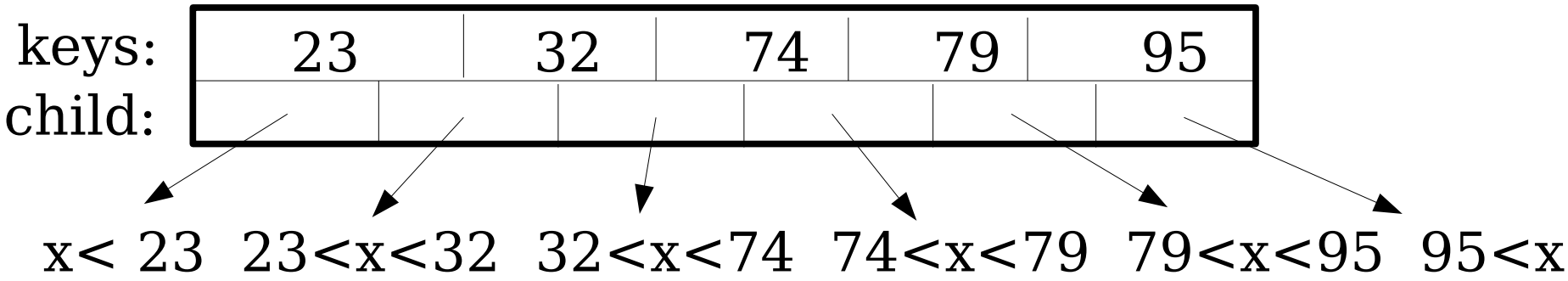
BTrees

Balanced trees which support O(log(n)) searching

BTrees

Balanced trees which support O(log(n)) searching

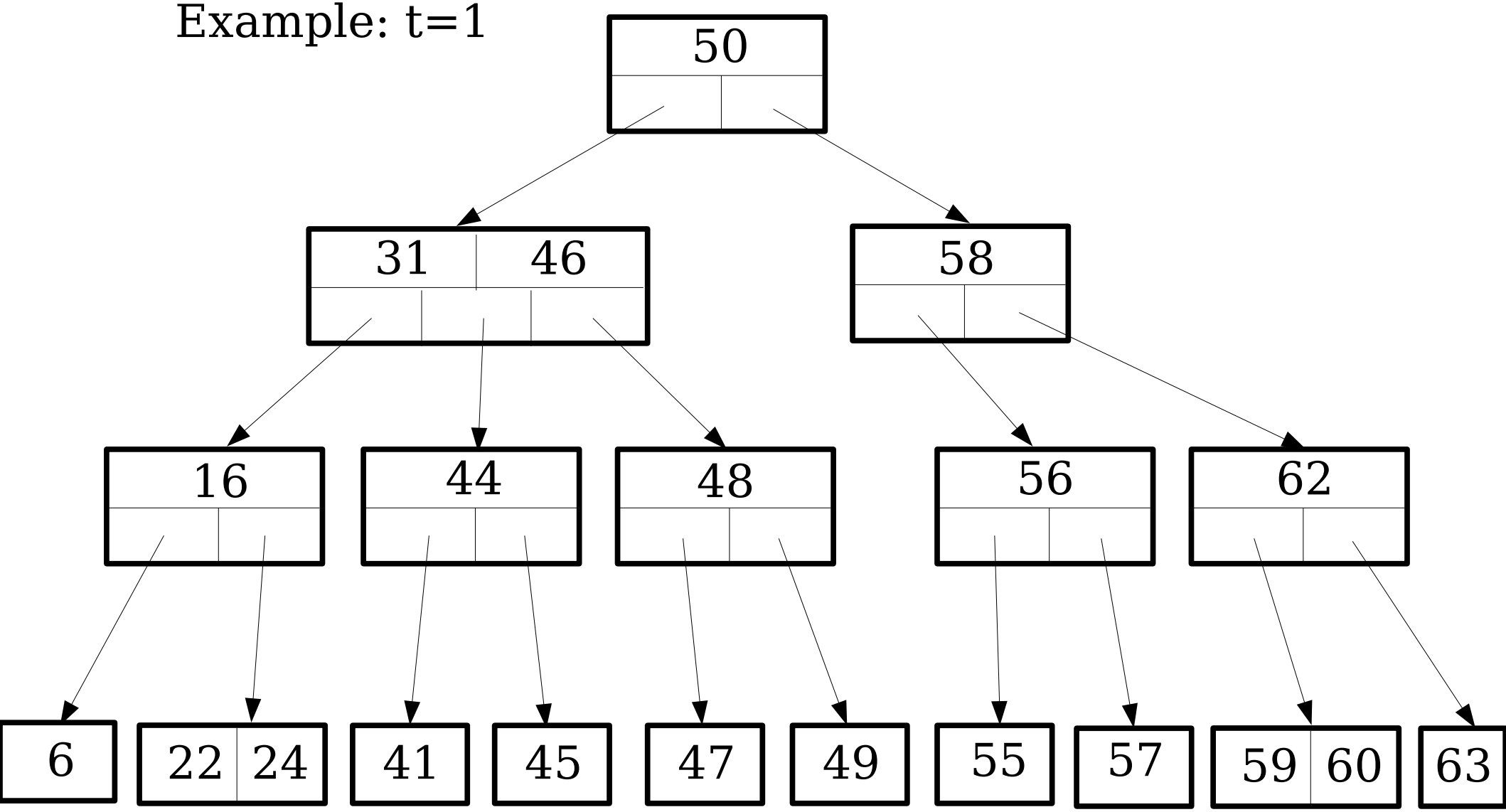
1. Every node has the following fields:
   - nkeys: The number of keys
   - keys: Array of keys stored in increasing order
   - leaf: True if no children exist for this node
   - child: Array of nkeys+1 children



2. Every leaf has the same depth
3. There is a number t s.t. all nodes other than root have at least t but no more than 2*t keys
4. A node is full if it has 2*t keys

BTrees

Example: t=1



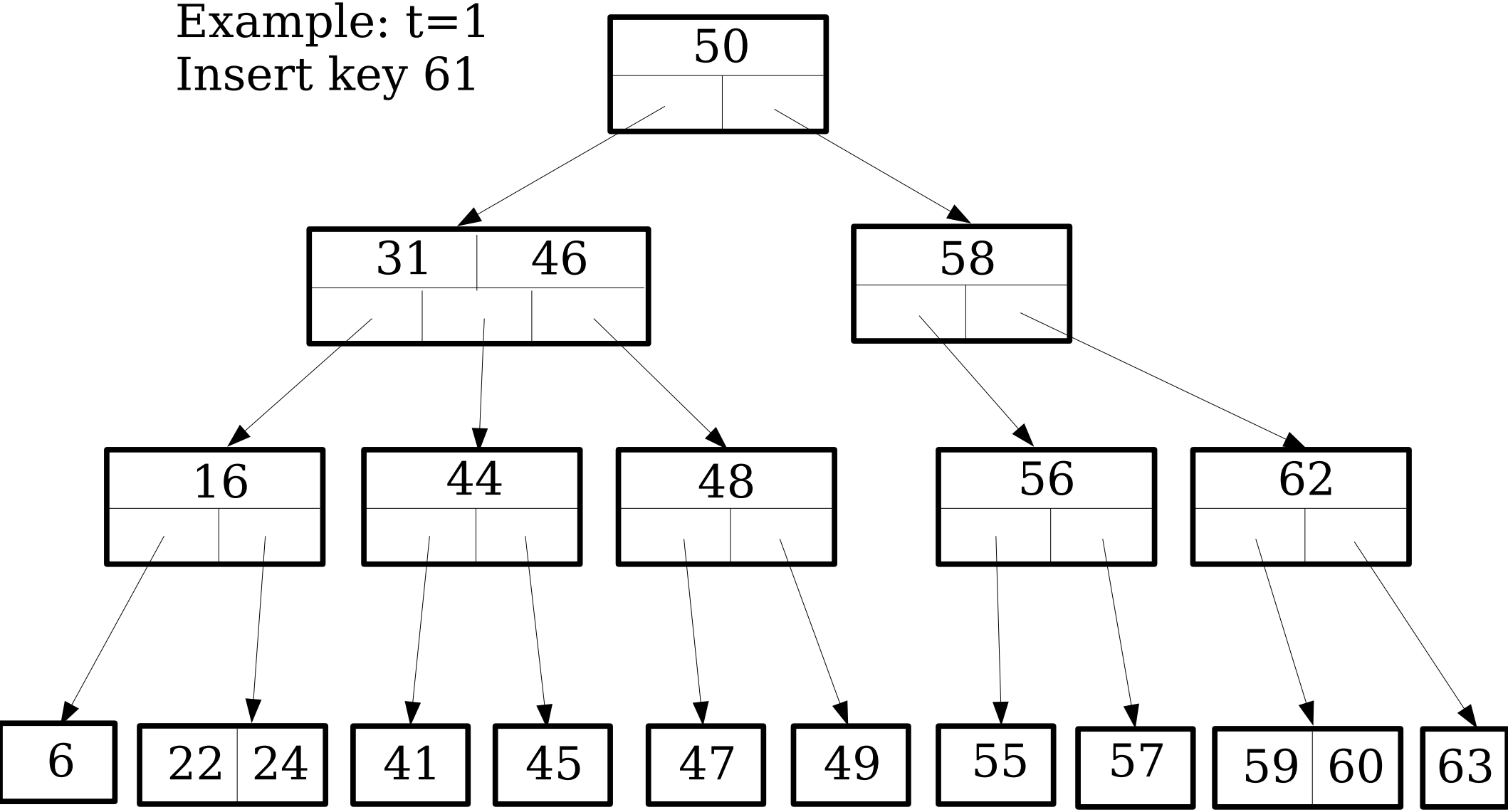
BTrees

Insertion of a node:

1. Compare down to leaf node
2. Put the new object in its rightful place so the leaf's list of keys is in increasing order
3. If the number of keys at the leaf is 2*t+1, split the node into two nodes of t keys each, the first t keys being in the left node and the last t keys being in the right node
4. Move the middle (t^{th}) key to the parent node and put it in its rightful place
5. If the number of keys in the parent is 2*t+1, repeat steps 3 and 4 on the parent
6. Continue until all nodes have a number of keys no greater than 2*t

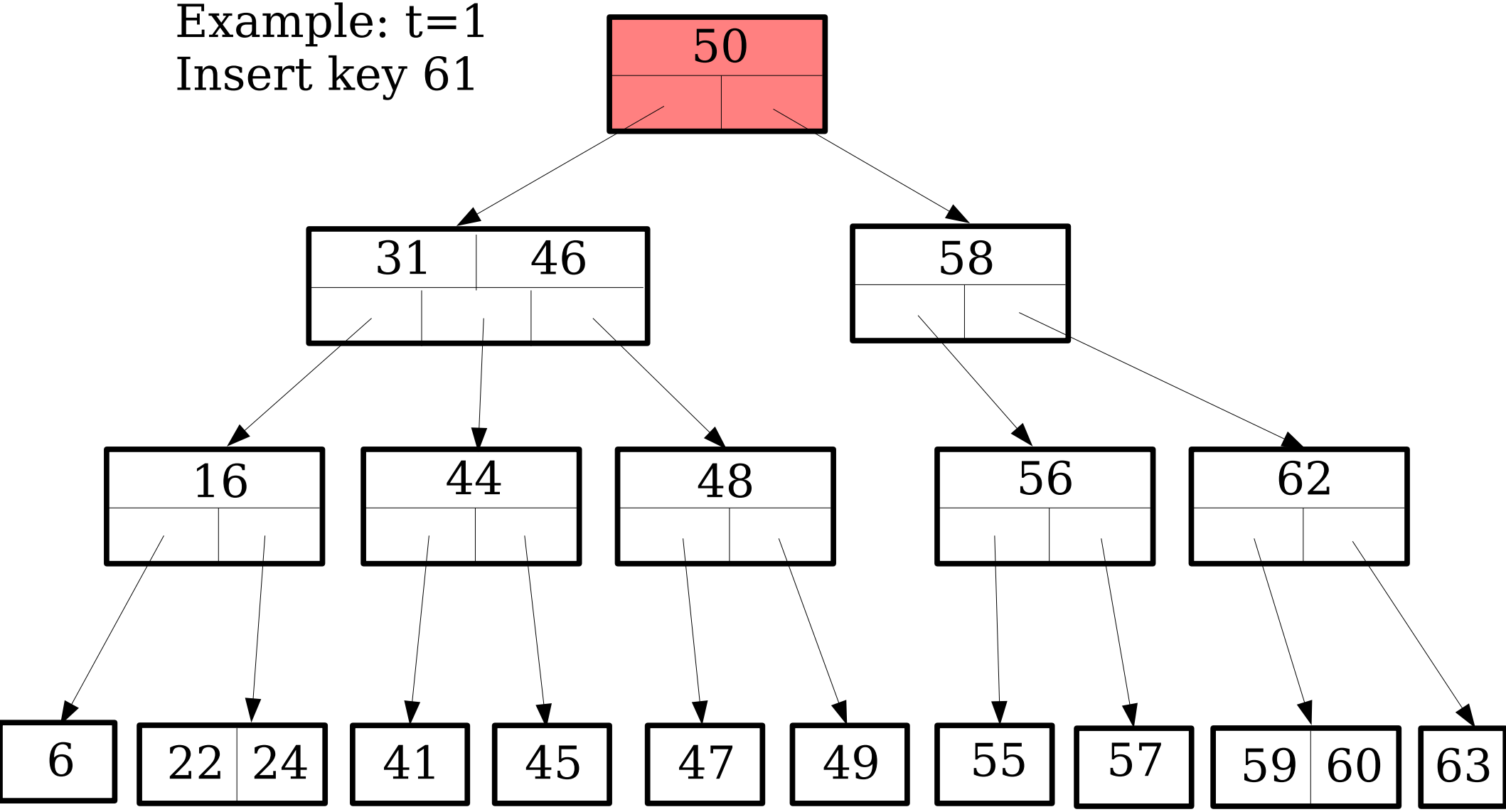
BTrees

Example: t=1
Insert key 61



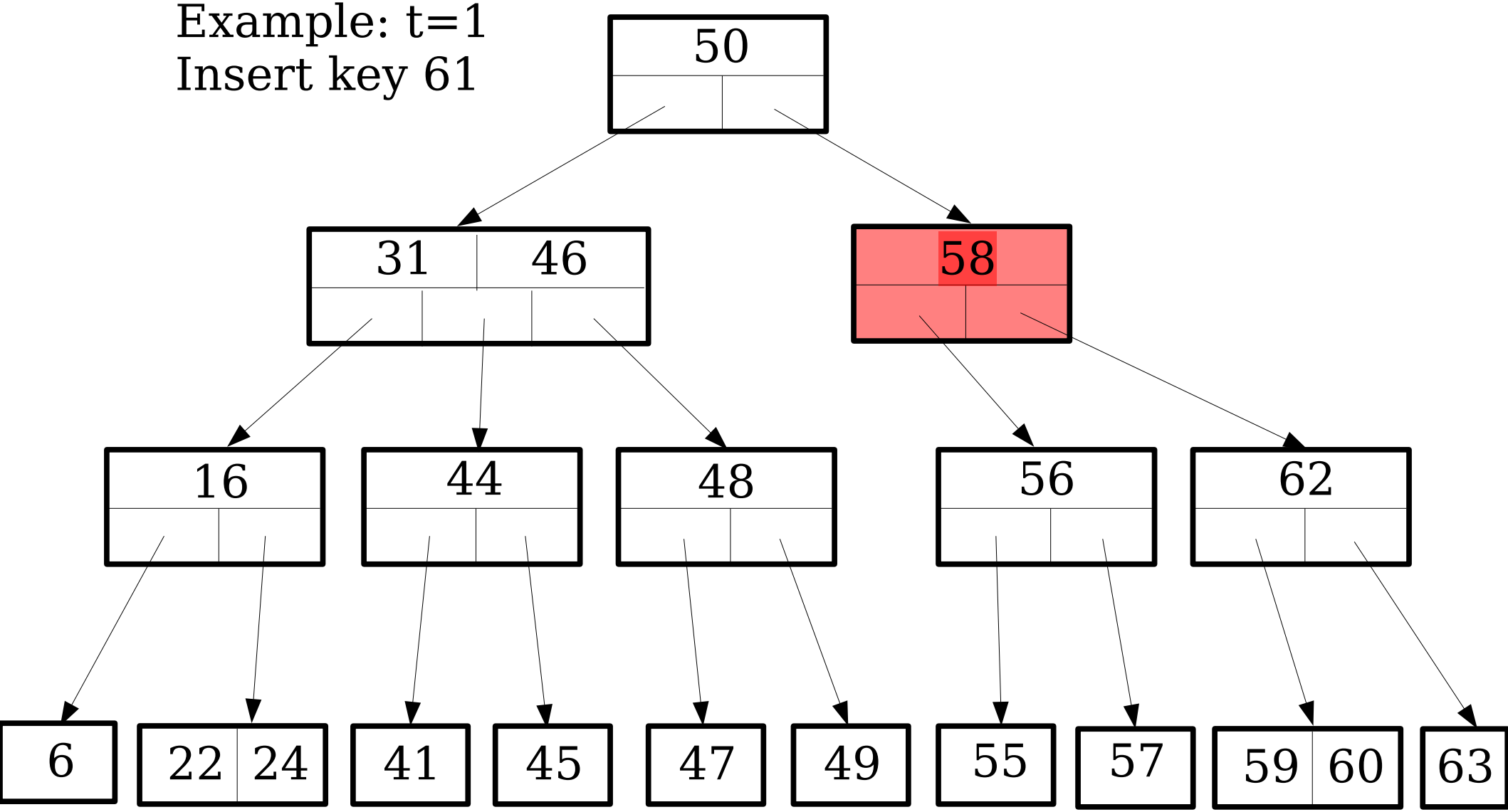
BTrees

Example: t=1
Insert key 61



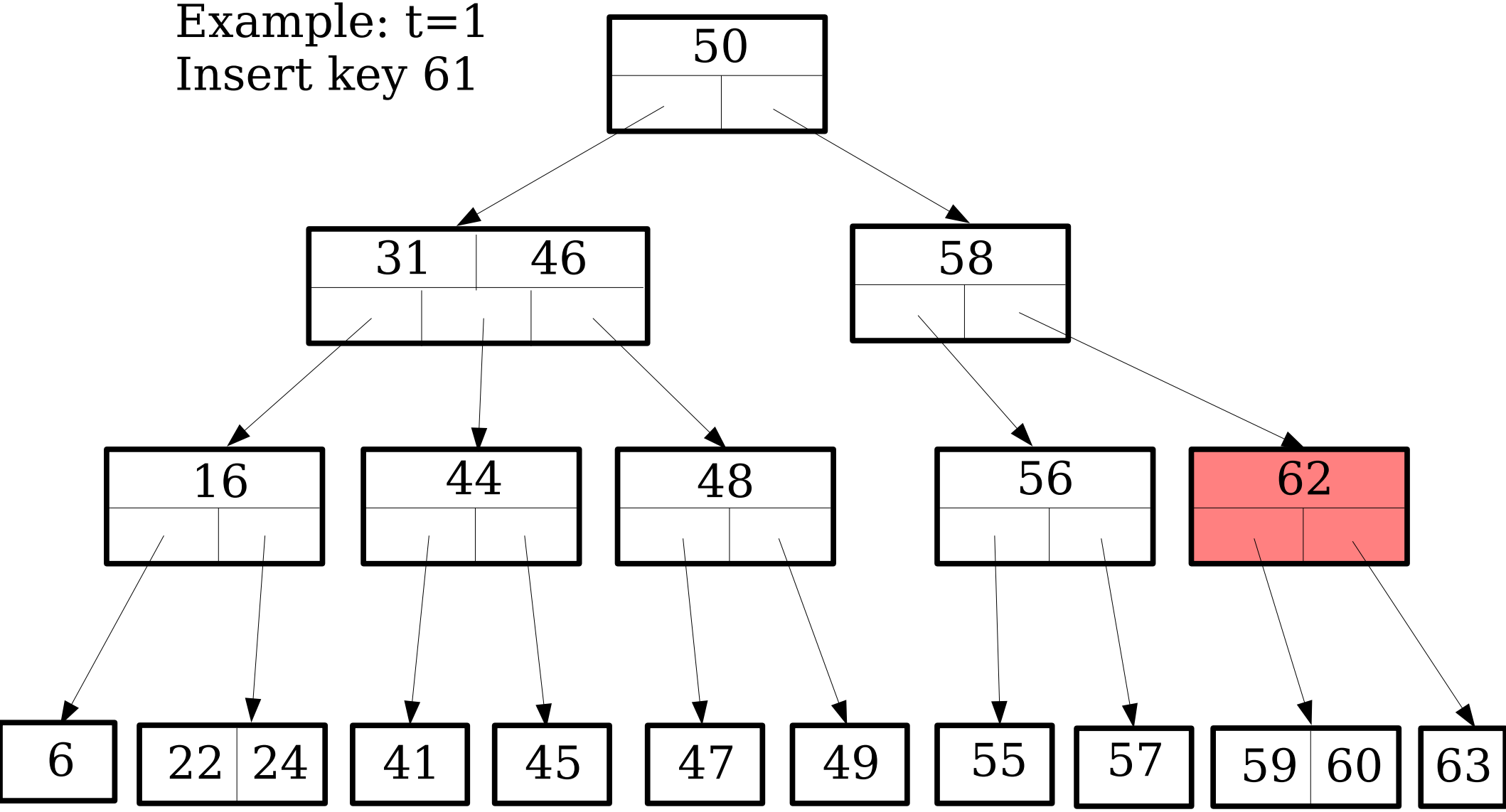
BTrees

Example: t=1
Insert key 61



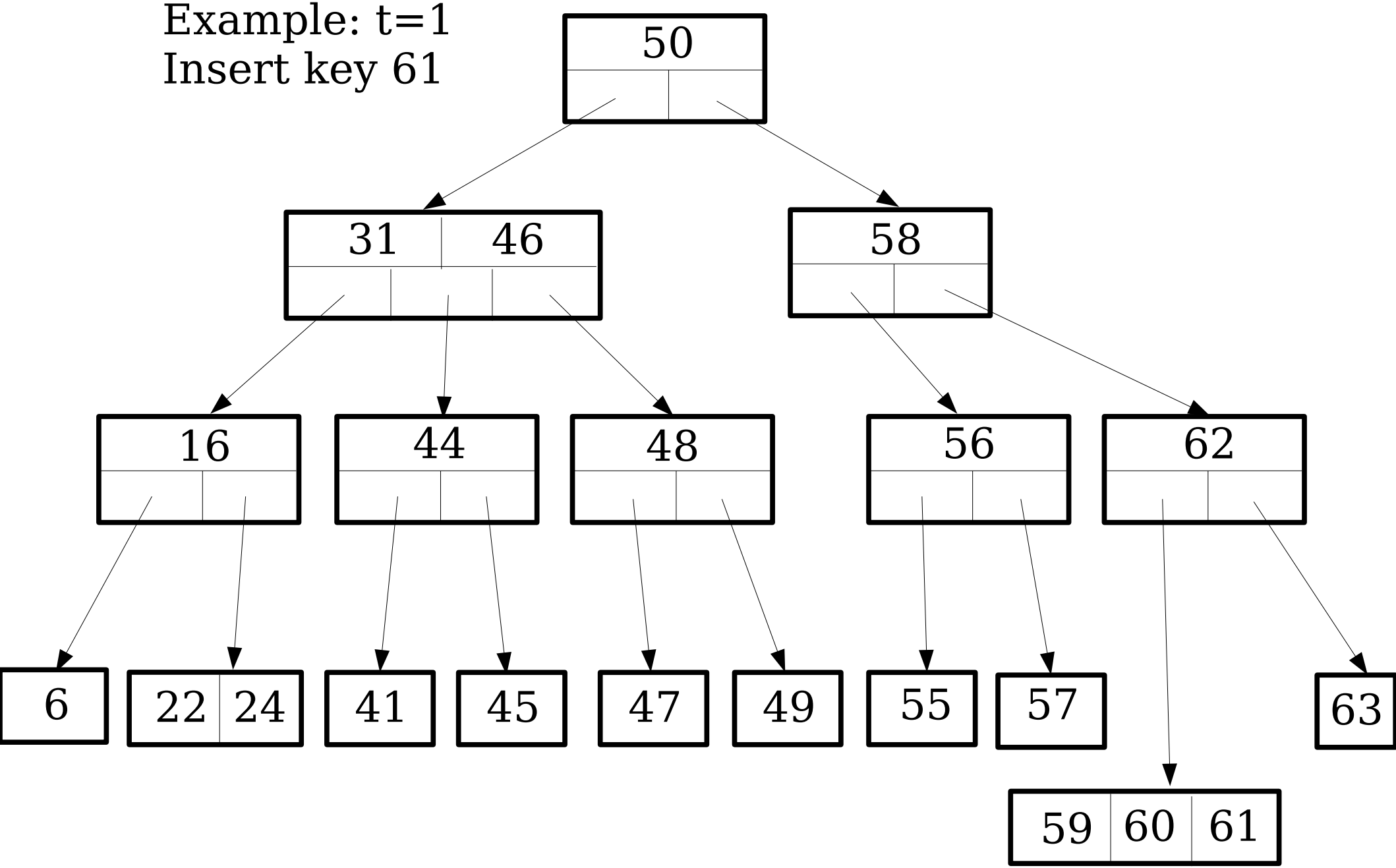
BTrees

Example: t=1
Insert key 61



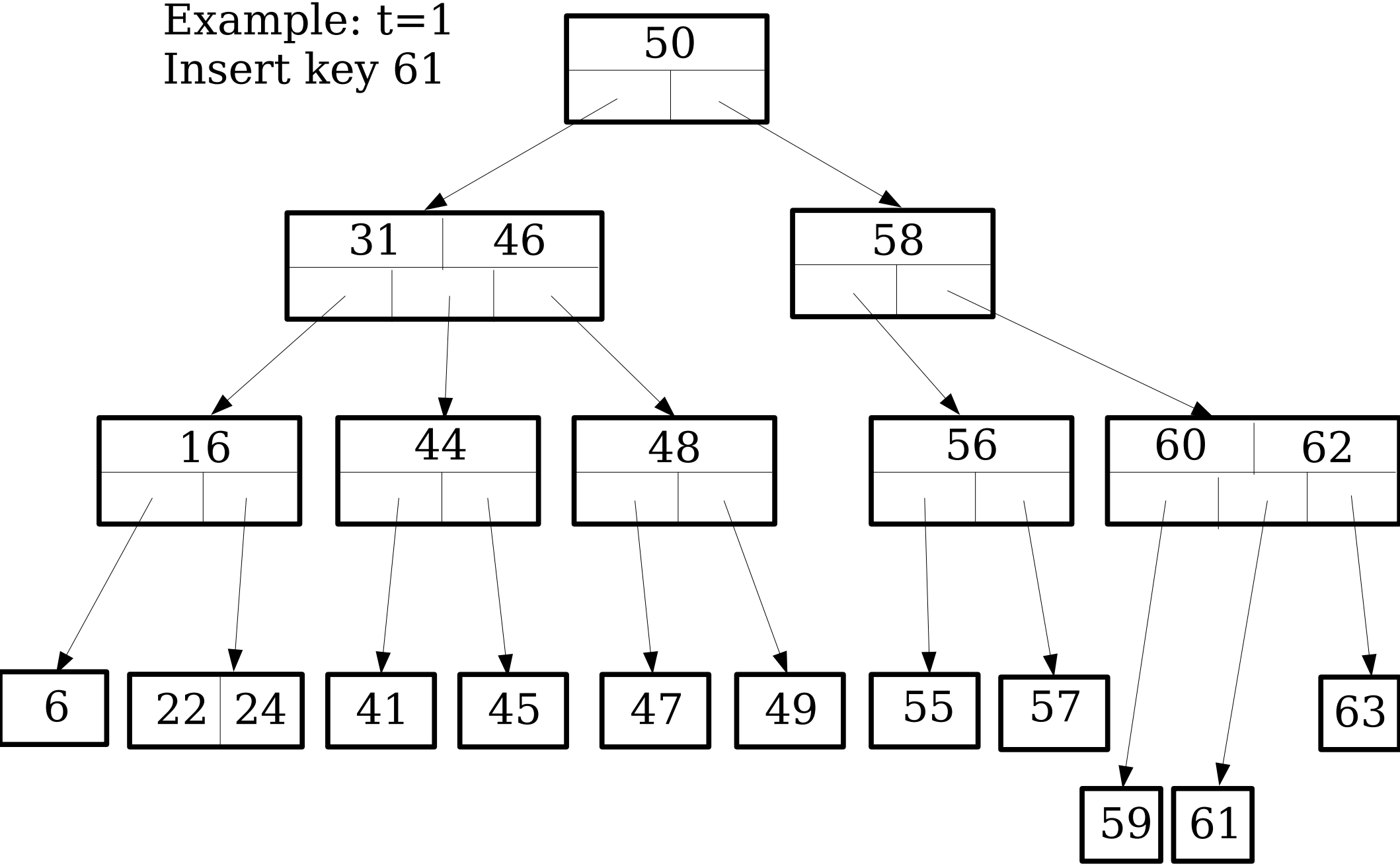
BTrees

Example: t=1
Insert key 61



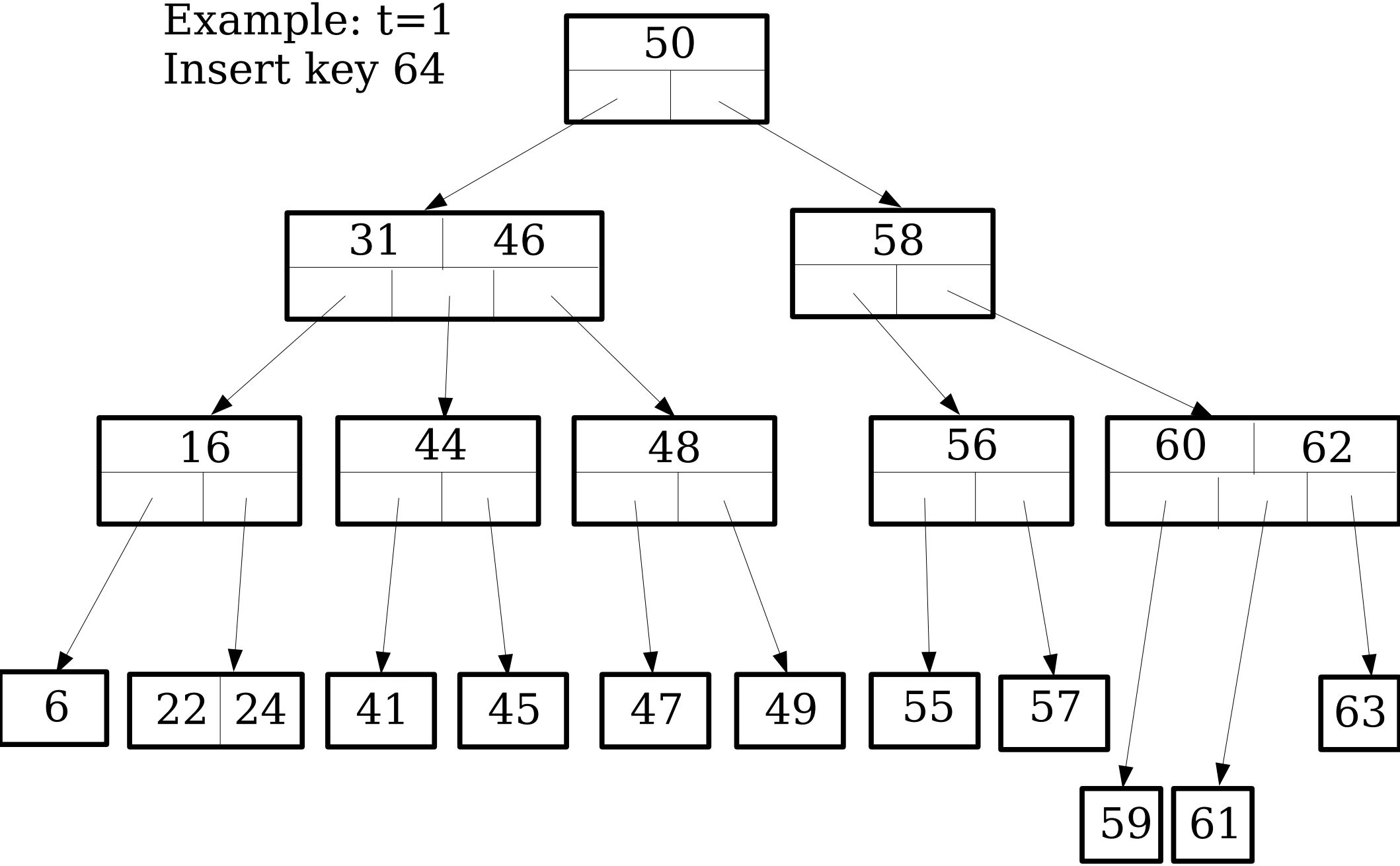
BTrees

Example: t=1
Insert key 61



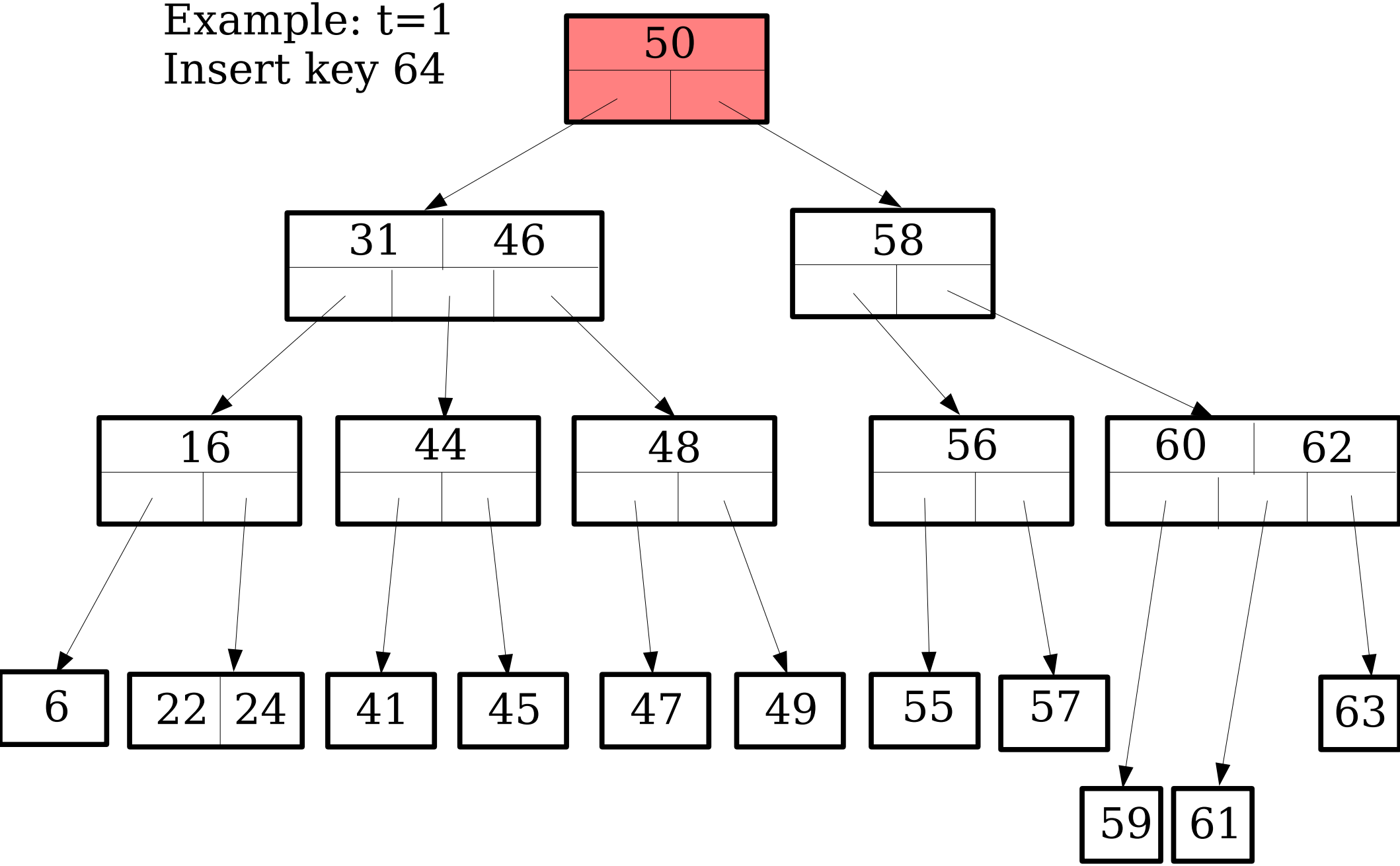
BTrees

Example: t=1
Insert key 64



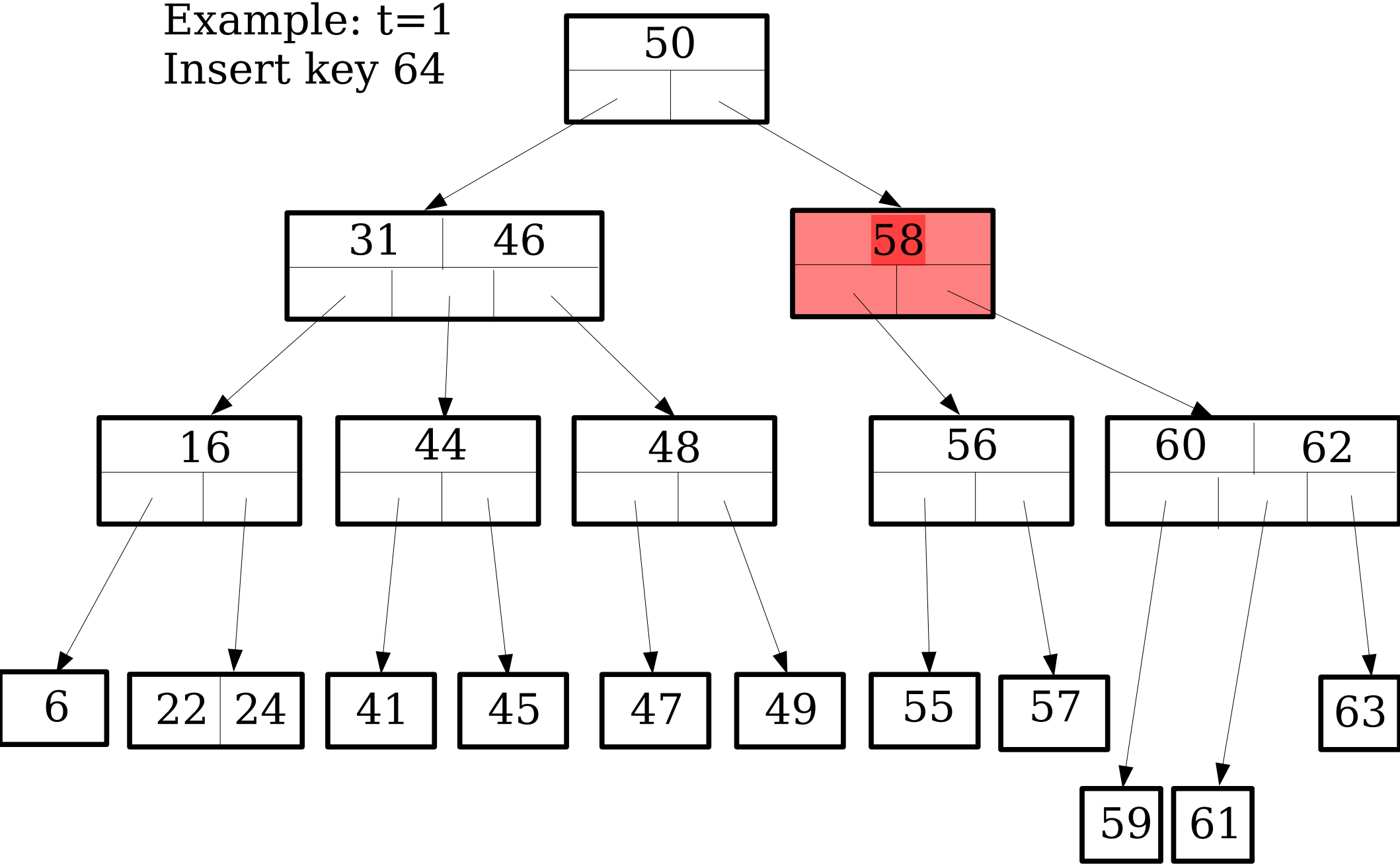
BTrees

Example: t=1
Insert key 64



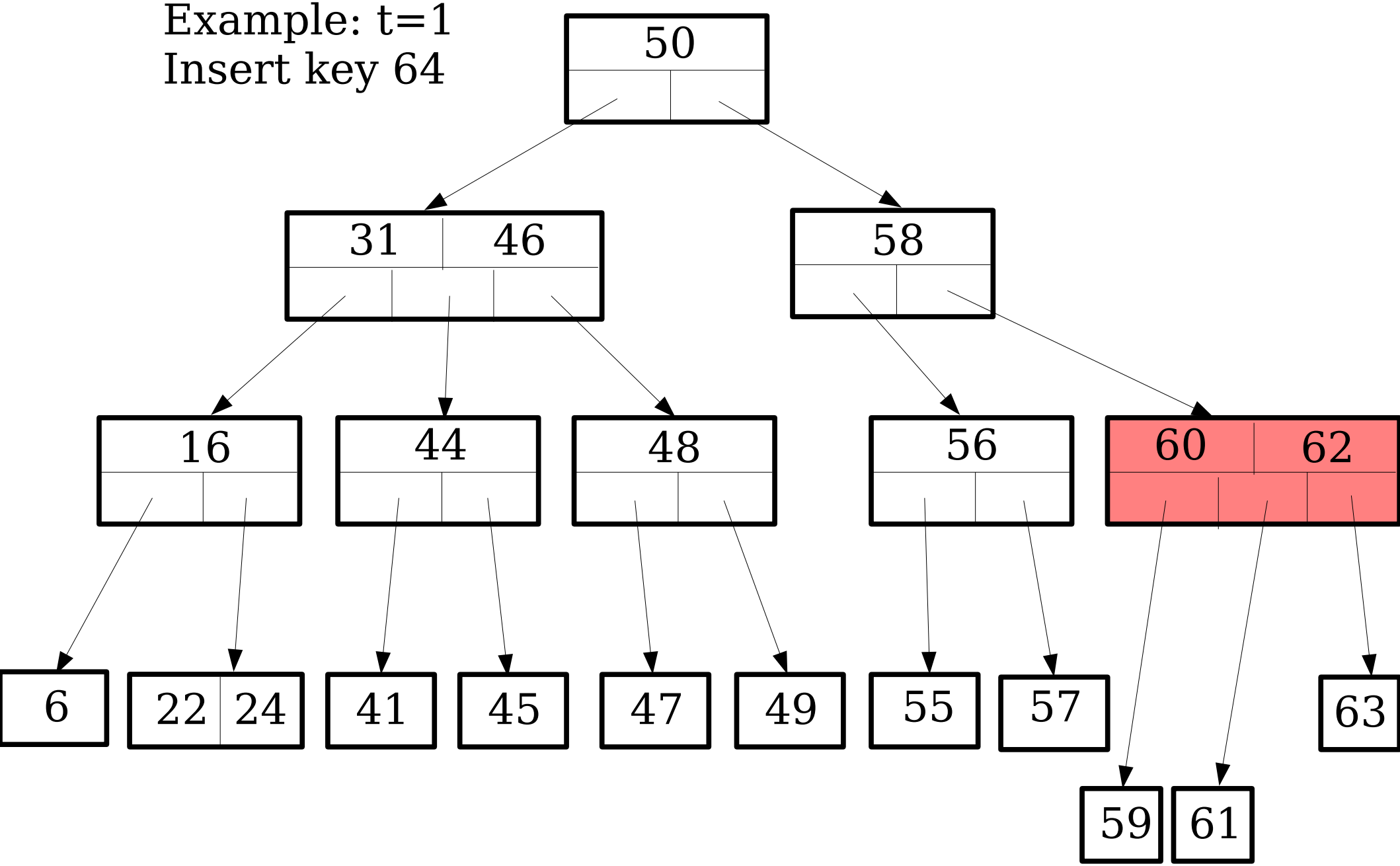
BTrees

Example: t=1
Insert key 64



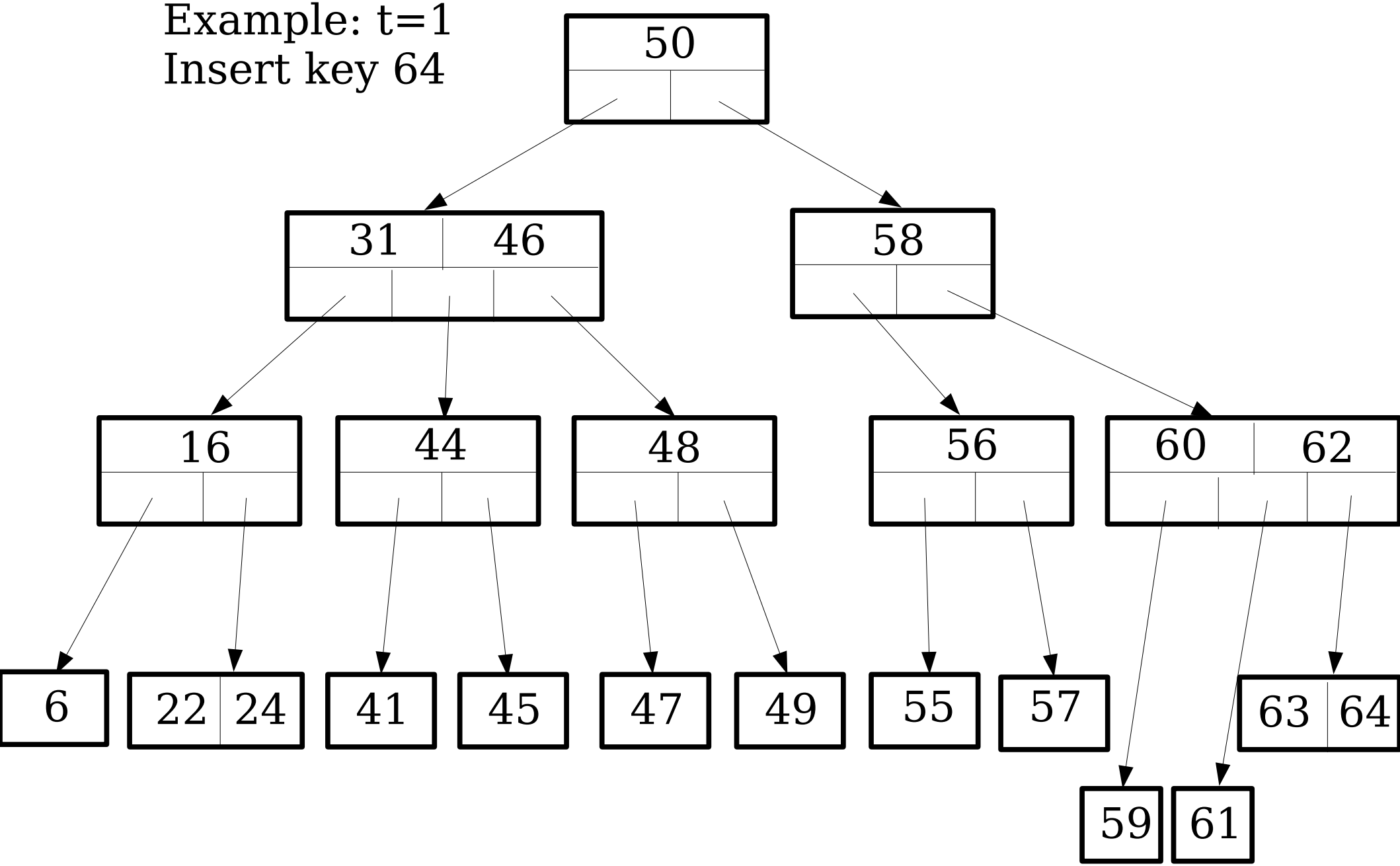
BTrees

Example: t=1
Insert key 64



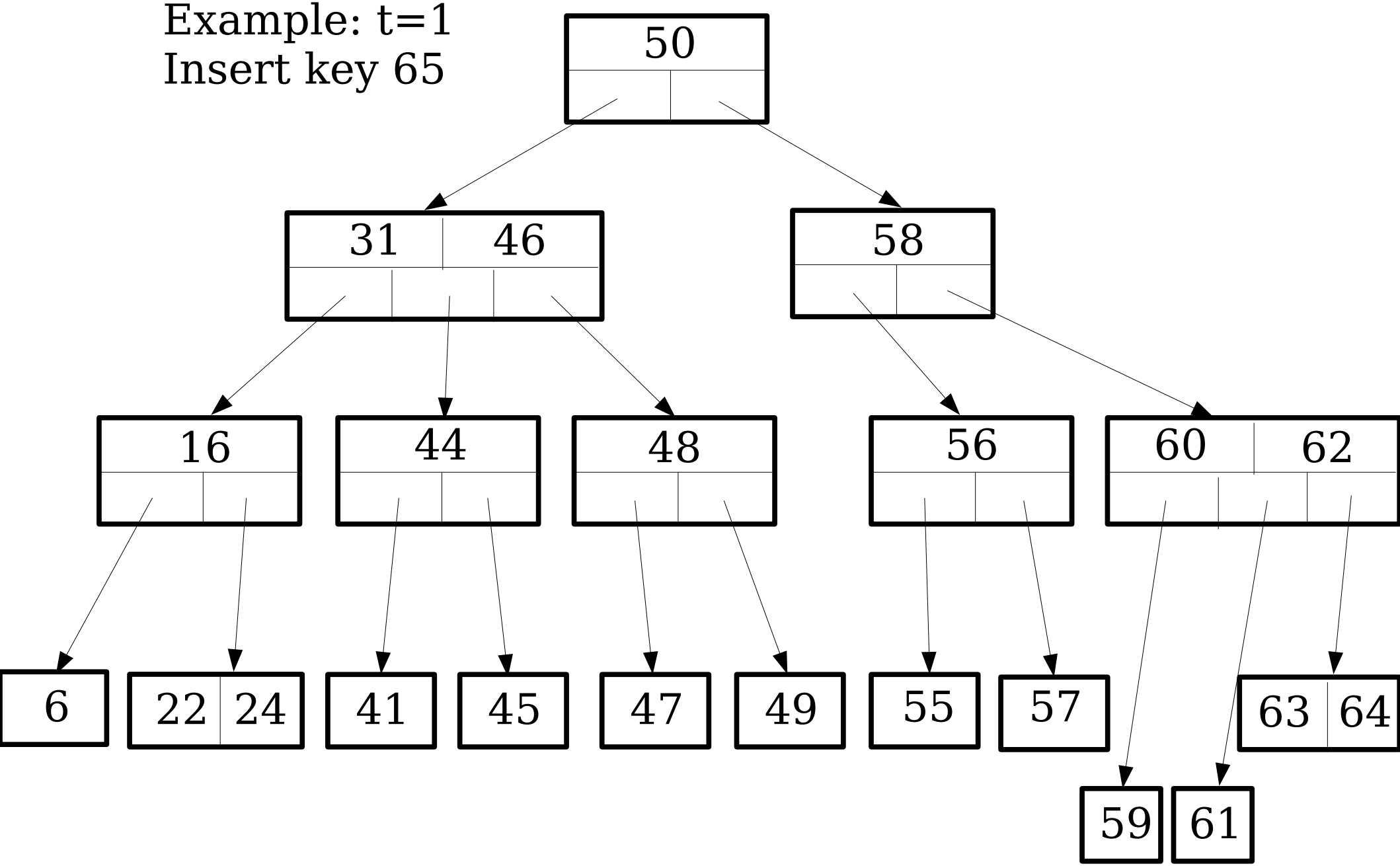
BTrees

Example: t=1
Insert key 64



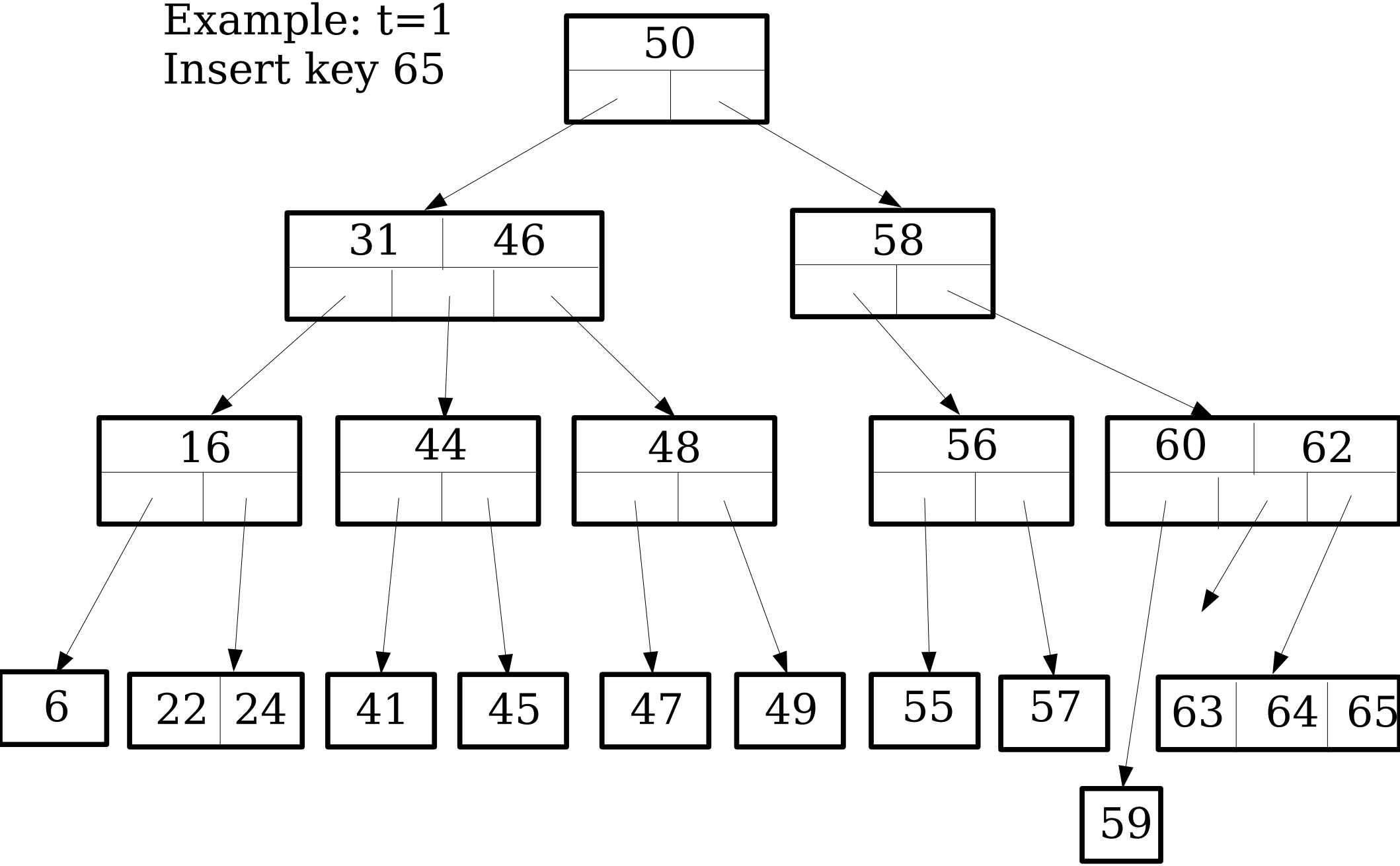
BTrees

Example: t=1
Insert key 65



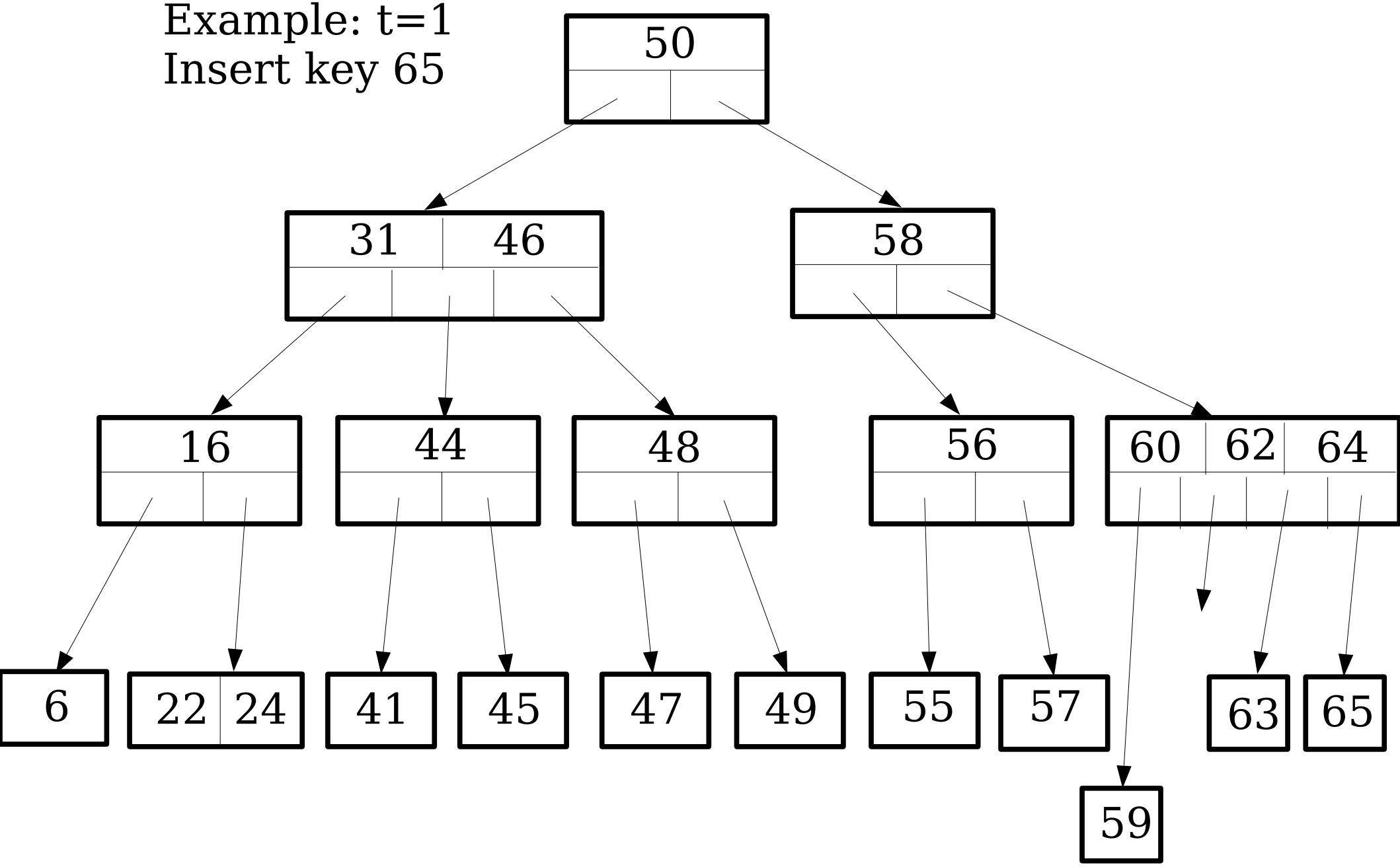
BTrees

Example: t=1
Insert key 65



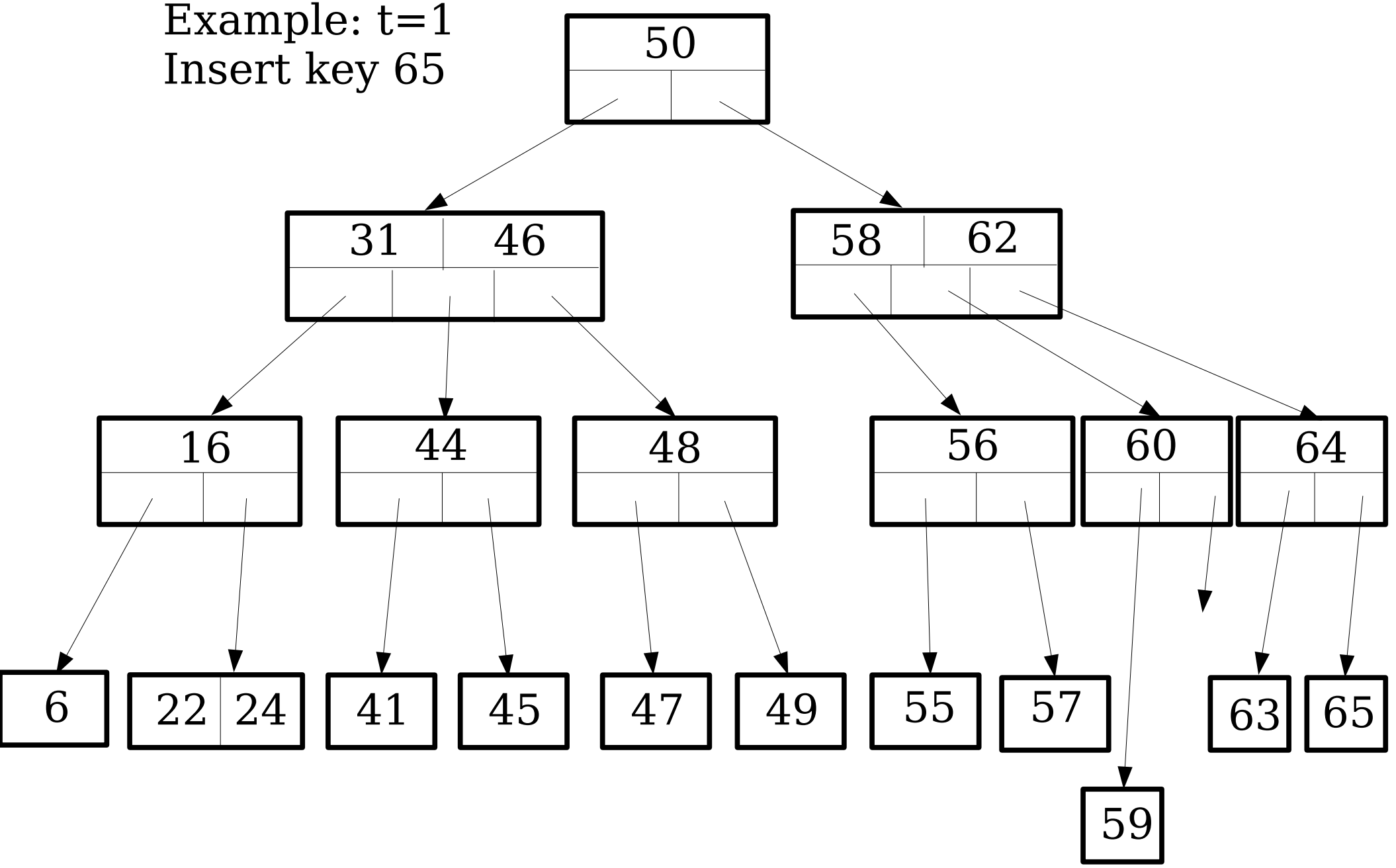
BTrees

Example: t=1
Insert key 65



BTrees

Example: t=1
Insert key 65



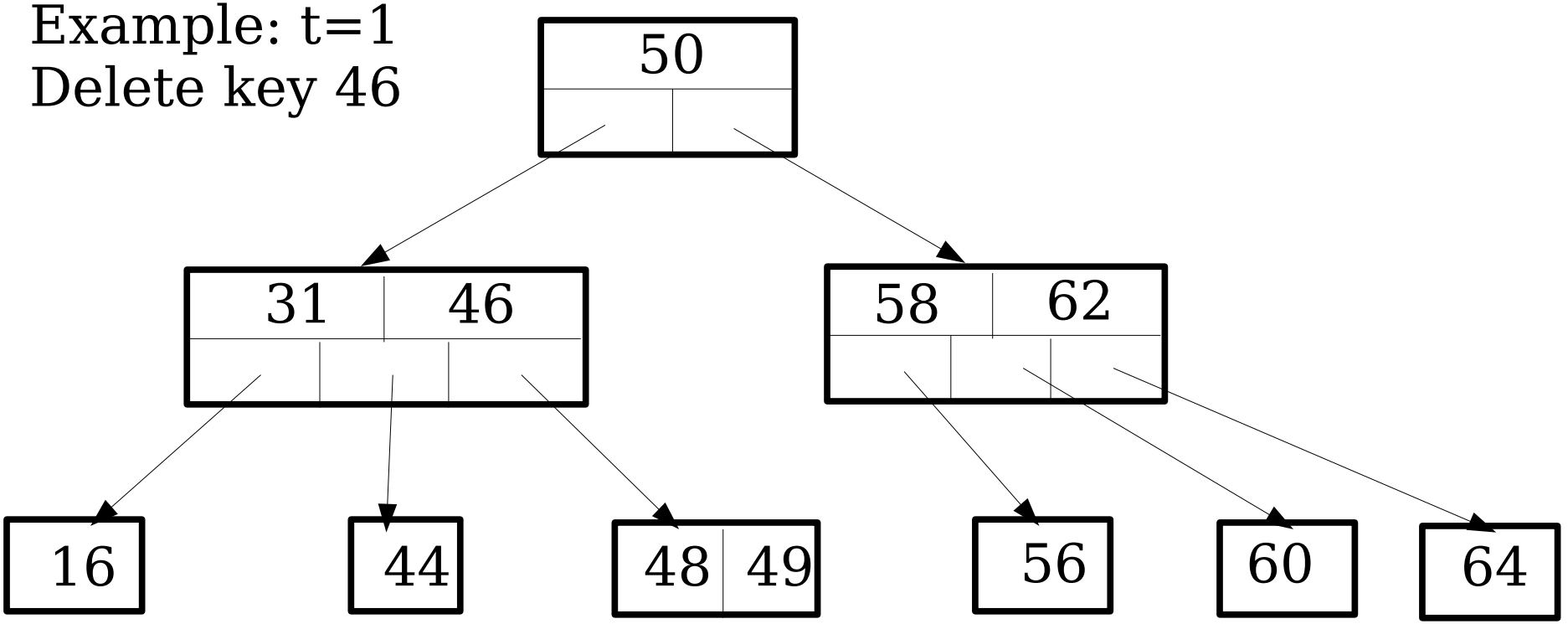
BTrees

Deletion of a node:

1. If the key to be deleted is in a node having two subtrees, replace it with the greatest key in the left subtree and delete the key from the node containing it in the left subtree
2. The largest key in the subtree will always be in a leaf node
3. Hence, delete the key from the leaf node
4. If the number of keys at the leaf node falls below the minimum, t, join the leaf node with its more populous neighbor, include the “middle” key from the node above
5. If the number of keys in the combined node is no greater than 2*t leave it
6. Otherwise, proceed as though inserting the middle node into the BTree

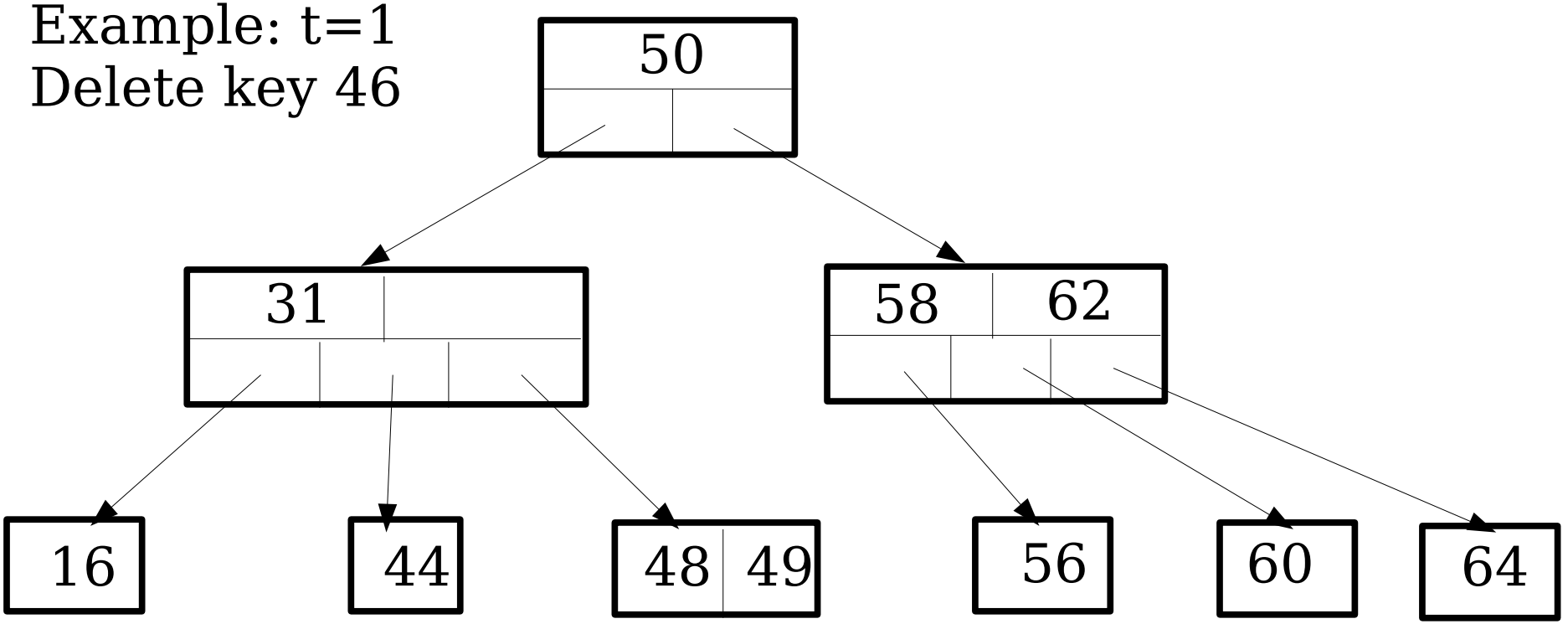
BTrees

Example: t=1
Delete key 46



BTrees

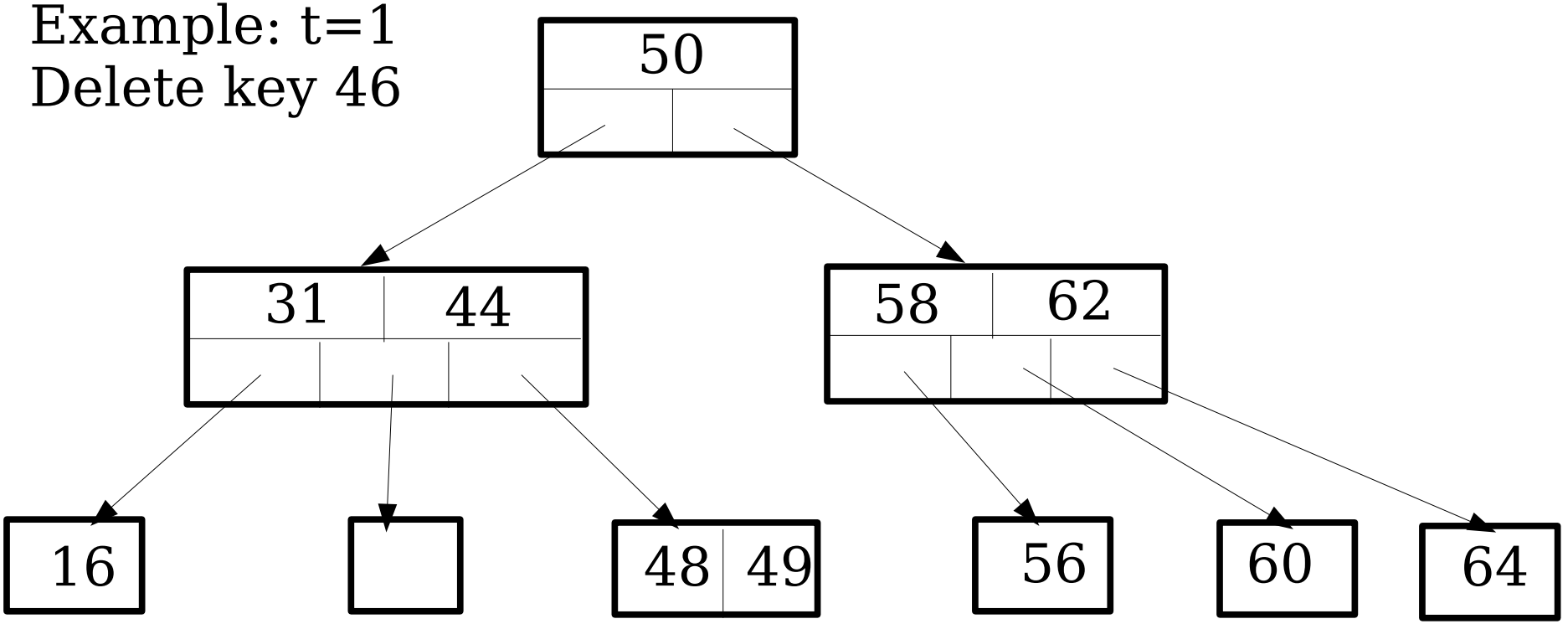
Example: t=1
Delete key 46



Find and delete key (46)

BTrees

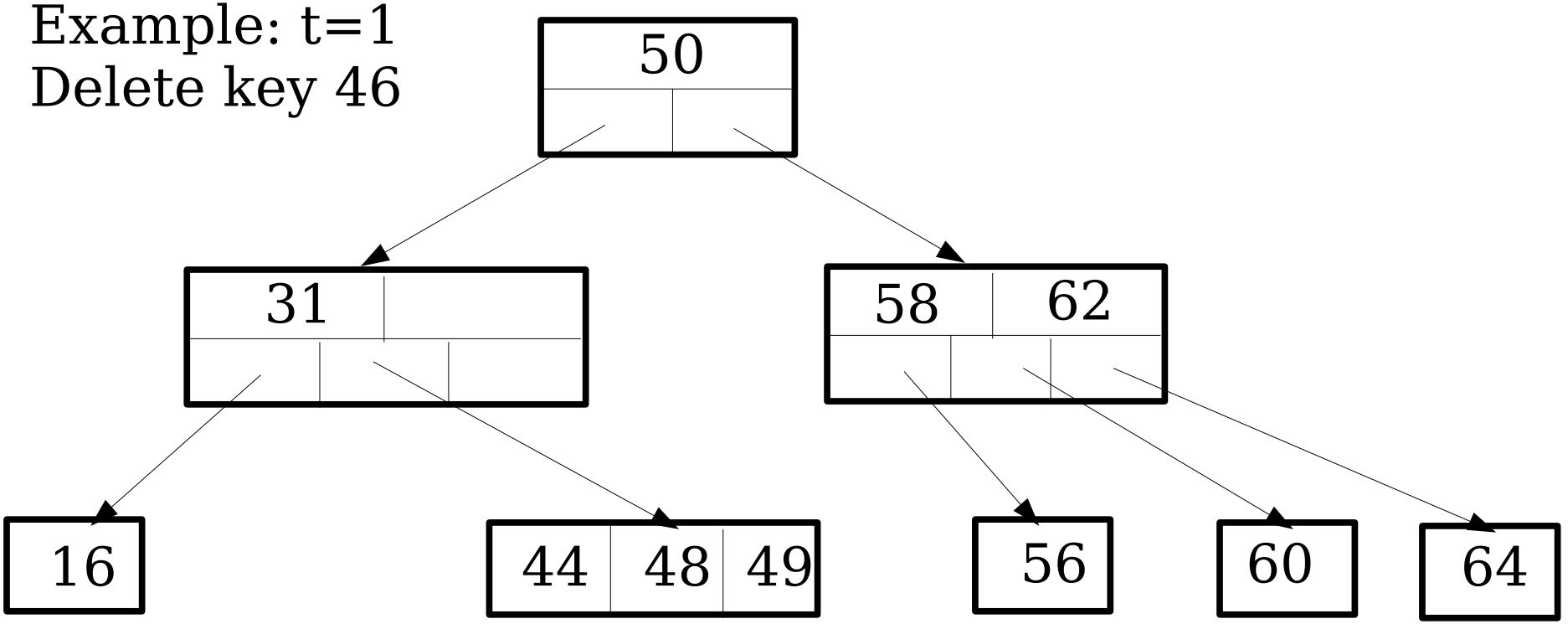
Example: t=1
Delete key 46



Move greatest node of left subtree to replace it

BTrees

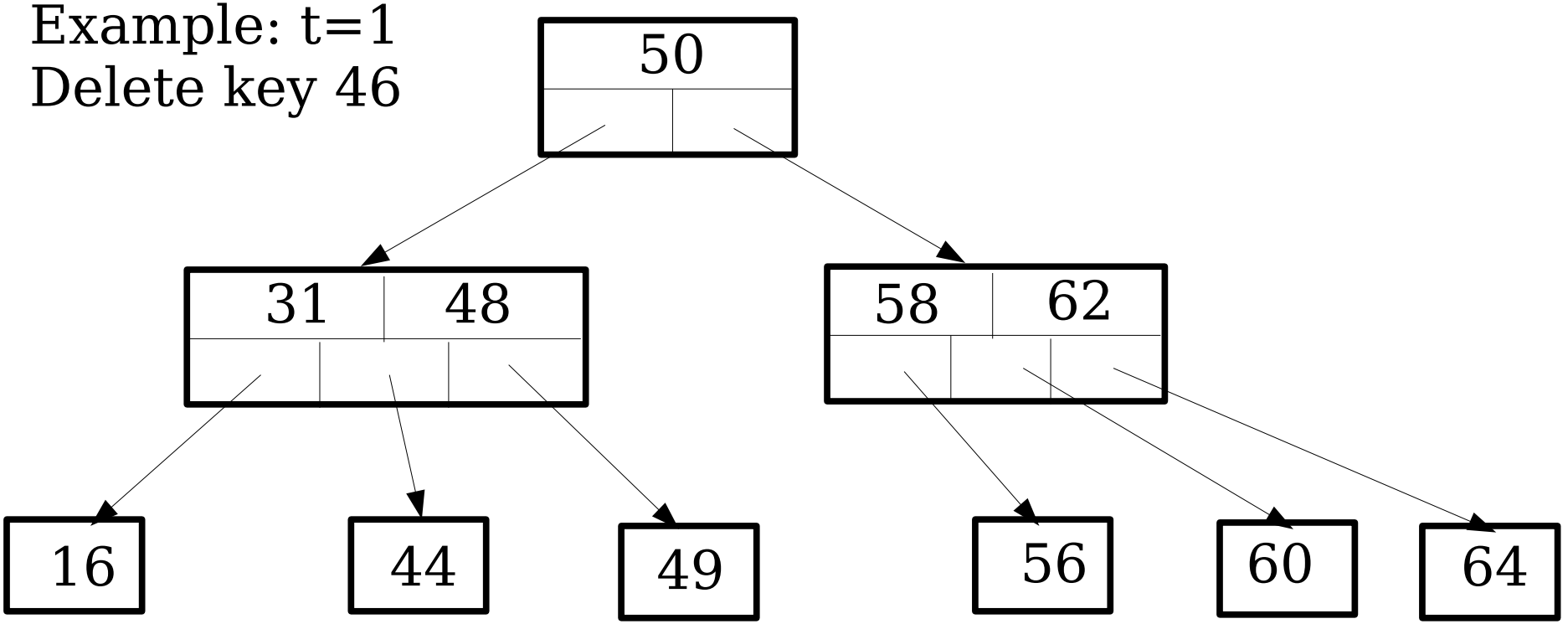
Example: t=1
Delete key 46



Combine two leaves with middle from above
The result is too many keys at the leaf

BTrees

Example: t=1
Delete key 46



Perform repair operation as in insertion