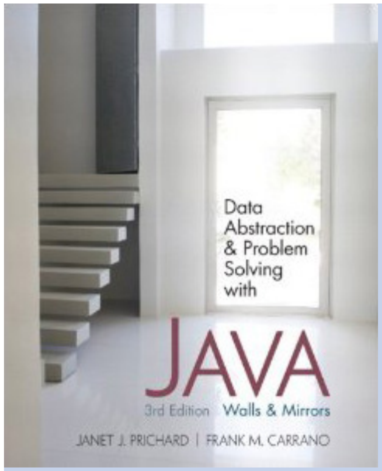


Chapter 15

External Methods

A Look At External Storage

- External storage
  - Exists beyond the execution period of a program
  - Generally, there is more external storage than internal memory
- Sequential access file
  - To access the data, you must advance the file window beyond all the intervening data
  - Resembles a linked list
- Random access file
  - Data can be accessed at a given position directly
  - Resembles an array
  - Essential for external tables

A Look At External Storage

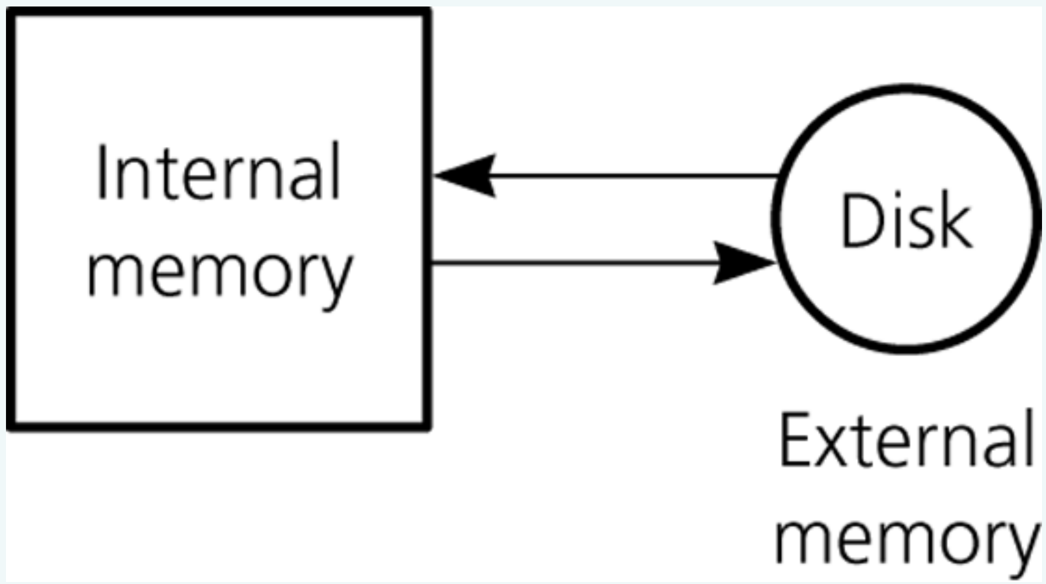


Figure 15-1

Internal and external memory

A Look At External Storage

- A file consists of data records
  - Records are organized into one or more blocks
    - The number of records in a block is a function of the size of the records
- Random access file
  - All input and output is at the block level
- Buffer
  - A location that temporarily stores data as it makes its way from one process or location to another
  - Used while transferring data between internal and external memory

A Look At External Storage

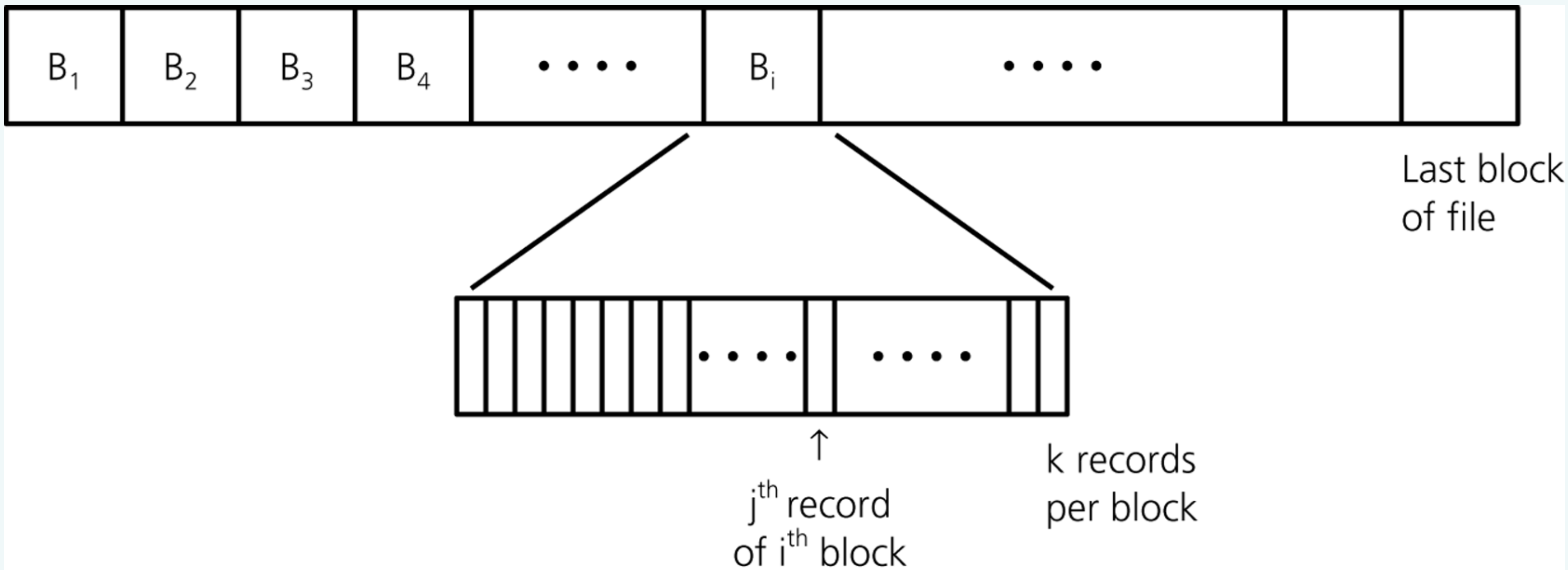


Figure 15-2

A file partitioned into blocks of records

A Look At External Storage

- Once the system has read a block into the buffer `buf`, the program can process the records in the block
- If the program modifies the records in `buf`, it must write `buf` back out to `dataFile`
- The number of block accesses should be reduced as much as possible
  - Block access time is the dominant factor when considering an algorithm's efficiency

Sorting Data in An External File

- The challenge with sorting data in an external file
  - An external file is too large to fit into internal memory all at once
  - Sorting algorithms presented earlier in the book assume that all the data to be sorted is available at one time in internal memory
- Solution
  - Use a modified version of mergesort

Sorting Data in An External File

- External mergesort
  - Phase 1
    - Read a block from F (data file to be sorted) into internal memory, sort its records by using an internal sort, and write the sorted block out to F_1 (a work file) before reading the next block from F
    - Repeat the above step for all the blocks of F
  - Phase 2 (a sequence of merge steps)
    - Each merge step
      - Merges pairs of sorted runs to form larger sorted runs
      - Doubles the number of blocks in each sorted run
      - Halves the total number of sorted runs
    - At the end
      - F_1 will contain all the records of the original file in sorted order

Sorting Data in An External File

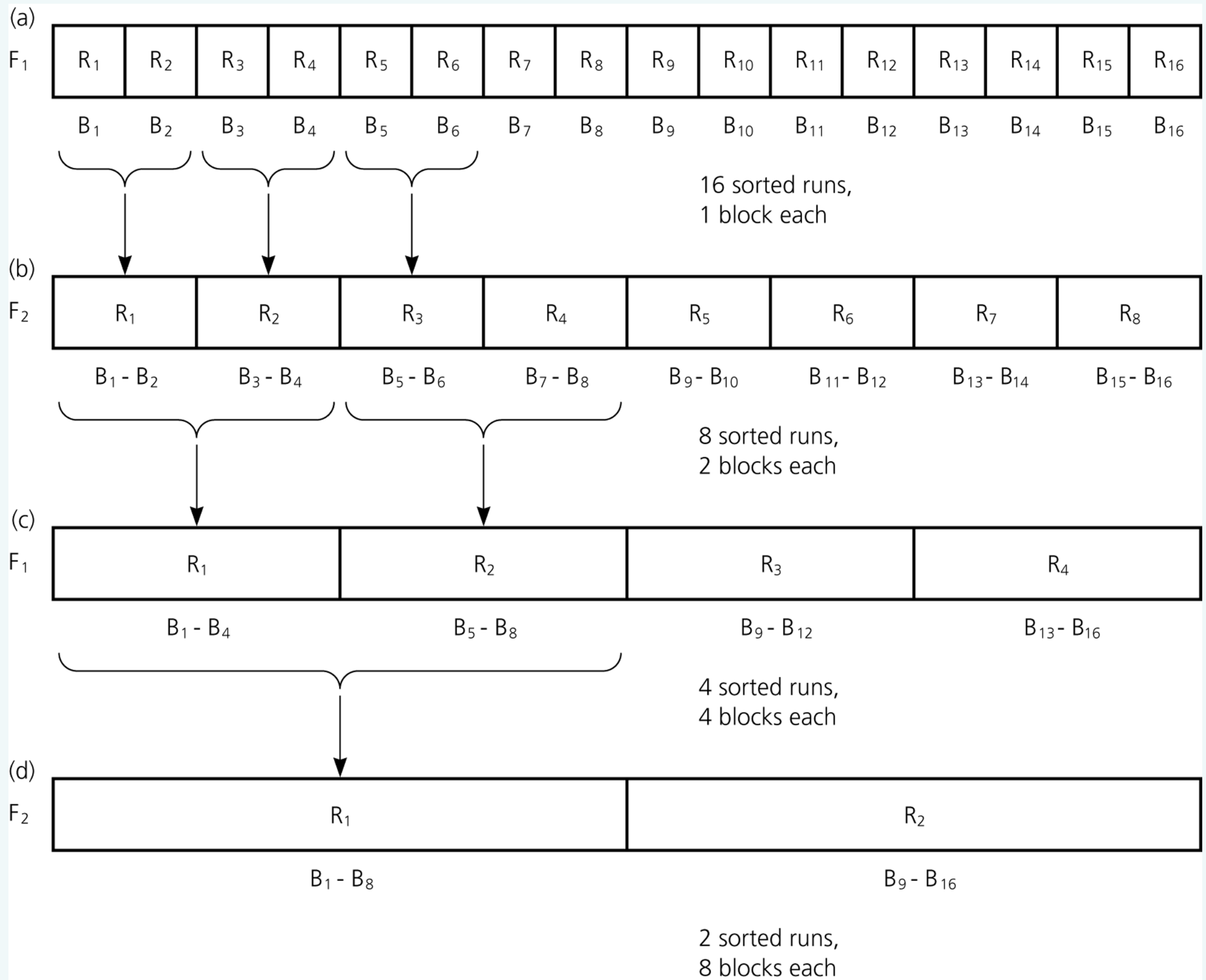


Figure 15-3

a) 16 sorted runs, 1 block each, in file F_1;
b) 8 sorted runs, 2 blocks each, in file F_2;
c) 4 sorted runs, 4 blocks each, in file F_1;
d) 2 sorted runs, 8 blocks each, in file F_2

External Tables

- External implementation of the ADT table
  - Records are stored in search-key order
    - The file can be traversed in sorted order
    - Main advantage
      - A binary search can be used to locate the block that contains a given search key
    - Main disadvantage
      - `tableInsert` and `tableDelete` operations can require many costly block accesses due to the need to shift records

External Tables

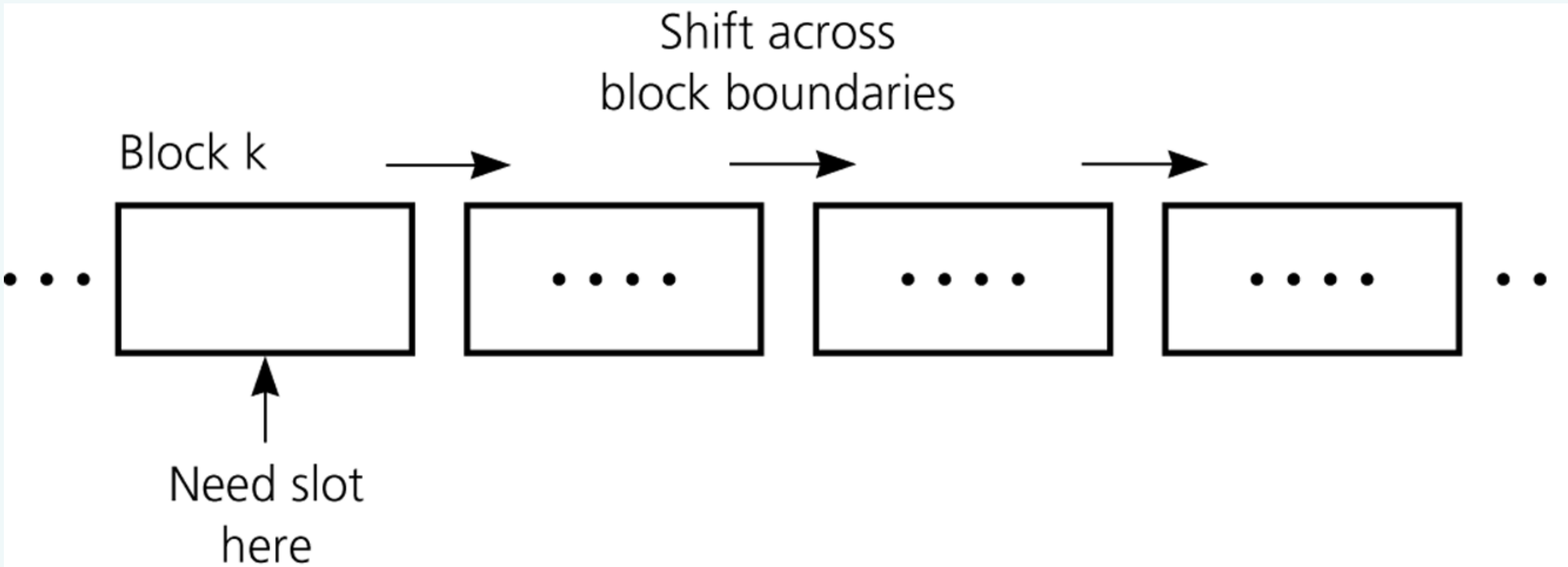


Figure 15-5

Shifting across block boundaries

Indexing An External File

- An index (or index file)
  - Used to locate items in an external data file
  - Contains an index record for each record in the data file

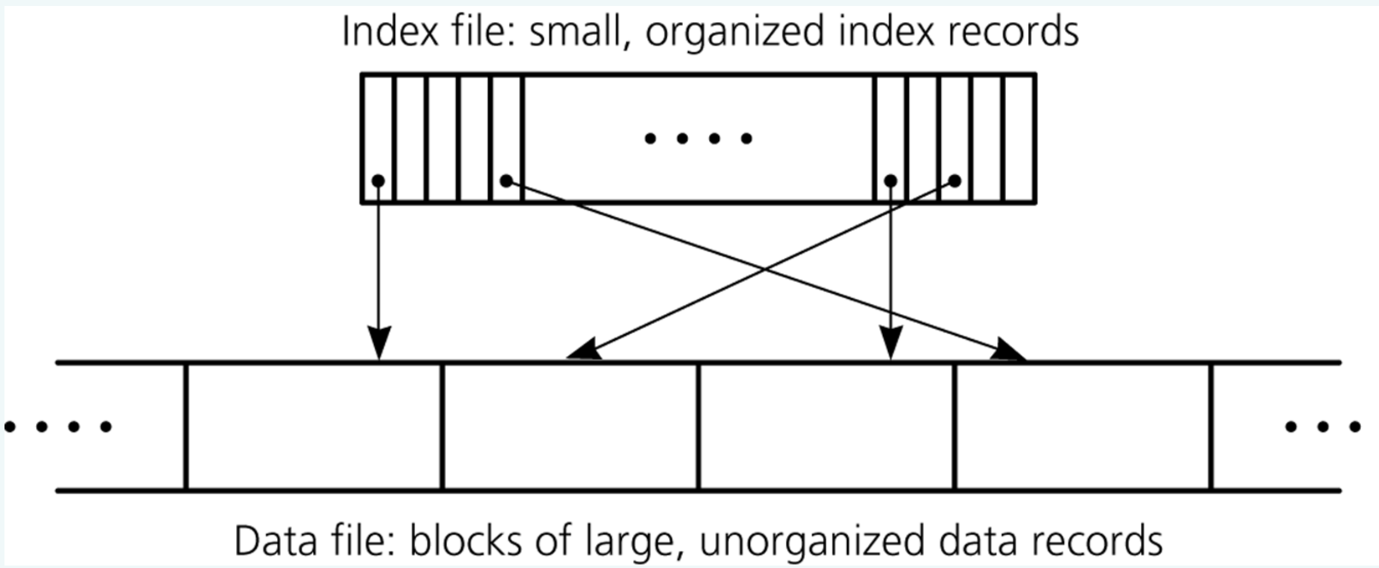


Figure 15-6

A data file with an index

Indexing An External File

- An index record has two parts
  - A key contains the same value as the search key of its corresponding record in the data file
  - A pointer shows the number of the block in the data file that contains the data record
- Advantages of an index file
  - An index file can often be manipulated with fewer block accesses than would be needed to manipulate the data file
  - Data records do not need to be shifted during insertions and deletions
  - Allows multiple indexing

Indexing An External File

- A simple scheme for organizing the index file
  - Store index records sequentially

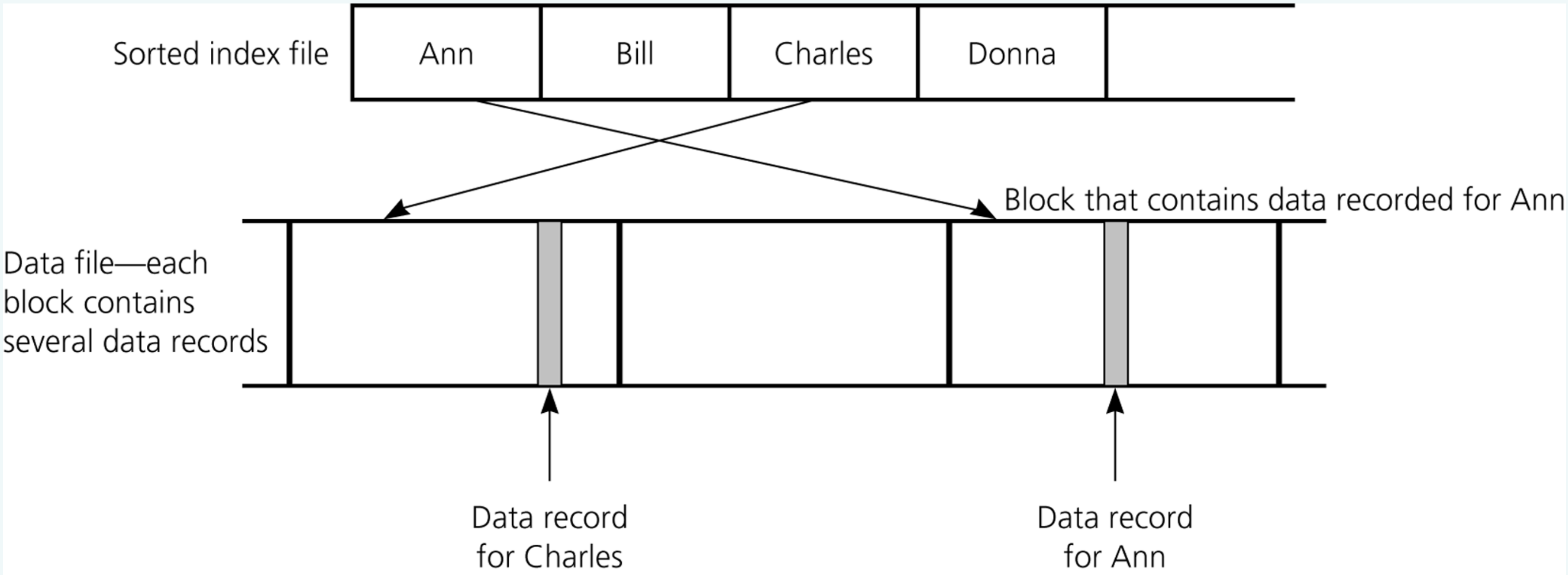


Figure 15-7

A data file with a sorted index file

Indexing An External File

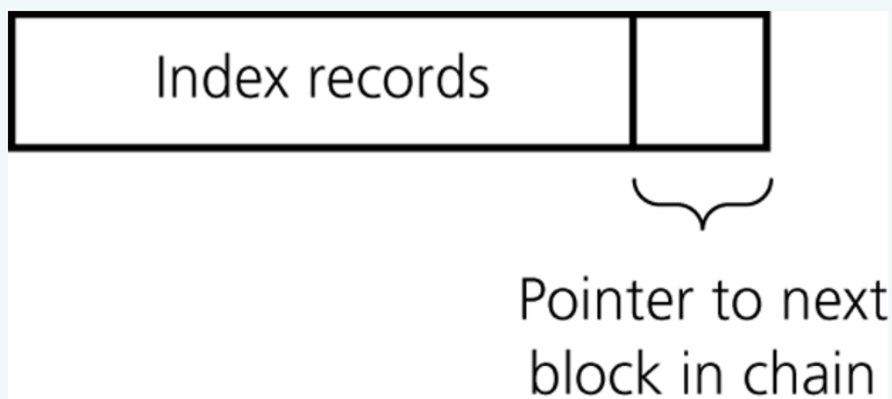
- Storing index records sequentially
  - `tableRetrieve` operation
    - Can be performed by using a binary search on the index file
  - `tableInsert` and `tableDelete` operations
    - Require only the shifting of index records, not data records
      - Benefits of shifting index records rather than data records
        - » Reduction in the maximum number of block accesses required
        - » Reduction in the time requirement for a single shift
  - More efficient than having a sorted data file
  - Not as efficient as using hashing or search trees to organize the index file

External Hashing

- The index file, not the data file, is hashed
  – Each entry `table[i]` is associated with a linked list of blocks of the index file
  – Each block of `table[i]`'s linked list contains index records whose keys hash into location `i`
  – To form the linked list, space must be reserved in each block for a block pointer
    - A block pointer is the integer block number of the next block in the chain

Figure 15-9

A single block with a pointer



External Hashing

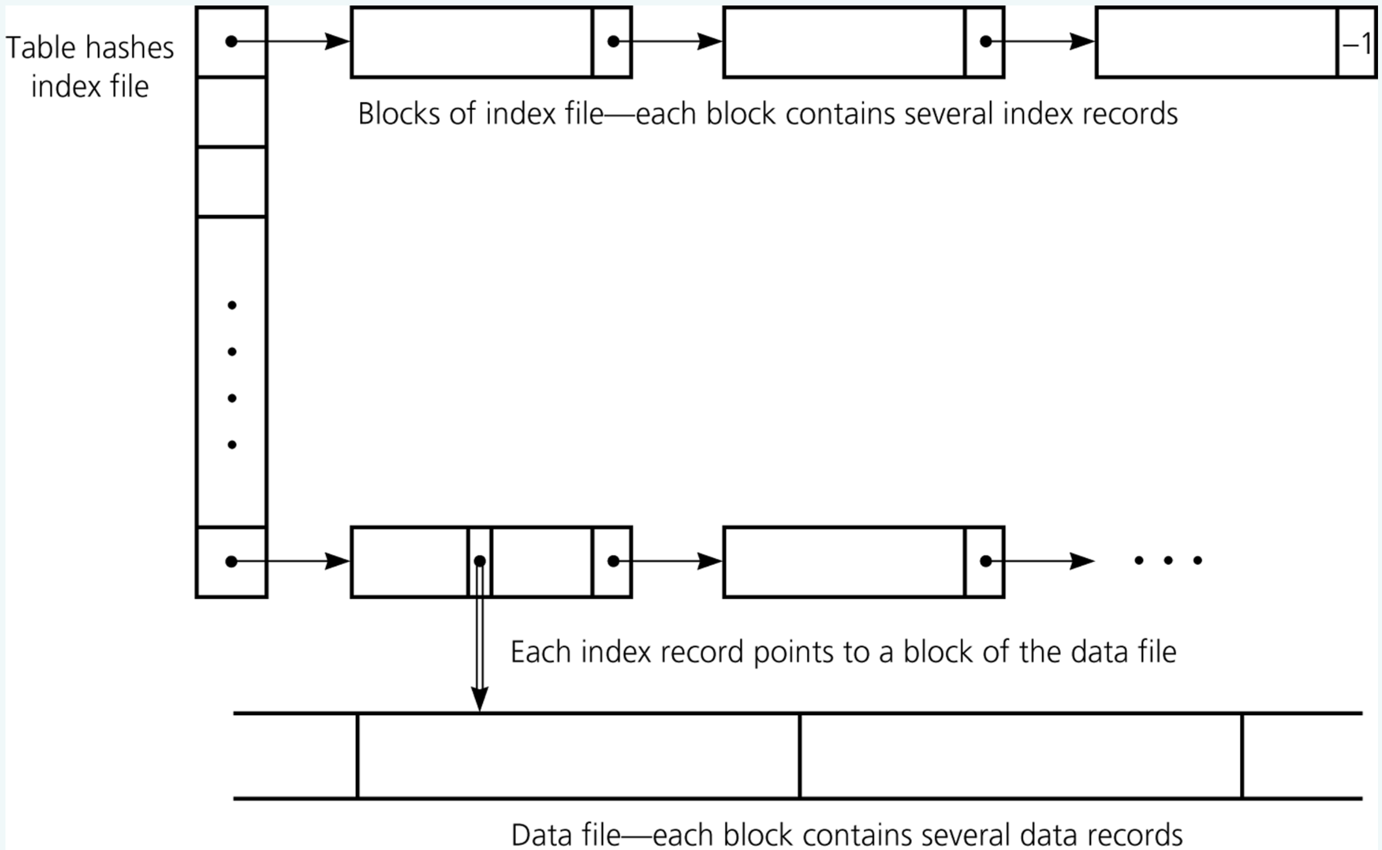


Figure 15-8

A hashed index file

External Hashing

- Retrieval under external hashing of an index file
  - Apply the hash function to the search key
  - Find the first block in the chain of index blocks (these blocks contain index records that hash into location i)
  - Search for the block with the desired index record
  - Retrieve the data item, if present

External Hashing

- Insertion under external hashing of an index file
  - Step 1: Insert the data record into the data file
    - New record can be inserted anywhere in the data file
  - Step 2: Insert a corresponding index record into the index file
    - For an index record that has key value `searchKey` and reference value p
      - Apply the hash function to `searchKey`, letting
        - » `i = h(searchKey)`
      - Insert the index record < `searchKey`, p> into the chain of blocks that the entry `table[i]` points to

External Hashing

- Deletion under external hashing of an index file
  - To delete the data record whose search key is `searchKey`
    - Step 1: Search the index file for the corresponding index record
      - Apply the hash function to `searchKey`, letting
        - » `i = hash(searchKey)`
      - Search the chain of index blocks pointed to by the entry `table[i]` for an index record whose key value equals `searchKey`
      - If an index record < `searchKey`, p> is found
        - » Note the block number `p`
        - » Delete the index record

External Hashing

- Deletion under external hashing of an index file
  - To delete the data record whose search key is `searchKey`
  - Step 2: Delete the data record from the data file
    - Access the block p
    - Search the block for the record
    - Delete the record
    - Write the block back to the file

External Hashing

- External hashing implementation
  - Should be chosen for performing the following operations on a large external table
    - `tableRetrieve`
    - `tableInsert`
    - `tableDelete`
  - Not practical for some operations, such as
    - Sorted traversal
    - Retrieval of the smallest or largest item
    - Range queries that require ordered data

B-Trees

- To organize the index file as an external search tree
  - Use block numbers for child pointers
    - A child pointer value of –1 is used as the `null` pointer

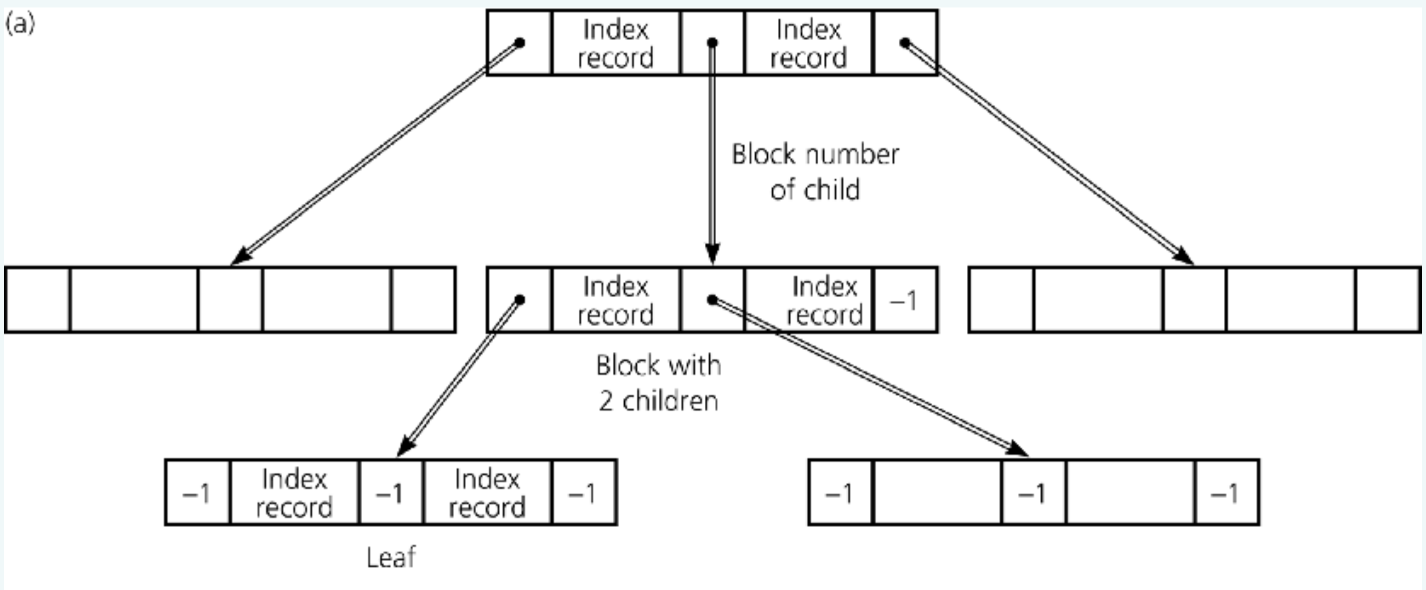


Figure 15-10a

a) Blocks organized into a 2-3 tree

B-Trees

- If the index file is organized into a 2-3 tree
  - Each node would contain
    - Either one or two index records, each of the form `<key, pointer>`
    - Three child pointers

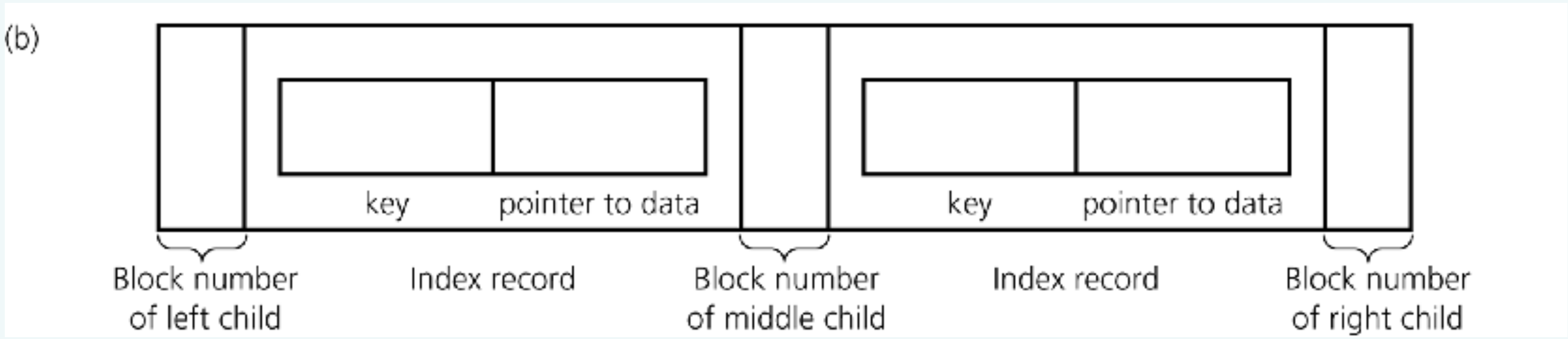


Figure 15-10b

b) a single node of the 2-3 tree

B-Trees

- An external 2-3 tree is adequate, but an improvement is possible
- To improve efficiency
  - Allow each node to have as many children as possible
    - In an external environment, the advantage of keeping a search tree short far outweighs the disadvantage of performing extra work at each node
    - Block size should be the only limiting factor for the number of children

B-Trees

- Binary search tree
  - If a node N has two children, it must contain one key value
- 2-3 tree
  - If a node N has three children, it must contain two key values
- General search tree
  - If a node N has m children, it must contain m – 1 key values

B-Trees

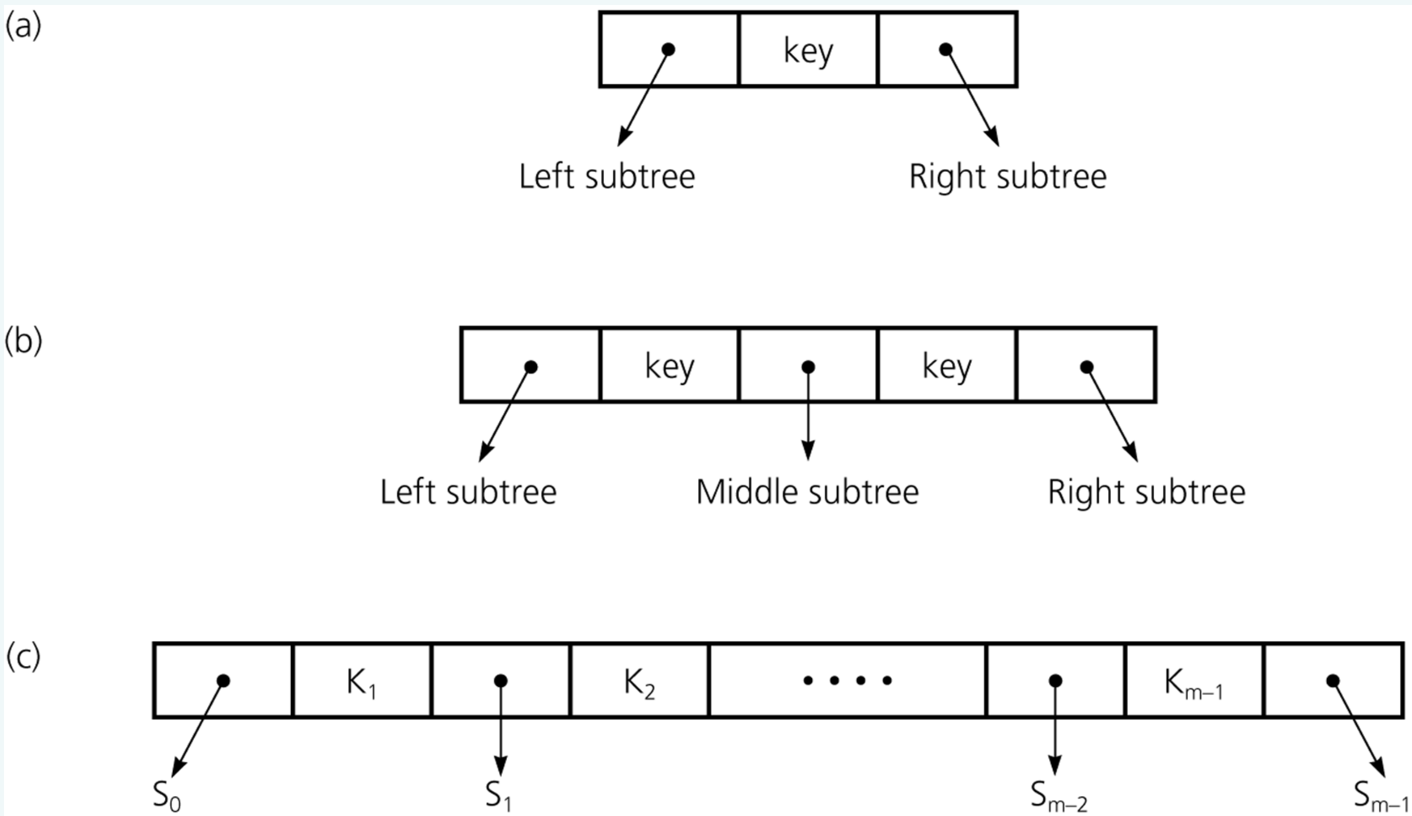


Figure 15-11

a) A node with two children; b) a node with three children; c) a node with *m* children

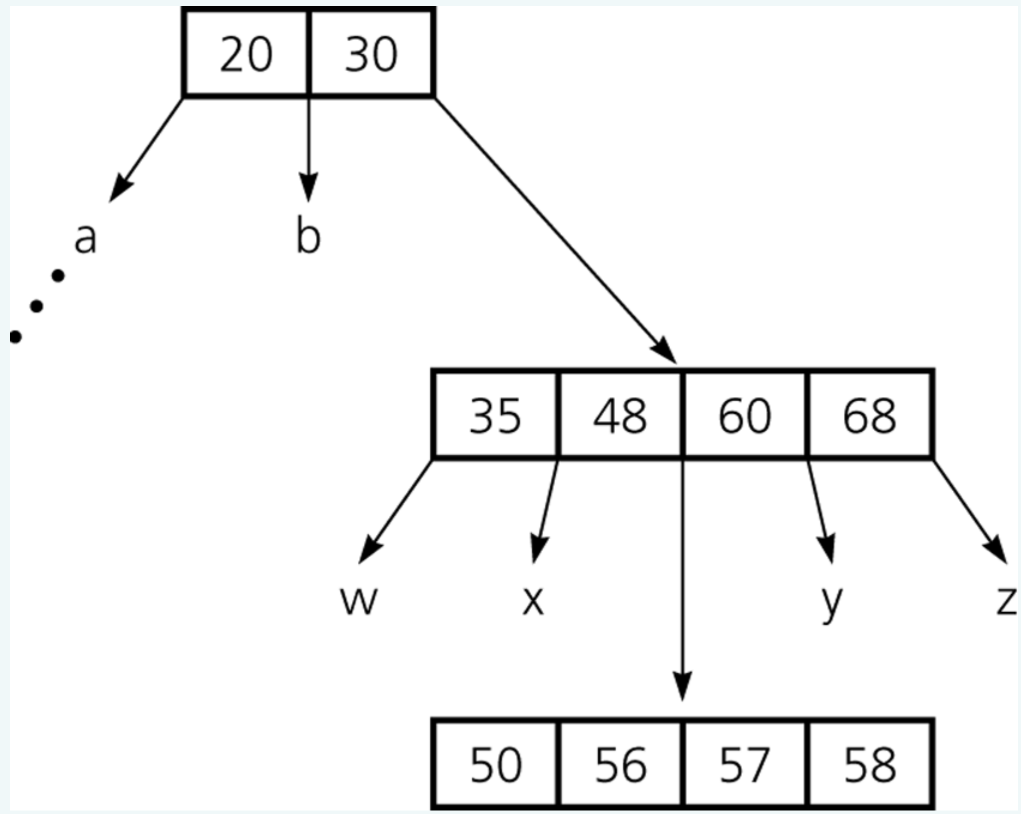
B-Trees

- B-tree of degree m
  - All leaves are at the same level
  - Nodes
    - Each node contains between m – 1 and [m/2] records
    - Each internal node has one more child than it has records
    - Exception: The root can contain as few as one record and can have as few as two children
  - Example
    - A 2-3 tree is a B-tree of degree 3

B-Trees

Figure 15-13

A B-tree of degree 5



B-Trees

- Insertion into a B-tree
  - Step 1: Insert the data record into the data file
  - Step 2: Insert a corresponding index record into the index file

B-Trees

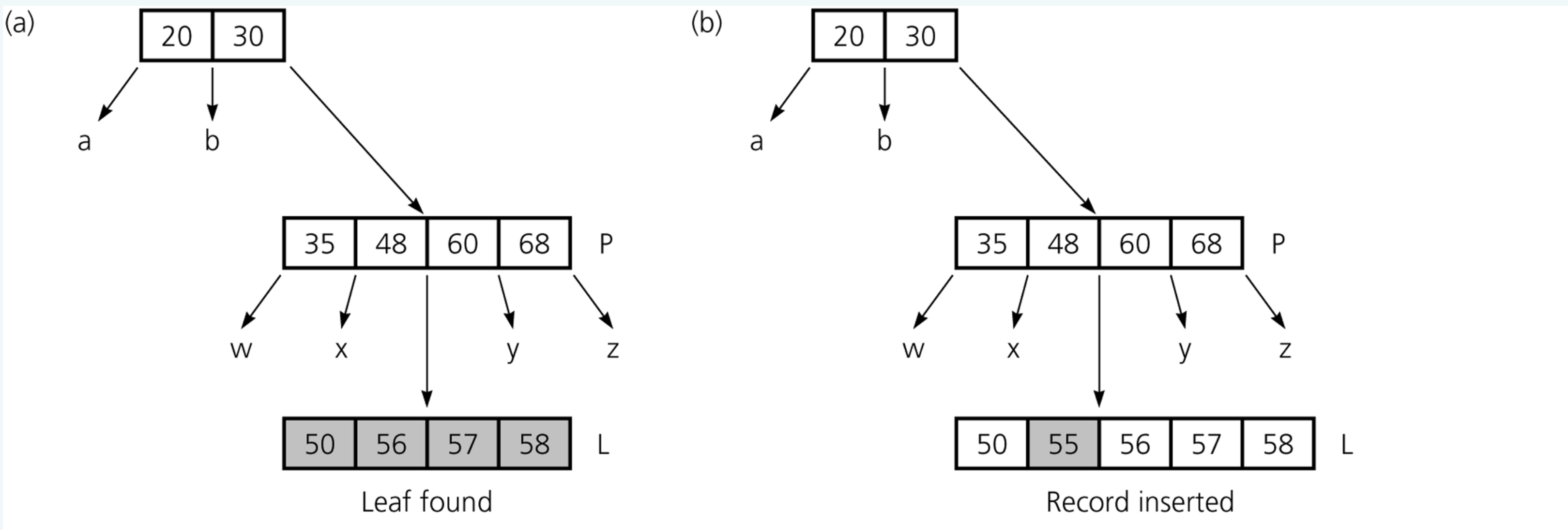


Figure 15-14a and b

The steps for inserting 55

B-Trees

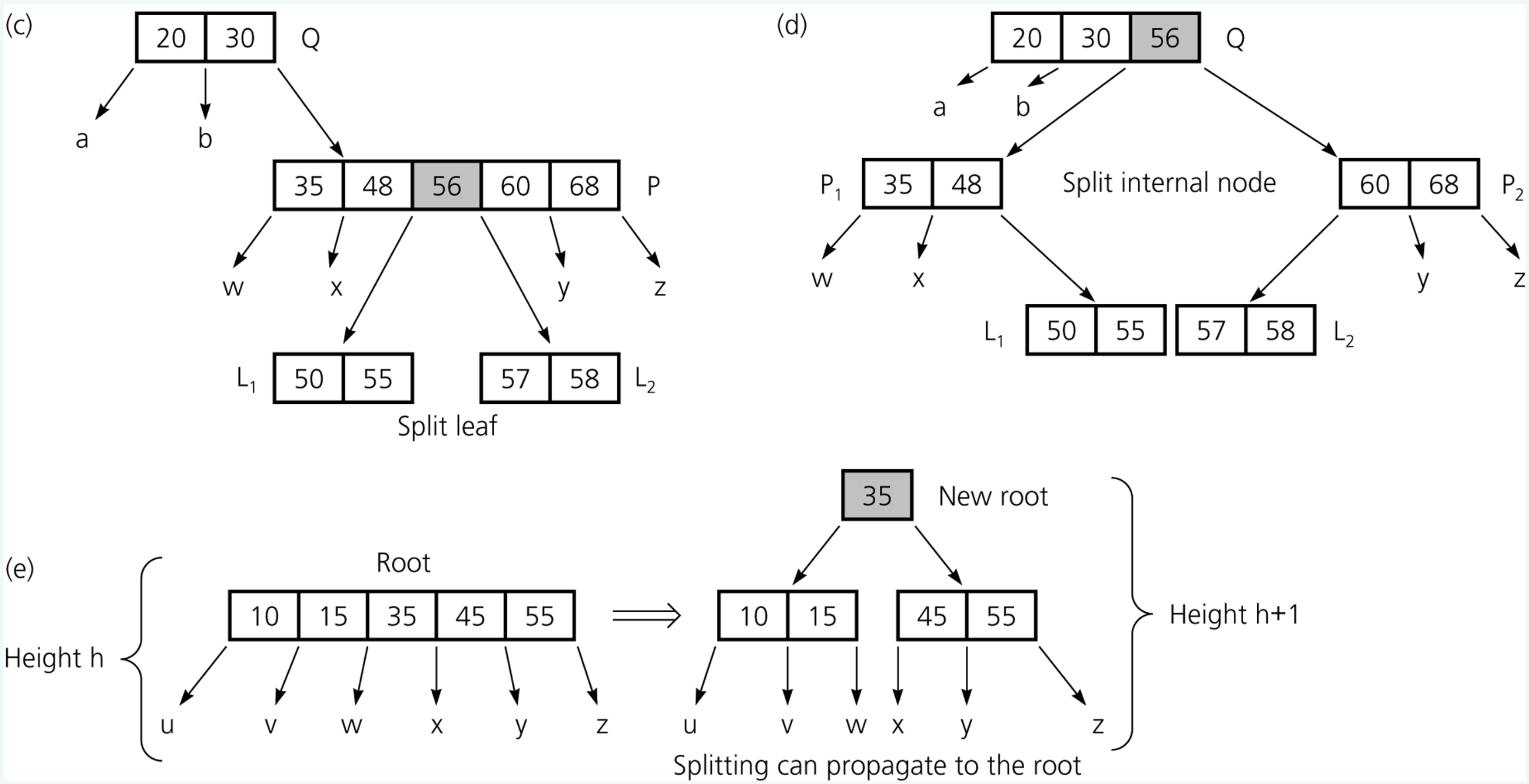


Figure 15-14c-e

The steps for inserting 55

B-Trees

- Deletion from a B-tree
  - Step 1: Locate the index record in the index file
  - Step 2: Delete the data record from the data file

B-Trees

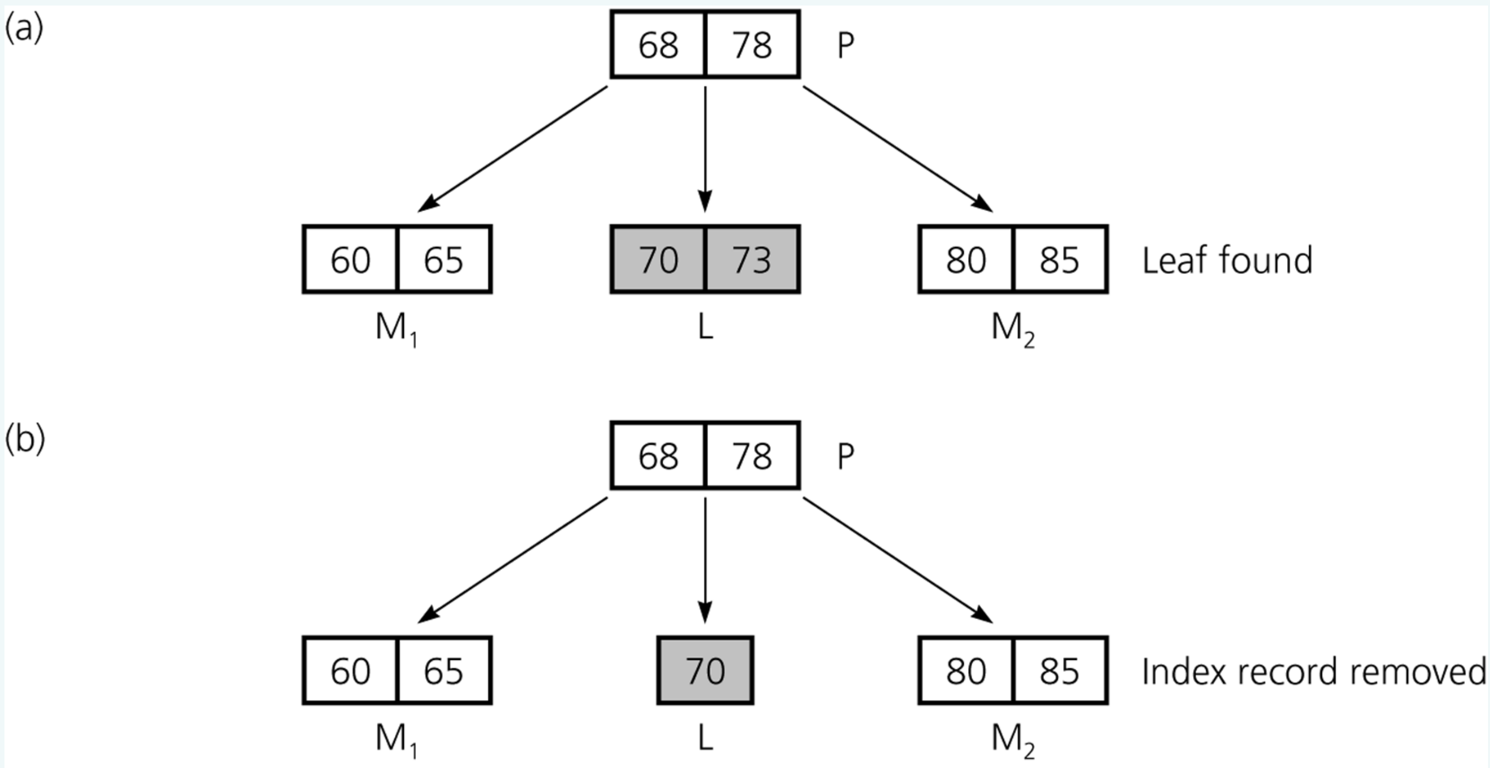


Figure 15-15a and b

The steps for deleting 73

B-Trees

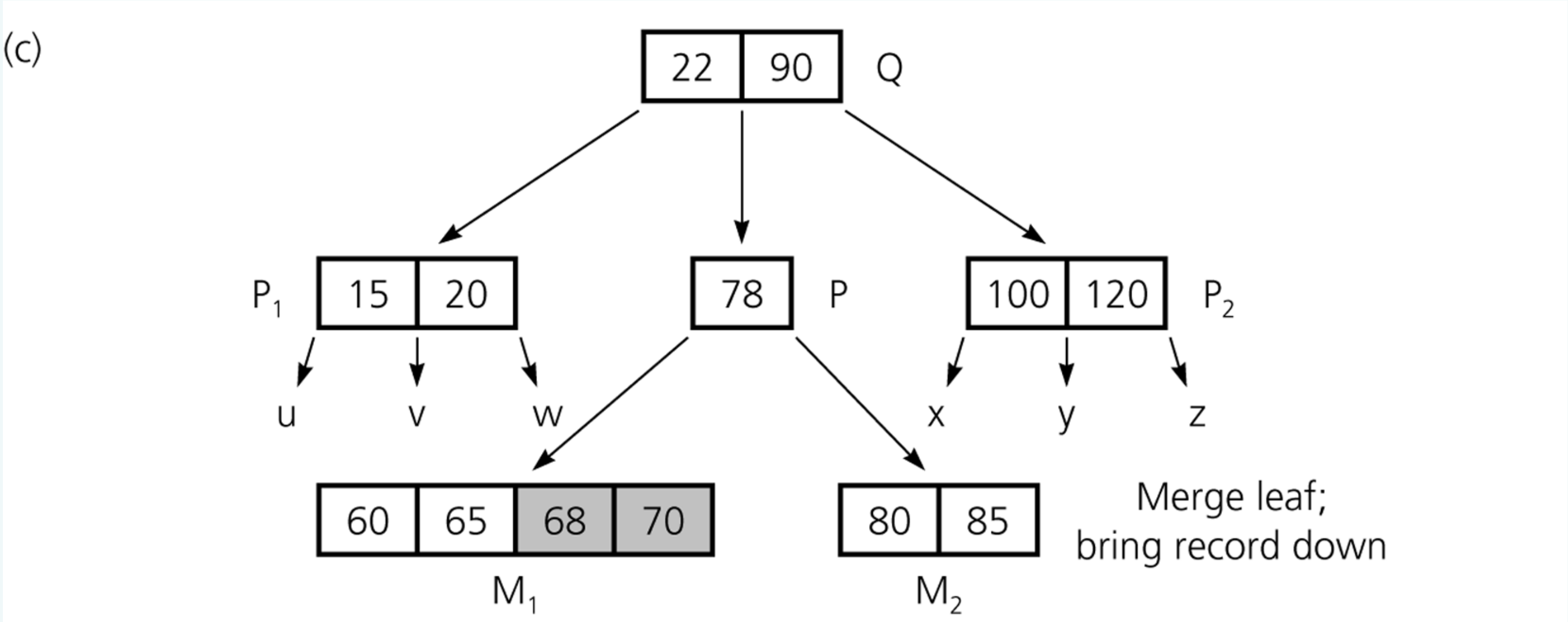


Figure 15-15c

The steps for deleting 73

B-Trees

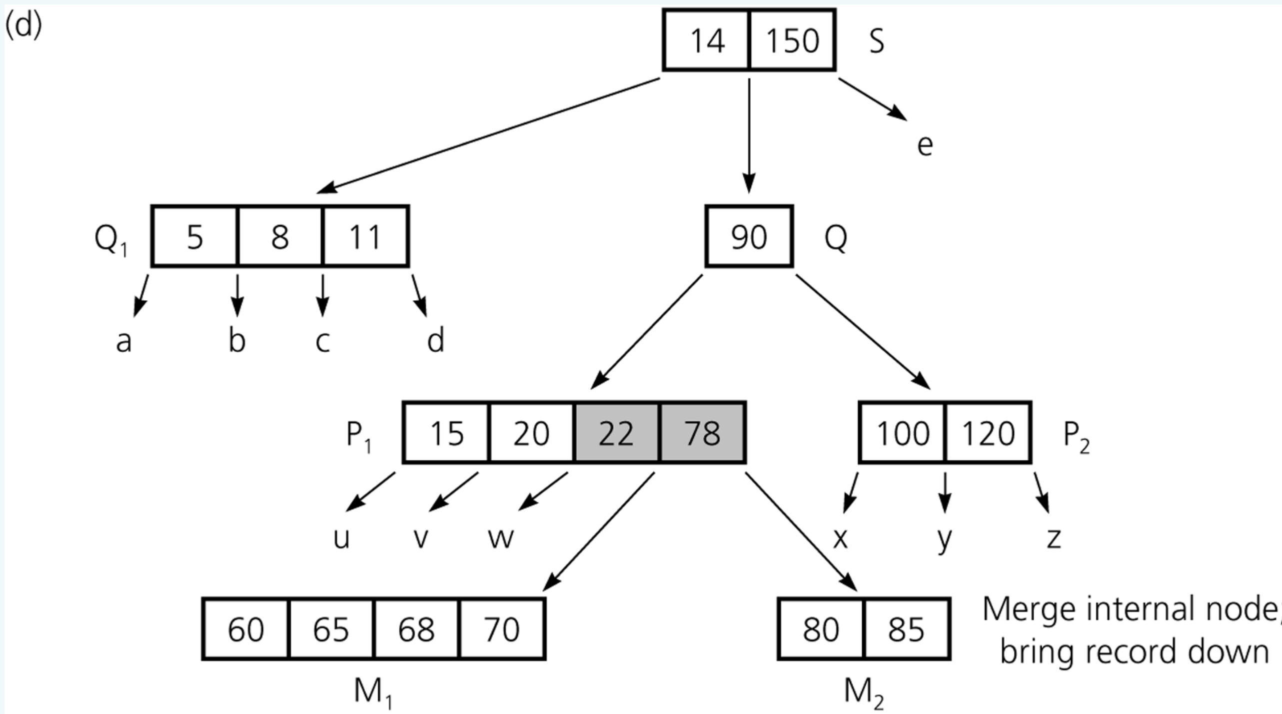


Figure 15-15d

The steps for deleting 73

B-Trees

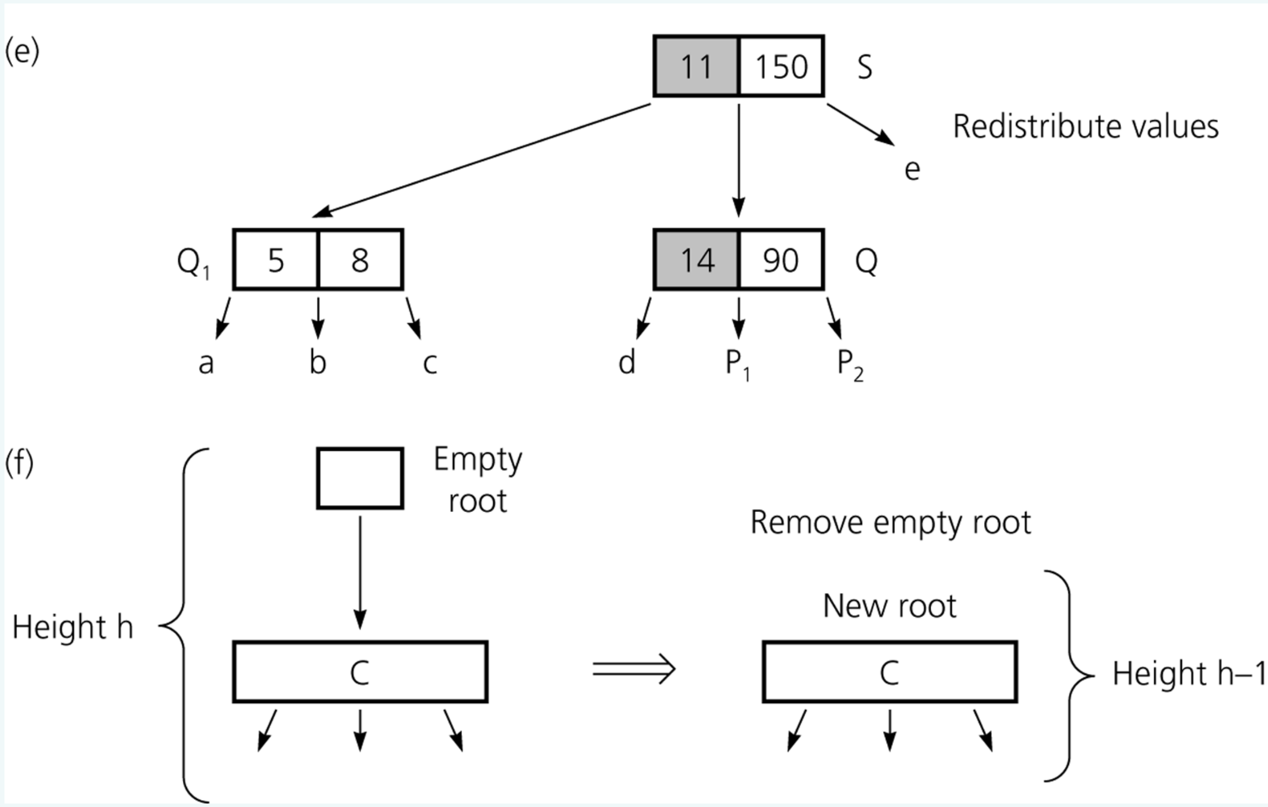


Figure 15-15e and f

The steps for deleting 73

Traversals

- Accessing only the search key of each record, not the data file
  - Not efficiently supported by the hashing implementation
  - Efficiently supported by the B-tree implementation
    - The search keys can be visited in sorted order by using an inorder traversal of the B-tree
- Accessing the entire data record
  - Not efficiently supported by the B-tree implementation

Multiple Indexing

- Advantage
  - Allows multiple data organizations
- Disadvantage
  - More storage space
  - Additional overhead for updating each index whenever the data file is modified

Multiple Indexing

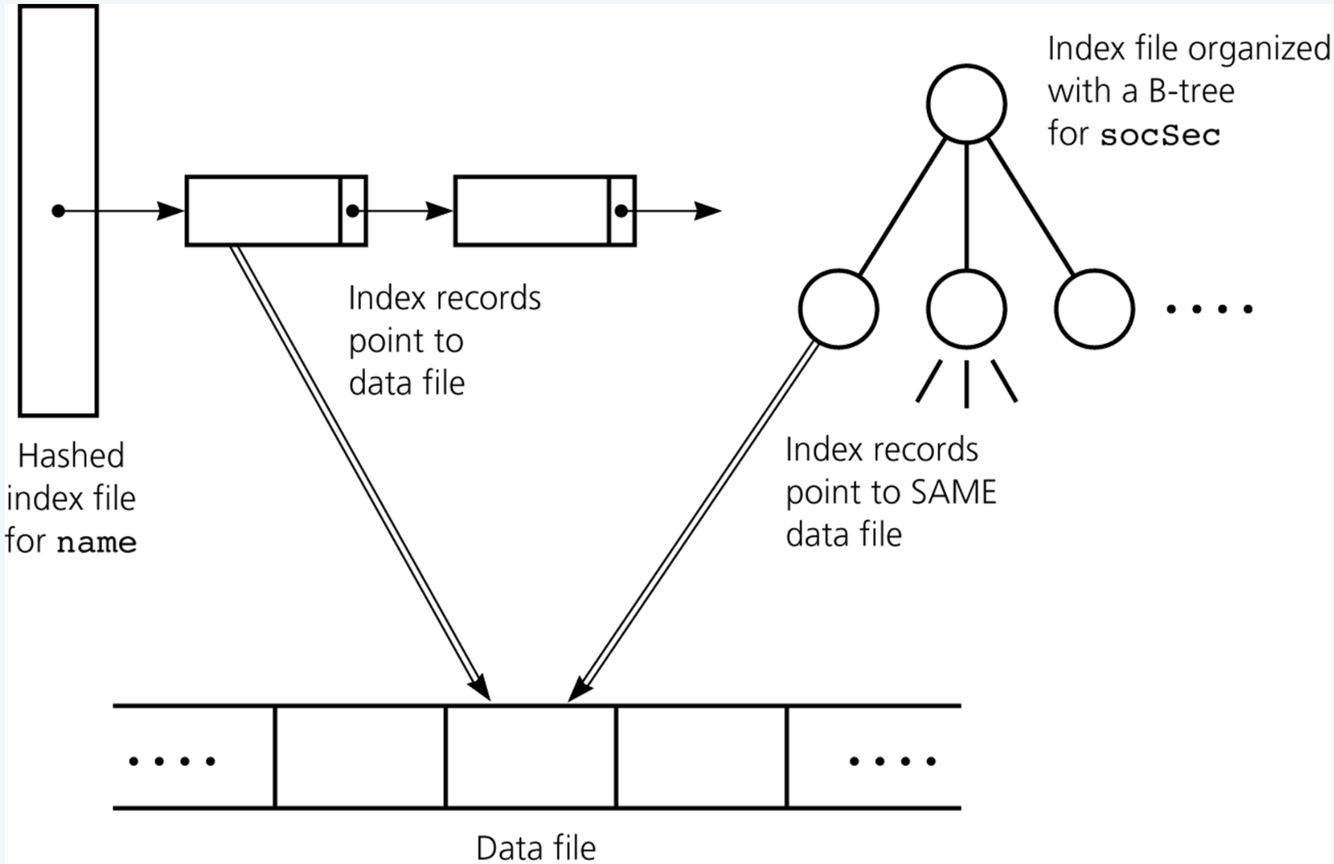


Figure 15-16

Multiple index files

Summary

- An external file is partitioned into blocks
  - Each block typically contains many data records
  - A block is generally the smallest unit of transfer between internal and external memory
- In a random access file, the i^{th} block can be accessed without accessing the blocks that precede it
- A modified mergesort algorithm can be used to sort an external file of records

Summary

- An index to a data file is a file that contains an index record for each record in the data file
- The index file can be organized using either hashing or a B-tree
  - These schemes allow you to perform the basic table operations by using only a few block accesses
- Several index files can be used with the same data file to perform different types of operations efficiently