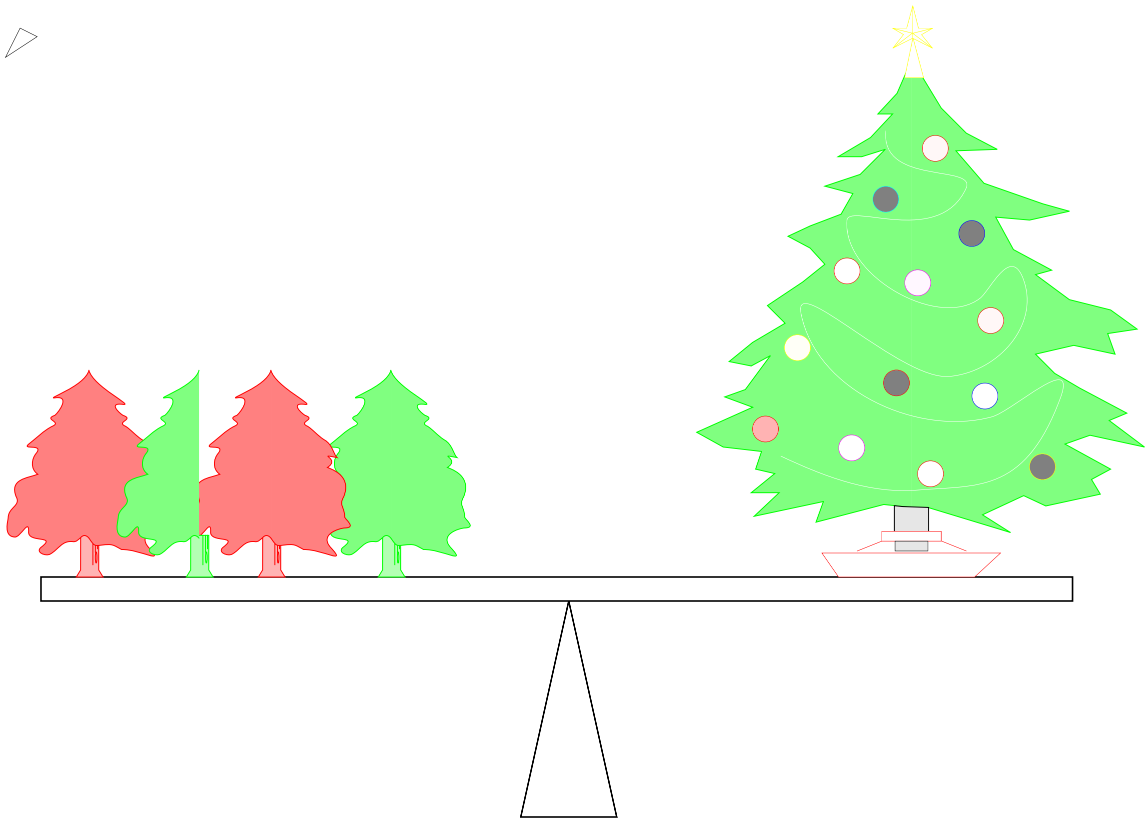
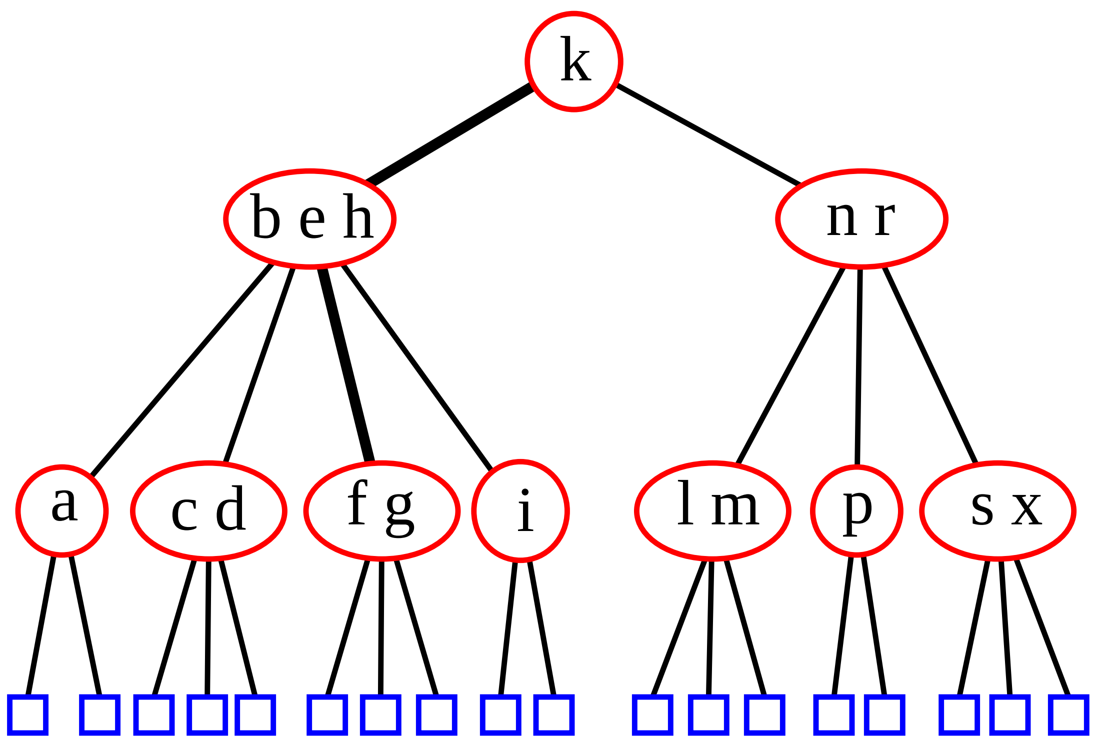
2-3-4 Trees and Red-Black Trees



2-3-4 Trees Revealed

- Nodes store 1, 2, or 3 keys and have 2, 3, or 4 children, respectively
- All leaves have the same depth



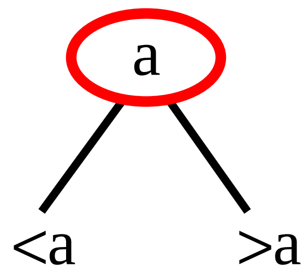
$$\frac{1}{2}\log(N+1) \leq \text{height} \leq \log(N+1)$$

2-3-4 Tree Nodes

- Introduction of nodes with more than 1 key, and more than 2 children

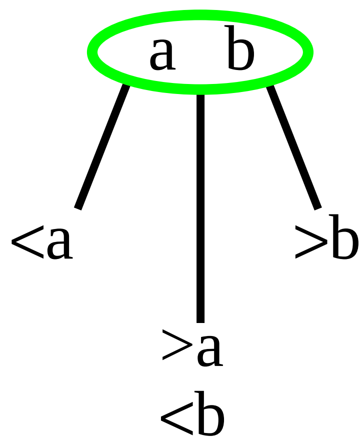
2-node:

- same as a binary node



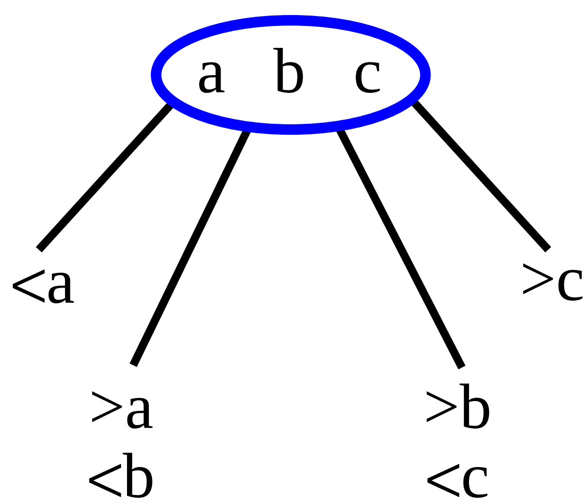
3-node:

- 2 keys, 3 links



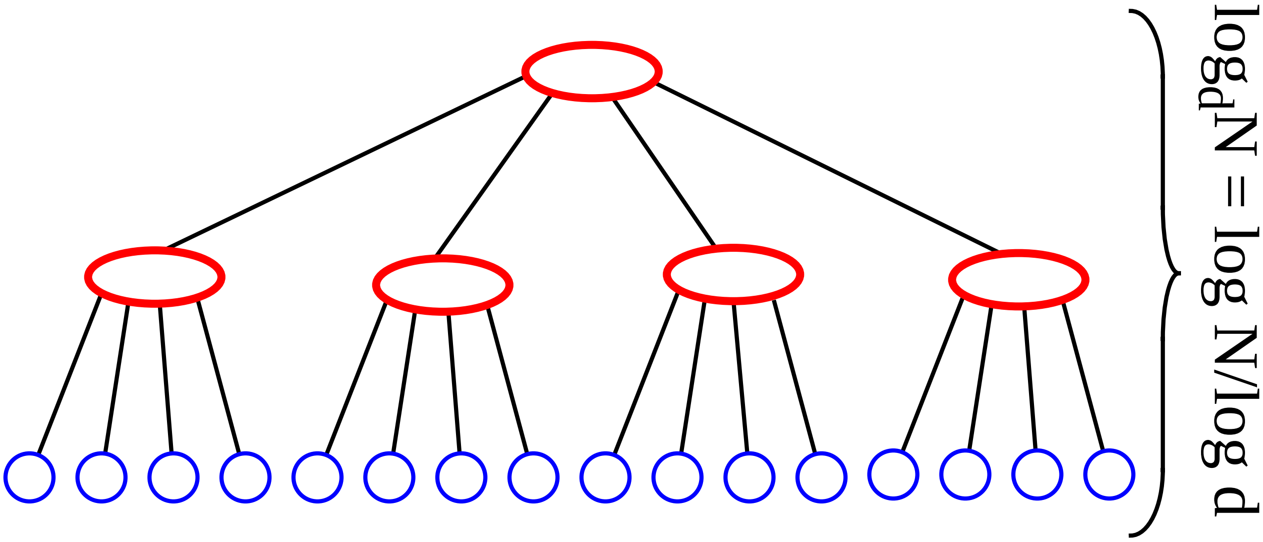
4-node:

- 3 keys, 4 links



Why 2-3-4?

- Why not minimize height by maximizing children in a "d-tree"?
- Let each node have d children so that we get O($\log_d$ N) search time! Right?

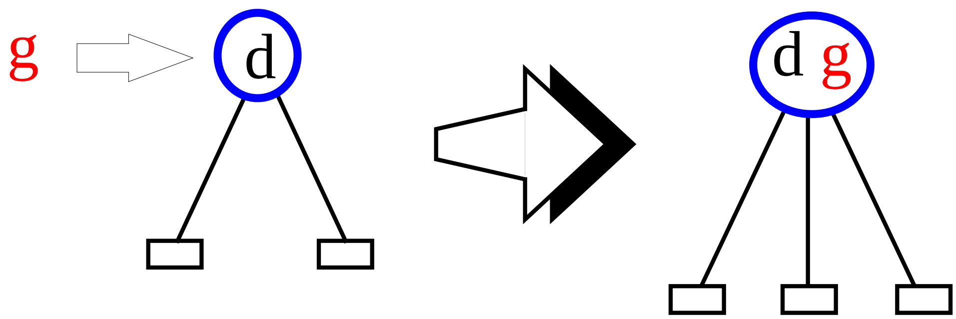


- That means if $d = N^{1/2}$, we get a height of 2
- However, searching out the correct child on each level requires O($\log N^{1/2}$) by binary search
- $2 \log N^{1/2}$ = O(log N) which is not as good as we had hoped for!
- 2-3-4-trees will guarantee O(log N) height using only 2, 3, or 4 children per node

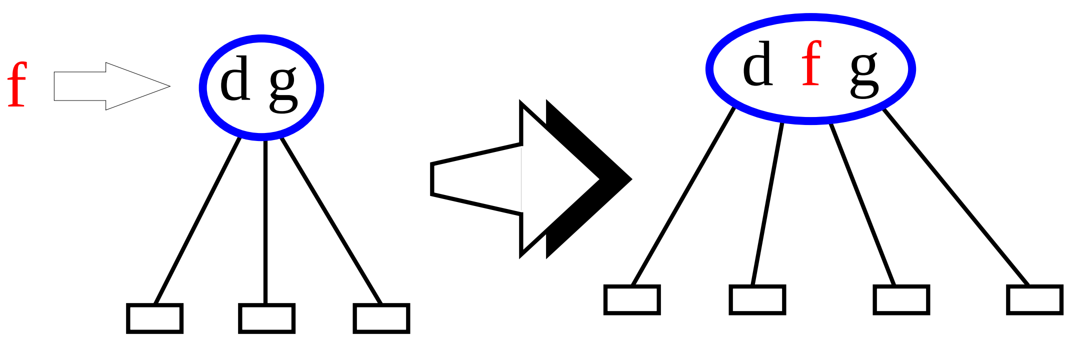
Insertion into 2-3-4 Trees

- Insert the new key at the lowest internal node reached in the search

  - ***2-node*** becomes ***3-node***



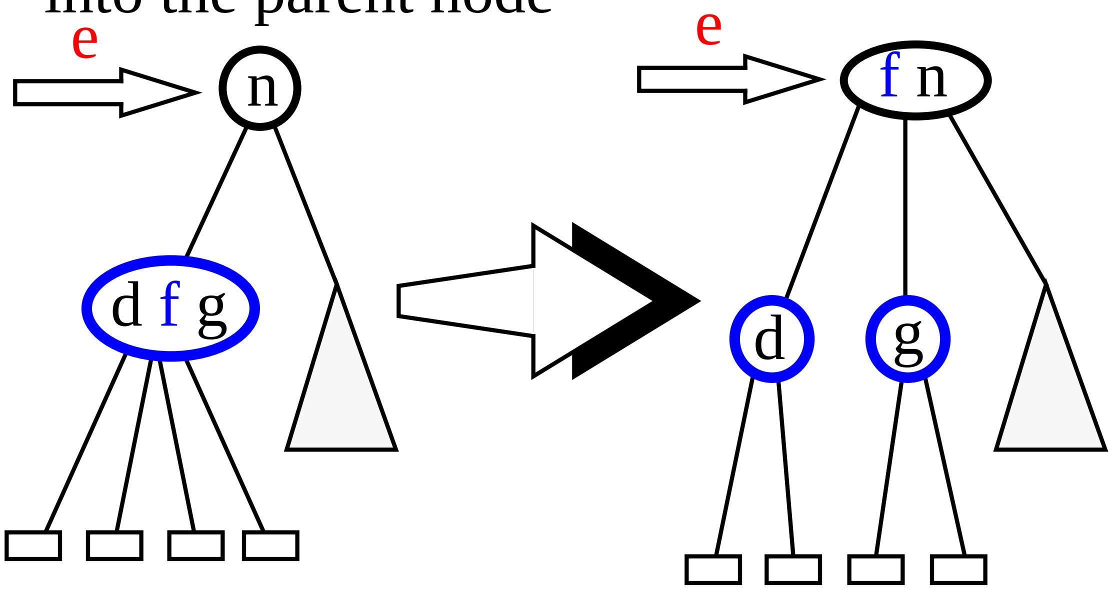
  - ***3-node*** becomes ***4-node***

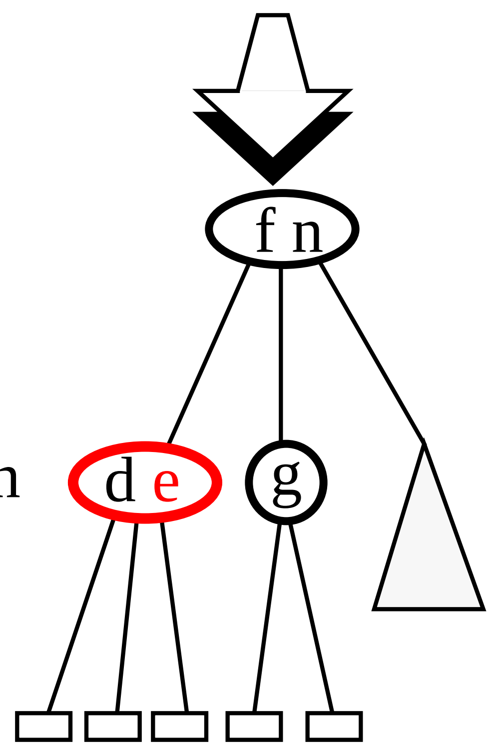


- What about a ***4-node***?
  - We can't insert another key!

Top Down Insertion

- In our way down the tree, whenever we reach a ***4-node***, we break it up into two ***2-nodes***, and move the middle element up into the parent node

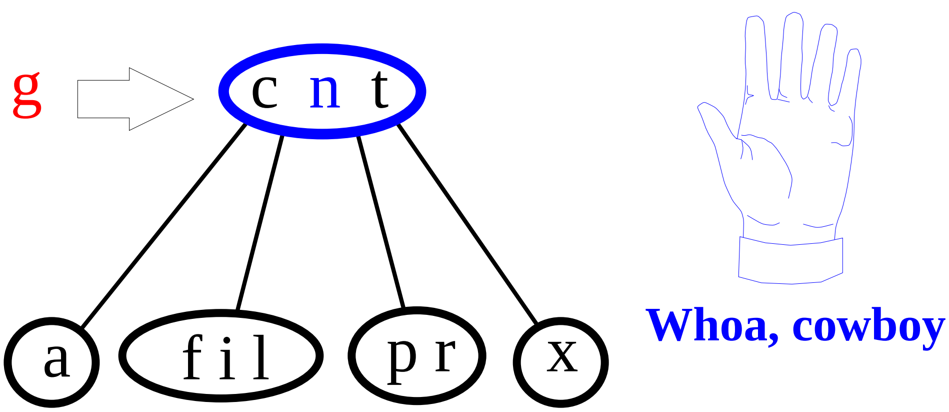


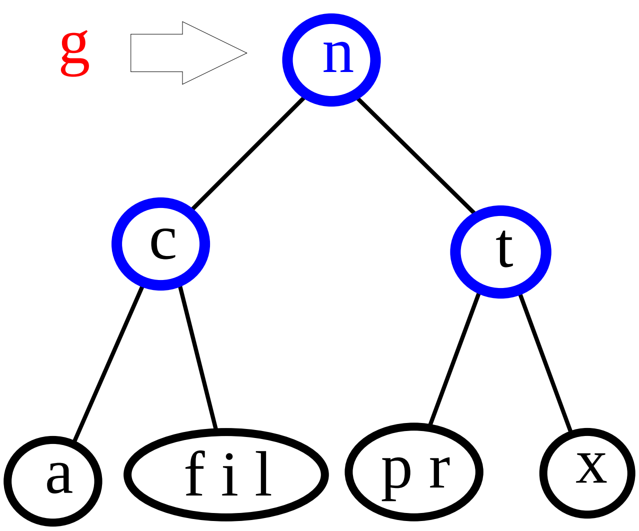


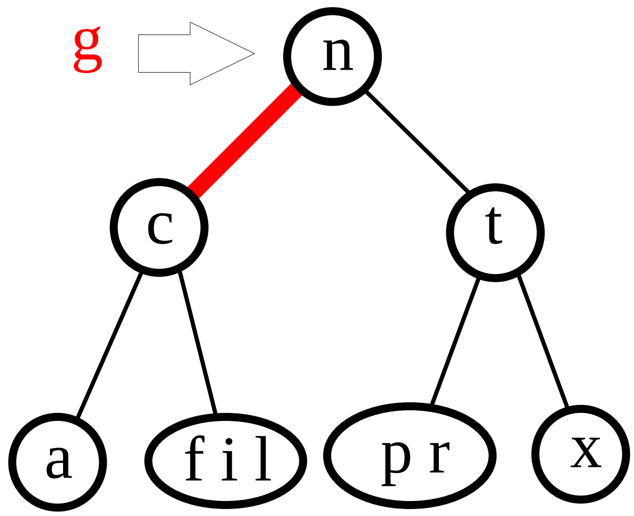
- Now we can perform the insertion using one of the previous two cases
- Since we follow this method from the root down to the leaf, it is called ***top down insertion***

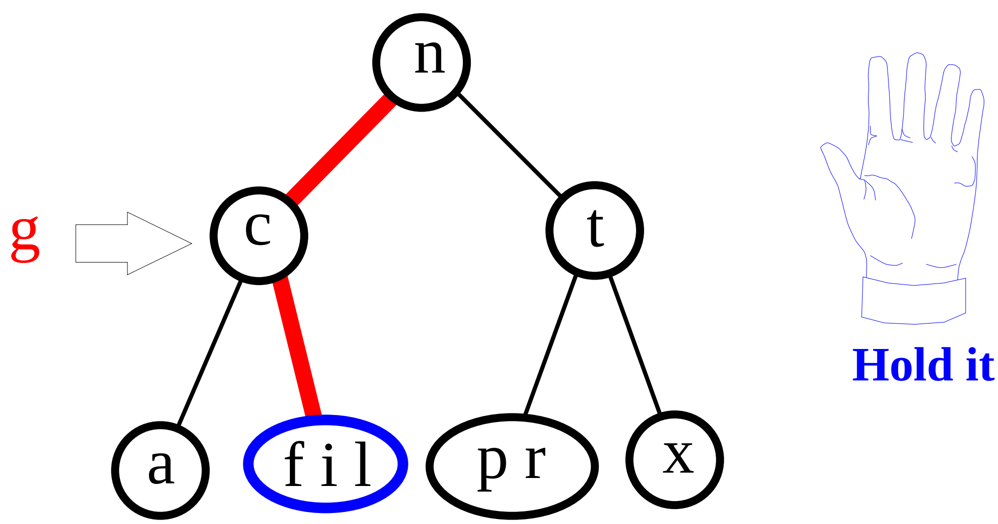
Splitting the Tree

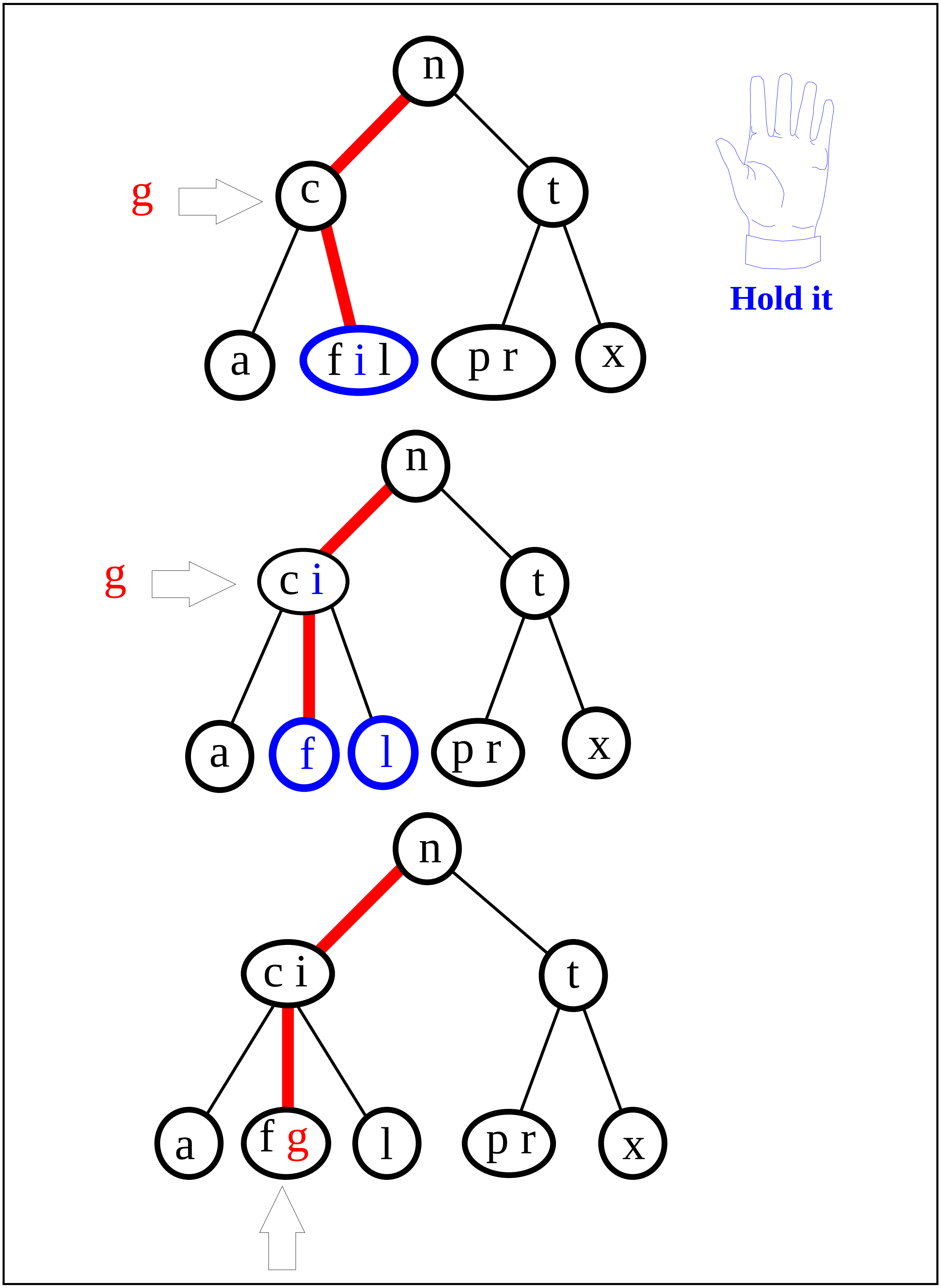
As we travel down the tree, if we encounter any ***4-node*** we will break it up into ***2-nodes***. This guarantees that we will never have the problem of inserting the middle element of a former ***4-node*** into its parent ***4-node***.






g
n
c
t
a
f i l
p r
x


g
n
c
t
a
f i l
p r
x
Hold it


n
c
t
g
a
f i l
p r
x
Hold it
n
c i
t
g
a
f
l
p r
x
n
c i
t
a
f g
l
p r
x

Time Complexity of Insertion in 2-3-4 Trees

Time complexity:

- A search visits O(log N) nodes
- An insertion requires O(log N) node splits
- Each node split takes constant time
- Hence, operations ***Search*** and ***Insert*** each take time O(log N)

Notes:

- Instead of doing splits top-down, we can perform them bottom-up starting at the insertion node, and only when needed. This is called ***bottom-up*** insertion.
- A deletion can be performed by ***fusing*** nodes (inverse of splitting), and takes O(log N) time