11 B-Trees

Copyright © 2004, Binnur Kurt

Content

- Introduction to multilevel indexing and B-trees
- Insertion in B Trees
- `Search` and `Insert` Algorithms
- Deletion in B Trees

Introduction to Multilevel Indexing and B-Trees

- Problems with simple indexes that are kept in disk:

1. Seeking the index is still slow (binary searching):
   - We do not want more than 3/4 seeks for a search
   - So, here $\log_2(N+1)$ is still slow:

N	$\log_2(N+1)$
15 keys	4
1,000	~10
100,000	~17
1,000,000	~20

Introduction to Multilevel Indexing and B-Trees

- Problems with simple indexes that are kept in disk:

2. Insertions and deletions should be as fast as searches:
   - In simple indexes, insertion or deletion take $O(n)$ disk accesses (since index should be kept sorted)

Indexing with Binary Search Trees

- ▶ We could use balanced binary search trees:
- ▶ **AVL** Trees
  - Worst-case search is $1.44 \times \log_2(N+2)$
  - 1,000,000 keys → 29 Levels
  - Still prohibitive
- ▶ **Paged Binary** Trees
  - Place subtree of size K in a single page
  - Worst-case search is $\log_{K+1}(N+1)$
  - K=511, N=134,217,727
  - Binary trees: 27 seeks, Paged Binary tree: 3 seeks
  - This is good but there are lots of difficulties in maintaining (doing insertions and deletions) in a paged binary tree

Multilevel Indexing

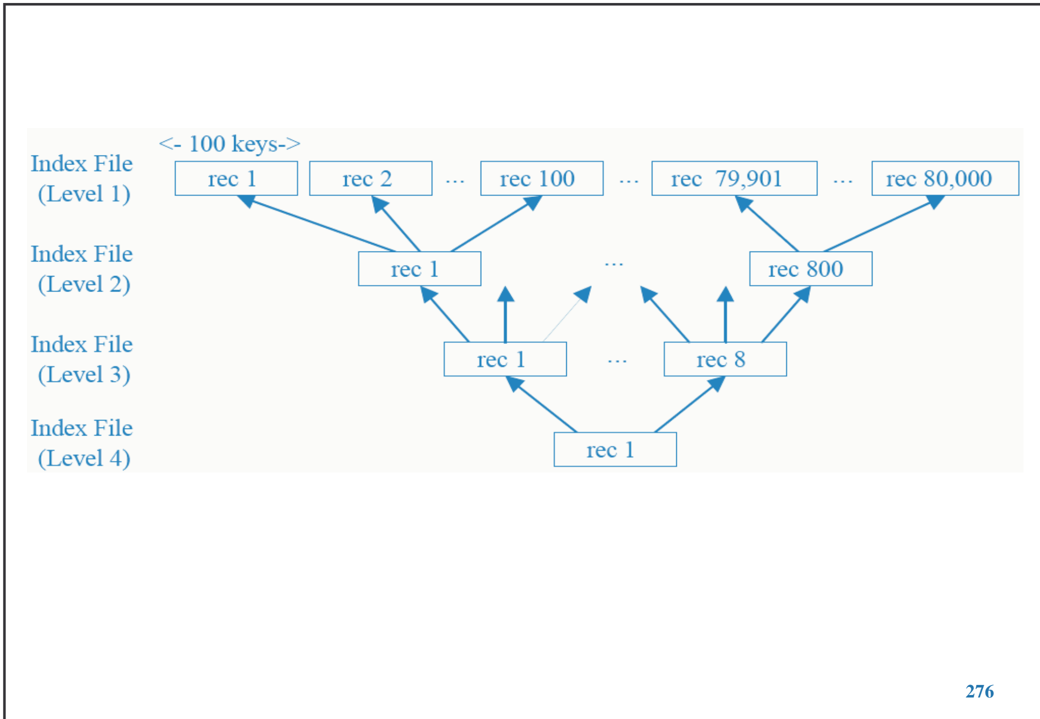
- ▶ Consider our 8,000,000 example with keysize =10B
- ▶ Index file size = 80 MB
- ▶ Each record in the index will contain 100 pairs (key,reference)
- ▶ A simple index would contain: 80,000 records:

  Too expensive to search (~16 seeks)

Multilevel Index

- Build an index of an index file
- How?
  - Build a simple index for the file, sorting keys using the method for external sorting previously studied
  - Build an index for this index
  - Build another index for the previous index, and so on
  - The index of an index stores the largest in the record it is pointing to.

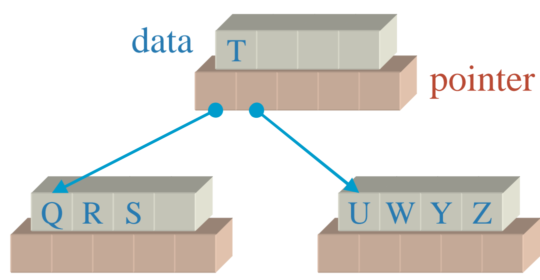
B-Trees 11



B-Trees

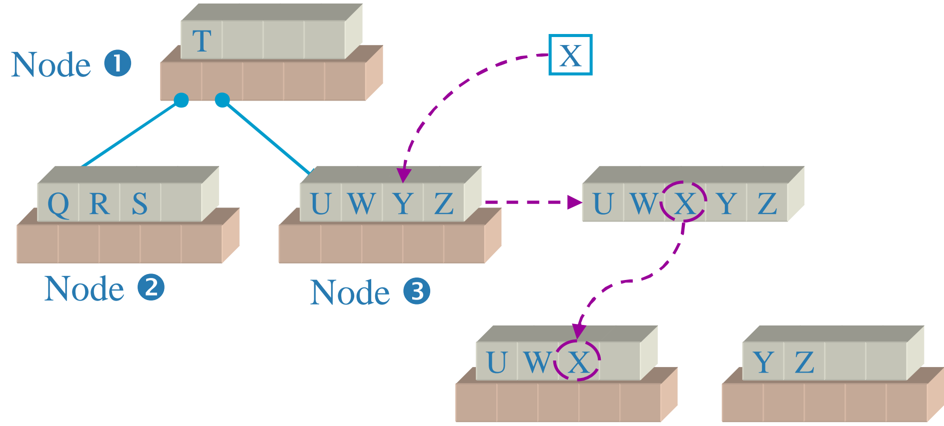
- Again an index record may contain 100 keys
- An index record may be half full (each index record may have from 50 to 100 keys)
- When insertion in an index record causes it to overfull
  - Split record in two
  - “Promote” the largest key in one of the records to the upper level

Example for Order = 4

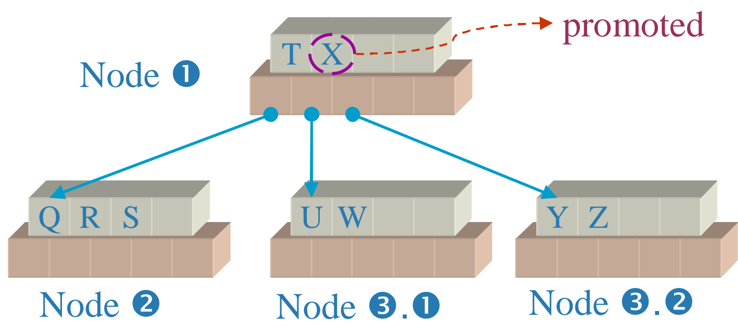


Inserting X

- X is between T and Z: insertion in node 3 splits it and generates a promotion of node X



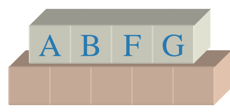
Promotion



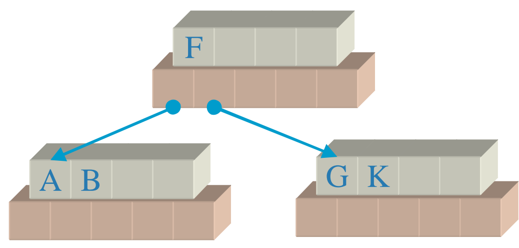
- Important: If Node 1 was full, this would generate a new split-promotion of Node 1. This could be propagated up to the root

Example of Insertions

- Inserting keys: order = 4
- A,G,F,B,K,D,H,M,J,E,S,I,R,X,C,L,N,T,U,P

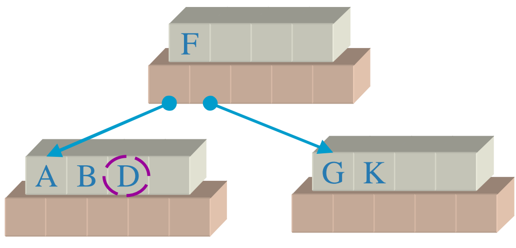


- Inserting K: Split and Promotion

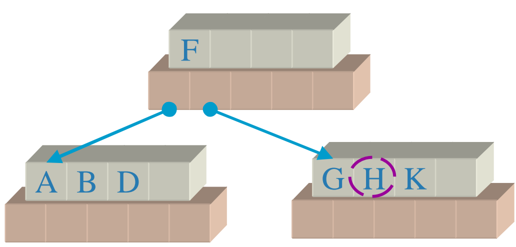


Insertion Example

- Inserting D

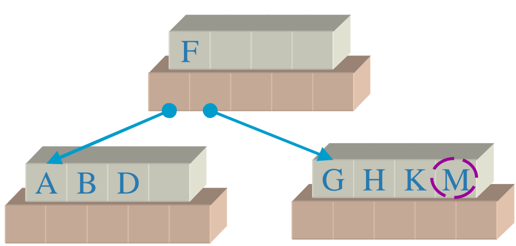


- Inserting H

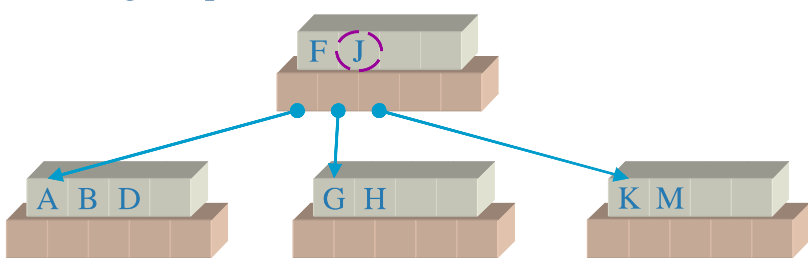


Insertion Example

- ▶ Inserting M

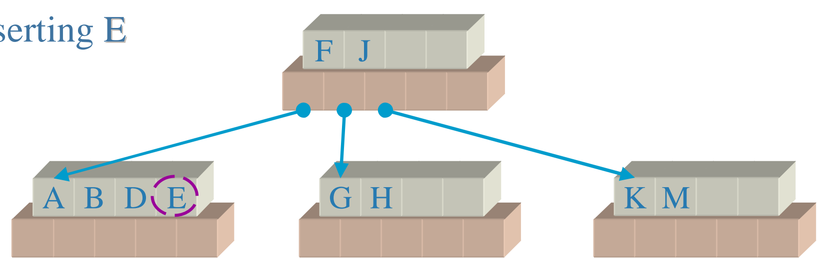


- ▶ Inserting J: Split and Promotion

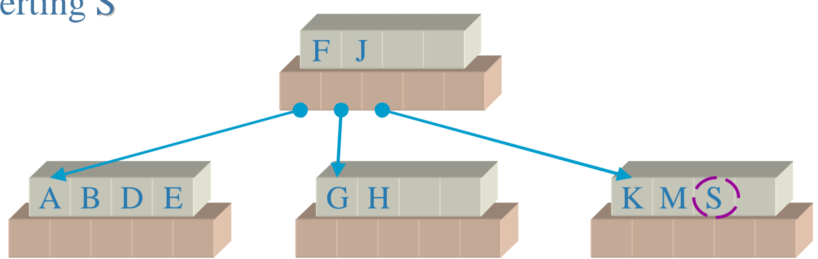


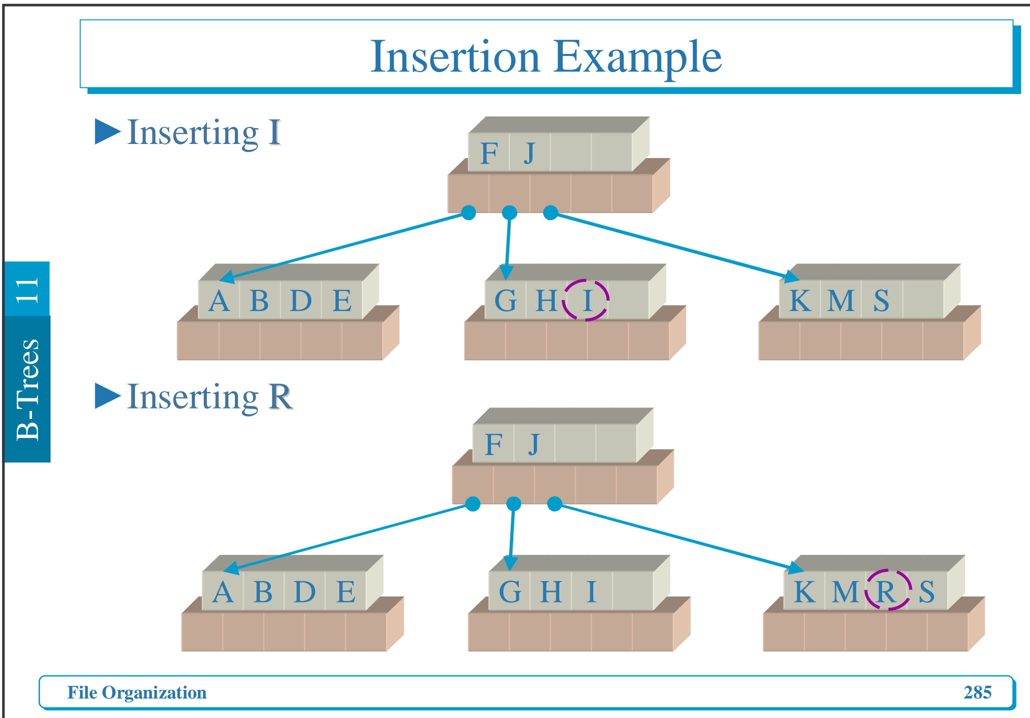
Insertion Example

- ▶ Inserting E

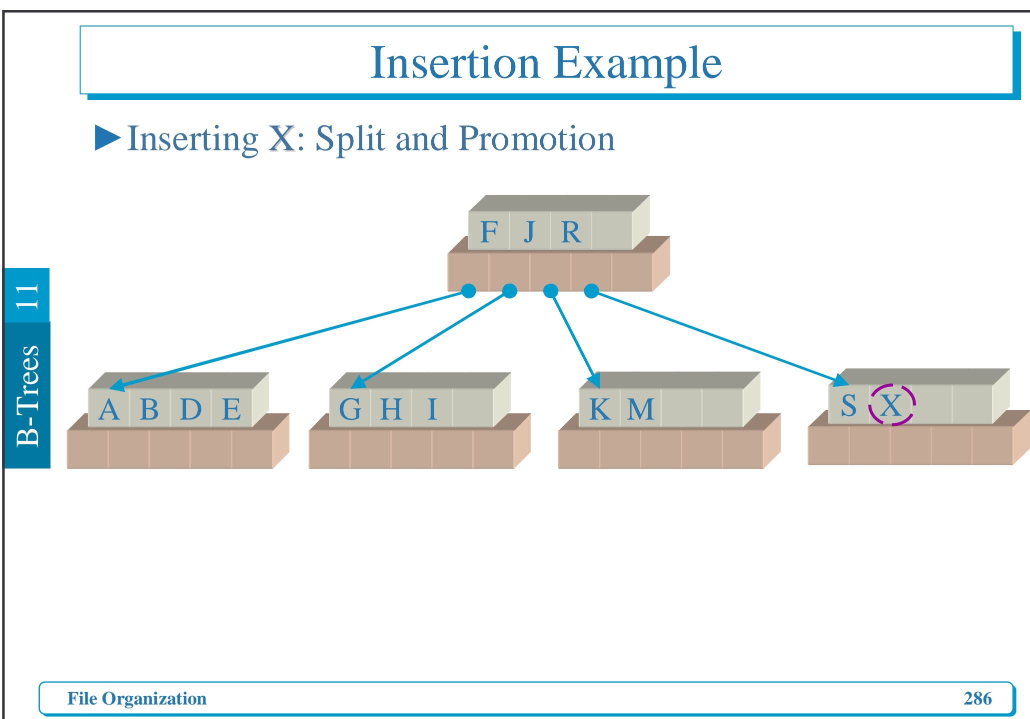


- ▶ Inserting S

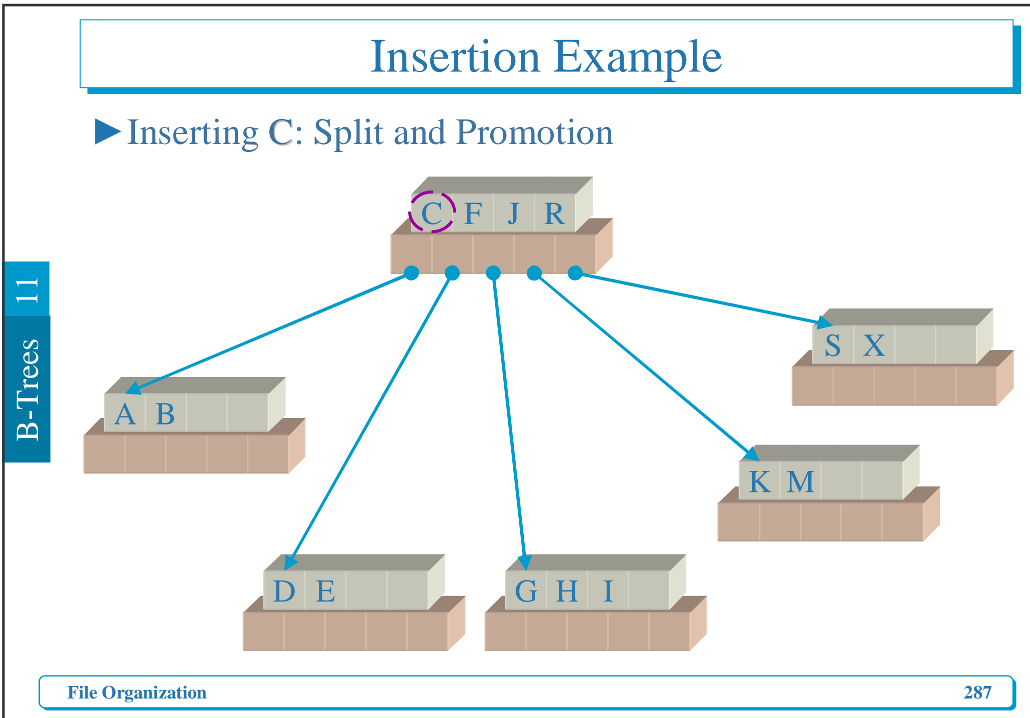


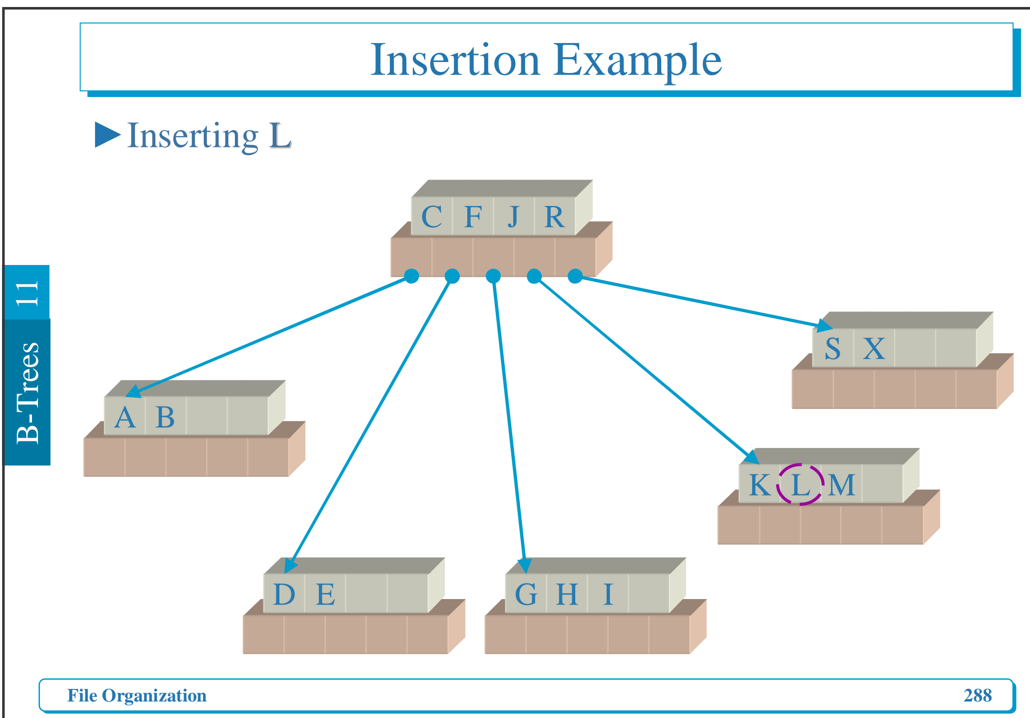


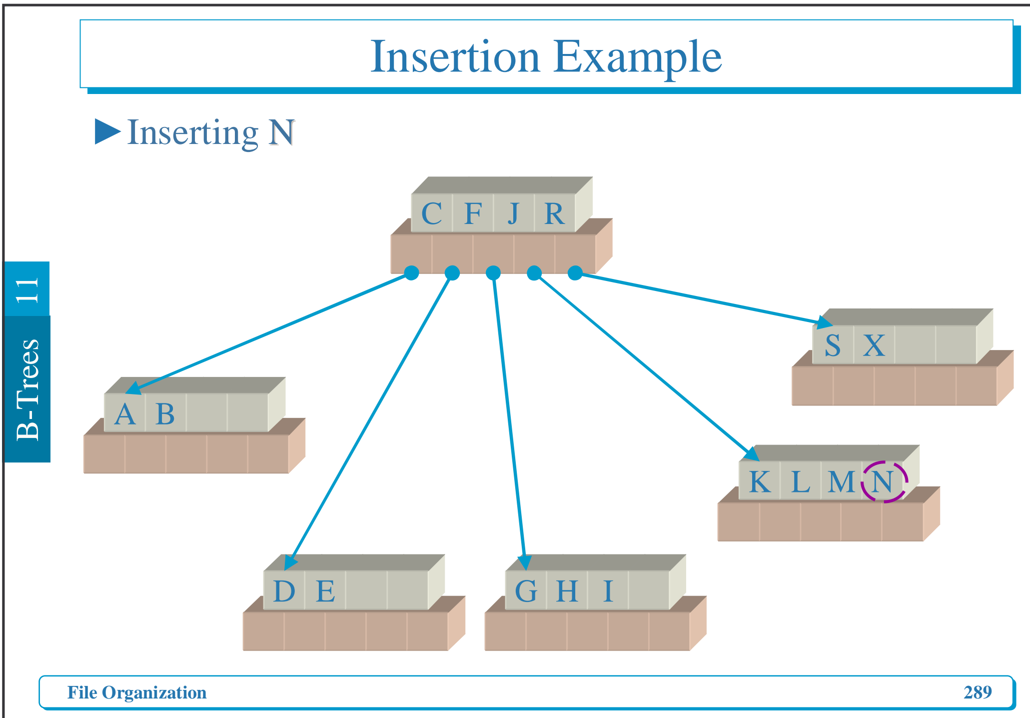
Insertion Example
Inserting I
F J
A B D E
G H I
K M S
Inserting R
F J
A B D E
G H I
K M R S
11
B-Trees
File Organization
285

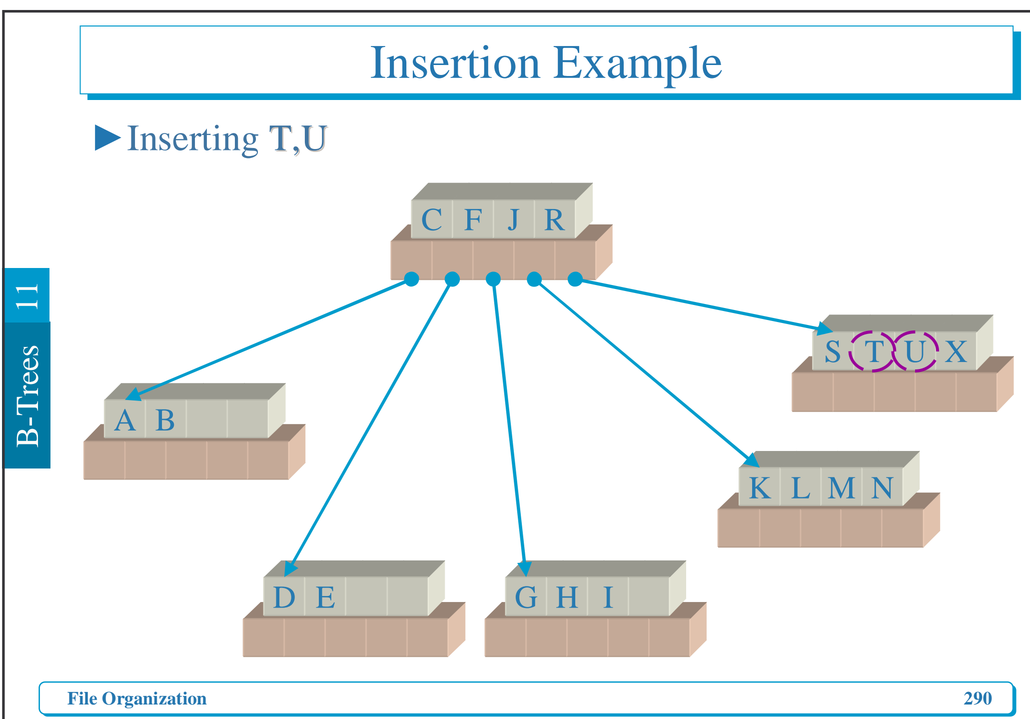


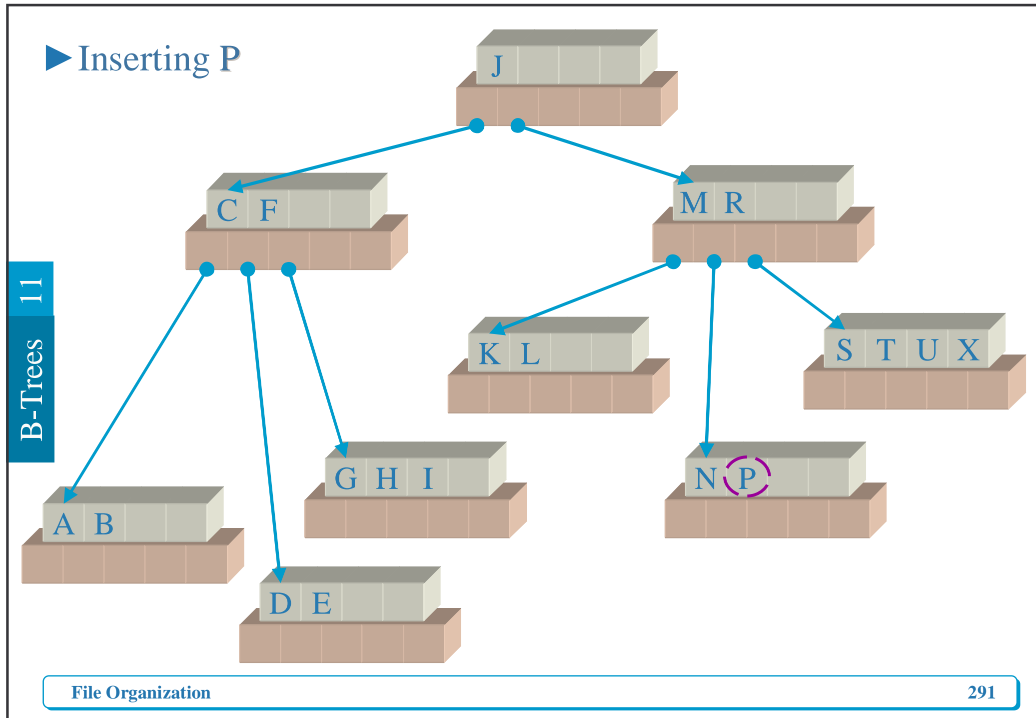
Insertion Example
Inserting X: Split and Promotion
F J R
A B D E
G H I
K M
S X
11
B-Trees
File Organization
286


Insertion Example
▶ Inserting C: Split and Promotion
C F J R
A B
D E
G H I
K M
S X
11
B-Trees
File Organization
287


Insertion Example
▶ Inserting L
C F J R
A B
D E
G H I
K L M
S X
11
B-Trees
File Organization
288


Insertion Example
Inserting N
C F J R
A B
D E
G H I
K L M N
S X
B-Trees 11
File Organization
289


Insertion Example
Inserting T,U
C F J R
A B
D E
G H I
K L M N
S T U X
B-Trees 11
File Organization
290



B-Tree Properties

- Properties of a B-tree of order *m*:

1. Every node has a maximum of m children
2. Every node, except for the root and the leaves, has at least *m*/2 children
3. The root has at least two children (unless it is a leaf)
4. All the leafs appear on the same level
5. The leaf level forms a complete index of the associated data file

Worst-case Search Depth

- The worst-case depth occurs when every node has the minimum number of children

Level	Minimum number of keys (children)
1 (root)	2
2	$2 \cdot \lceil m/2 \rceil$
3	$2 \cdot \lceil m/2 \rceil \cdot \lceil m/2 \rceil = 2 \cdot \lceil m/2 \rceil^2$
4	$2 \cdot \lceil m/2 \rceil^3$
...	...
d	$2 \cdot \lceil m/2 \rceil^{d-1}$

Example

- Assume that we have *N* keys in the leaves

$$N \geq 2 \cdot (m/2)^{d-1}$$

So,

$$d \leq 1 + \log_{m/2}(N/2)$$

For *N*=1,000,000 and order *m*=512, we have

$$d \leq 1 + \log_{256}(N/2)$$

$$d \leq 3.37$$

- There is at most 3 levels in a B-tree of order 512 holding 1,000,000 keys.

B-Trees 11

Outline of Search Algorithm

- Search (KeyType key)

1. Find leaf: find the leaf that could contain key, loading all the nodes in the path from root to leaf into an array in main memory
2. Search for key in the leaf which was loaded into main memory

B-Trees 11

Deletions from B-Tree

- The rules for deleting a key K from a node n :

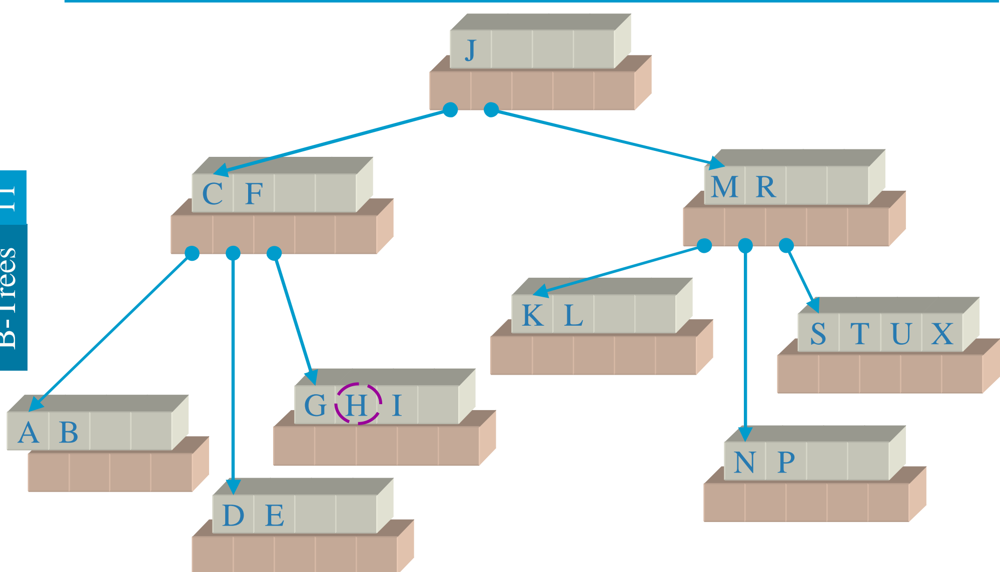
1. If n has more than the minimum number of keys and K is not the largest key in n, simply delete K from n.
2. If n has more than the minimum number of keys and K is the largest key in n, delete K from n and modify the higher level indexes to reflect the new largest key in n.
3. If n has exactly the minimum number of keys and one of the siblings has “few enough keys”, **merge** n with its sibling and delete a key from the parent node

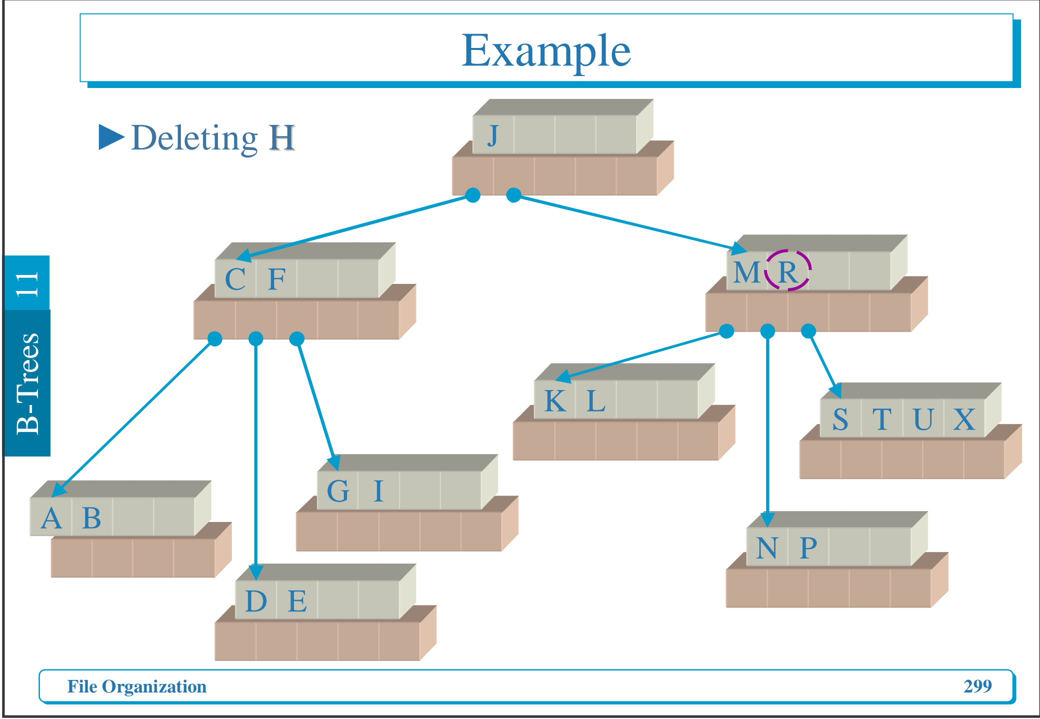
Deletions from B-Tree (Con't)

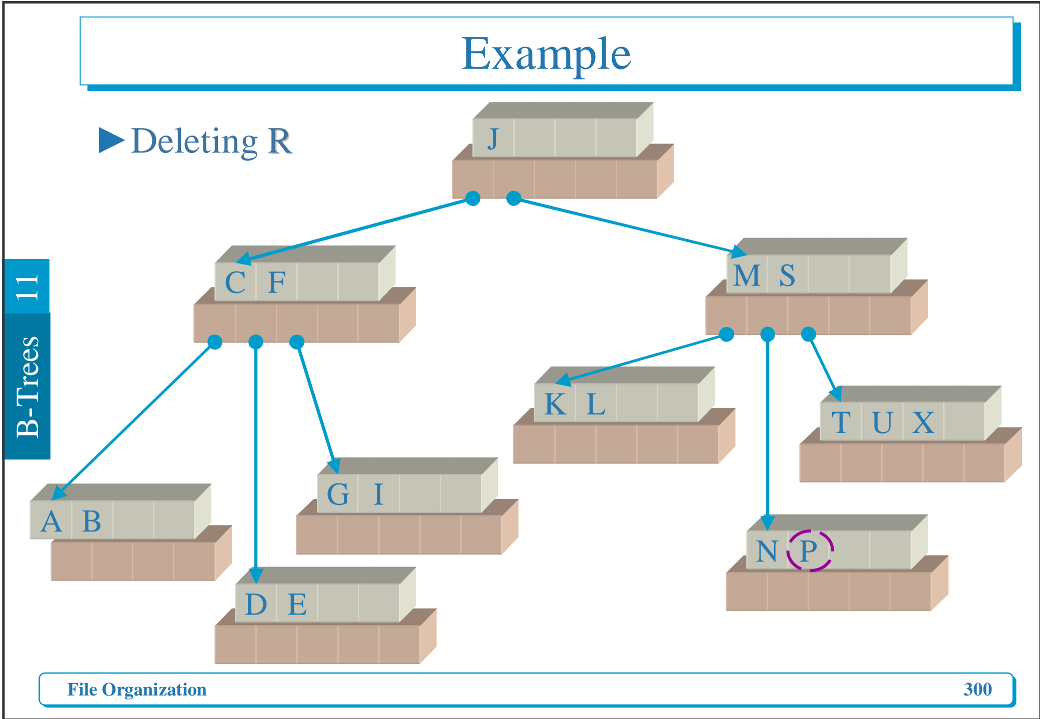
- The rules for deleting a key *K* from a node *n* :

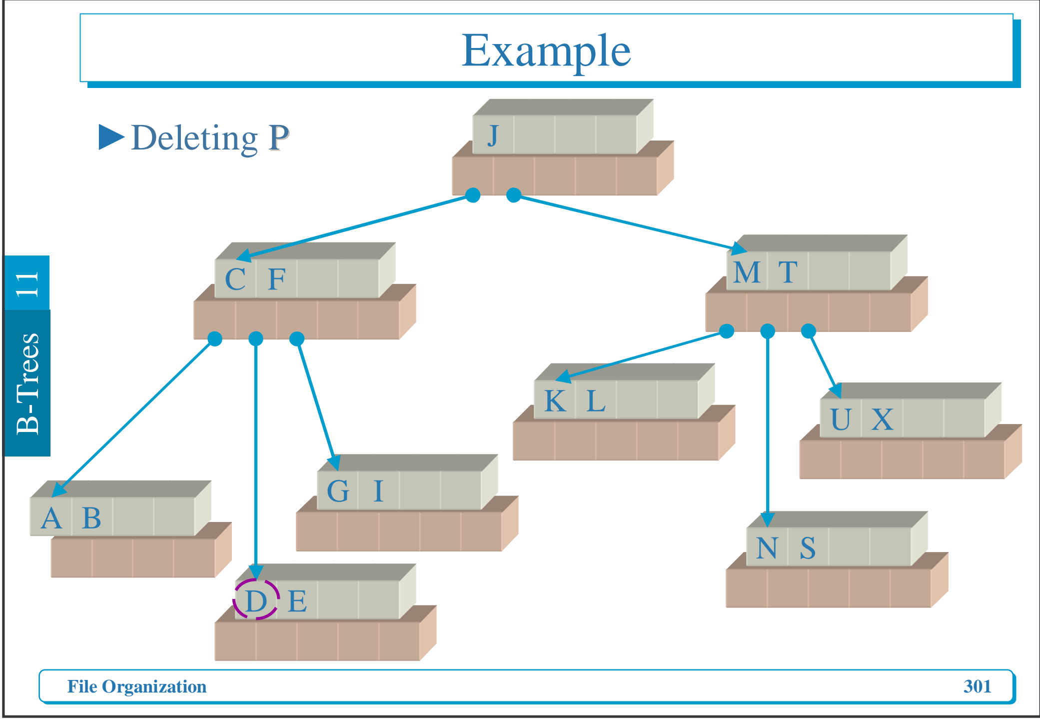
4. If *n* has exactly the minimum number of keys and one of the siblings has extra keys, **redistribute** by moving some keys from a sibling to *n*, and modify higher levels to reflect the new largest keys in the affected nodes

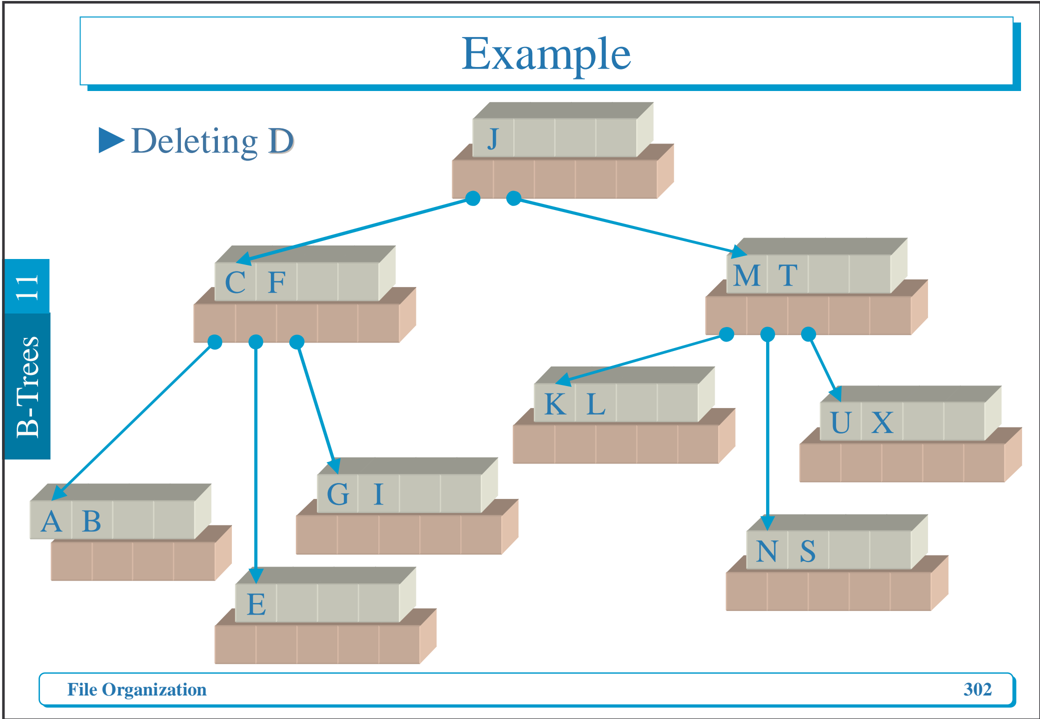
Example

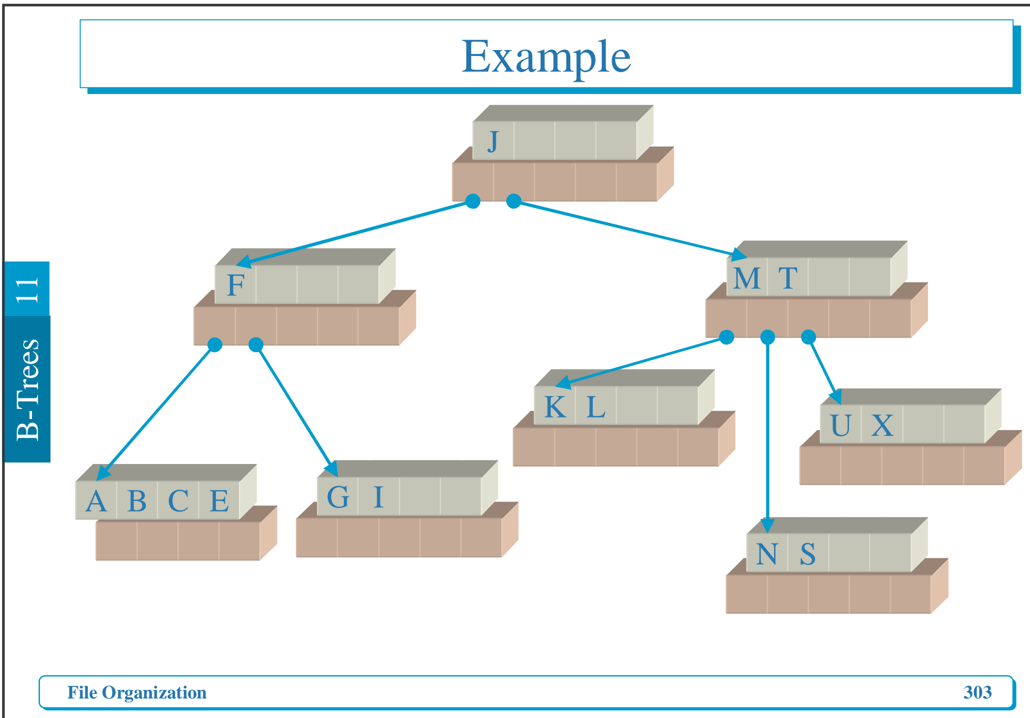


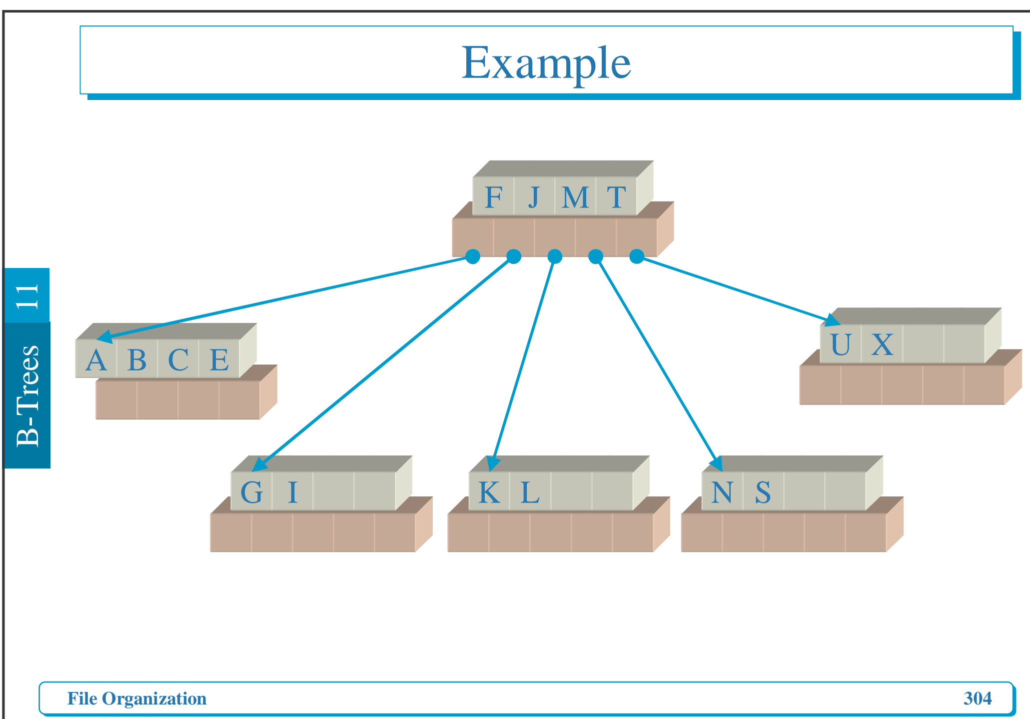

Example
Deleting H
J
C F
M R
A B
D E
G I
K L
N P
S T U X
B-Trees
11
File Organization
299


Example
Deleting R
J
C F
M S
A B
D E
G I
K L
N P
T U X
B-Trees
11
File Organization
300


Example
Deleting P
J
C F
M T
K L
U X
A B
G I
N S
D E
B-Trees
11
File Organization
301

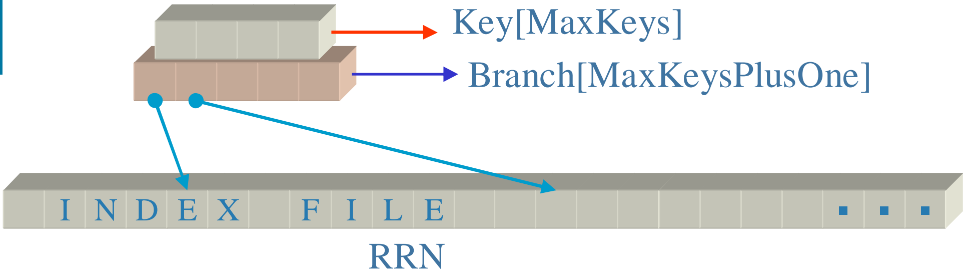

Example
Deleting D
J
C F
M T
K L
U X
A B
G I
N S
E
B-Trees
11
File Organization
302


Example
J
F
M T
K L
U X
A B C E
G I
N S
11
B-Trees
File Organization
303


Example
F J M T
A B C E
U X
G I
K L
N S
11
B-Trees
File Organization
304

Implementation

```
typedef struct {
    int Count; // Number of keys stored in the current
    ItemType Key[MaxKeys];
    long Branch[MaxKeysPlusOne];
} NodeType;
```



B-Trees 11

Implementation (Con't)

```
typedef struct {
    int Count; // Number of keys stored in the current
    ItemType Key[MaxKeys];
    long Branch[MaxKeysPlusOne];
    bool Search(KeyFieldType SearchKey,
                ItemType & Item);
} NodeType;
```


B-Trees 11

bool Search(KeyFieldType SearchKey,...)

```
{
  long CurrentRoot;
  int Location;
  bool Found;
 Found = false;
 CurrentRoot = Root;
 while ((CurrentRoot != -1L) && (!Found)) {
   fseek(hFile,CurrentRoot * NodeSize, SEEK_SET);
   fread((unsigned char *)(&CurrentNode),NodeSize,1,hFile);
   if (SearchNode(SearchKey, Location)) {
     Found = true;
     Item = CurrentNode.Key[Location];
   } else   CurrentRoot = CurrentNode.Branch[Location + 1];
 }
 return Found;
}
```


bool SearchNode(KeyFieldType Target,...)

```
bool SearchNode(const KeyFieldType Target, int & Location) const {
  bool Found=false;
  if (strcmp(Target, CurrentNode.Key[0].KeyField)<0)  Location=-1L;
  else
  {
     Location = CurrentNode.Count - 1;
     while ((strcmp(Target, CurrentNode.Key[Location].KeyField) < 0)
              && (Location > 0))
        Location--;
        if (strcmp(Target, CurrentNode.Key[Location].KeyField) == 0)
          Found = true;
       }
  return Found;
}
```