Data Structures & Algorithms

Lecture 5: Advanced Data Structures

B-Trees

Lecturer: Dr. Nguyen Hai Minh
nhminh@fit.hcmus.edu.vn

OUTLINE

- Introduction
- Basic operations on B-Tree
- Deleting a key from a B-Tree

Motivatio of B-Trees

- So far, we have assumed that we can store an entire data structure in main memory
- What if we have so much data that it won't fit?
- Storing it on disk requires different approach to efficiency
- Assuming that a disk spins at 3600 RPM, one revolution occurs in 1/60 of a second, or 16.7ms
- Crudely speaking, one disk access takes about the same time as 200,000 instructions

Motivation of B-Trees

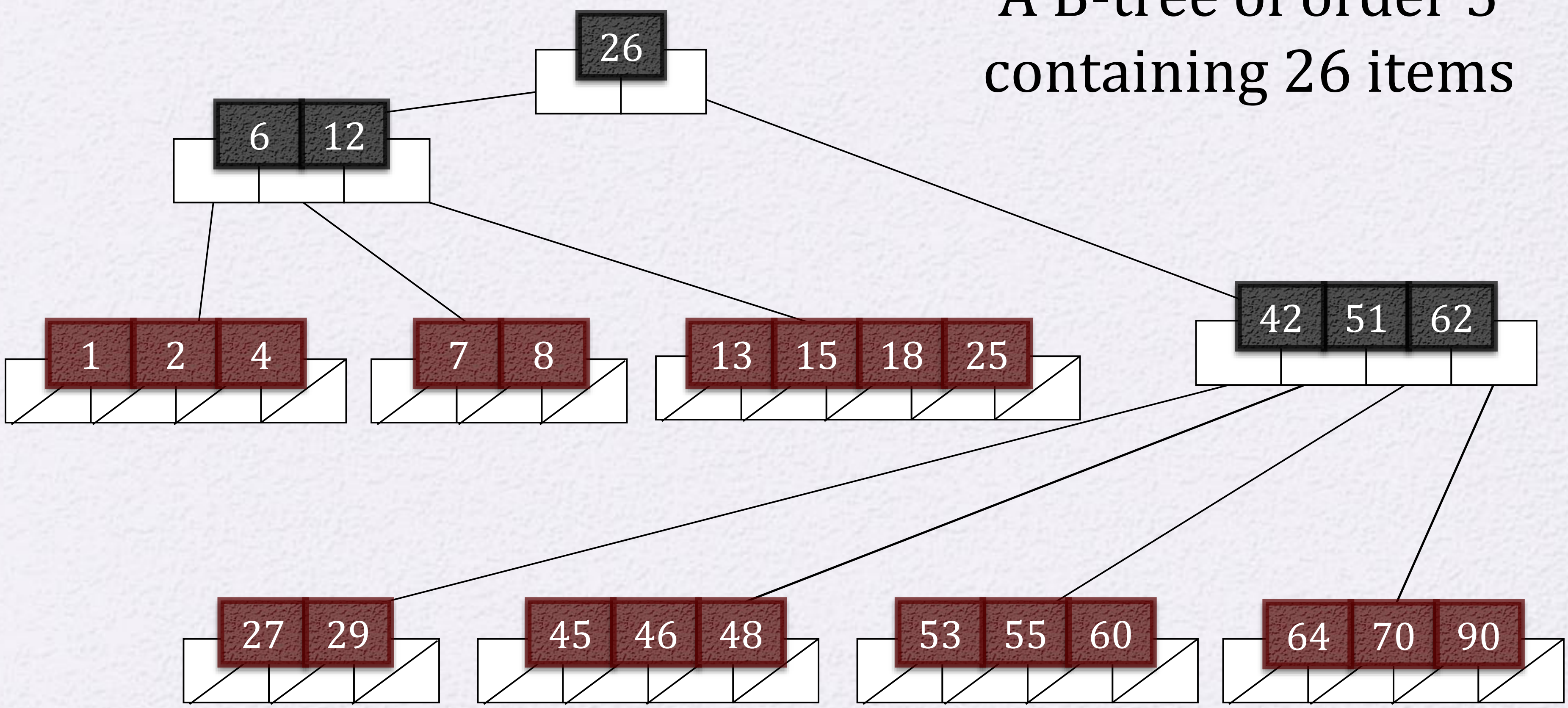
- Assume that we use an AVL tree to store about 20 million records
- We end up with a very deep binary tree with lots of different disk accesses; $\mathbf{\log_2 20{,}000{,}000 \approx 24}$, so this takes about **0.2** seconds
- We know we can't improve on the $\log_2 n$ lower bound on search for a binary tree
- But, the solution is to use more branches and thus reduce the height of the tree!
- As branching increases, depth decreases

Definition of a B-Tree

- A B-tree of order m is an m-way tree (i.e., a tree where each node may have up to m children) in which:
  1. The number of keys in each non-leaf node is one less than the number of its children and these keys partition the keys in the children in the fashion of a search tree
  2. All leaves are on the same level
  3. All non-leaf nodes except the root have at least $\lceil m/2 \rceil$ children
  4. The root is either a leaf node, or it has from 2 to m children
  5. A leaf node contains $\lceil m/2 \rceil - 1$ to $m - 1$ keys
- The number m should always be odd

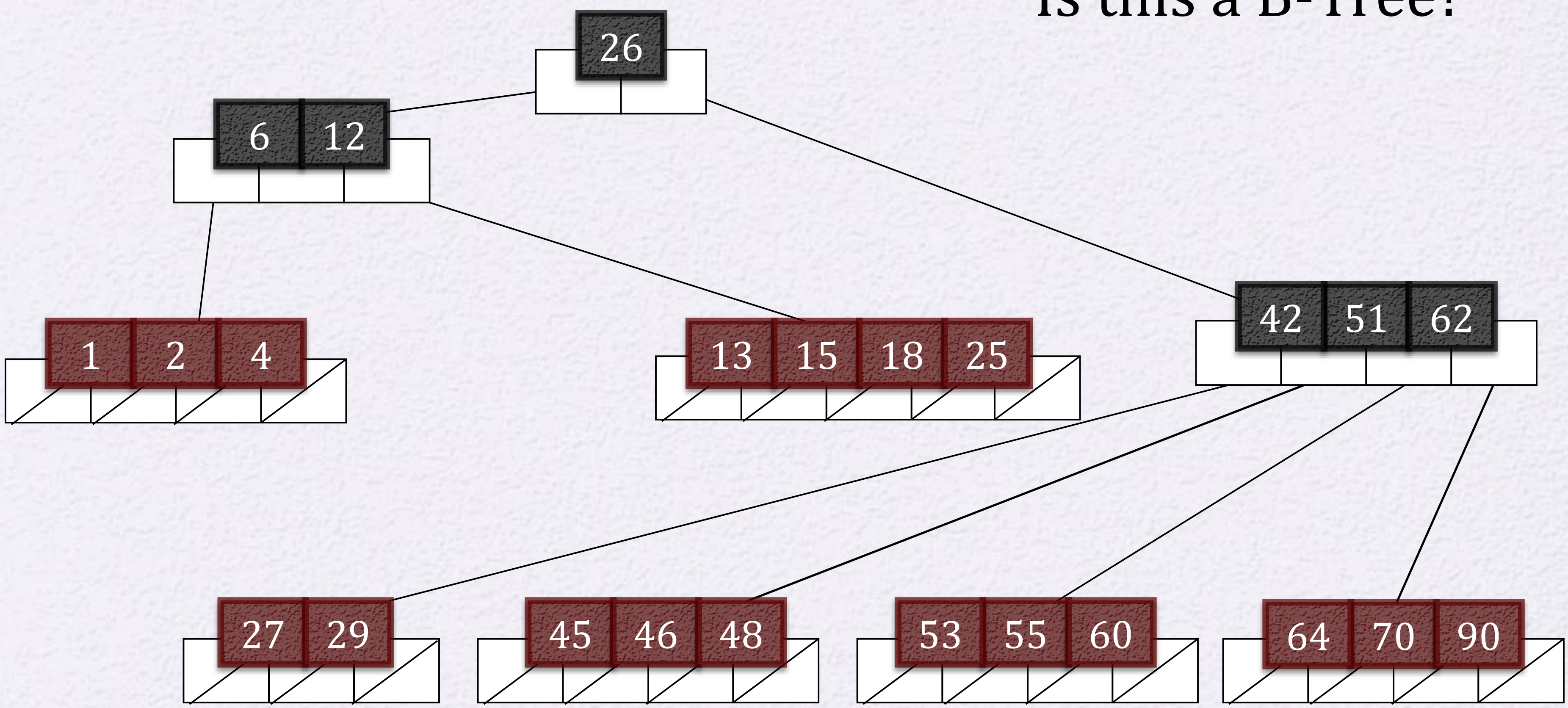
An example B-Tree

A B-tree of order 5 containing 26 items



An example B-Tree

Is this a B-Tree?



B-Tree's maximum heigh

- ❑ Let h be the height of B-Tree

→ The maximum number of nodes in a m-order B-Tree is:

$$\boxed{m^{h+1} - 1}$$

B-Tree's maximum heigh

- The maximum number of items in a B-tree of order m and height h:

root	$m - 1$
level 1	$m(m - 1)$
level 2	$m^2(m - 1)$
. . .	
level h	$m^h(m - 1)$

- So, the total number of items is

$$(1 + m + m^2 + m^3 + \ldots + m^h)(m - 1) =$$
$$[(m^{h+1} - 1)/\ (m - 1)]\ (m - 1) = \boldsymbol{m^{h+1} - 1}$$

- When $m = 5$ and $h = 2$ this gives $5^3 - 1 = 124$

Inserting into a B-Tree

- Attempt to insert the new key into a leaf
- If leaf becomes too big,
  - Split the leaf into two
  - Promoting the middle key to the leaf’s parent
- If the parent becomes too big
  - Split the parent into two
  - Promoting the middle key
- This strategy might have to be repeated all the way to the top
- If necessary, the root is split in two and the middle key is promoted to a new root, *making the tree one level higher*

Inserting into a B-Tree Example

- Suppose we start with an empty B-tree and keys arrive in the following order:1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45

- We want to construct a B-tree of order 5

Inserting into a B-Tree Example

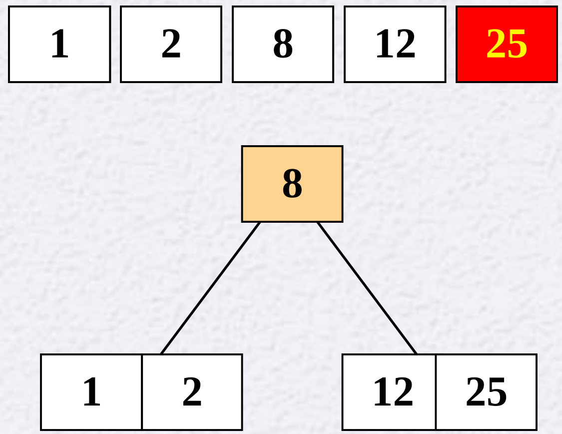
1 12 8 2 25 5 14
28 17 7 52 16 48
68 3 26 29 53 55
45

1	2	8	12

Insert **1** Insert **12** Insert **8** Insert **2**

Inserting into a B-Tree Example

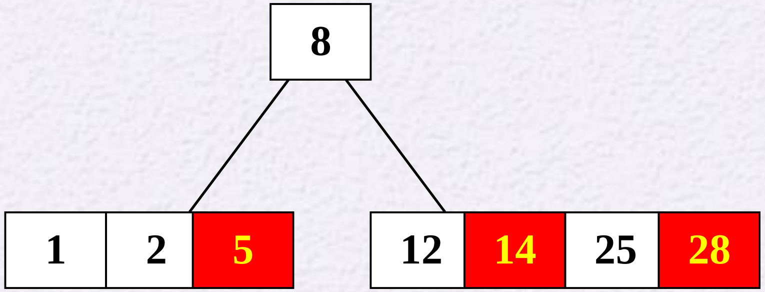
1 12 8 2 **25** 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45



Insert **25**

Inserting into a B-Tree Example

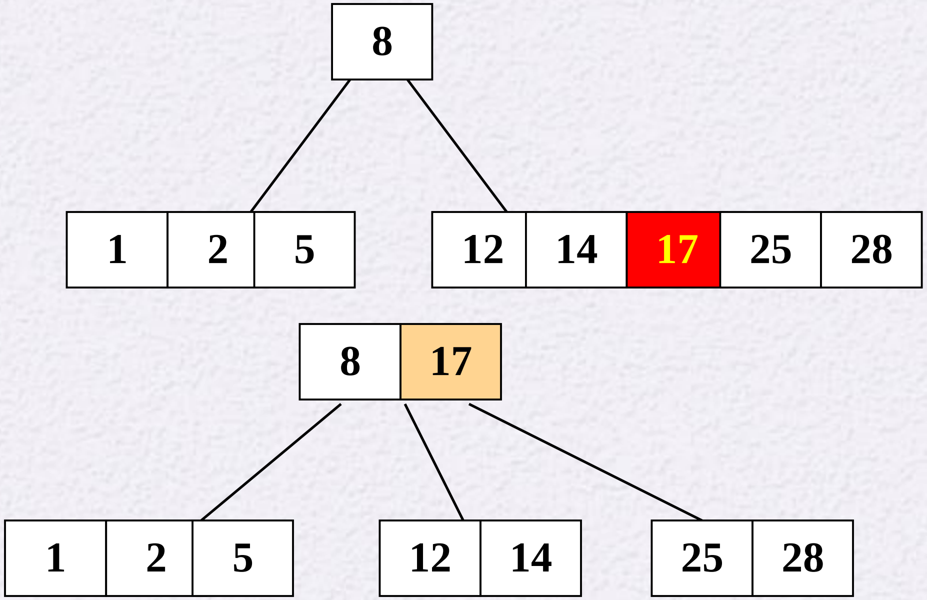
1 12 8 2 25 **5 14**
28 17 7 52 16 48
68 3 26 29 53 55
45



Insert **5, 14, 28**

Inserting into a B-Tree Example

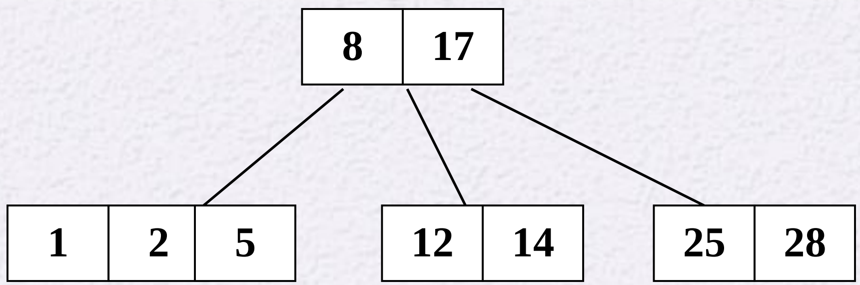
1 12 8 2 25 5 14 28 **17** 7 52 16 48 68 3 26 29 53 55 45

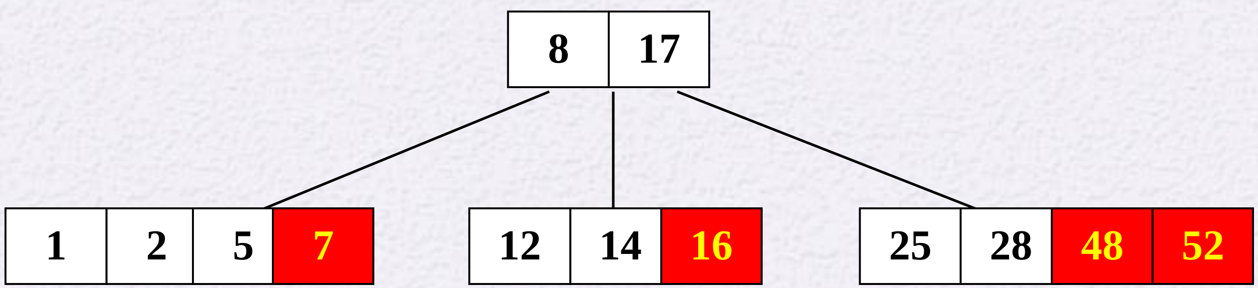


Insert **17**

Inserting into a B-Tree Example

1 12 8 2 25 5 14
28 17 **7 52 16 48**
68 3 26 29 53 55
45

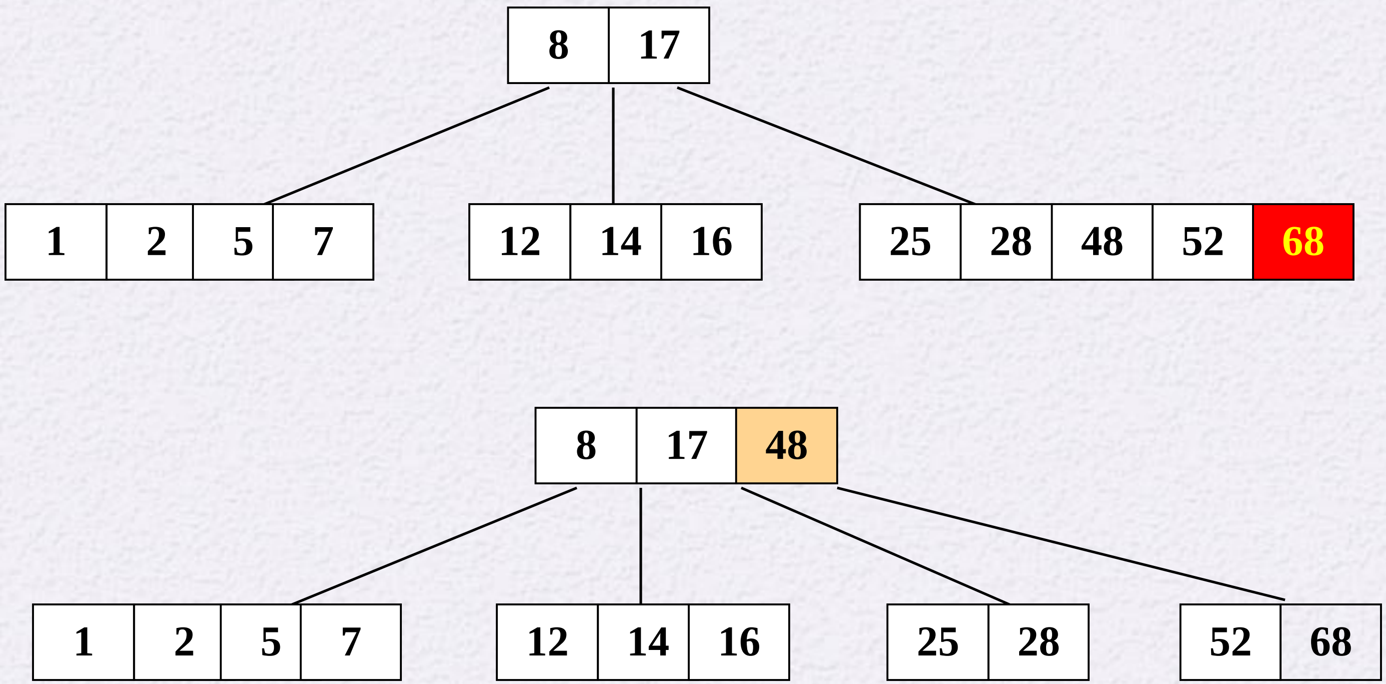




Insert **7, 52, 16, 48**

Inserting into a B-Tree Example

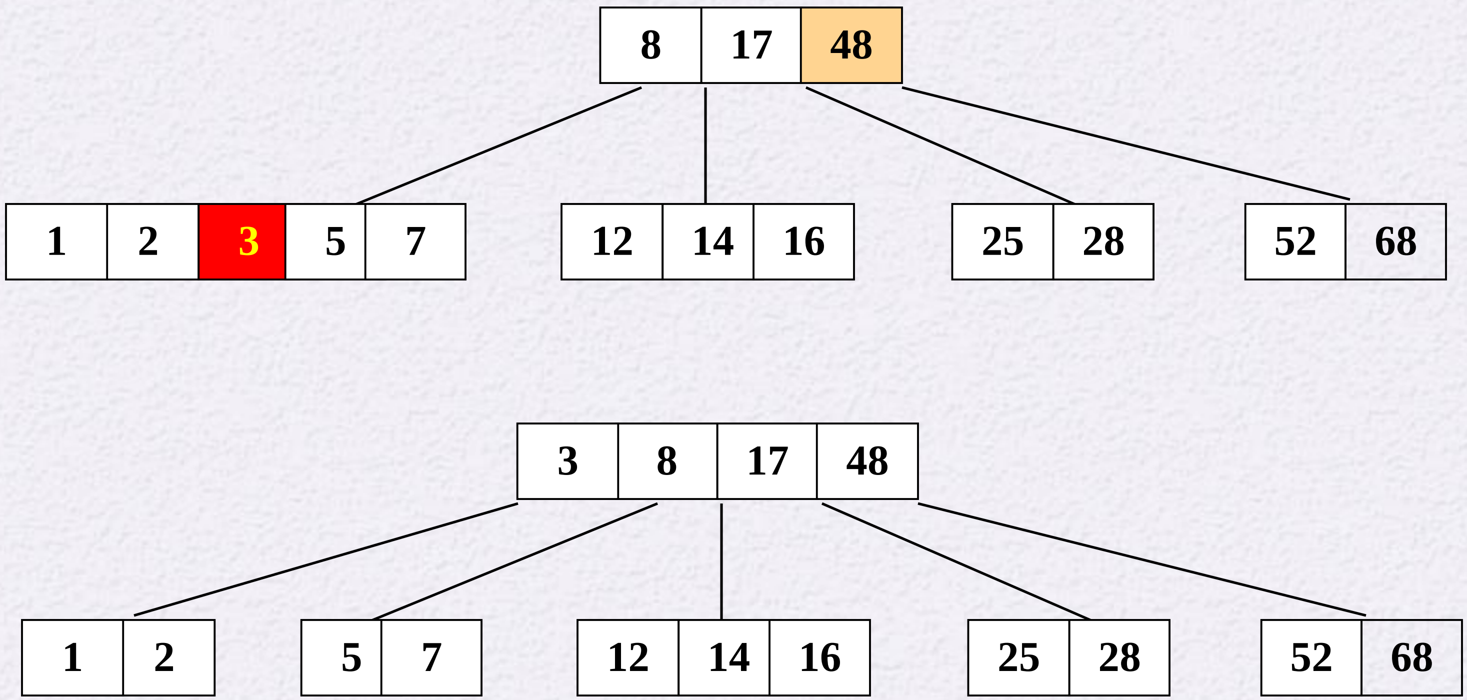
1 12 8 2 25 5 14 28 17 7 52 16 48 **68** 3 26 29 53 55 45



Insert **68**

Inserting into a B-Tree Example

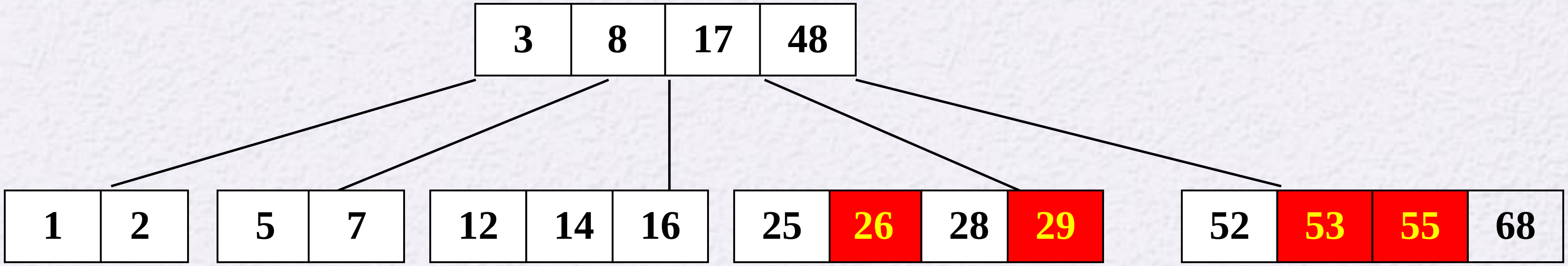
1 12 8 2 25 5 14
28 17 7 52 16 48
68 **3** 26 29 53 55
45



Insert **3**

Inserting into a B-Tree Example

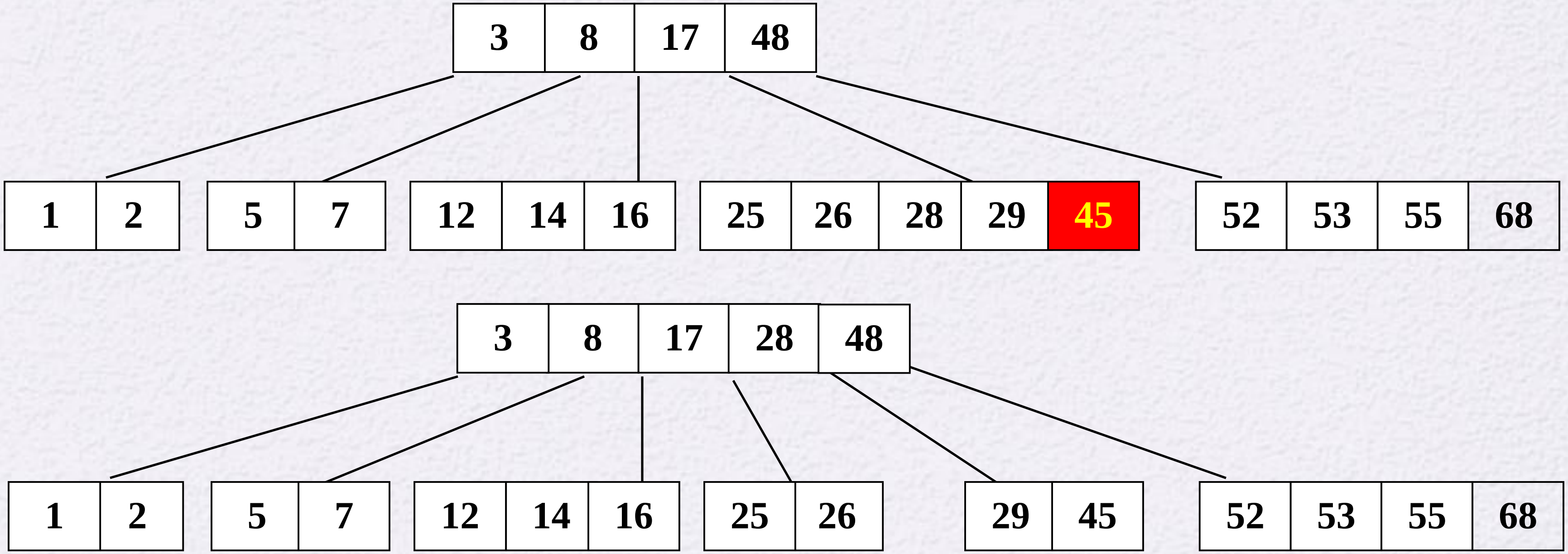
1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 **26 29 53 55** 45



Insert **26, 29, 53, 55**

Inserting into a B-Tree Example

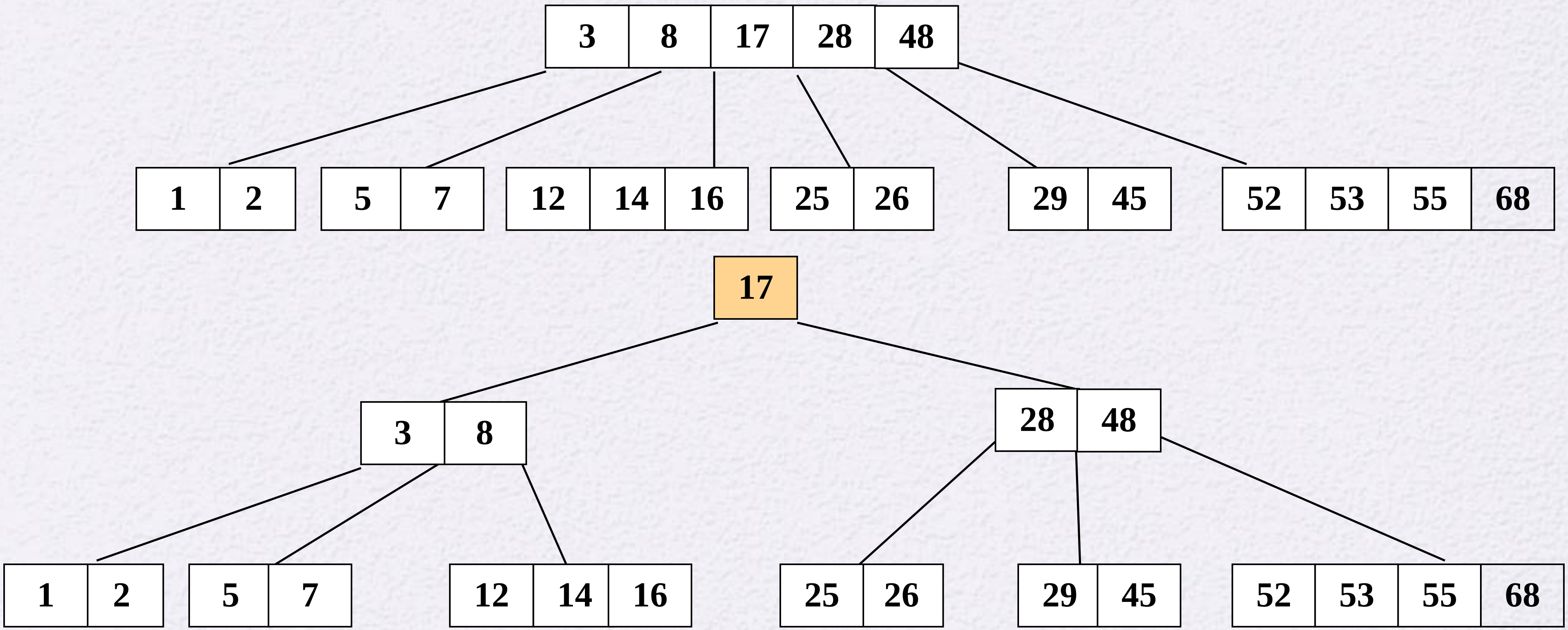
1 12 8 2 25 5 14
28 17 7 52 16 48
68 3 26 29 53 55
45



Insert **45**

Inserting into a B-Tree Example

1 12 8 2 25 5 14
28 17 7 52 16 48
68 3 26 29 53 55
45



Insert **45**

Removal from a B-tree

- Remove a key *k from* a leaf.

1. If *k* is in a leaf node, and removing it doesn't cause that leaf node to have too few keys, then simply remove *k*.
2. If *k* is NOT in a leaf then it is guaranteed (by the nature of a B-tree) that its predecessor or successor will be in a leaf -- in this case we can delete *k* and promote the predecessor or successor of *k* to *k*'s position.

Removal from a B-tree (2)

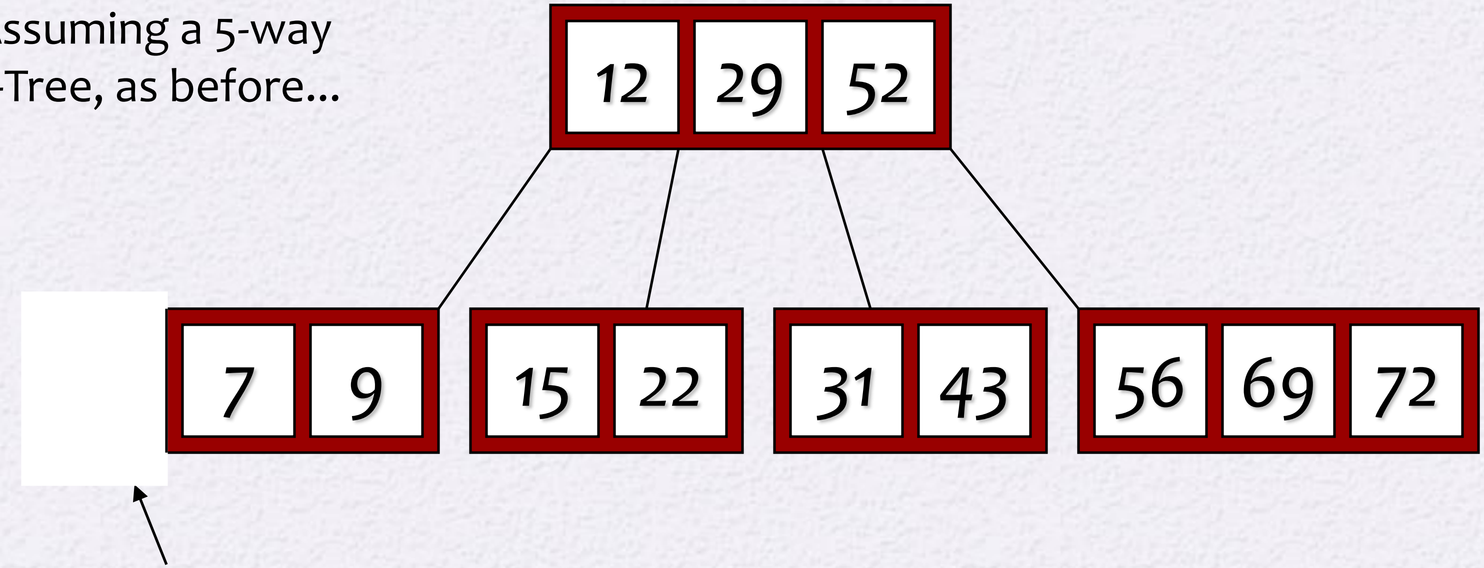
- (1) & (2) may lead to a leaf node *L* has less than min. number of keys

→ Look at the siblings immediately adjacent to the leaf

- 3: if one of them has more than the min. number of keys then we can promote one of its keys to the parent and take the parent key into *L*
- 4: otherwise, combine *L* and one of its neighbours with their shared parent, repeat the process up to the root, if required

Type #1: Simple leaf deletion

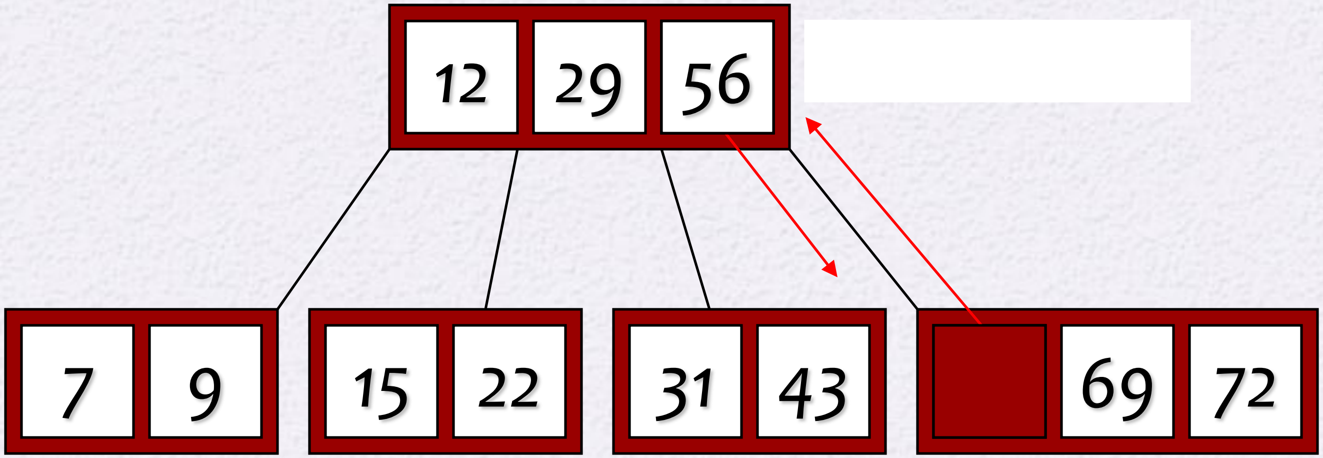
Assuming a 5-way B-Tree, as before...



Delete 2: Since there are enough keys in the node, just delete it

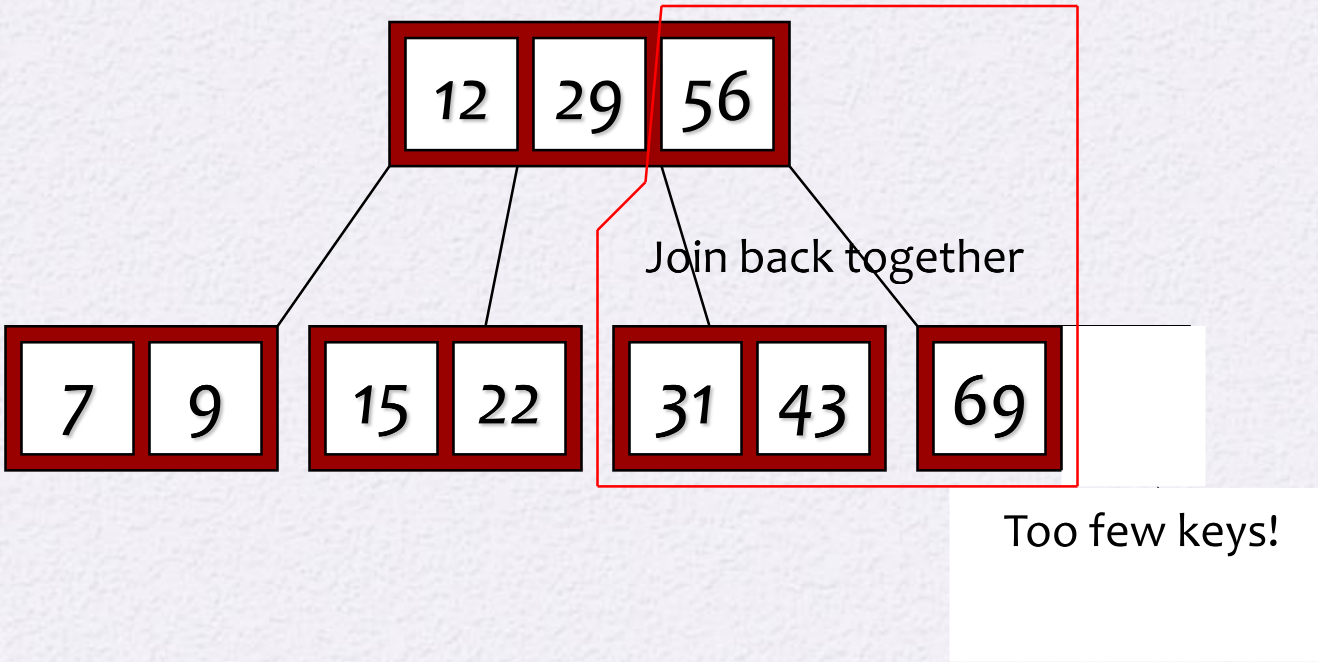
Note when printed: this slide is animated

Type #2: Simple non-leaf deletion



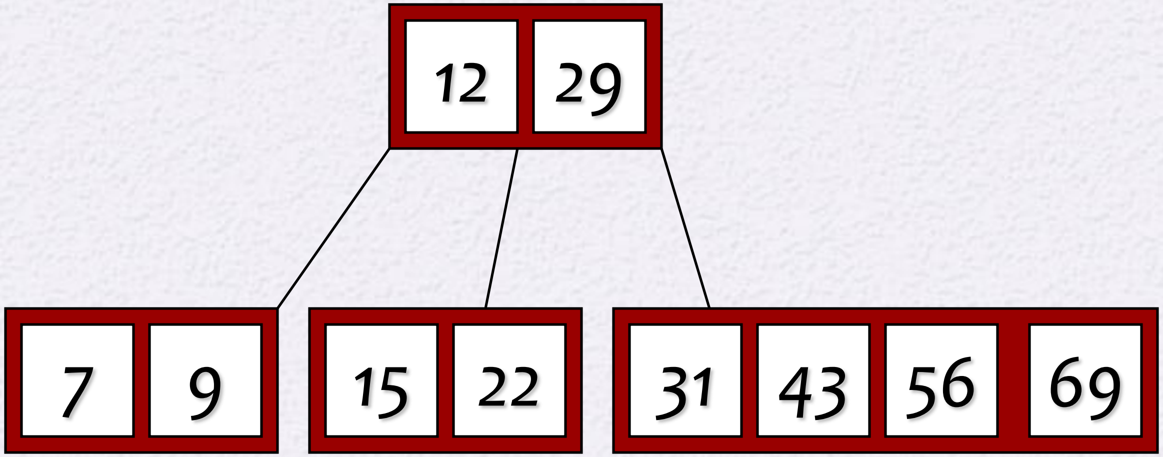
Note when printed: this slide is animated

Type #4: Too few keys in node and its siblings



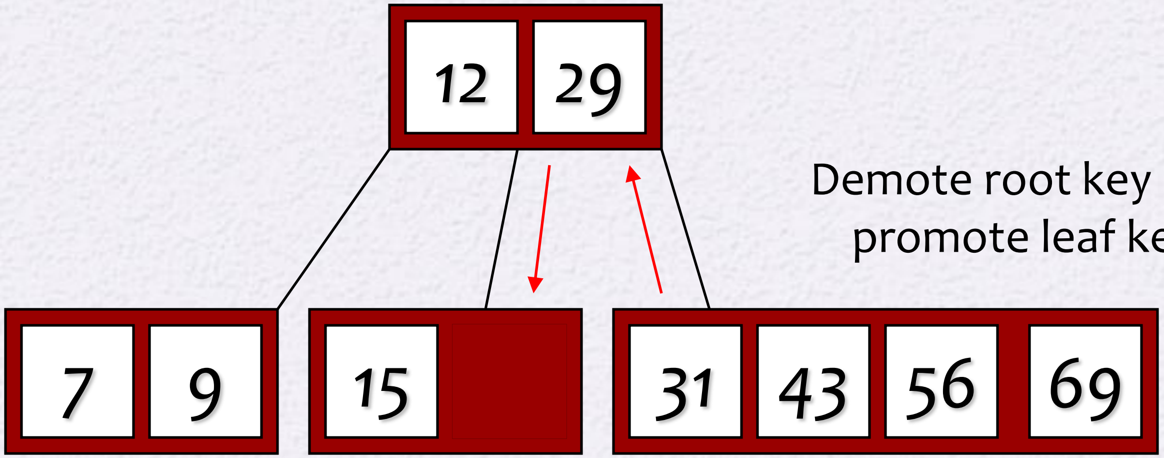
Note when printed: this slide is animated

Type #4: Too few keys in node and its siblings



Note when printed: this slide is animated

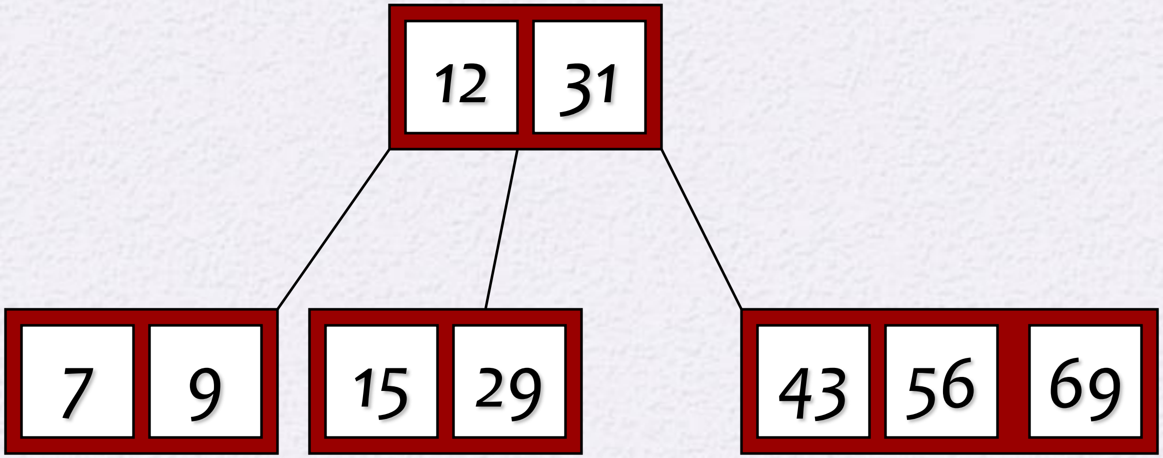
Type #3: Enough siblings



Demote root key and
promote leaf key

Note when printed: this slide is animated

Type #3: Enough siblings



Note when printed: this slide is animated

Exercise in Removal from a B-Tree

- Given 5-way B-tree created by these data:

3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56

- Add these further keys: 2, 6,12

- Delete these keys: 4, 5, 7, 3, 14

Reasons for using B-Trees

- When searching tables held on disc, the cost of each disc transfer is high but doesn't depend much on the amount of data transferred, especially if consecutive items are transferred
  - If we use a B-tree of order 101, say, we can transfer each node in one disc read operation
  - A B-tree of order 101 and height 3 can hold $101^4 - 1$ items (approximately 100 million) and any item can be accessed with 3 disc reads (assuming we hold the root in memory)

- If we take m = 3, we get a **2-3 tree**, in which non-leaf nodes have two or three children (i.e., one or two keys)
  - B-Trees are always balanced (since the leaves are all at the same level), so 2-3 trees make a good type of balanced tree

Comparing Trees

- Binary trees
  - Can become *unbalanced* and *lose* their good time complexity (big O)
  - AVL trees are strict binary trees that *overcome the balance problem*
  - Heaps remain balanced but only *prioritise* (not order) the keys
- Multi-way trees
  - B-Trees can be *m*-way, they can have any (odd) number of children
  - One B-Tree, the 2-3 (or 3-way) B-Tree, *approximates* a permanently balanced binary tree, exchanging the AVL tree's balancing operations for insertion and (more complex) deletion operations

Q&A