields of the file – **search field**
- Search tree can be stored on disk by assigning each tree node to a disk block.
- Goal of balanced search tree:

  "To make the search speed uniform - average time to find any random key is roughly the same."
- We want nodes to be as full as possible and delete the empty nodes.

B-Tree of order p

- Each internal node is of the form

  $<P_1, <K_1, Pr_1>, P_2, <K_2, Pr_2>, \ldots, <K_{q-1}, Pr_{q-1}>, P_q>$

P_i = tree pointer – pointing to another node in the tree

Pr_i = data pointer – pointing to the record of the search key field K_i

$K_1 < K_2 < K_3 < \ldots < K_{q-1}$

- Each node has at most p tree pointers
- Each node, except root and leaf nodes has atleast $\lceil p/2 \rceil$ tree pointers.
- All leaf nodes are at the same level. All the pointers of leaf nodes are NULL.

B Tree – Example

- Insertion – 8 5 1 7 3 12 9 6
- Order(ρ) of node is 3

B Tree – Example

- Deletion – 5 12 9
- Order(ρ) of node is 3

B+ Tree

- Data pointers are stored only at the leaf nodes of the tree.
- Pointers in internal nodes (tree pointers) points to other tree nodes.
- Pointers in leaf nodes (data pointers) points to data file records.
- Every leaf node contains a pointer (P_{next}) that points to the next leaf node.
- Structure of leaf nodes differ from structure of internal nodes.
- Leaf nodes have an entry for every value of the search field, along with a data pointer to the record – if search field is a key
- For a non-key search field, pointer points to a block containing pointers to the data file blocks, creating an extra level of indirection.

B+ Tree - Example

- Insertion – 8 5 1 7 3 12 9 6
- Order(ρ) of internal node is 3 and order(ρ) of leaf node is 2

B+ Tree – Example

- Deletion – 5 12 9