CS 764: Topics in Database Management Systems

Lecture 9: B-tree Locking

Xiangyao Yu

10/5/2020

Today's Paper: B-tree Locking

Efficient Locking for Concurrent Operations on B-Trees

PHILIP L. LEHMAN
Carnegie-Mellon University
and
S. BING YAO
Purdue University

The B-tree and its variants have been found to be highly useful (both theoretically and in practice) for storing large amounts of information, especially on secondary storage devices. We examine the problem of overcoming the inherent difficulty of concurrent operations on such structures, using a practical storage model. A single additional "link" pointer in each node allows a process to easily recover from tree modifications performed by other concurrent processes. Our solution compares favorably with earlier solutions in that the locking scheme is simpler (no read-locks are used) and only a (small) constant number of nodes are locked by any update process at any given time. An informal correctness proof for our system is given.

Key Words and Phrases: database, data structures, B-tree, index organizations, concurrent algorithms, concurrency controls, locking protocols, correctness, consistency, multiway search trees
CR Categories: 3.73, 3.74, 4.32, 4.33, 4.34, 5.24

1. INTRODUCTION

The B-tree [2] and its variants have been widely used in recent years as a data structure for storing large files of information, especially on secondary storage devices [7]. The guaranteed small (average) search, insertion, and deletion time for these structures makes them quite appealing for database applications.

A topic of current interest in database design is the construction of databases that can be manipulated concurrently and correctly by several processes. In this

ACM Trans. Database Syst. 1981

Agenda

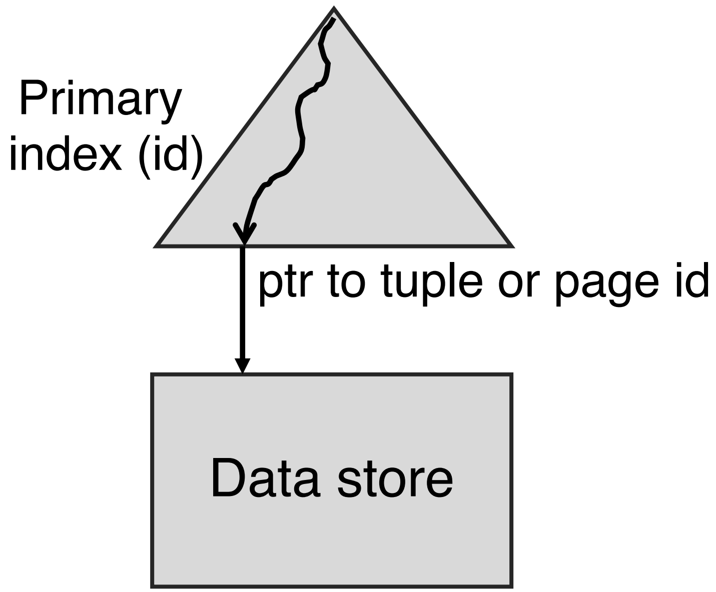
Index in OLTP database

B tree, B+ tree, and B* tree

B^{link}-tree

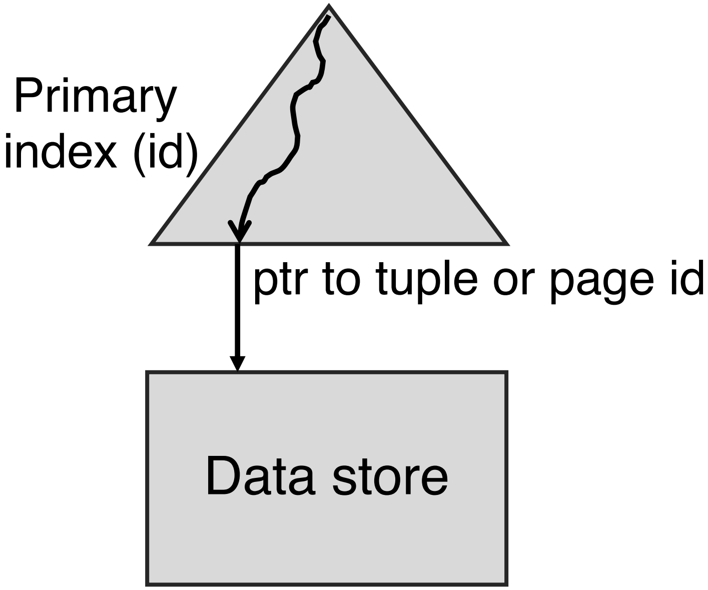
Index in an OLTP Database

Select name
From student
Where id=xxx

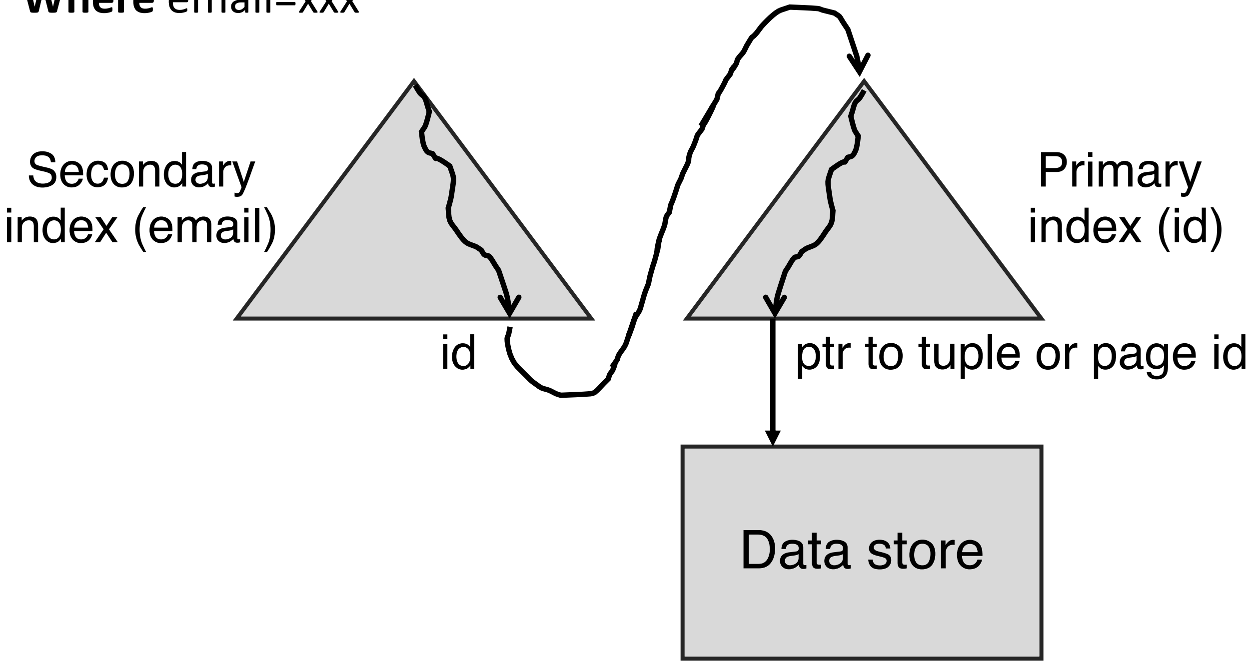


Index in an OLTP Database

Select name
From student
Where id=xxx



Select name
From student
Where email=xxx

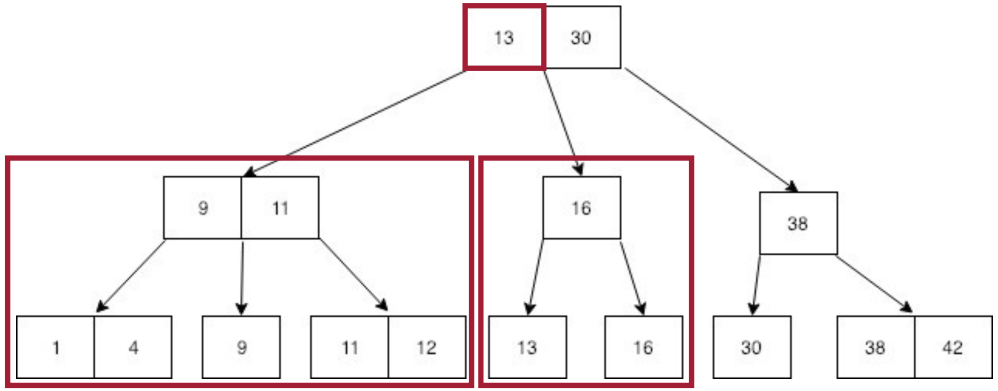


B-tree

Balanced tree data structure

- Data is sorted
- Supports: search, sequential scan, insets, and deletes

Algorithm	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)



B-tree

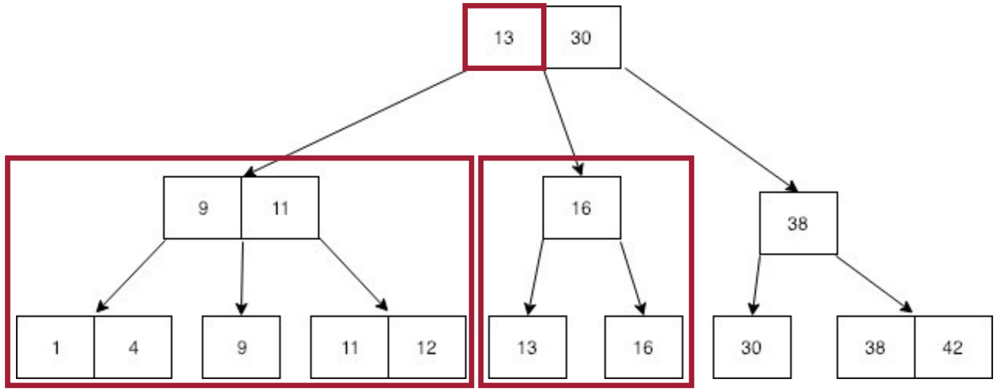
Balanced tree data structure

- Data is sorted
- Supports: search, sequential scan, inserts, and deletes

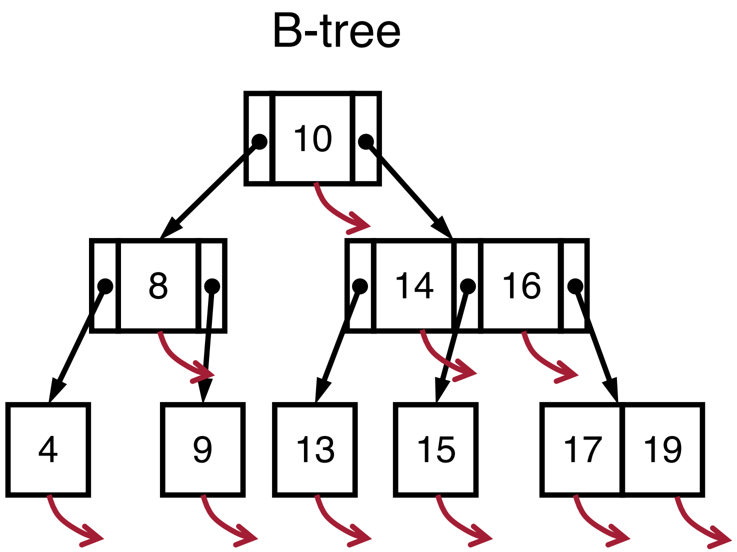
Properties

- Every node has at most *m* children.
- Every non-leaf node (except root) has at least ⌈*m*/2⌉ child nodes.
- All the leaf nodes of the B-tree must be at the same level.

Algorithm	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

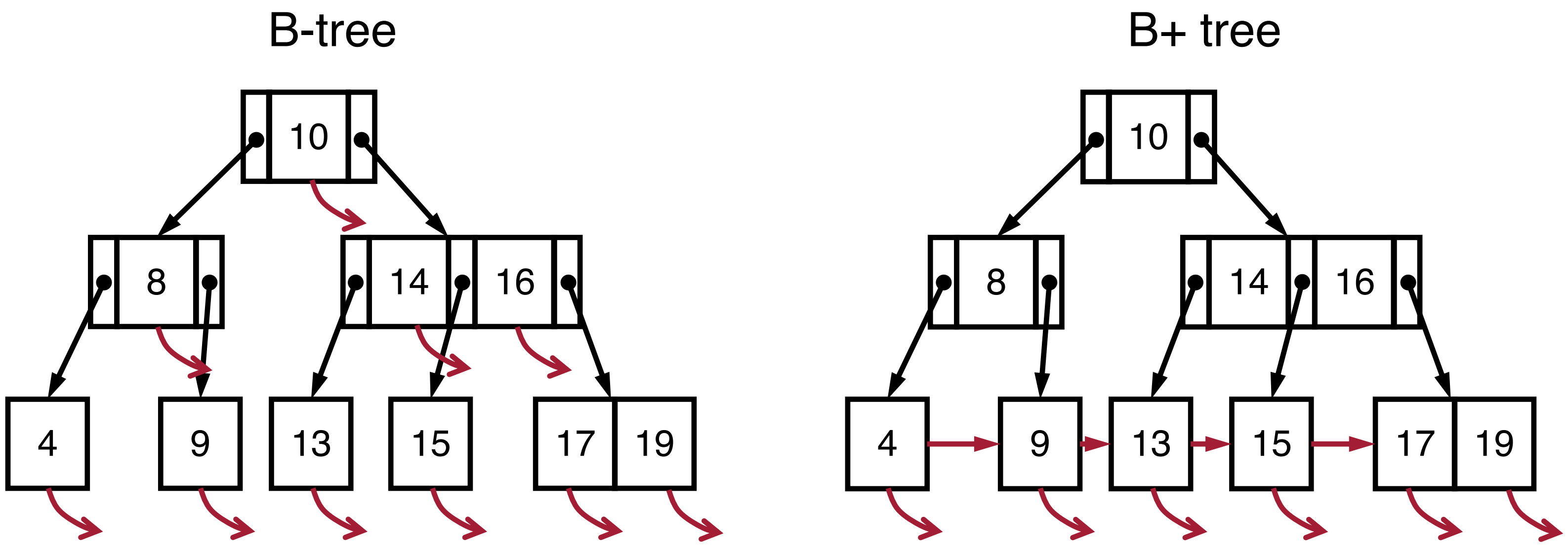


B-tree vs. B+ Tree vs. B* Tree



B-tree: data pointers stored in all nodes

B-tree vs. B+ Tree vs. B* Tree

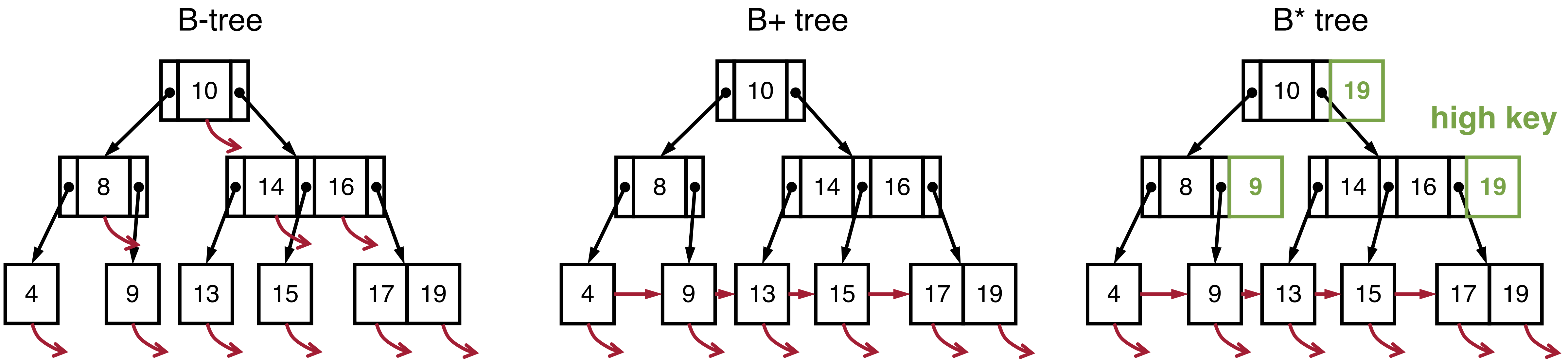


B-tree: data pointers stored in all nodes

B+ tree:

- Data pointers stored only in leaf nodes
- The leaf nodes are linked

B-tree vs. B+ Tree vs. B* Tree



B-tree: data pointers stored in all nodes

B+ tree:

- Data pointers stored only in leaf nodes
- The leaf nodes are linked

B* tree is a misused term in B-tree literature

- Typically means a variant of B+ tree in which each node is least 2/3 full
- In this paper: B+ tree with high key appended to non-leaf nodes (upper bound on values)

B* Tree Structure

Within each node, keys in ascending order

Each node contains at least *k* keys and at most *2k* keys (*k* is a tree parameter)

Values stored in a subtree are bounded by the the two key values

$$K_{i-1} < v \leq K_i$$

Example: search key 53

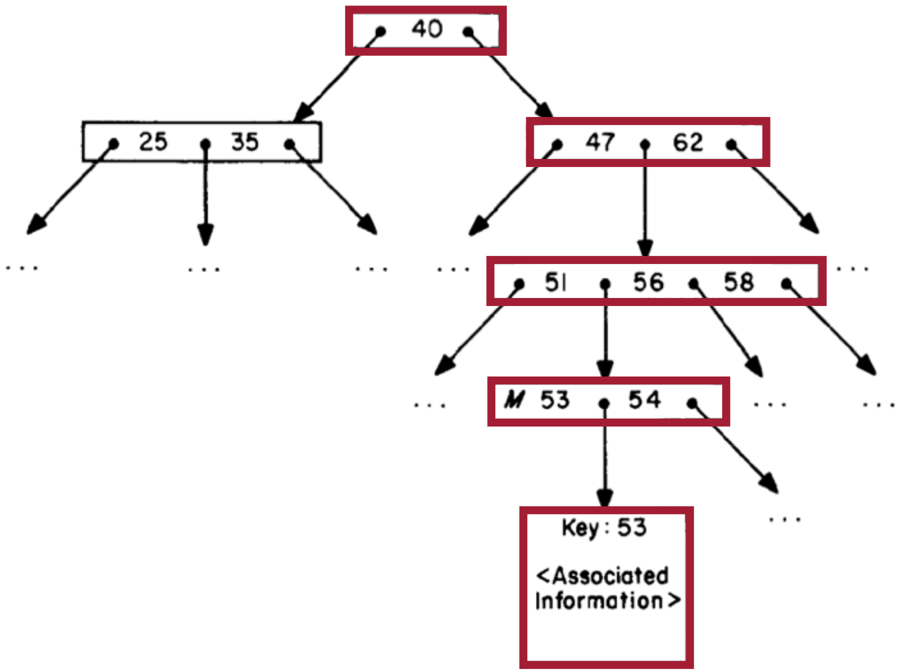


Fig. 2. An example B*-tree (with parameter $k = 2$).

B* Tree Insertion

Insert to leaf if the leaf node has fewer than *2k* entries

If leaf has *2k* entries, split the node into two nodes (split may happen recursively)

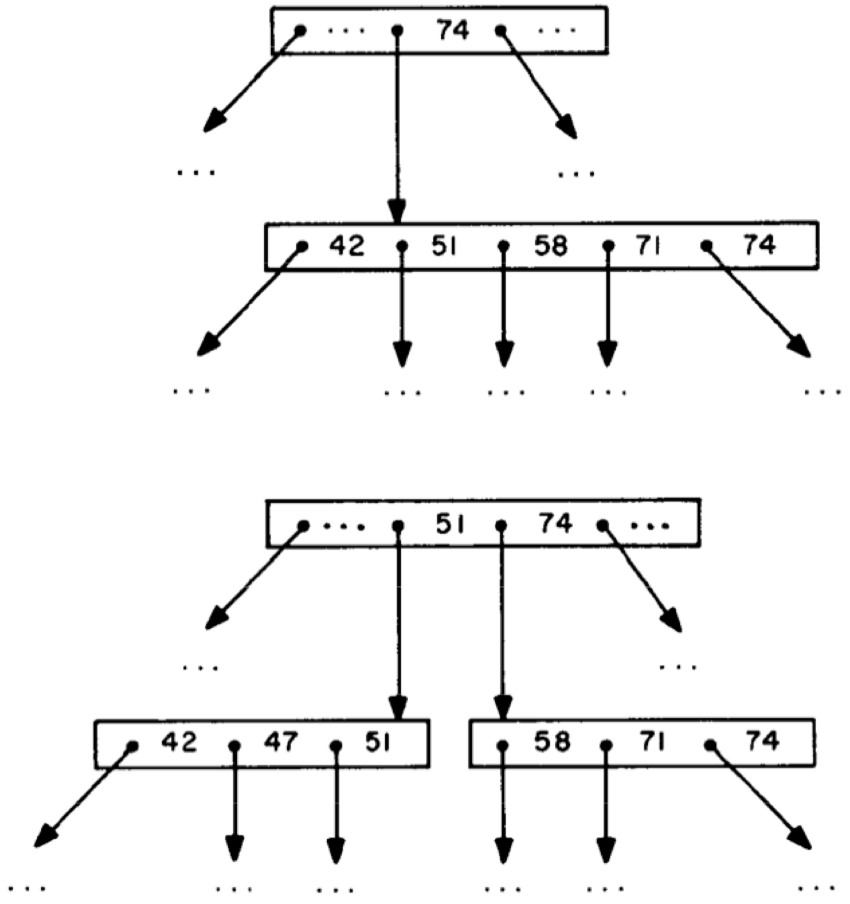
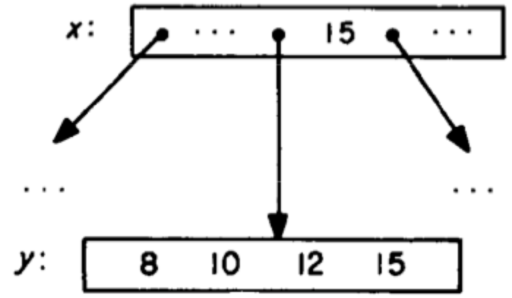


Fig. 4. Splitting a node after adding "47" ($k = 2$).

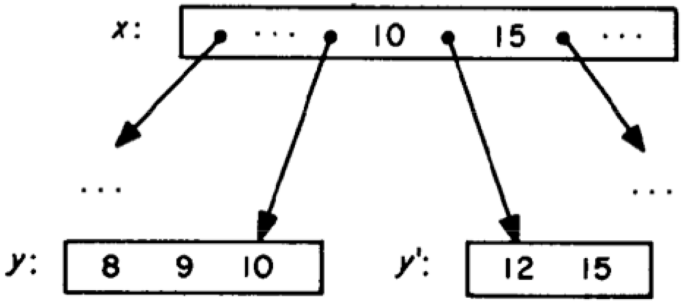
Challenge of Concurrent Operations

Concurrent search and insert operations may cause problems

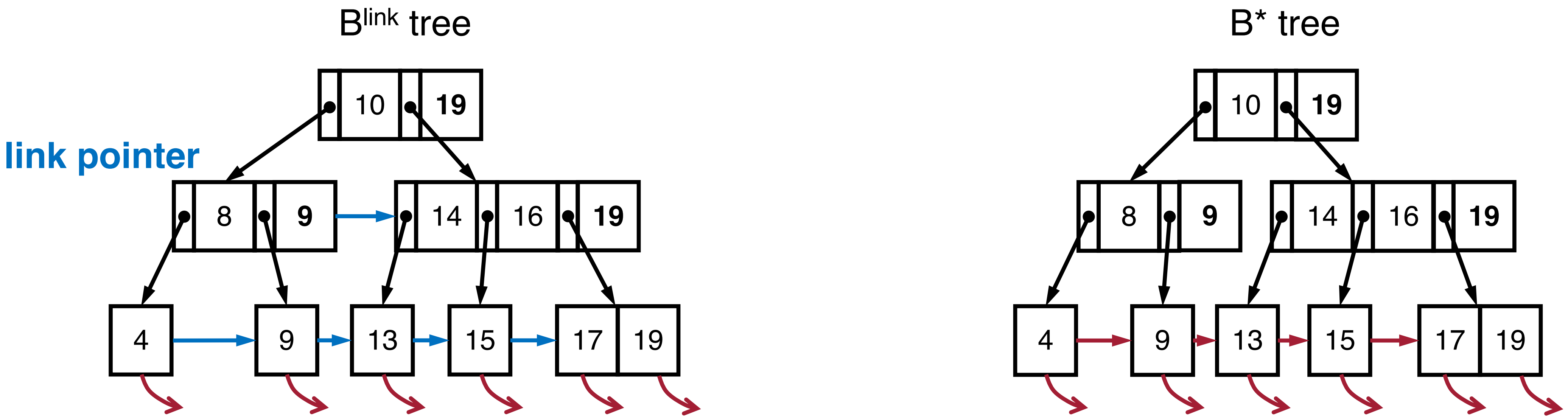
	search(15)	insert(9)
1.	$C \leftarrow \text{read}(x)$	
2.		$A \leftarrow \text{read}(x)$
3.	examine C; get ptr to y	
4.		examine A; get ptr to y
5.		$A \leftarrow \text{read}(y)$
6.		insert 9 into A; must split into A, B
7.		put(B, y')
8.		put(A, y)
9.		Add to node x a pointer to node y'.
10.	$C \leftarrow \text{read}(y)$	
11.	*error*: 15 not found!	



(a)



B^{link}-Tree

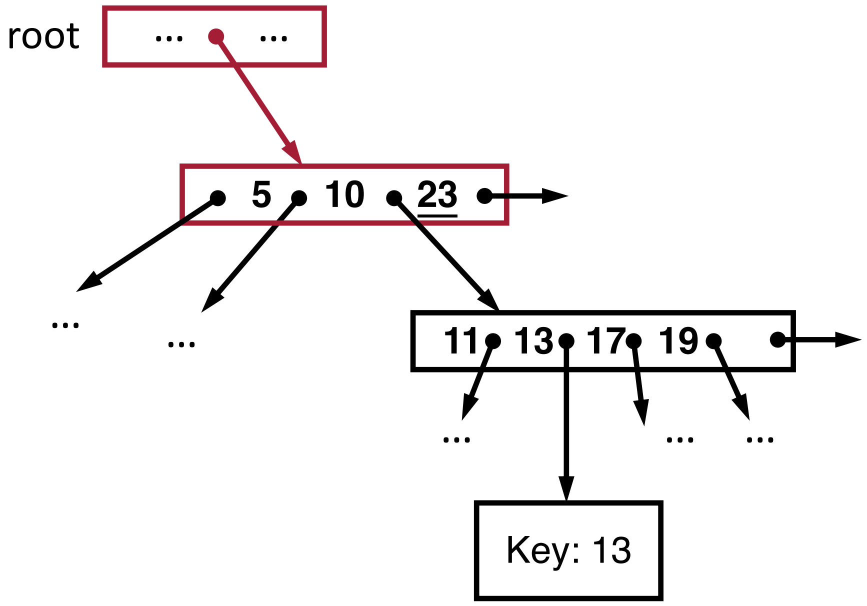


Adds a link field that points to the next node at the same level of the tree as the current node

The link pointer of the rightmost node on a level is a null pointer

B^{link}-Tree: Search Algorithm

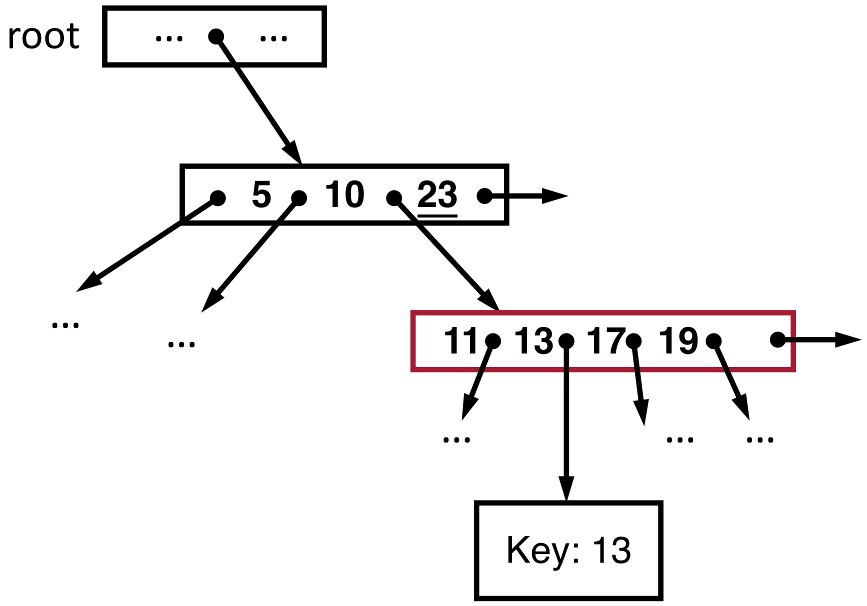
```
procedure search(v)
current ← root;                                          /* Get ptr to root node */
A ← get(current);                                    /* Read node into memory */
while current is not a leaf do
begin                                                      /* Scan through tree */
   current ← scannode(v, A);              /* Find correct (maybe link) ptr */
   A ← get(current)                                  /* Read node into memory */
end;
                                                 /* Now we have reached leaves. */
while t ← scannode(v, A) = link ptr of A do
                                         /* Keep moving right if necessary */
begin
   current ← t;
   A ← get(current)                                                /* Get node */
end;
                     /* Now we have the leaf node in which v should exist. */
if v is in A then done "success" else done "failure"
```



Example: search Key=13

B^{link}-Tree: Search Algorithm

```
procedure search(v)
current ← root;                                          /* Get ptr to root node */
A ← get(current);                                        /* Read node into memory */
while current is not a leaf do
begin                                                    /* Scan through tree */
   current ← scannode(v, A);                             /* Find correct (maybe link) ptr */
   A ← get(current)                                      /* Read node into memory */
end;
                                                         /* Now we have reached leaves. */
while t ← scannode(v, A) = link ptr of A do
                                                         /* Keep moving right if necessary */
begin
   current ← t;
   A ← get(current)                                      /* Get node */
end;
                              /* Now we have the leaf node in which v should exist. */
if v is in A then done "success" else done "failure"
```

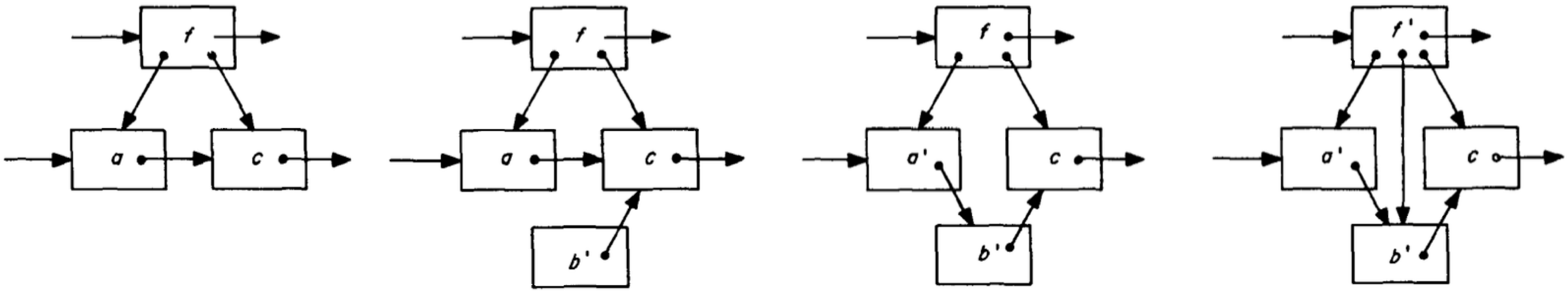


Example: search Key=13

B^{link}-Tree: Insert Algorithm

Insert to leaf if the leaf node if not full

Illustration of node split (node *a* is split into *a'* and *b'*)



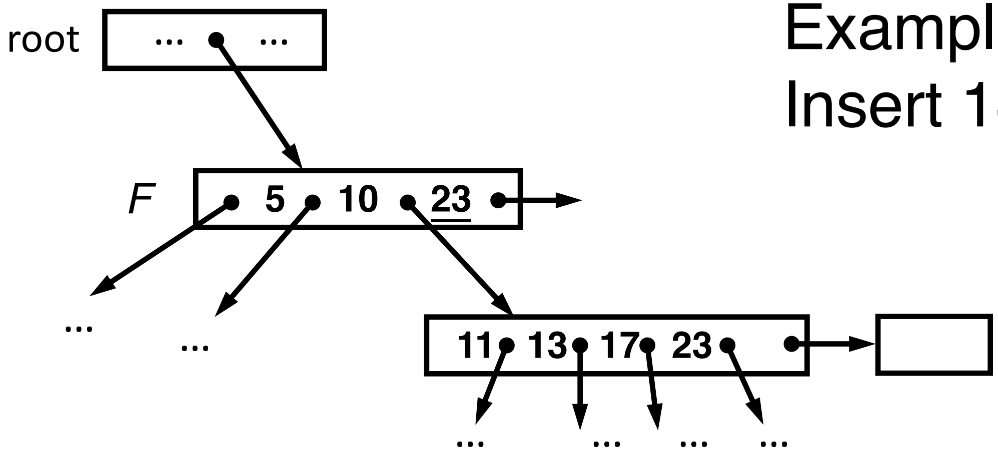
Before split   Step 1   Step 2   Step 3

B^{link}-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                          /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                      /* Scan down tree */
  t ← current;
  current ← scannode(v, A);
  if new current was not link pointer in A then
    push(t);                                               /* Remember node at that level */
  A ← get(current)
end;
lock(current);                                             /* We have a candidate leaf */
A ← get(current);
move.right;                                                /* If necessary */
if v is in A then stop "v already exists in tree";         /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
  A ← node.insert(A, w, v);                                /* Exact manner depends if current is a leaf */
  put(A, current);
  unlock(current);                                         /* Success—done backtracking */
end else begin                                             /* Must split node */
  u ← allocate(1 new page for B);
  A, B ← rearrange old A, adding v and w, to make 2 nodes,
    where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
  y ← max value stored in new A;                           /* For insertion into parent */
  put(B, u);                                               /* Insert B before A */
  put(A, current);                                         /* Instantaneous change of 2 nodes */
  oldnode ← current;                                       /* Now insert pointer in parent */
  v ← y;
  w ← u;
  current ← pop(stack);                                    /* Backtrack */
  lock(current);                                           /* Well ordered */
  A ← get(current);
  move.right;                                              /* If necessary */
  unlock(oldnode);
  goto Doinsertion                                         /* And repeat procedure for parent */
end
```

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                      /* Move right if necessary */
  lock(t);                                                 /* Note left-to-right locking */
  unlock(current);
  current ← t;
A ← get(current);
end
```

Example:
Insert 14

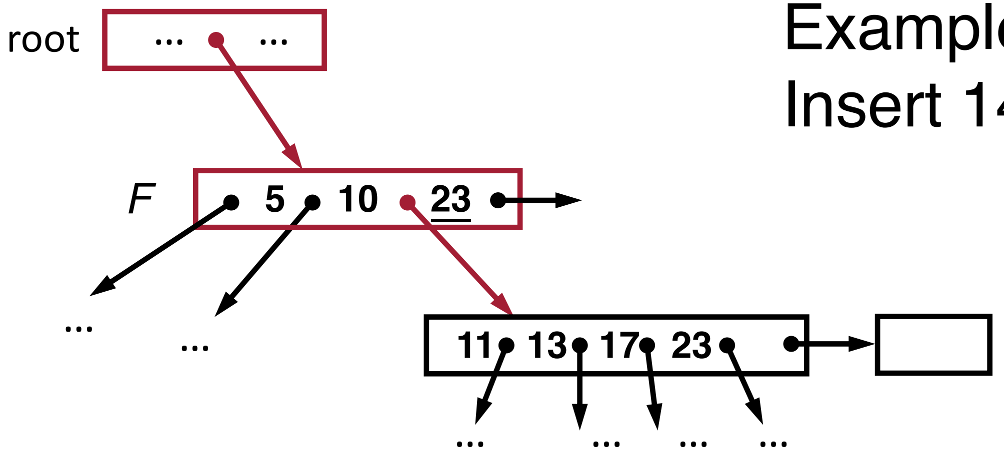


Blink-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                        /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                    /* Scan down tree */
   t ← current;
   current ← scannode(v, A);
   if new current was not link pointer in A then
      push(t);                                           /* Remember node at that level */
   A ← get(current)
end;
lock(current);                                           /* We have a candidate leaf */
A ← get(current);
move.right;                                              /* If necessary */
if v is in A then stop "v already exists in tree";       /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
   A ← node.insert(A, w, v);                             /* Exact manner depends if current is a leaf */
   put(A, current);
   unlock(current);                                      /* Success—done backtracking */
end else begin                                           /* Must split node */
   u ← allocate(1 new page for B);
   A, B ← rearrange old A, adding v and w, to make 2 nodes,
      where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
   y ← max value stored in new A;                        /* For insertion into parent */
   put(B, u);                                            /* Insert B before A */
   put(A, current);                                      /* Instantaneous change of 2 nodes */
   oldnode ← current;                                    /* Now insert pointer in parent */
   v ← y;
   w ← u;
   current ← pop(stack);                                 /* Backtrack */
   lock(current);                                        /* Well ordered */
   A ← get(current);
   move.right;                                           /* If necessary */
   unlock(oldnode);
   goto Doinsertion                                      /* And repeat procedure for parent */
end
```

stack = { F
root, }

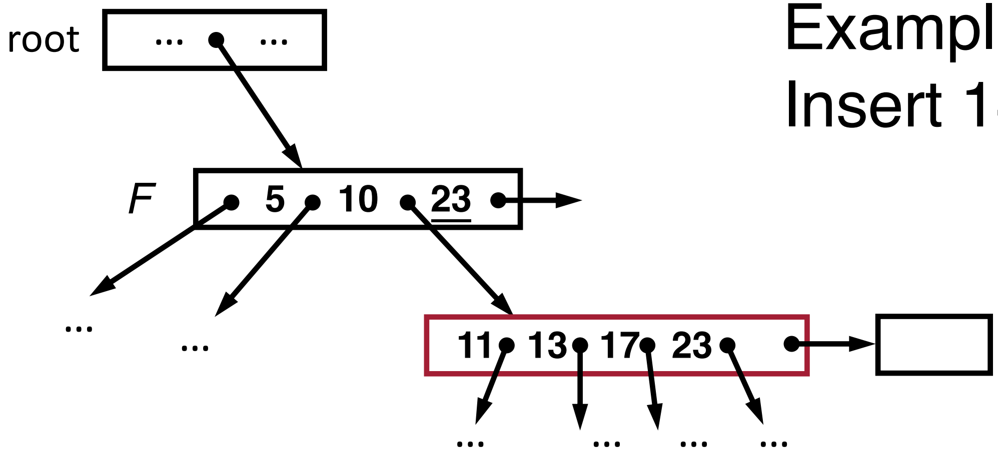
Example:
Insert 14




```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do       /* Move right if necessary */
begin                                                    /* Note left-to-right locking */
   lock(t);
   unlock(current);
   current ← t;
A ← get(current);
end
```

B^{link}-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                    /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                /* Scan down tree */
   t ← current;
   current ← scannode(v, A);
   if new current was not link pointer in A then
      push(t);                                       /* Remember node at that level */
   A ← get(current)
end;
lock(current);                                       /* We have a candidate leaf */
A ← get(current);
move.right;                                          /* If necessary */
if v is in A then stop "v already exists in tree";   /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
   A ← node.insert(A, w, v);                         /* Exact manner depends if current is a leaf */
   put(A, current);
   unlock(current);                                  /* Success—done backtracking */
end else begin                                       /* Must split node */
   u ← allocate(1 new page for B);
   A, B ← rearrange old A, adding v and w, to make 2 nodes,
      where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
   y ← max value stored in new A;                    /* For insertion into parent */
   put(B, u);                                        /* Insert B before A */
   put(A, current);                                  /* Instantaneous change of 2 nodes */
   oldnode ← current;                                /* Now insert pointer in parent */
   v ← y;
   w ← u;
   current ← pop(stack);                             /* Backtrack */
   lock(current);                                    /* Well ordered */
   A ← get(current);
   move.right;                                       /* If necessary */
   unlock(oldnode);
   goto Doinsertion                                  /* And repeat procedure for parent */
end
```



Example:
Insert 14

initially, w is the data page to be inserted

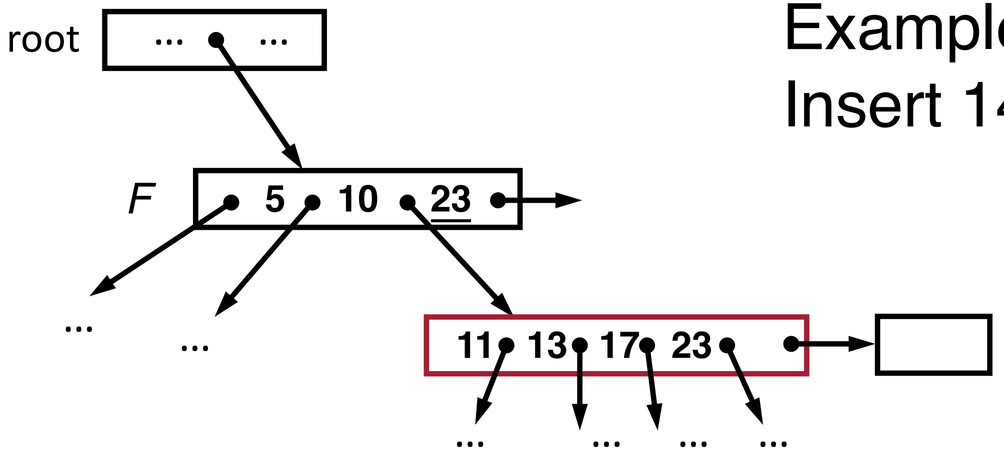
```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                /* Move right if necessary */
   lock(t);                                          /* Note left-to-right locking */
   unlock(current);
   current ← t;
A ← get(current);
end
```

B^{link}-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                        /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                    /* Scan down tree */
  t ← current;
  current ← scannode(v, A);
  if new current was not link pointer in A then
    push(t);                                             /* Remember node at that level */
  A ← get(current)
end;
lock(current);                                           /* We have a candidate leaf */
A ← get(current);
move.right;                                              /* If necessary */
if v is in A then stop "v already exists in tree";       /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
  A ← node.insert(A, w, v);                              /* Exact manner depends if current is a leaf */
  put(A, current);
  unlock(current);                                       /* Success—done backtracking */
end else begin                                           /* Must split node */
  u ← allocate(1 new page for B);
  A, B ← rearrange old A, adding v and w, to make 2 nodes,
    where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
  y ← max value stored in new A;                         /* For insertion into parent */
  put(B, u);                                             /* Insert B before A */
  put(A, current);                                       /* Instantaneous change of 2 nodes */
  oldnode ← current;                                     /* Now insert pointer in parent */
  v ← y;
  w ← u;
  current ← pop(stack);                                  /* Backtrack */
  lock(current);                                         /* Well ordered */
  A ← get(current);
  move.right;                                            /* If necessary */
  unlock(oldnode);
  goto Doinsertion                                       /* And repeat procedure for parent */
end
```

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                    /* Move right if necessary */
  lock(t);                                               /* Note left-to-right locking */
  unlock(current);
  current ← t;
A ← get(current);
end
```

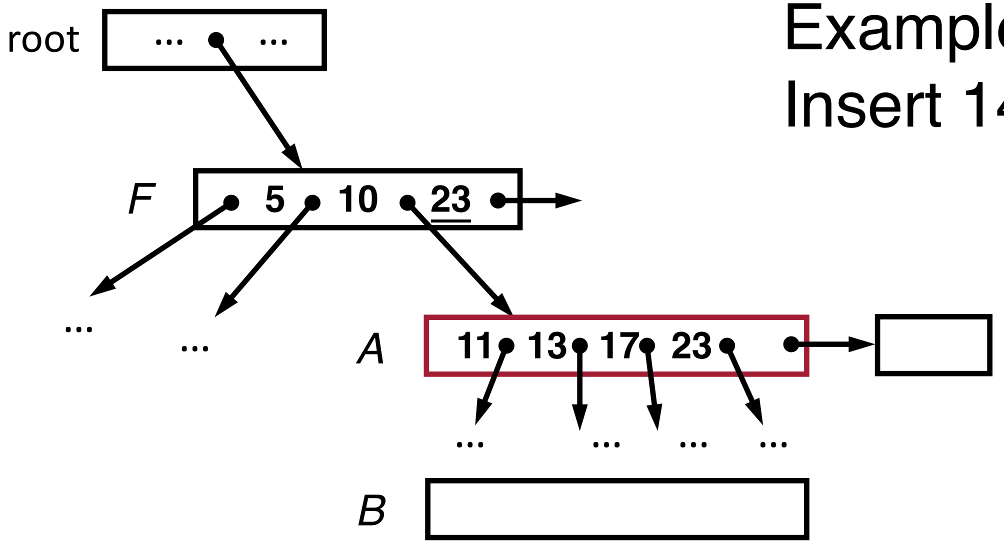
Example:
Insert 14



B^{link}-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                          /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                      /* Scan down tree */
  t ← current;
  current ← scannode(v, A);
  if new current was not link pointer in A then
    push(t);                                               /* Remember node at that level */
  A ← get(current)
end;
lock(current);                                             /* We have a candidate leaf */
A ← get(current);
move.right;                                                /* If necessary */
if v is in A then stop "v already exists in tree";         /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
  A ← node.insert(A, w, v);                /* Exact manner depends if current is a leaf */
  put(A, current);
  unlock(current);                                         /* Success—done backtracking */
end else begin                                             /* Must split node */
  u ← allocate(1 new page for B);
  A, B ← rearrange old A, adding v and w, to make 2 nodes,
    where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
  y ← max value stored in new A;                           /* For insertion into parent */
  put(B, u);                                               /* Insert B before A */
  put(A, current);                                         /* Instantaneous change of 2 nodes */
  oldnode ← current;                                       /* Now insert pointer in parent */
  v ← y;
  w ← u;
  current ← pop(stack);                                    /* Backtrack */
  lock(current);                                           /* Well ordered */
  A ← get(current);
  move.right;                                              /* If necessary */
  unlock(oldnode);
  goto Doinsertion                                         /* And repeat procedure for parent */
end
```

Example:
Insert 14



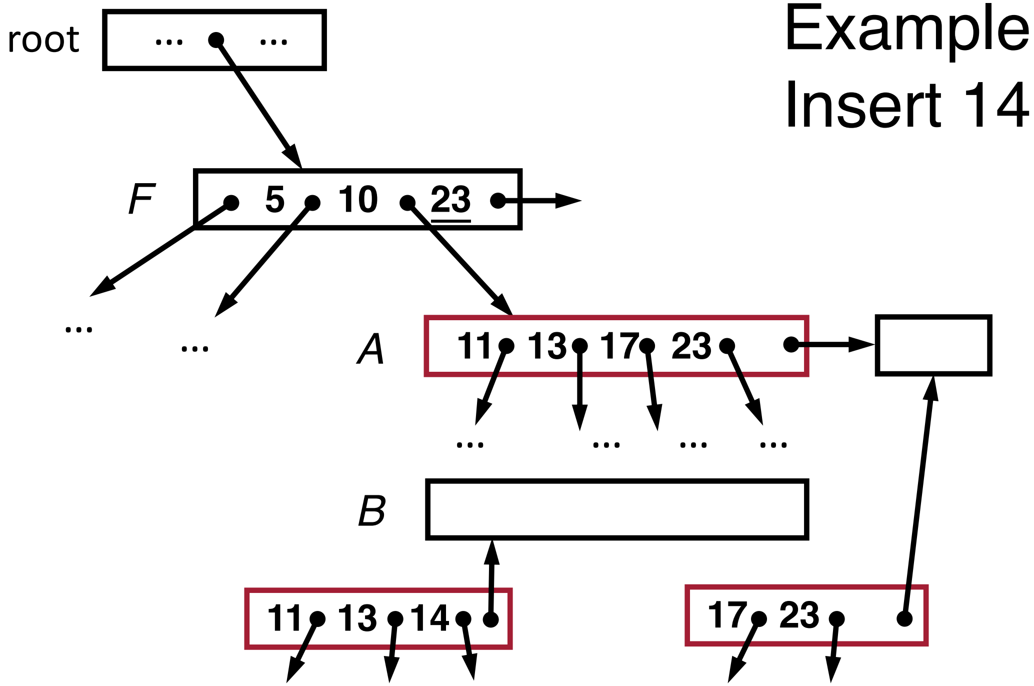
Allocate new block on disk

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                      /* Move right if necessary */
  lock(t);                                                 /* Note left-to-right locking */
  unlock(current);
  current ← t;
A ← get(current);
end
```

B^{link}-Tree: Insert Algorithm

Example:
Insert 14

```
procedure insert(v)
initialize stack;                                          /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                      /* Scan down tree */
   t ← current;
   current ← scannode(v, A);
   if new current was not link pointer in A then
      push(t);                                             /* Remember node at that level */
   A ← get(current)
end;
lock(current);                                             /* We have a candidate leaf */
A ← get(current);
move.right;                                                /* If necessary */
if v is in A then stop "v already exists in tree";         /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
   A ← node.insert(A, w, v);                               /* Exact manner depends if current is a leaf */
   put(A, current);
   unlock(current);                                        /* Success—done backtracking */
end else begin                                             /* Must split node */
   u ← allocate(1 new page for B);
   A, B ← rearrange old A, adding v and w, to make 2 nodes,
      where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
   y ← max value stored in new A;                          /* For insertion into parent */
   put(B, u);                                              /* Insert B before A */
   put(A, current);                                        /* Instantaneous change of 2 nodes */
   oldnode ← current;                                      /* Now insert pointer in parent */
   v ← y;
   w ← u;
   current ← pop(stack);                                   /* Backtrack */
   lock(current);                                          /* Well ordered */
   A ← get(current);
   move.right;                                             /* If necessary */
   unlock(oldnode);
   goto Doinsertion                                        /* And repeat procedure for parent */
end
```



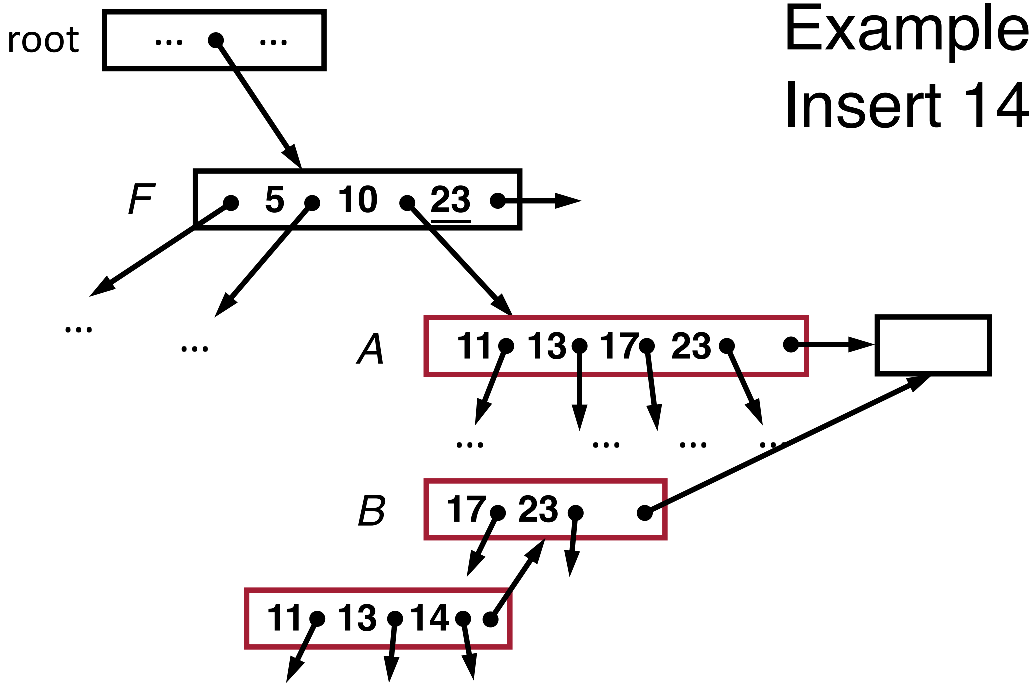
create two pages in memory

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                      /* Move right if necessary */
   lock(t);                                                /* Note left-to-right locking */
   unlock(current);
   current ← t;
A ← get(current);
end
```

B^{link}-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                         /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                     /* Scan down tree */
  t ← current;
  current ← scannode(v, A);
  if new current was not link pointer in A then
    push(t);                                              /* Remember node at that level */
  A ← get(current)
end;
lock(current);                                            /* We have a candidate leaf */
A ← get(current);
move.right;                                               /* If necessary */
if v is in A then stop "v already exists in tree";        /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
  A ← node.insert(A, w, v);                               /* Exact manner depends if current is a leaf */
  put(A, current);
  unlock(current);                                        /* Success—done backtracking */
end else begin                                            /* Must split node */
  u ← allocate(1 new page for B);
  A, B ← rearrange old A, adding v and w, to make 2 nodes,
    where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
  y ← max value stored in new A;                          /* For insertion into parent */
  put(B, u);                                              /* Insert B before A */
  put(A, current);                                        /* Instantaneous change of 2 nodes */
  oldnode ← current;                                      /* Now insert pointer in parent */
  v ← y;
  w ← u;
  current ← pop(stack);                                   /* Backtrack */
  lock(current);                                          /* Well ordered */
  A ← get(current);
  move.right;                                             /* If necessary */
  unlock(oldnode);
  goto Doinsertion                                        /* And repeat procedure for parent */
end
```

Example:
Insert 14



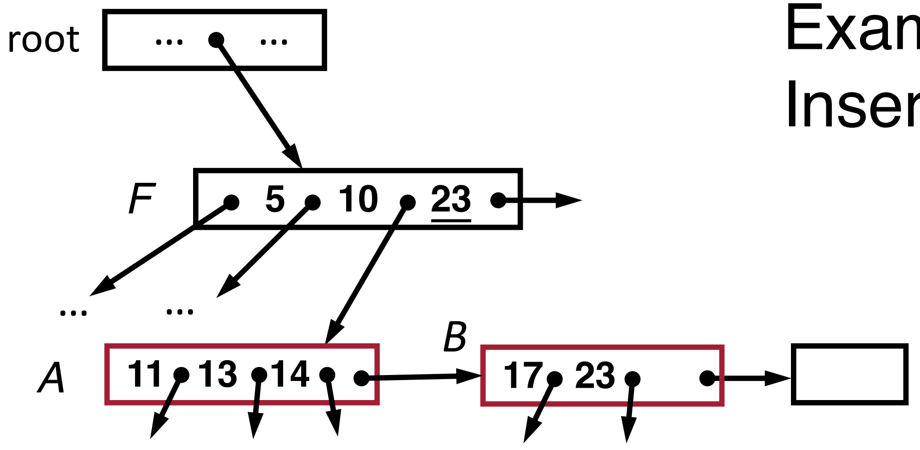
update the two disk pages (page B first)

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                     /* Move right if necessary */
  lock(t);                                                /* Note left-to-right locking */
  unlock(current);
  current ← t;
A ← get(current);
end
```

B^{link}-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                          /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                      /* Scan down tree */
  t ← current;
  current ← scannode(v, A);
  if new current was not link pointer in A then
    push(t);                                               /* Remember node at that level */
  A ← get(current)
end;
lock(current);                                             /* We have a candidate leaf */
A ← get(current);
move.right;                                                /* If necessary */
if v is in A then stop "v already exists in tree";         /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
  A ← node.insert(A, w, v);                                /* Exact manner depends if current is a leaf */
  put(A, current);
  unlock(current);                                         /* Success—done backtracking */
end else begin                                             /* Must split node */
  u ← allocate(1 new page for B);
  A, B ← rearrange old A, adding v and w, to make 2 nodes,
    where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
  y ← max value stored in new A;                           /* For insertion into parent */
  put(B, u);                                               /* Insert B before A */
  put(A, current);                                         /* Instantaneous change of 2 nodes */
  oldnode ← current;                                       /* Now insert pointer in parent */
  v ← y;
  w ← u;
  current ← pop(stack);                                    /* Backtrack */
  lock(current);                                           /* Well ordered */
  A ← get(current);
  move.right;                                              /* If necessary */
  unlock(oldnode);
  goto Doinsertion                                         /* And repeat procedure for parent */
end
```

Example:
Insert 14

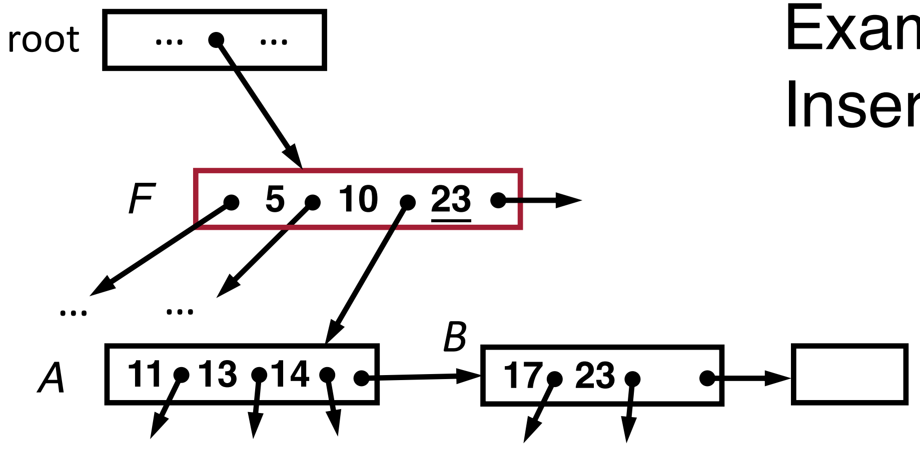


update the two disk pages (page B first)

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                      /* Move right if necessary */
  lock(t);                                                 /* Note left-to-right locking */
  unlock(current);
  current ← t;
A ← get(current);
end
```

B^link-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                   /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                               /* Scan down tree */
  t ← current;
  current ← scannode(v, A);
  if new current was not link pointer in A then
    push(t);                                        /* Remember node at that level */
  A ← get(current)
end;
lock(current);                                      /* We have a candidate leaf */
A ← get(current);
move.right;                                         /* If necessary */
if v is in A then stop "v already exists in tree";  /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
  A ← node.insert(A, w, v);                         /* Exact manner depends if current is a leaf */
  put(A, current);
  unlock(current);                                  /* Success—done backtracking */
end else begin                                      /* Must split node */
  u ← allocate(1 new page for B);
  A, B ← rearrange old A, adding v and w, to make 2 nodes,
    where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
  y ← max value stored in new A;                    /* For insertion into parent */
  put(B, u);                                        /* Insert B before A */
  put(A, current);                                  /* Instantaneous change of 2 nodes */
  oldnode ← current;                                /* Now insert pointer in parent */
  v ← y;
  w ← u;
  current ← pop(stack);                             /* Backtrack */
  lock(current);                                    /* Well ordered */
  A ← get(current);
  move.right;                                       /* If necessary */
  unlock(oldnode);
  goto Doinsertion                                  /* And repeat procedure for parent */
end
```



Example:
Insert 14

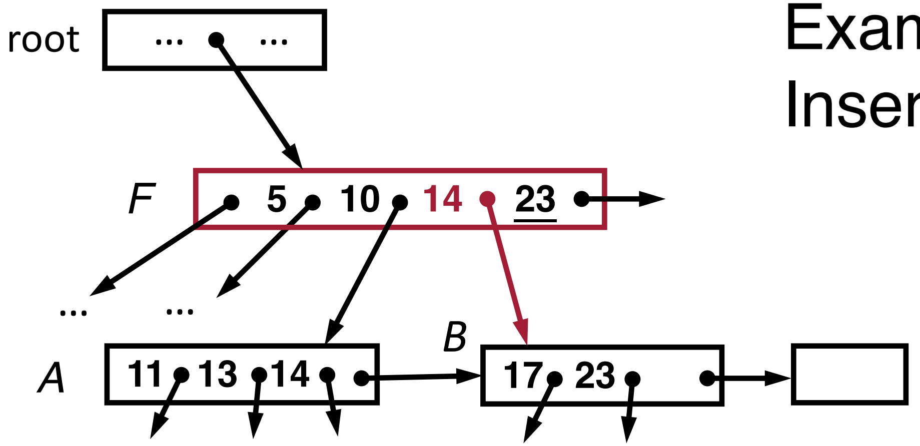
try to insert (key=14, ptr=*B*) to *F*

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                               /* Move right if necessary */
  lock(t);                                          /* Note left-to-right locking */
  unlock(current);
  current ← t;
A ← get(current);
end
```

B^{link}-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                    /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                /* Scan down tree */
  t ← current;
  current ← scannode(v, A);
  if new current was not link pointer in A then
    push(t);                                         /* Remember node at that level */
  A ← get(current)
end;
lock(current);                                       /* We have a candidate leaf */
A ← get(current);
move.right;                                          /* If necessary */
if v is in A then stop "v already exists in tree";   /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
  A ← node.insert(A, w, v);                          /* Exact manner depends if current is a leaf */
  put(A, current);
  unlock(current);                                   /* Success—done backtracking */
end else begin                                       /* Must split node */
  u ← allocate(1 new page for B);
  A, B ← rearrange old A, adding v and w, to make 2 nodes,
    where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
  y ← max value stored in new A;                     /* For insertion into parent */
  put(B, u);                                         /* Insert B before A */
  put(A, current);                                   /* Instantaneous change of 2 nodes */
  oldnode ← current;                                 /* Now insert pointer in parent */
  v ← y;
  w ← u;
  current ← pop(stack);                              /* Backtrack */
  lock(current);                                     /* Well ordered */
  A ← get(current);
  move.right;                                        /* If necessary */
  unlock(oldnode);
  goto Doinsertion                                   /* And repeat procedure for parent */
end
```

Example:
Insert 14

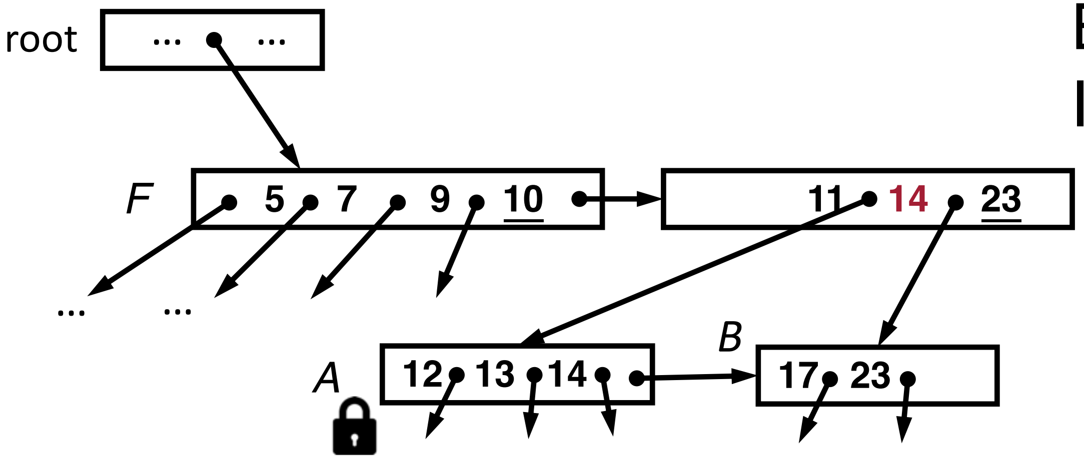


insert (key=14, ptr=B) to F

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                /* Move right if necessary */
  lock(t);                                           /* Note left-to-right locking */
  unlock(current);
  current ← t;
A ← get(current);
end
```

Blink-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                          /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                      /* Scan down tree */
   t ← current;
   current ← scannode(v, A);
   if new current was not link pointer in A then
      push(t);                                             /* Remember node at that level */
   A ← get(current)
end;
lock(current);                                             /* We have a candidate leaf */
A ← get(current);
move.right;                                                /* If necessary */
if v is in A then stop "v already exists in tree";         /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
   A ← node.insert(A, w, v);                               /* Exact manner depends if current is a leaf */
   put(A, current);
   unlock(current);                                        /* Success—done backtracking */
end else begin                                             /* Must split node */
   u ← allocate(1 new page for B);
   A, B ← rearrange old A, adding v and w, to make 2 nodes,
      where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
   y ← max value stored in new A;                          /* For insertion into parent */
   put(B, u);                                              /* Insert B before A */
   put(A, current);                                        /* Instantaneous change of 2 nodes */
   oldnode ← current;                                      /* Now insert pointer in parent */
   v ← y;
   w ← u;
   current ← pop(stack);                                   /* Backtrack */
   lock(current);                                          /* Well ordered */
   A ← get(current);
   move.right;                                             /* If necessary */
   unlock(oldnode);
   goto Doinsertion                                        /* And repeat procedure for parent */
end
```



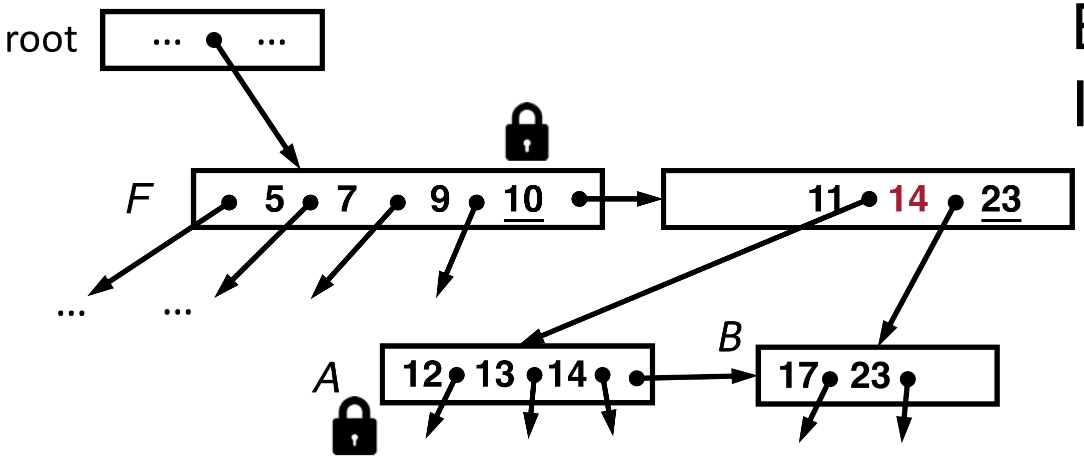
Example:
Insert 14

At most three locks are being during an insert

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                      /* Move right if necessary */
   lock(t);                                                /* Note left-to-right locking */
   unlock(current);
   current ← t;
A ← get(current);
end
```

B^{link}-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                        /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                    /* Scan down tree */
  t ← current;
  current ← scannode(v, A);
  if new current was not link pointer in A then
    push(t);                                             /* Remember node at that level */
  A ← get(current)
end;
lock(current);                                           /* We have a candidate leaf */
A ← get(current);
move.right;                                              /* If necessary */
if v is in A then stop "v already exists in tree";       /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
  A ← node.insert(A, w, v);            /* Exact manner depends if current is a leaf */
  put(A, current);
  unlock(current);                                       /* Success—done backtracking */
end else begin                                           /* Must split node */
  u ← allocate(1 new page for B);
  A, B ← rearrange old A, adding v and w, to make 2 nodes,
    where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
  y ← max value stored in new A;                         /* For insertion into parent */
  put(B, u);                                             /* Insert B before A */
  put(A, current);                                       /* Instantaneous change of 2 nodes */
  oldnode ← current;                                     /* Now insert pointer in parent */
  v ← y;
  w ← u;
  current ← pop(stack);                                  /* Backtrack */
  lock(current);                                         /* Well ordered */
  A ← get(current);
  move.right;                                            /* If necessary */
  unlock(oldnode);
  goto Doinsertion                                       /* And repeat procedure for parent */
end
```



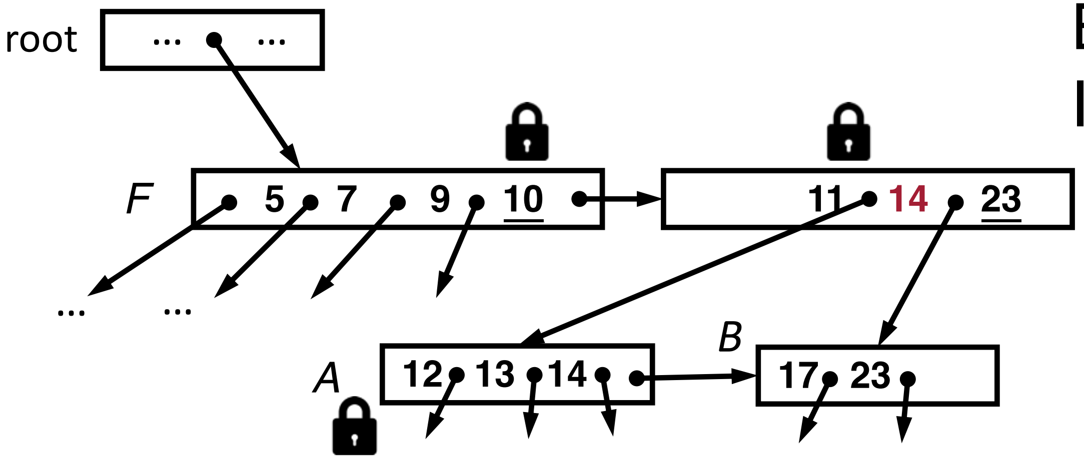
Example:
Insert 14

At most three locks are being during an insert

```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                          /* Move right if necessary */
  lock(t);                                     /* Note left-to-right locking */
  unlock(current);
  current ← t;
A ← get(current);
end
```

B^{link}-Tree: Insert Algorithm

```
procedure insert(v)
initialize stack;                                          /* For remembering ancestors */
current ← root;
A ← get(current);
while current is not a leaf do
begin                                                      /* Scan down tree */
  t ← current;
  current ← scannode(v, A);
  if new current was not link pointer in A then
    push(t);                                               /* Remember node at that level */
  A ← get(current)
end;
lock(current);                                             /* We have a candidate leaf */
A ← get(current);
move.right;                                                /* If necessary */
if v is in A then stop "v already exists in tree";         /* And t points to its record */
w ← pointer to pages allocated for record associated with v;
Doinsertion:
if A is safe then
begin
  A ← node.insert(A, w, v);                                /* Exact manner depends if current is a leaf */
  put(A, current);
  unlock(current);                                         /* Success—done backtracking */
end else begin                                             /* Must split node */
  u ← allocate(1 new page for B);
  A, B ← rearrange old A, adding v and w, to make 2 nodes,
    where (link ptr of A, link ptr of B) ← (u, link ptr of old A);
  y ← max value stored in new A;                           /* For insertion into parent */
  put(B, u);                                               /* Insert B before A */
  put(A, current);                                         /* Instantaneous change of 2 nodes */
  oldnode ← current;                                       /* Now insert pointer in parent */
  v ← y;
  w ← u;
  current ← pop(stack);                                    /* Backtrack */
  lock(current);                                           /* Well ordered */
  A ← get(current);
  move.right;                                              /* If necessary */
  unlock(oldnode);
  goto Doinsertion                                         /* And repeat procedure for parent */
end
```



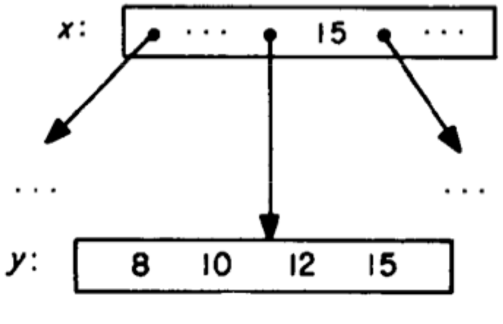
Example:
Insert 14

At most three locks are being during an insert

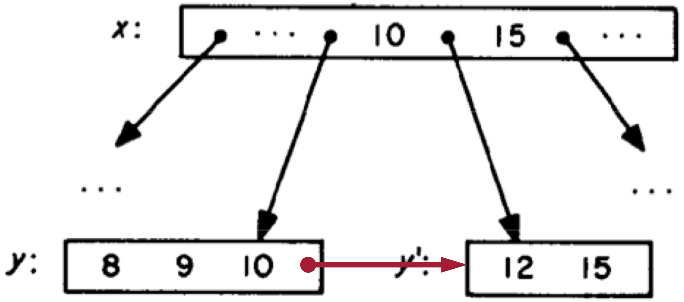
```
procedure move.right
while t ← scannode(v, A) is a link pointer of A do
begin                                                      /* Move right if necessary */
  lock(t);                                                 /* Note left-to-right locking */
  unlock(current);
  current ← t;
A ← get(current);
end
```

Revisit Concurrent Operations

	search(15)	insert(9)
1.	$C \leftarrow$ read(x)	
2.		$A \leftarrow$ read(x)
3.	examine C; get ptr to y	
4.		examine A; get ptr to y
5.		$A \leftarrow$ read(y)
6.		insert 9 into A; must split into A, B
7.		put(B, y')
8.		put(A, y)
9.		Add to node x a pointer to node y'.
10.	$C \leftarrow$ read(y)	
11.	*error*: 15 not found!	



(a)



key=15 is less than max key in node y

Follow the link ptr to the next leaf node and 15 is found!

Other Issues

Delete: allow fewer than k entries in a leaf node

- Observations: insertions are much more frequent than deletions

Deadlock freedom: locks are acquired bottom-up and left to right => total order

Livelock: keep following the link pointer due to node splits

Q/A – B-tree Locking

B+ tree vs. B* tree?

Which variant of B-tree are modern DBMSs using?

Would a left pointer add benefit?

Experimental comparison

What's the typical value of *k*?

Binary search within a node?

Disk utilization w.r.t. deletion

Deadlock vs. livelock?

Before Next Lecture

Submit review before next lecture

- C. Mohan, et al. [ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging](). ACM Trans. Database Syst. 1992.