CPSC 332: File Structures and Database Systems

Basics of FD and Normalization for Relational Databases

Week 13, Lecture 02

Kanika Sood

Slide: Image Courtesy: Fundamentals of Database Systems, By Navathe [Pearson]
Reference Textbook: Elmasri, Ramez, and Sham Navathe. Fundamentals of database systems. Pearson, 2017.

Outline

- Single-level Indexes
  - Primary
  - Clustering
  - Secondary
- Multilevel indexes
- Trees
- Dynamix Indexes using B Trees

Chapter 17

Why indexing is relevant?

Single-level Ordered Indexes

- **Ordered Index**
  - ~ Index in textbook
  - Ordered index: file
- **Indexing field**
  - Single Attribute
  - **Index Attribute**
  - Index: stores each values of index field + list of pointers -> all disk blocks
- Index values: **ordered**
  - Ordering key field
  - Binary Search
  - Ordered

Types of Ordered Indexes

1. Primary Index
   a. Index -> Ordering key field -> ordered file
   b. Ordering key field: Unique => all records
2. Clustering Index
   a. Ordering field: ❌ Key field
   b. Numerous records => same value
   c. Use: Clustering Index
   d. Data File: Clustered File

File: Max. 1 ordering field: Either 1. Or 2.

3. Secondary Index
   – On any non-ordering field

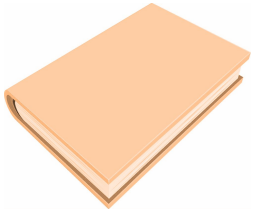
Data file => several secondary indexes

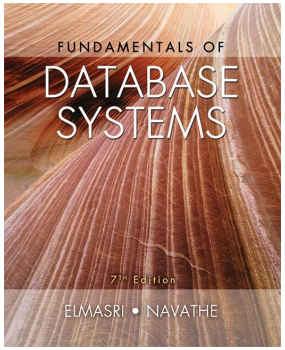
Primary Index

- Ordered file with two fields
  - Primary key: PK
  - Pointer to a disk block
- 1 index entry in index file => each block in data file
- Index entry: PK value for 1st record
- Indexes => dense or sparse
- **Dense index**
  - Index entry for every record in file
- **Sparse index**
  - Entries for some search values
- # entries: # disk blocks in ordered data file

Index File (IF)

- 1000 page book
  - 2000 page index
  - Index Advantageous ?
  - Direct access file from memory
- Index properties?
  - Short





Fundamentals of Database Systems, 4th Edition

By Ramez Elmasri and Shamkant Navathe

Table of Contents

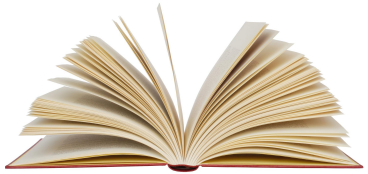
A. Short Table of Contents

(This Includes part and chapter titles only)

Search time $\propto$ Memory size

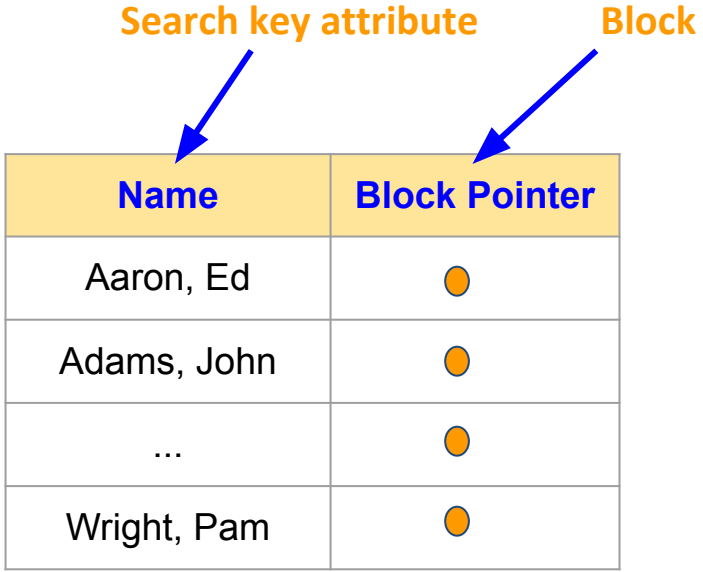
An index is relevant only if it is short

- Always sorted
- Else: still searching



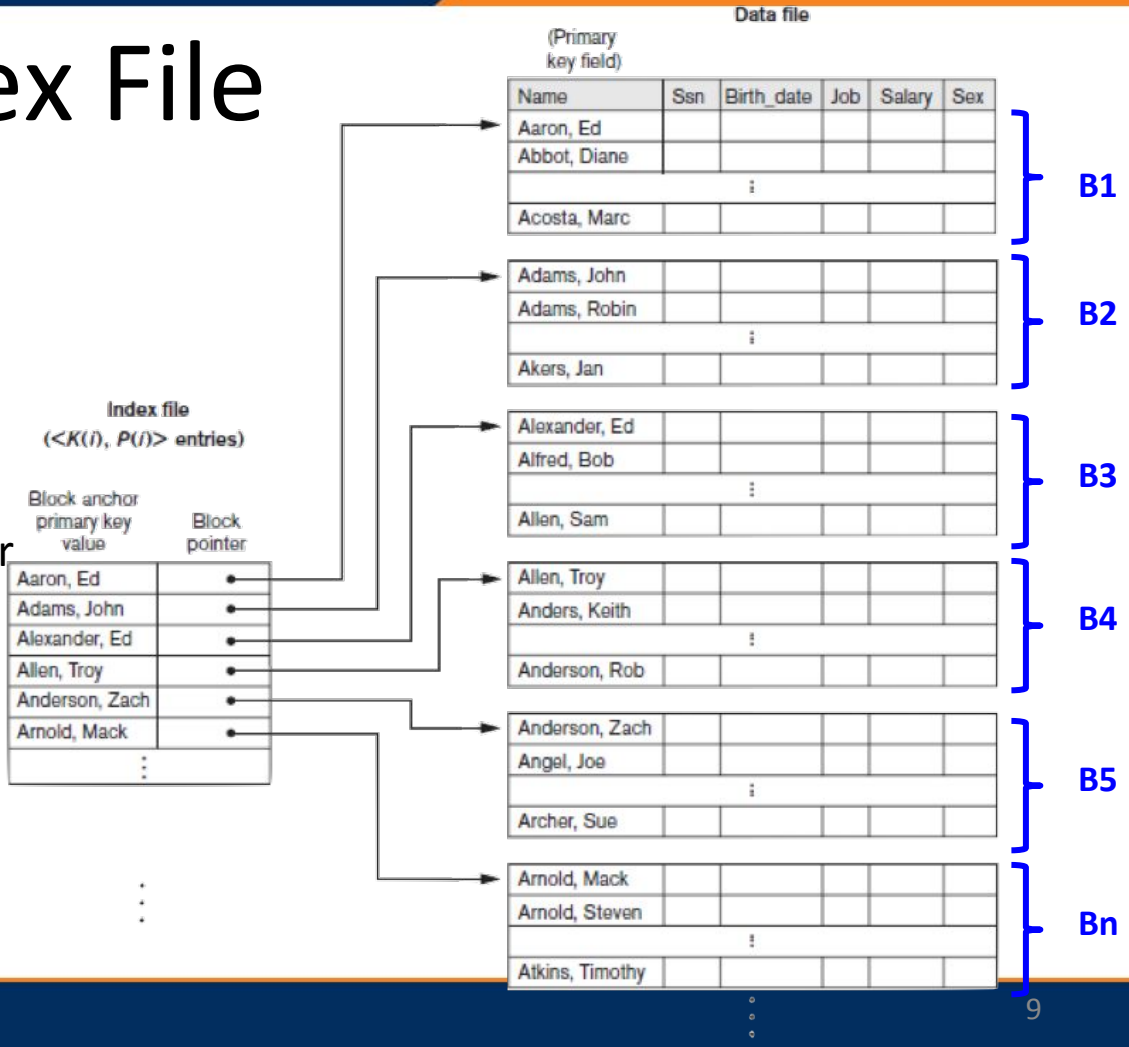
Index File

- size(Index File) << size(main file)
- Index file
  - 2 fields
    - **Search key attribute (SK)**
    - **Block pointer**
- Index file: small
  - How?
    - # attributes
    - # records



Example: Index File

- Index file: small
  - Reduce breadth: # attributes
    - 1 vs n
  - Reduce length: # records
    - Book index: all text or selective text?
    - Index: Selective data
      - 1 member of each block



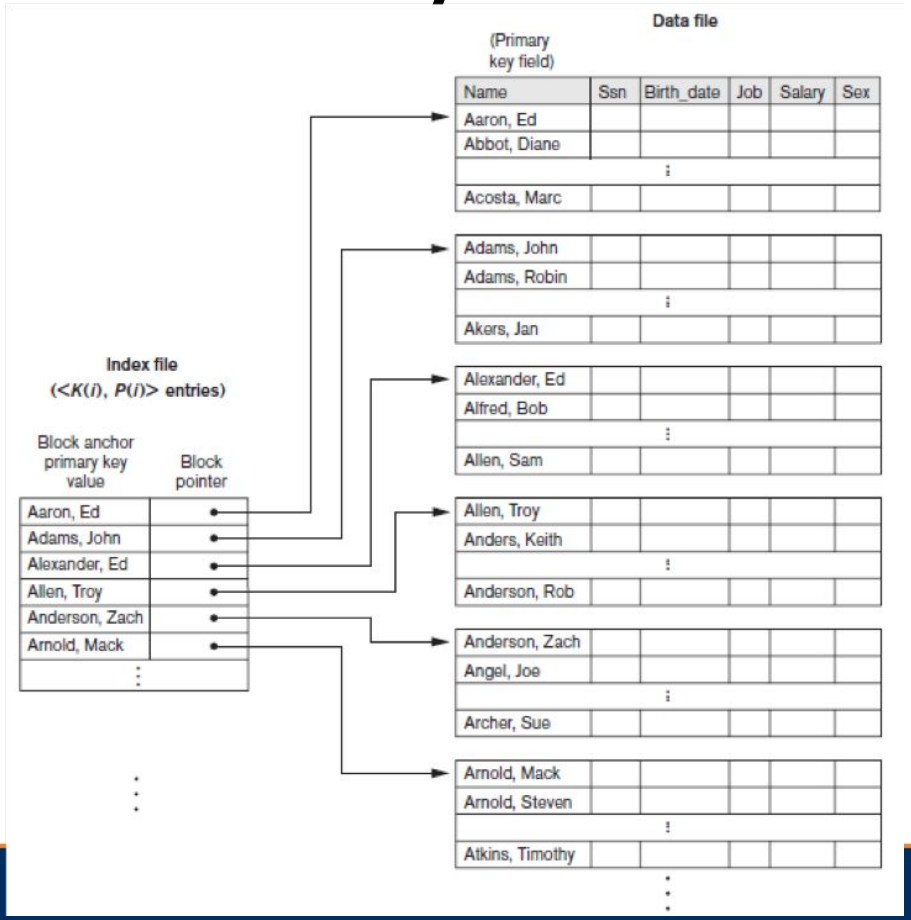
Index File

- Total: 10,000 records
- Blocks: 1000
- 10 records/block
  - B1: R1,R2, … R10
  - B2: R11, R12,... R20

- ***Can you search for R35 now***
  - ***Approach***

SK	Block Pointer
...	P(R1)
...	P(R11)
...	P(R21)
...	P(R31)
...	...

Primary Index



Primary indexing

- Main File: Sorted
- Search key: Prime key
  - Unique
- 1 index entry: each block in data file
- Each index entry: PK value => 1st record in block
- Total # entries in index = # disk blocks in **ordered** data file
- **Anchor record/ Block anchor**: 1st record in each block
- Indexes
  - **Dense indexing**: All file entries => index file
  - **Sparse indexing**: Not all file entries => index file
    - Distinct values for PK => index file

Primary Indexing

Dense indexing: All file entries => index file

Sparse indexing: Not all file entries => index file

Is primary indexing sparse or dense?

Primary indexing

- Index size: small
  - Fewer entries
  - Fewer attributes (1)
- Advantage of index
  - Binary Search on main file: All records
    - $\text{Log}_2 b$ block accesses
      - b: # blocks
  - Binary Search on main file: fewer records
    - Faster
    - $\text{Log}_2 b_i + 1$ block accesses
      - b_i: # blocks, i: # records

Searching without indexing

- Main file: 30,000 records
- Each record: 100 Bytes
- Block size: 1024 Bytes

Access time to search a file with, without index?

- **Blocking factor**: *lower_bound*(Block size/record size): *lower_bound(*1024/100)= 10
- **# blocks for main file** = # records in main file/blocking factor
  - 30,000/10 = 3,000
  - ***Access time without***: $\log_2 b = \log_2 3000$ = *upper_bound(*11.55) = 12
  - 12 blocks

Searching with index

- Main file: 30,000 records
- Each record: 100 Bytes
- Block size: 1024 Bytes
- Primary indexing
  - Index File: 15 Bytes
- **Blocking factor**: *lower_bound*(Block size/record size): *lower_bound(*1024/**15**)= 68
- **# blocks for index file** = # records in index file/blocking factor
  - # records in index file = # blocks in main file= 3,000
  - 3,000/68 = 44.18 = 45
  - ***Access time with*** Primary index: $\log_2 b = \log_2 45$ = *upper_bound(*5.49) = 6+1 = 7
  - 7 blocks v/s 12 blocks

Primary Index: Disadvantage

- **Major problem: insertion and deletion of records**
  - Move records around and change index values

- Solutions
  - Use unordered **overflow file**
  - Use **linked list of overflow records**

Clustering Indexing

- Ordered file with 2 fields
  - Non-key field: **Clustering field**
  - Block Pointer

- For each **distinct** value of non-key field => entry in Index file
  - Block Pointer: For **1st block** with existence ONLY
  - Example: B1: A B C B2: B C D B3: D E E B4: E E E B5: F F F

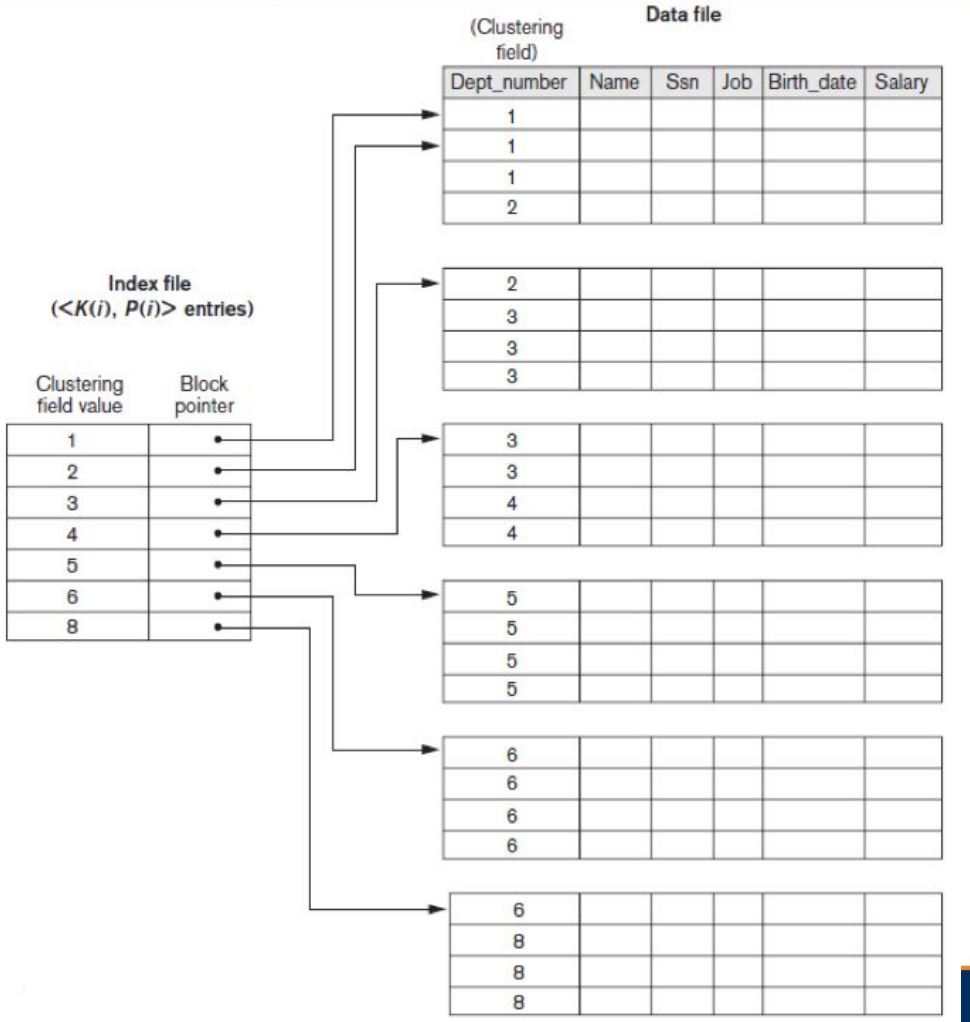
Clustering Indexing

Dense indexing: All file entries => index file

Sparse indexing: Not all file entries (only distinct ones) => index file

Is clustering indexing sparse or dense?

Clustering Index



Secondary Indexing

- Primary index: exists
- Purpose: secondary access
- Main file => unsorted
- 2 fields
  - Indexing field
  - Block pointer (or record pointer)
- Why not sort main file
  - Sorted wrt 1 field (CWID)
  - Get records => alphabetical order
    - unsorted
- Binary search => possible

Secondary Indexing

- Primary index
- Secondary index
- Multiple indexes on one main file
  - Possible
- Why multiple indexes?
  - Sort by CWID
  - Sort by Name
  - Multiple needs
    - Access file differently: **Better access**

Fast access: Additional memory for extra index

 # block access = $\log_2 n + 1$

Secondary Indexing

- Total: 10,000 records
- Blocks: 1000
- 10 records/block
  - B1: R1,R12, … R18
  - B2: R4, R2, ... R21

- ***Can you search for R35 now***
  - ***Approach***

Usually need more storage space, longer search time than primary index.

Secondary Indexing

Dense indexing: All file entries => index file

Sparse indexing: Not all file entries (only distinct ones) => index file

Is secondary indexing sparse or dense?

Secondary Indexing

- Unordered main file
- 2 fields
  - Search key field: Key / Non-key?
  - Block Pointer
- # entries in index file
  - Same as main file
- SK field: Pick 1
  - Sort

Why Secondary Indexing?

 # records of index file = # records of main file

Why do indexing ?

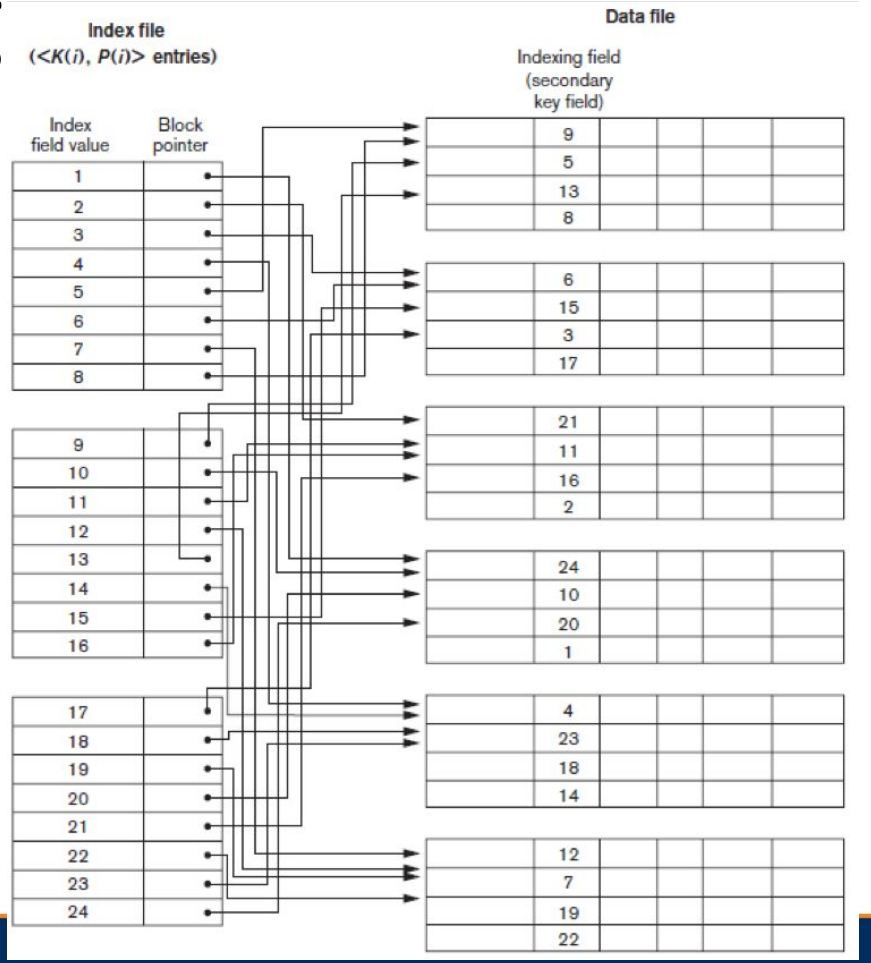
Answer: Why Secondary Indexing

 # records of index file = # records of main file

Why do indexing ?

1. Index file: sorted
   - Binary search
2. # records: same
   - Record size? Same or not

Secondary Indexing



Class Exercise

- Can the search key field be repeated for Secondary indexing?
- If yes
  - Does each record get an entry in index file?

Secondary Indexing

Secondary indexing: non-key field

- Options for implementing:
  a. Include duplicate entries => 1 for each occurence [**Dense index**]
     - 1 B1
     - 1 B3
     - …
  b. Block ptr: variable length:
     - 1 B1, B3
     - 2 B4
     - …
  c. Block pointer [**Non-dense index**]
     - 1 B200 B200: B1, B3

Types: Single-level ordered indexes

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

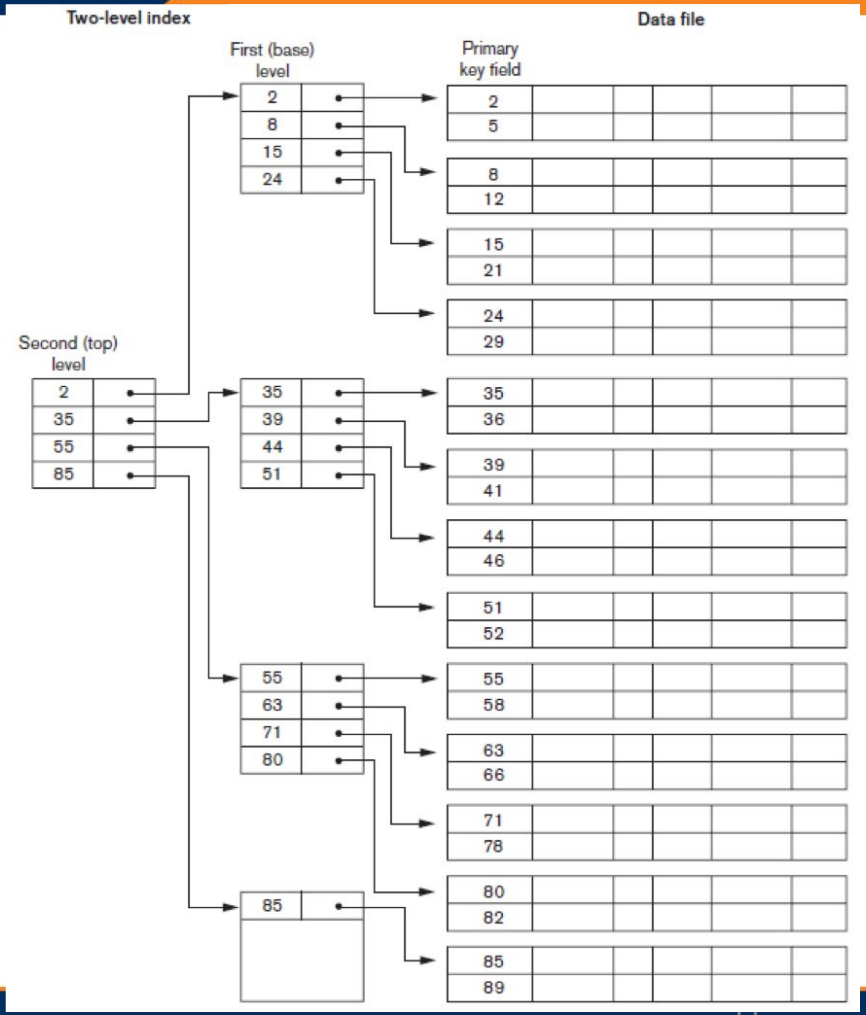
Multilevel Indexing

- Indexing at multiple levels
  - First level OR Base level indexing:
    - Ordered file
    - 2 fields
      - Search Key SK: Distinct value
      - Block Pointer
    - Directly attached to block
- Second level indexing
  - First level IF: Sorted data file
  - 1 entry => for each block of 1st level
  - Pointers: Record => block from level 1

Multilevel Indexing

- First Level
  - Base level
- Second Level
  - Primary index to 1st level
- Third Level
  - Primary index to 2nd level

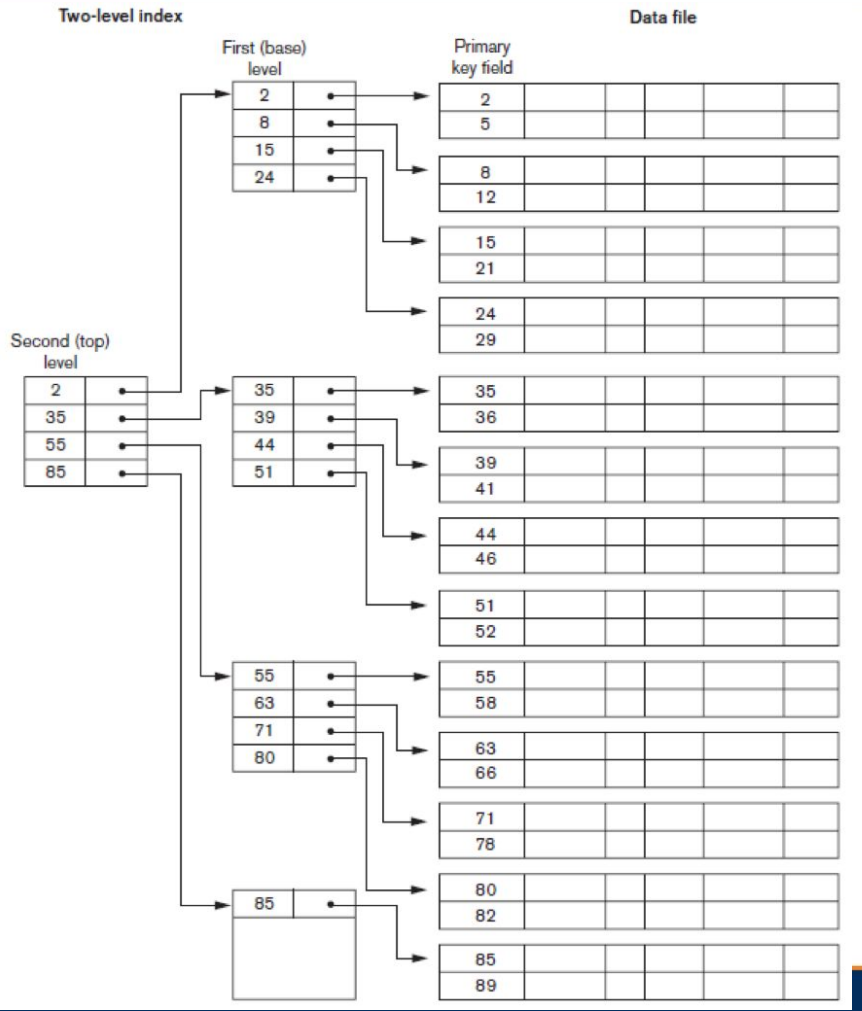
- # Levels:
  - Depend on:
    - # records,
    - # blocks



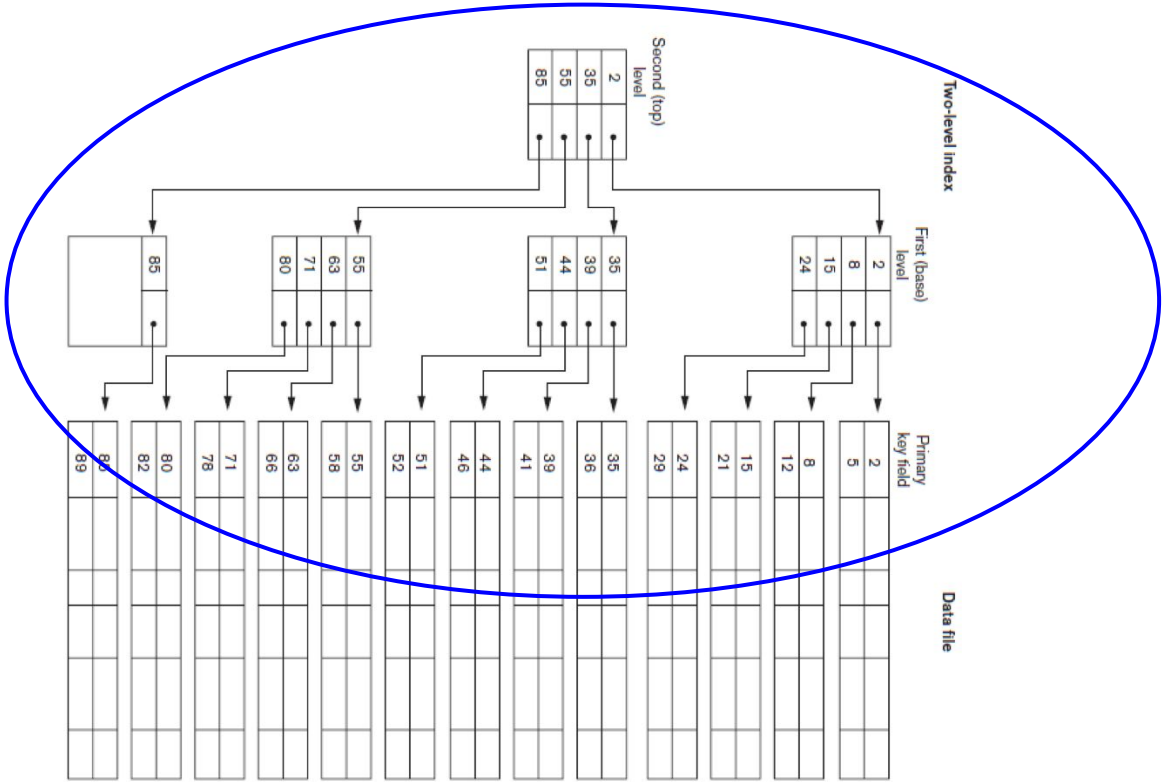
Why multilevel indexing

- Index too big
  - Index in index file (IF)
- Fast search of index
  - Index IF

Multilevel Indexing



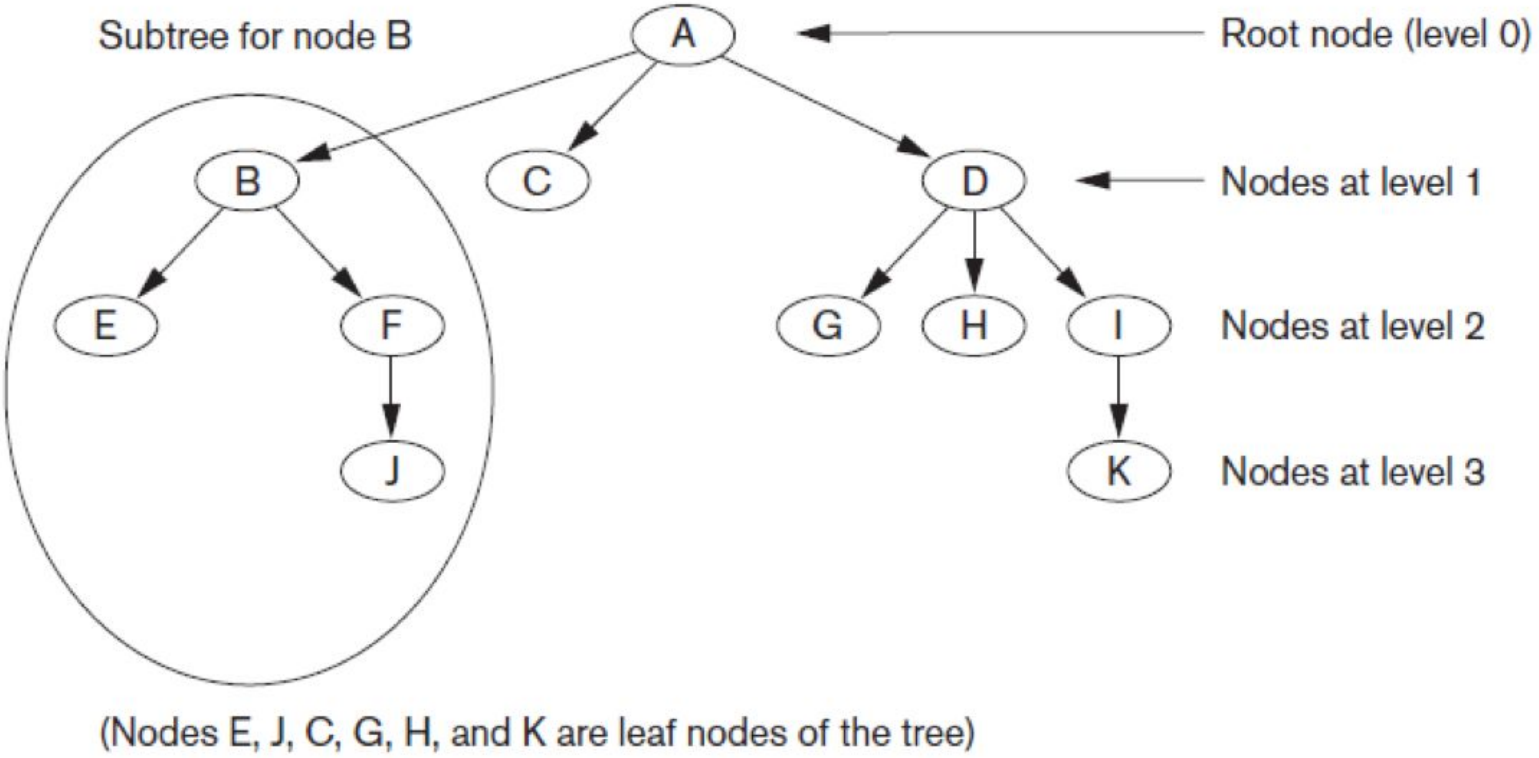
Multilevel Indexing



Dynamic Multilevel Indexes

- B Trees, B+ Trees
- Terminology
  - **Tree**: Formed of nodes
  - **Node**: Each node
    - 1 parent [except root]
    - >= 0 children
  - **Leaf** node
    - No child nodes
  - Non leaf node
    - Internal node
  - **Subtree** of node
    - Node + all descendant nodes

Tree Data Structure

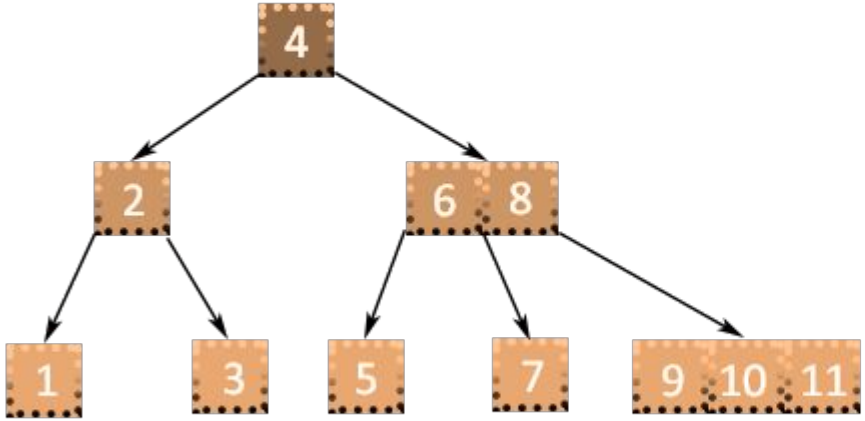


Why are we looking at trees?

Search tree => Used to do guide search for a record

Trees

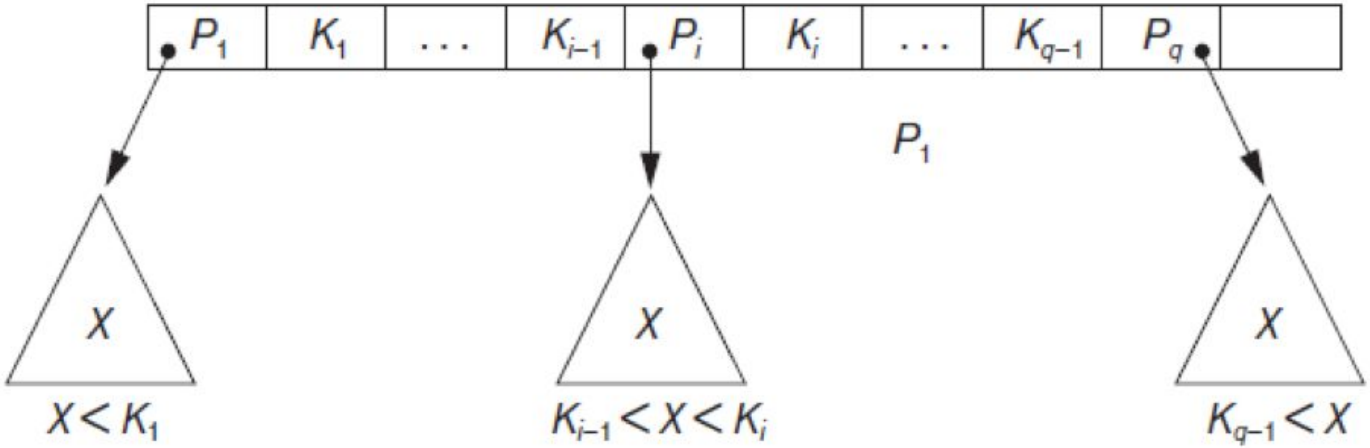
- Left child <= Parent <= Right child



- Node: Tree Pointers, values, record pointers

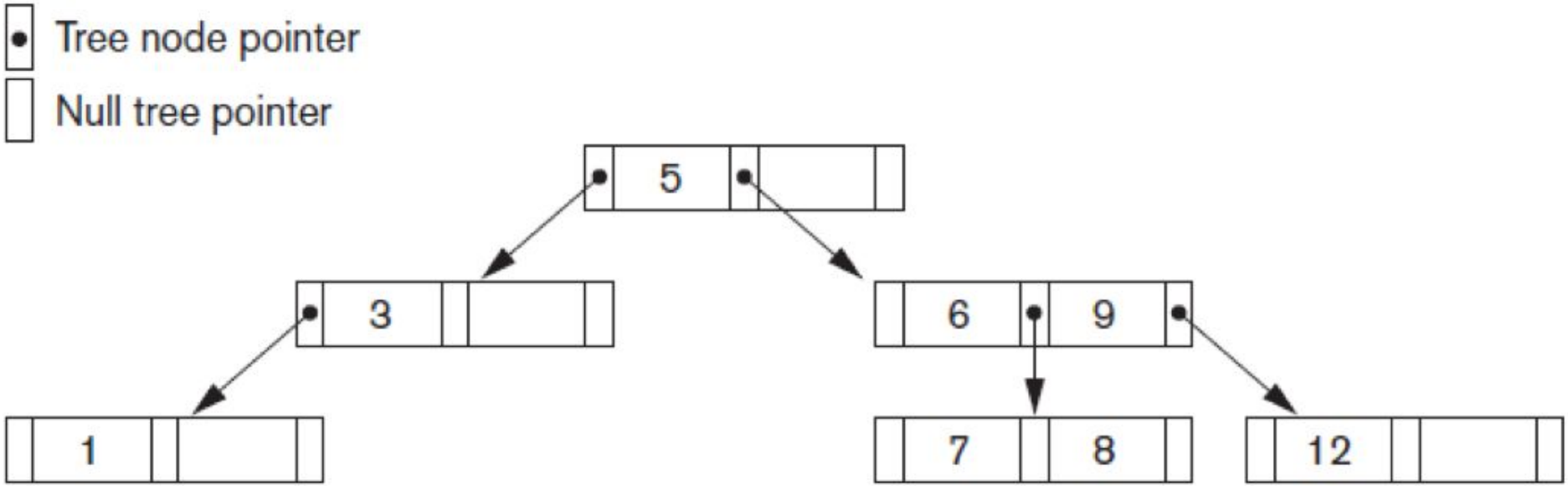
Search Tree

- ~ Multilevel indexing
- Search Tree



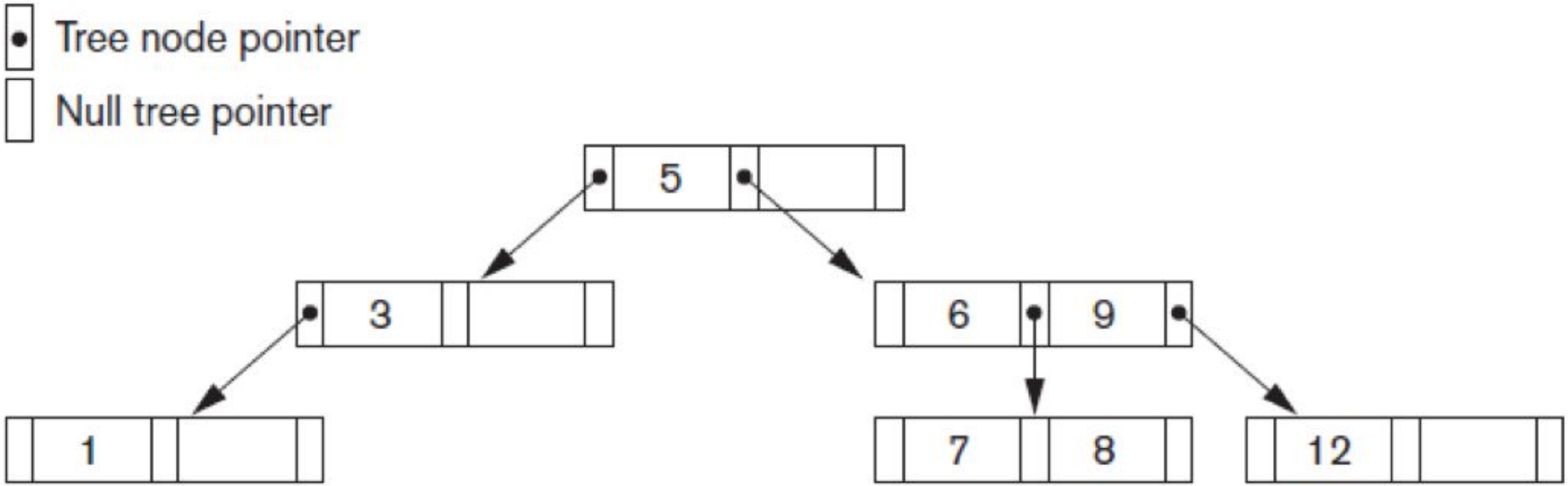
Search Tree

- Order: 3



Search Tree

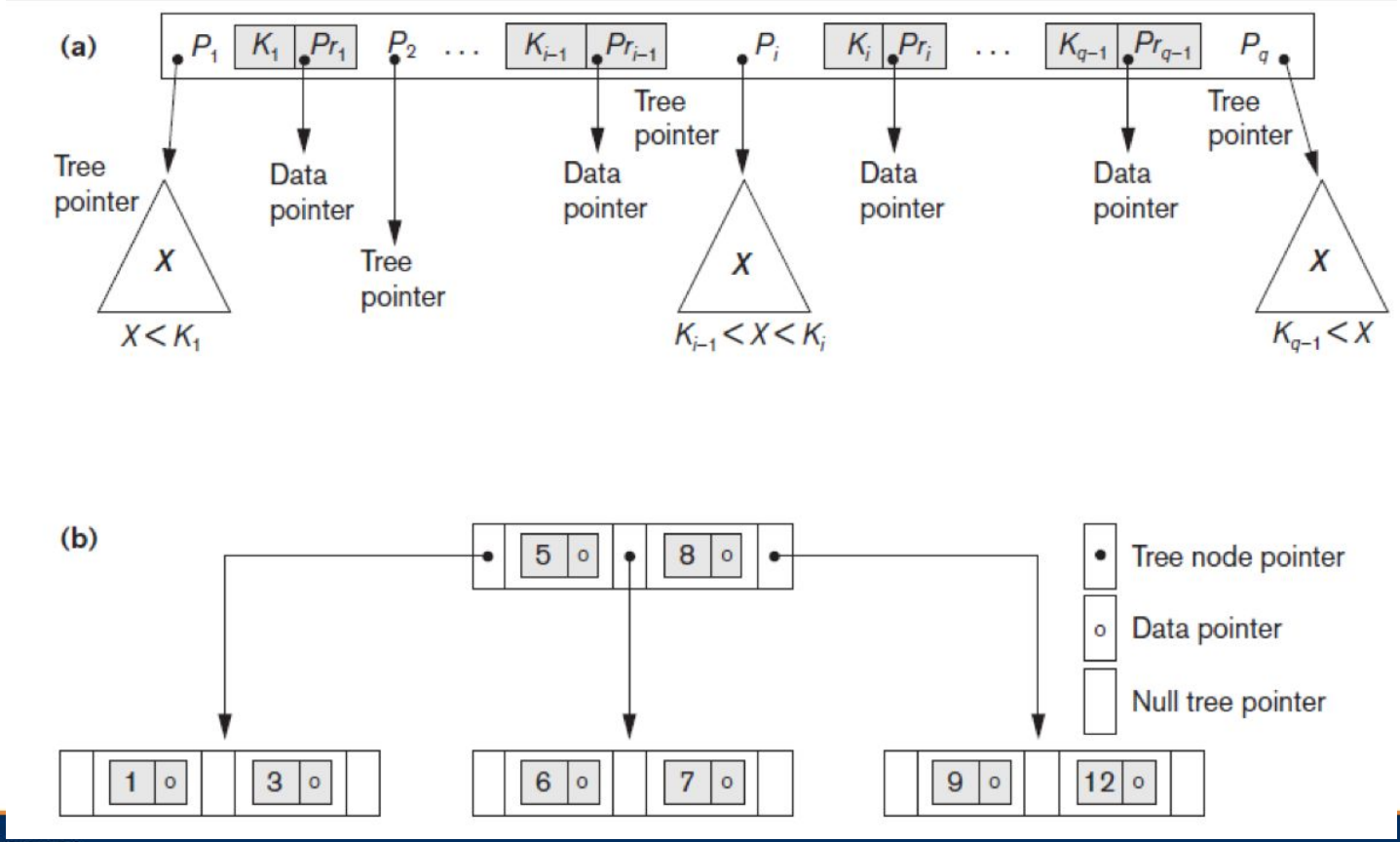
- Insertion, Deletion



B Trees

- Multilevel access structure
- Tree is always balanced
- Space wasted by deletion never becomes excessive
  - Each node is at least half-full
- Each node in a B-tree of order p can have at most p-1 search values

B Trees


(a)
P_1 K_1 Pr_1 P_2 ... K_{i-1} Pr_{i-1} P_i K_i Pr_i ... K_{q-1} Pr_{q-1} P_q
Tree pointer
Data pointer
Tree pointer
Data pointer
Tree pointer
Data pointer
Data pointer
Tree pointer
X
X
X
$X < K_1$
$K_{i-1} < X < K_i$
$K_{q-1} < X$
(b)
5
8
1
3
6
7
9
12
Tree node pointer
Data pointer
Null tree pointer

B Trees and B+ Trees

- B Trees
  - Pointers to data blocks => store in **leaf nodes** and **internal nodes**
- B+ Trees
  - Pointers to data blocks => store in **leaf nodes** ONLY

Summary

- Single-level Indexes
  - Primary
  - Clustering
  - Secondary
- Multilevel Indexing
- Dynamic Multilevel Indexes
- Search Trees
- B Trees