CSc 134
Database Management and File Organization

9. Indexing Structures for Files

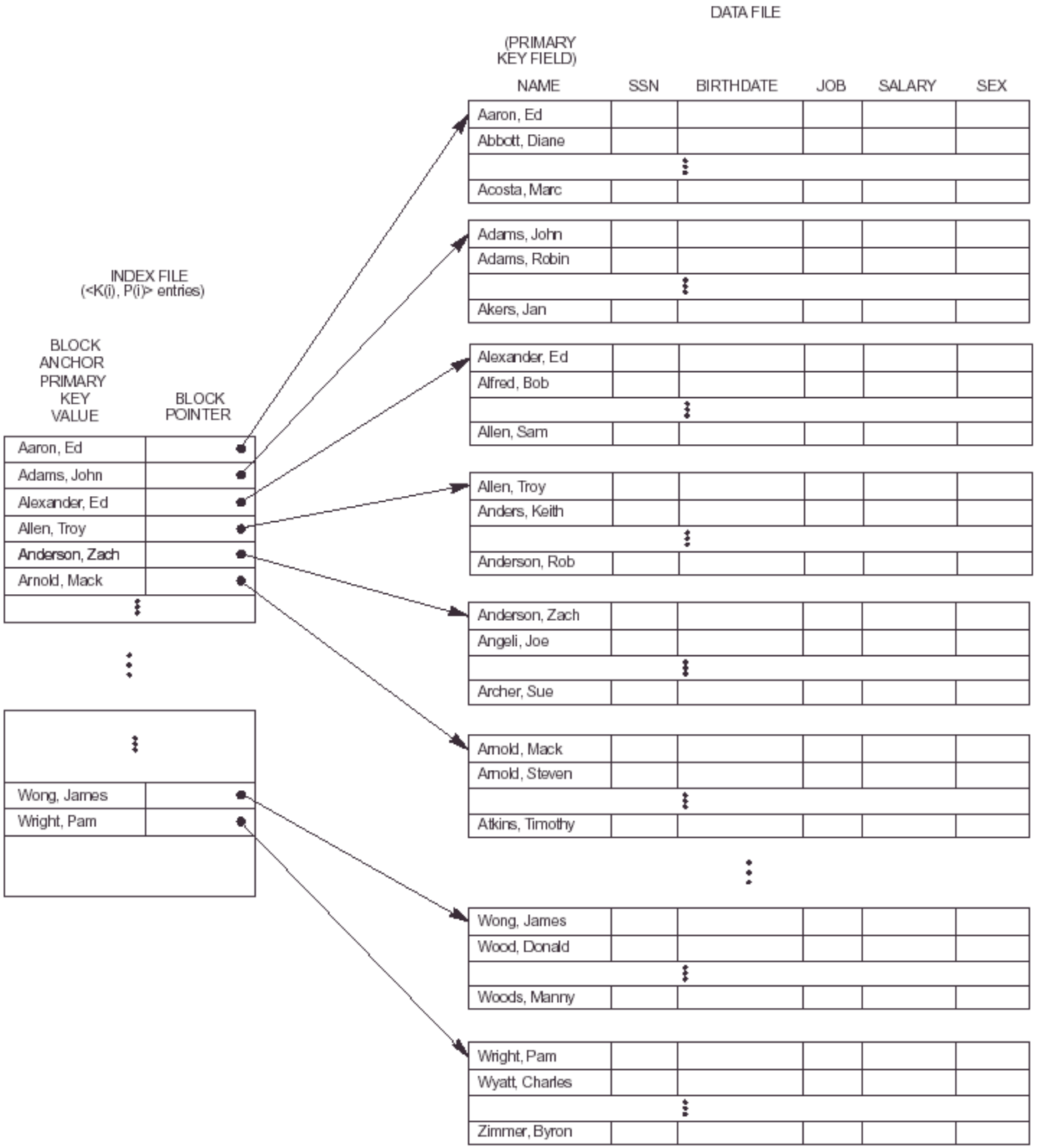
Ying Jin
Computer Science Department
California state University, Sacramento

Single-level ordered indexes

- Structure
  - Ordered index file
    - Small, easy to do binary search
  - A list of pointers to disk blocks
- Dense index vs. sparse (nondense) index
  - Dense: One index entry for every record of data files.
  - Sparse: Has index entries for only some of the search values.
- Primary index
- Clustering index
- Secondary index

Primary Indexes

- Specified on the *ordering key field* of an ordered file of records.
- Ordering key field is used to physically order the file records on disk.
- Index file consists of index entry (or index record)
- index entry: <primary key, pointer to disk block>
- index entry i: <K(i), P(i)>
- 1 index entry --- 1 block in the data file
- Anchor record of the block (block anchor): first record in each block of the data file.
- Dense index or spare index ☺?


DATA FILE
(PRIMARY KEY FIELD)
NAME
SSN
BIRTHDATE
JOB
SALARY
SEX
Aaron, Ed
Abbott, Diane
Acosta, Marc
Adams, John
Adams, Robin
Akers, Jan
Alexander, Ed
Alfred, Bob
Allen, Sam
Allen, Troy
Anders, Keith
Anderson, Rob
Anderson, Zach
Angeli, Joe
Archer, Sue
Arnold, Mack
Arnold, Steven
Atkins, Timothy
Wong, James
Wood, Donald
Woods, Manny
Wright, Pam
Wyatt, Charles
Zimmer, Byron
INDEX FILE
(<K(i), P(i)> entries)
BLOCK ANCHOR PRIMARY KEY VALUE
BLOCK POINTER
Aaron, Ed
Adams, John
Alexander, Ed
Allen, Troy
Anderson, Zach
Arnold, Mack
Wong, James
Wright, Pam

Access a record

- Load blocks of index files
- Search in index files
- load corresponding data file

Average time to access a record

- B: block size, R: record size, r: total number of records.
- blocking factor bfr=$\lfloor B/R \rfloor$ records per block
- A file has b block: b = (r/brf)
- A file with b block, binary search to find a specific record needs to access $\lceil \log_2 b \rceil$ block
  - Linear search b/2.
- Example1

  r=30,000, B=1024, R=100.
  - How many block accesses are needed for a binary search on the data file?

Example 1 (Cont.)

- Use primary index
  - ordering key field: V=9 bytes, a block pointer P=6 bytes.

Insertion and Deletion

- Insertion
  - Solution 1
    - Move space for the new record
    - May change some index entries
  - Solution 2
    - Using an unordered overflow file
  - Solution 3
    - Linked list of overflow records
- Deletion
  - Deletion markers.

Clustering Indexes

- Files are physically ordered on non-key field – clustering field
- Clustering field does not required to have distinct value.
- Clustering index
- <index field, pointer>
  - 1 entry for each distinct value of the clustering field.
  - a pointer to the first block in the data file that has a record with that value for its clustering field.
- Figure

Insertion

- Make insertion more efficient:

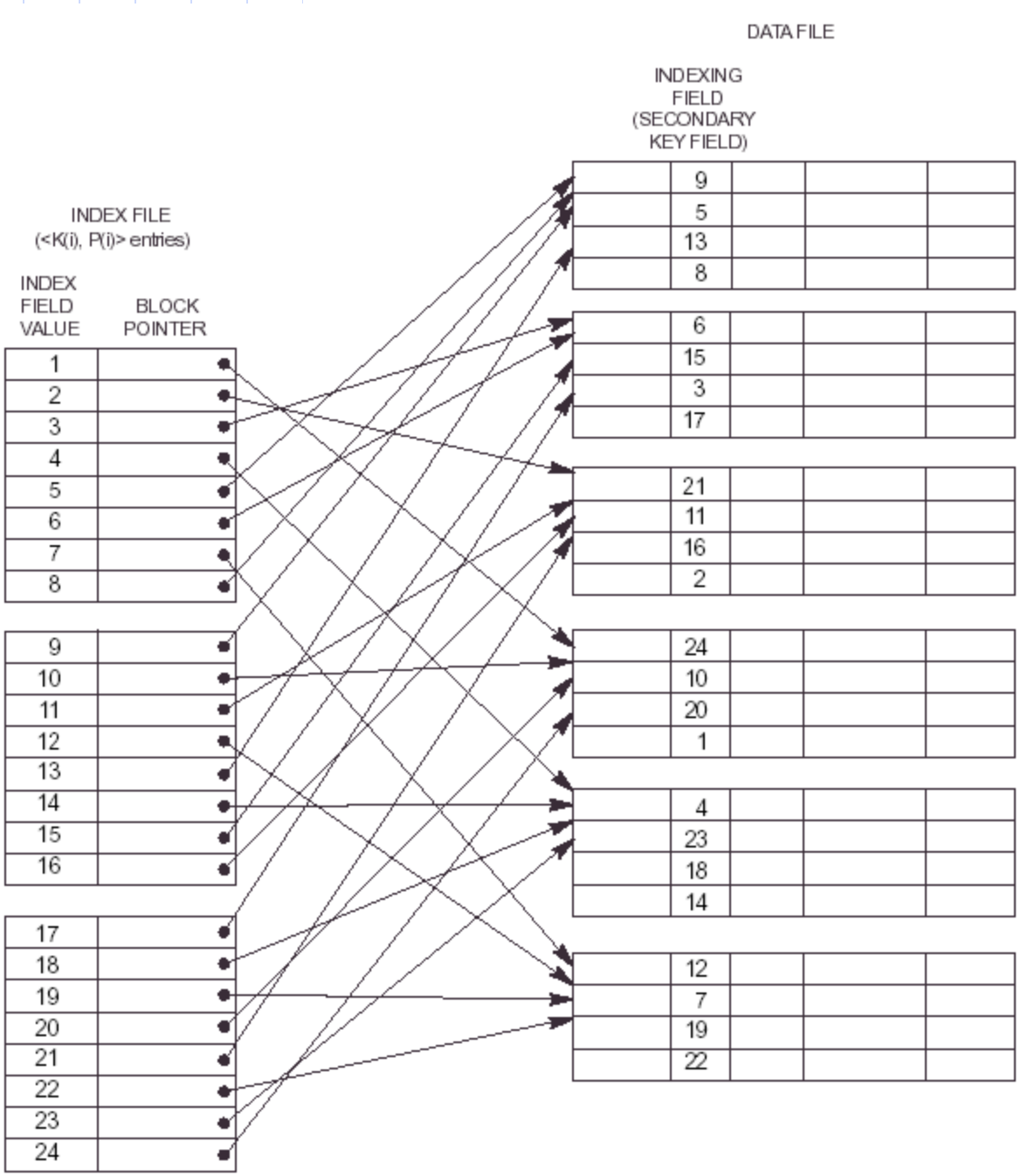
  It is common to reserve a whole block (or a cluster of contiguous blocks) for each value of the clustering field.
- Figure

Secondary Indexes

- Provide a secondary means of accessing a file for which some primary access already exists
- Index entry
  - index field: nonordering field of the data file
  - pointer: either a block pointer or a record pointer

Secondary indexes

- Secondary key: Index field has a distinct value for every record
- One index entry for each record in the data file
- e.g. Block pointer (figure)
  - Load and search index block
  - Load appropriate block
  - A search for the desired record within the block
- Dense index


DATA FILE
INDEXING FIELD (SECONDARY KEY FIELD)
INDEX FILE (<K(i), P(i)> entries)
INDEX FIELD VALUE
BLOCK POINTER
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
9
5
13
8
6
15
3
17
21
11
16
2
24
10
20
1
4
23
18
14
12
7
19
22

Example 2

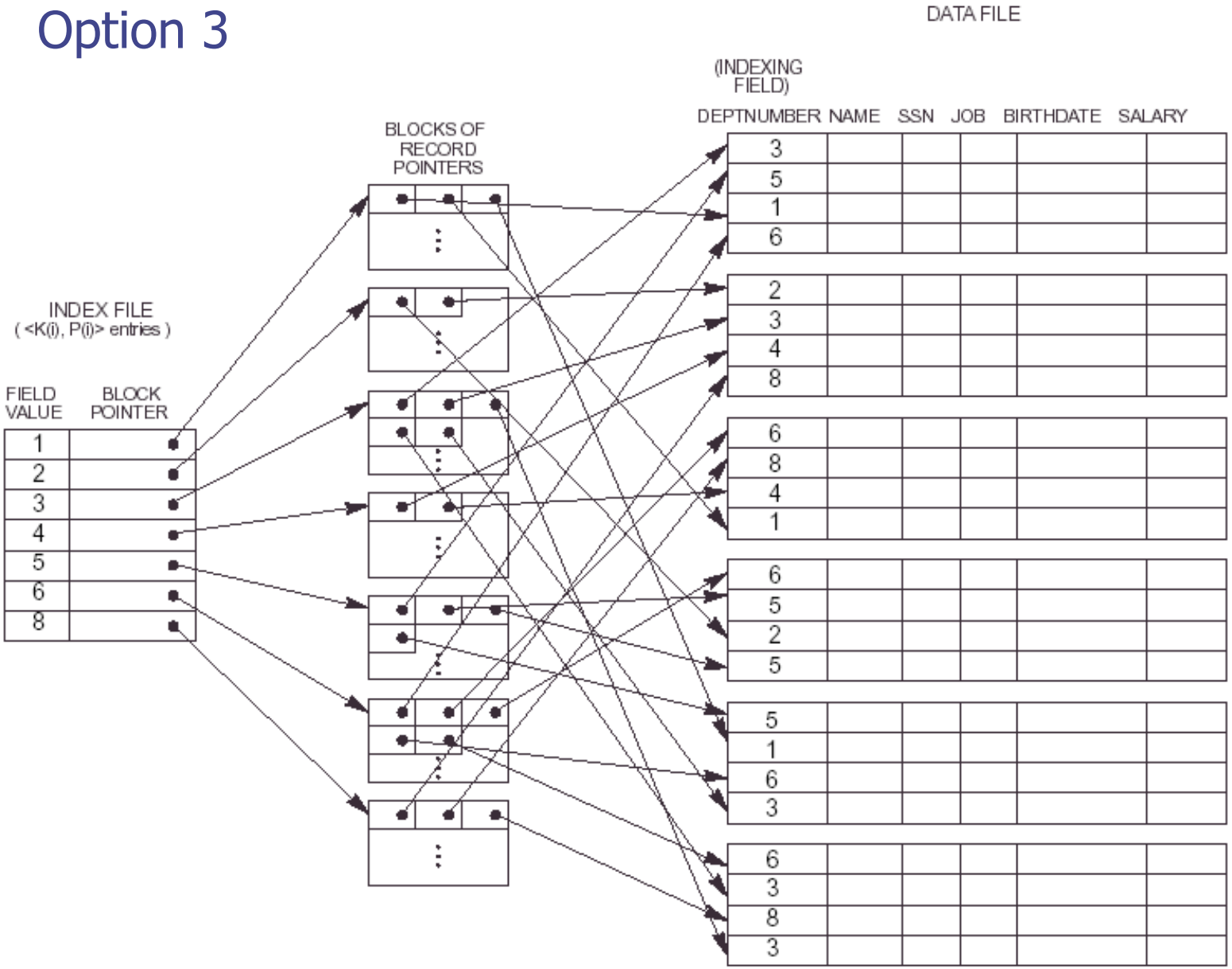
- r = 30,000, R=100, B=1024, b=3000
- Linear search
  - Require b/2=3000/2 = 1500 block accesses on the average
- Secondary index on non-ordering key field

  v=9, P=6

Secondary index on a non-key field

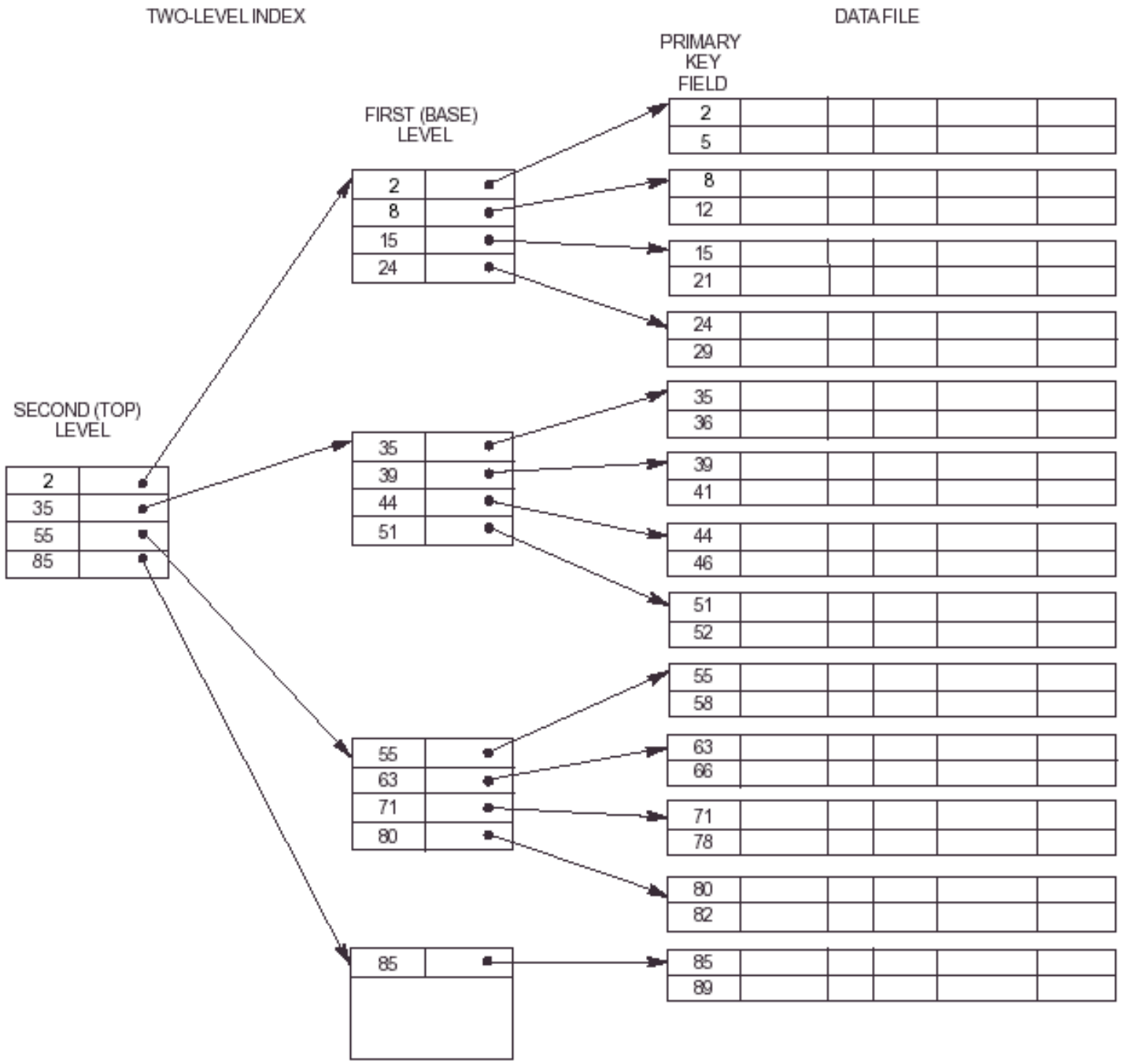
- Numerous records in the data file can have the same value for the indexing field
- Options for implementation
  1. Several index entries with the same K(i) value – one for each record.
  2. Variable-length records for the index entries
     <K(i), <P(i,1),..., P(i,k)>
  3. <K(i), P(i)> points to a block of record pointers (figure)
     If some K(i) occurs in too many records (pointers cannot fit in one block), a cluster or linked list of blocks is used. Figure

Option 3


DATA FILE
(INDEXING FIELD)
DEPTNUMBER NAME SSN JOB BIRTHDATE SALARY
BLOCKS OF RECORD POINTERS
INDEX FILE
(<K(i), P(i)> entries)
FIELD VALUE
BLOCK POINTER
1
2
3
4
5
6
8
3 5 1 6
2 3 4 8
6 8 4 1
6 5 2 5
5 1 6 3
6 3 8 3

Multilevel Indexes

- Idea
  - If the first level needs more than one block of disk storage → require a second level index
  - If the second level needs more than one block of disk storage → require a third level index
  - ...
  - Repeat the preceding process until all the entries of some index level t fit in a single block


TWO-LEVEL INDEX
DATA FILE
PRIMARY KEY FIELD
FIRST (BASE) LEVEL
SECOND (TOP) LEVEL
2
35
55
85
2
8
15
24
35
39
44
51
55
63
71
80
85
2
5
8
12
15
21
24
29
35
36
39
41
44
46
51
52
55
58
63
66
71
78
80
82
85
89

Multilevel indexes (Cont.)

- bfr_i is called fan-out of the multilevel index. Refered as fo.
- In any level, bfr_i = fo (# of record per block)
  - Reason: all index entries are the same size
- r1: total number of records in level 1
- the 1st level needs (r1/fo) block

  = # of entries needed at the 2nd index level

  = r2 = (r1/fo)

Multilevel indexes (Cont.)

- the 2nd level needs (r2/fo) block
  = # of entries needed at the 3rd index level
  = r3 = (r2/fo)
- Approximately t level index, such that r1/ $((fo)^t) >= 1$
- A multilevel index with r1 first-level entries will have approximately t level, where
  $t = \log_{fo} (r1)$

B-Tree

- All leaf nodes are at the same level
- Figure
- Internal node of a B-tree

  <P1, <K1, Pr1>, P2, <K2, Pr2>, ..., <Kq-1, Prq-1>, Pq>

  Pi – tree pointer

  Pri – data pointer (a pointer to the record whose search key field value = Ki, or to the data file block containing that record)

B-Tree (Cont.)

- Within each node, K1<K2<...<Kq-1
- For all search key field values X in the subtree pointed at by Pi
  - Ki-1 < X< Ki, where 1<i<q
  - X<Ki, where i=1
  - Ki-1<X, where i=q;
- Leaf nodes have the same structure as internal nodes except that all of their tree pointers are null

B^+ Tree

- Data pointers are stored only at the leaf nodes of the tree
- The leaf nodes have an entry for every value of the search field, along with a data pointer to the record (or block) if the search field is a key field
- The structure of leaf nodes differs from the structure of internal nodes
- Internal notes of a B+ tree (figure)

Leaf node of a B^+ tree

- <<K1, Pr1>, <K2, Pr2>, … <Kq-1, Prq-1>, Pnext>

  Pri is a data pointer

  Pnext points to the next leaf node of the B^+ tree
- K1 < K2 <… < Kq-1
- Figure

These slides are based on the textbook:

R. Elmaseri and S. Navathe, *Fundamentals of Database Systems*, 6th Edition, Addison-Wesley. Chapter 18.