Full

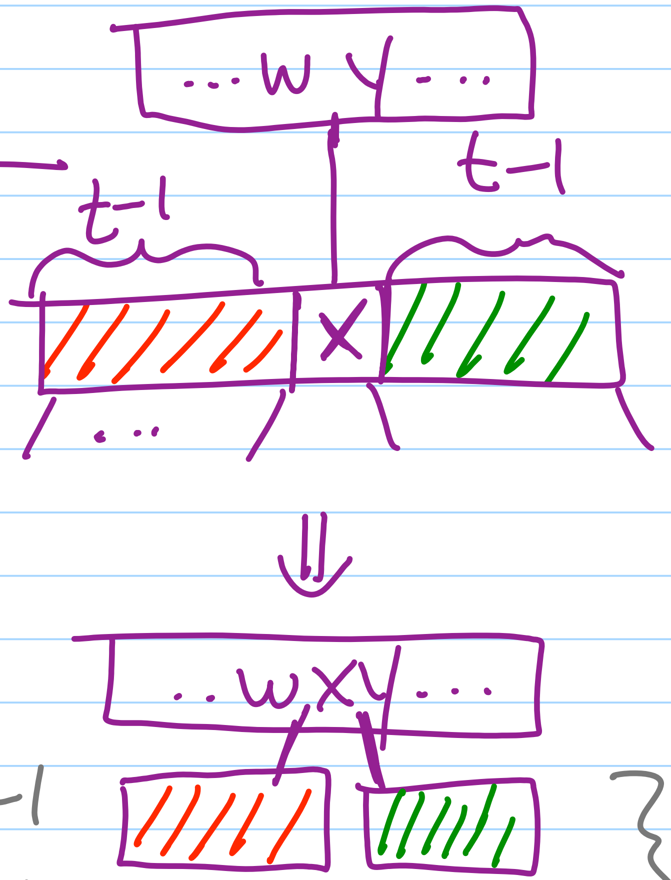
a → ab → abc

$t=2$

1, 2 or 3 tags

2, 3, or 4 children

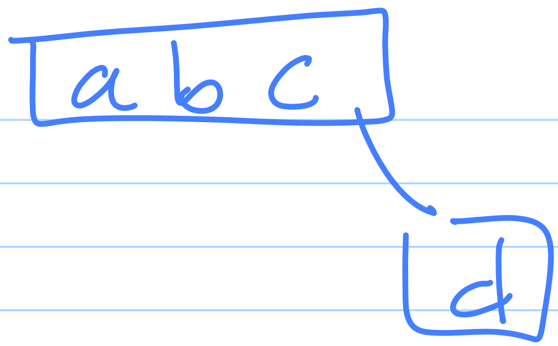
Split



Full node
$2t-1$ tags
$2t$ children

$t-1$ tags
$+t$ children

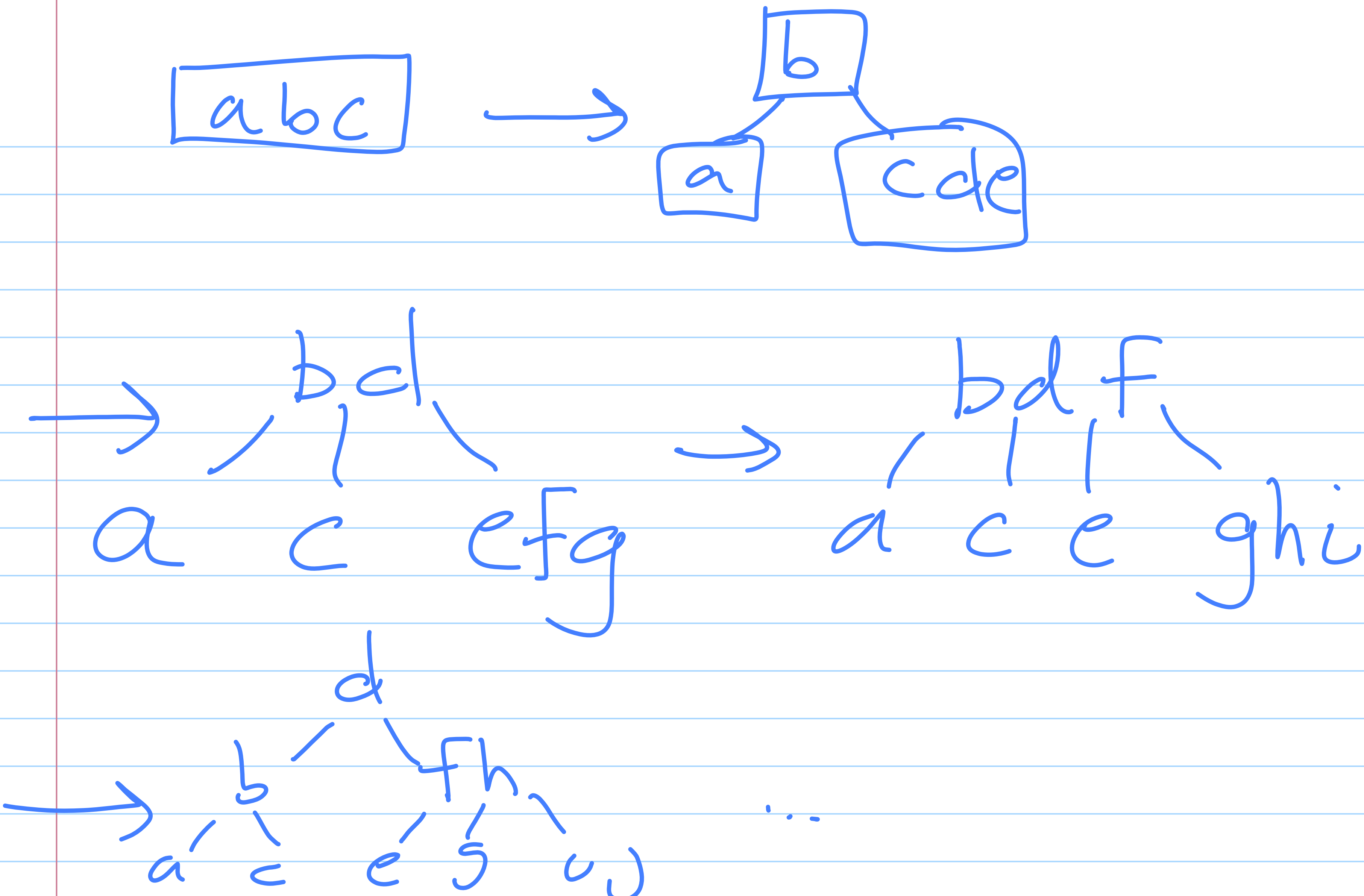
min size allowed!



? not balanced
can't do this!

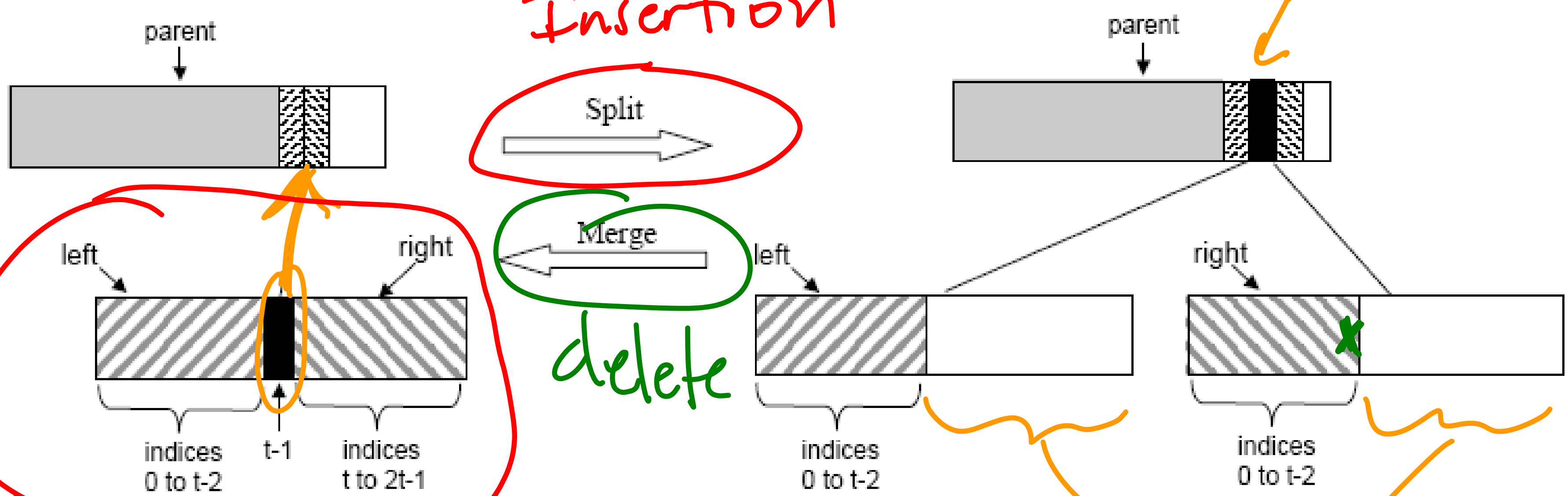
Insert:

Follow path to the leaf where the new item will be inserted, split any full node encountered on the way.


abc
b
a
cde
bd
a
c
efg
bdf
a
c
e
ghi
d
b
fh
a
c
e
g
i
j
...

Split (+ Merge)

Insertion



delete

2t children

2t-1 elements

full & we want to split

room for growth

Top-Down Insertion

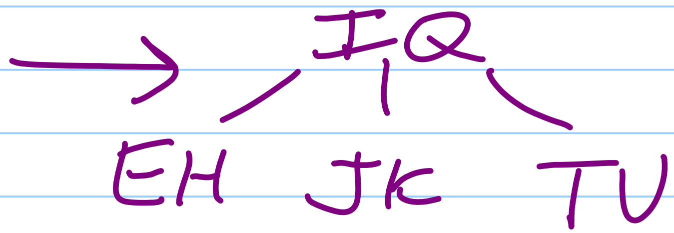
Follow path to leaf where you'd insert (with natural extension of binary search tree insertion)

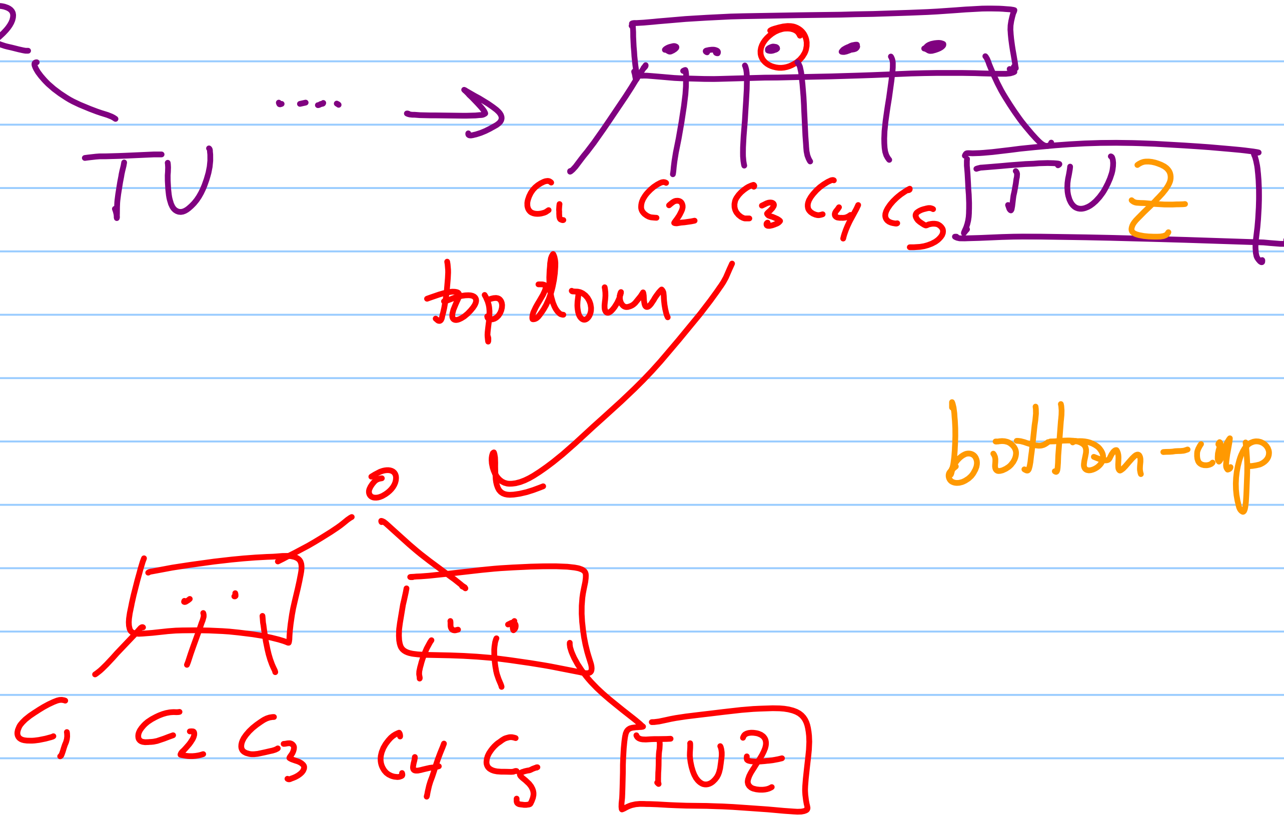
Whenever a full node (2t children) encountered split it & then continue

Goal: minimize possibility of a page fault occuring twice on same page

$t=3$

5 elements is a full node





Insert Z

bottom-up

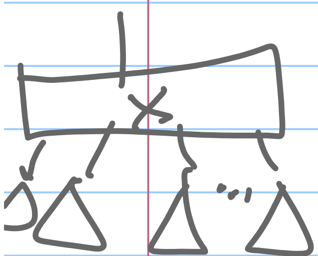
Bottom-Up Insertion

Do standard insertion in leaf if there is room.

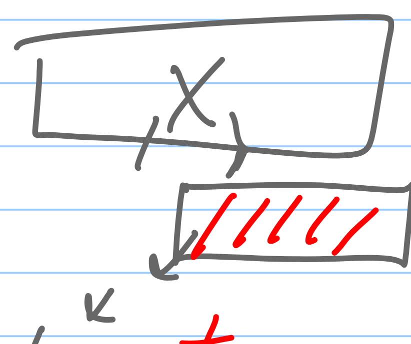
Otherwise split leaf (which could propogate to the root). Stop as soon as parent is not full.

Reduces unnecessary splits.

Overview of B-tree Deletion



Like binary search tree, for removing element in an internal node, then replace x by its successor + remove the successor



hold this in memory until successor is found to replace it

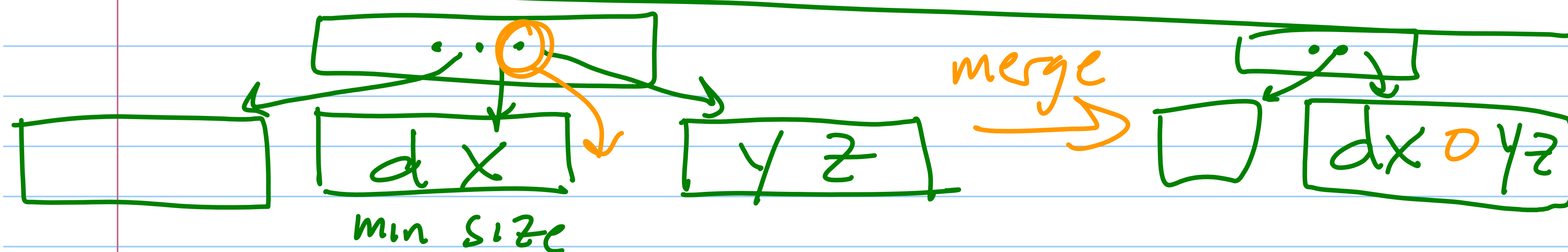
from marked node take leftmost child until reach leaf + succ. leftmost element

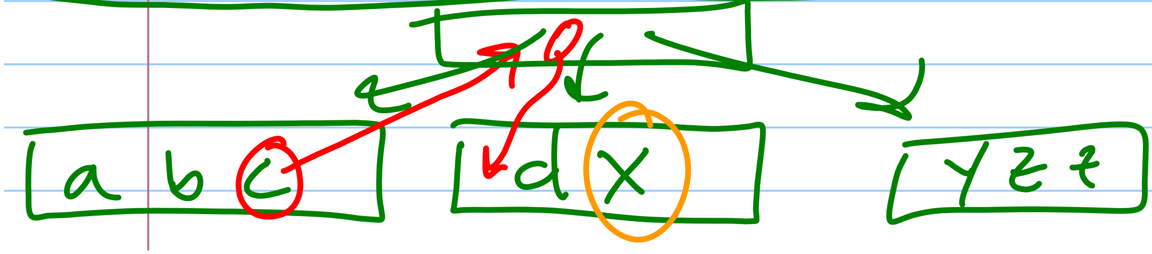
Focus on removing an element in a leaf

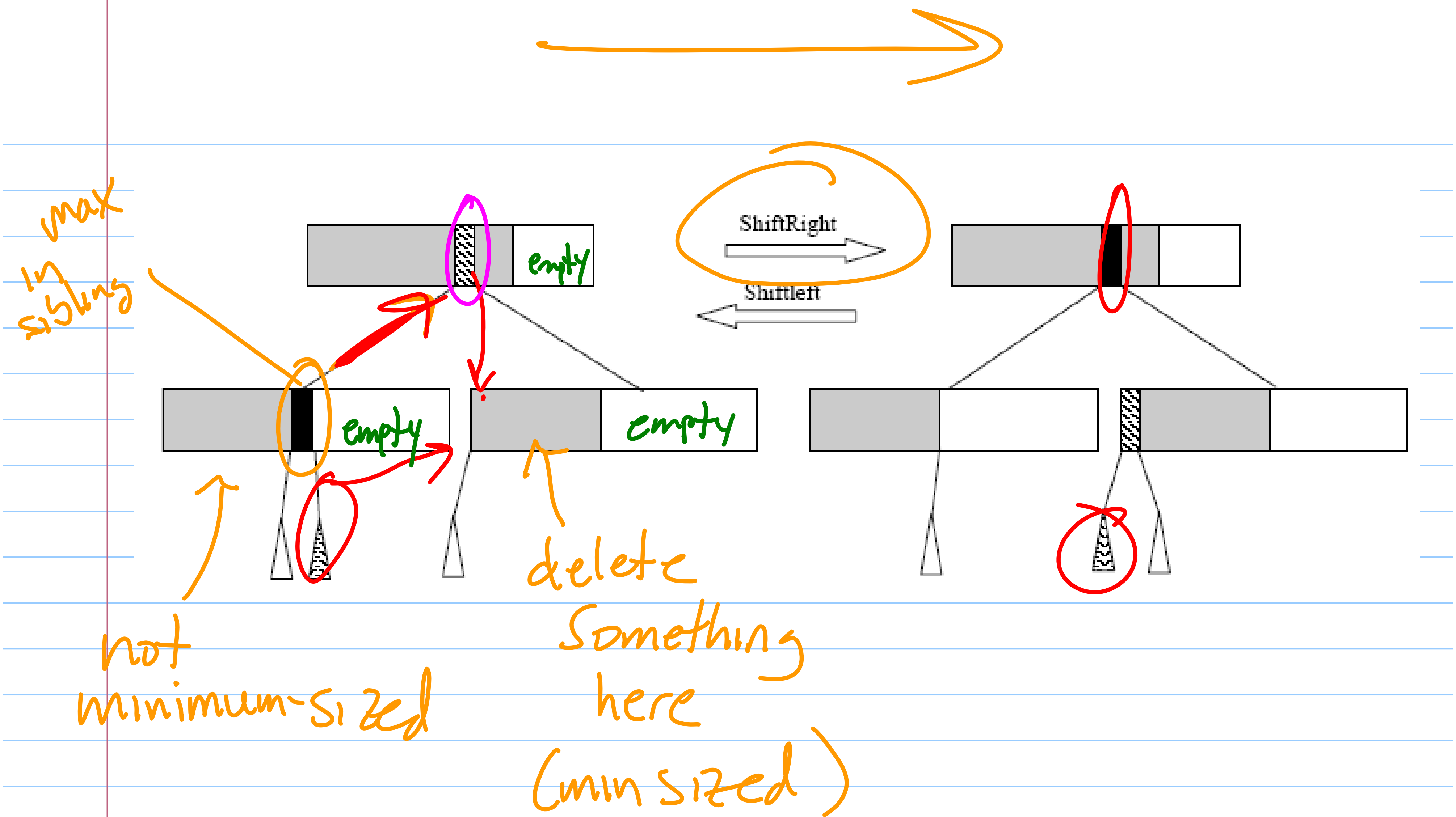
t=3

2-5 elements

leaf $\boxed{c\ d\ m\ \cancel{x}}$ ← not minimum sized







max in sibling
empty
ShiftRight
Shiftleft
empty
empty
not minimum-sized
delete Something here (min sized)

Basic Flow for B-tree deletion
Top-down

On search to leaf (for element to delete or its successor)

If a minimum-sized node is encountered "fix that" by
① merging or ② shift left/right

Ensures that the leaf holding element to remove is not min sized

Pictorial Summary of B-Tree Deletion

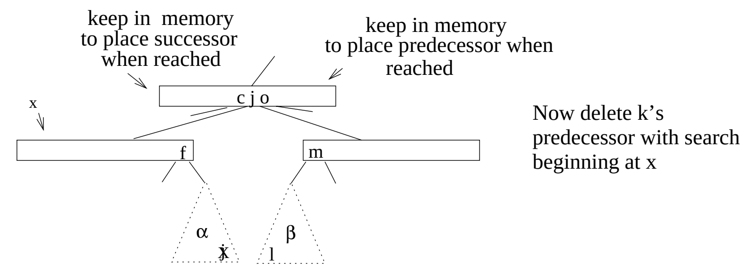
Key idea is to ensure as you move down the tree that for the nodes visited (i.e. those on the path from the root to the node with the key to delete), the number of keys is always at least one more than the minimum allowed (i.e. at least 1 + (t-1) = t) with the exception of the root.

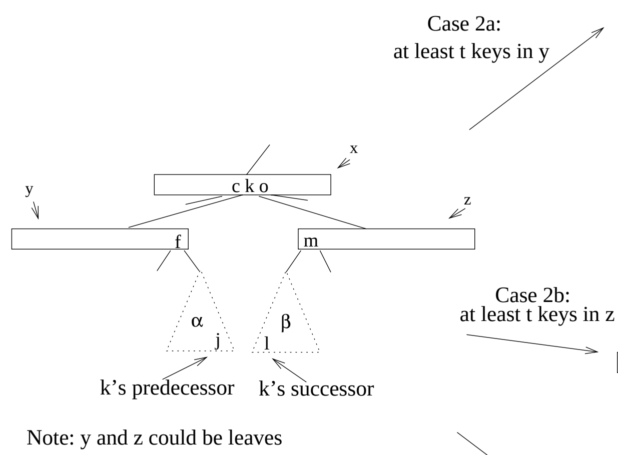
Repeat until Case 1 is reached:

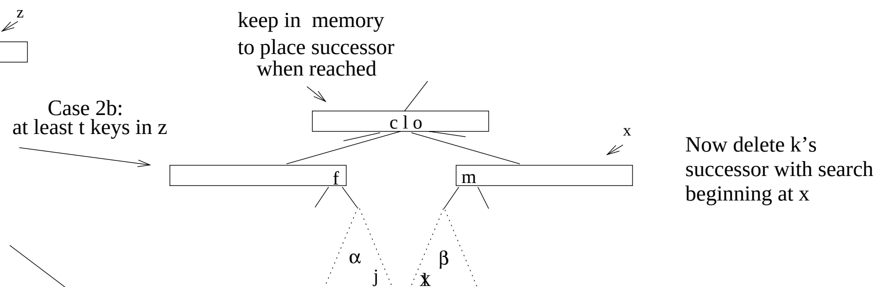
Let x be the current node when searching for the node with key k to delete.

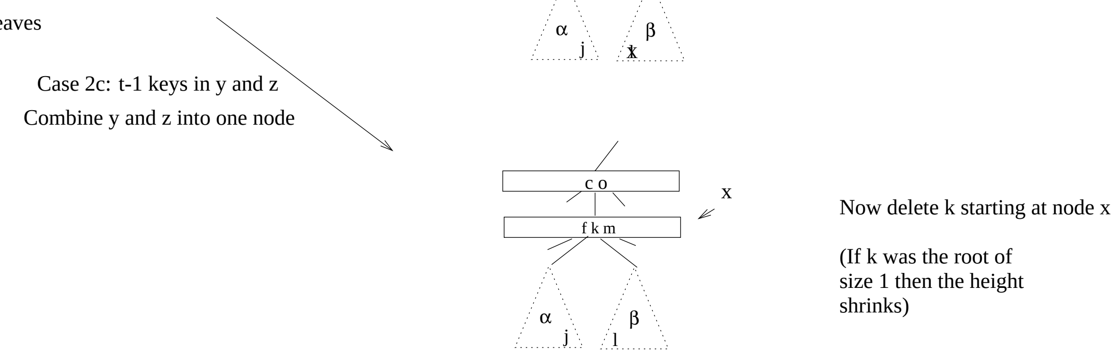
Case 1: x is a leaf (which must then contain k). Then just delete k

Case 2: k is in x where x is not a leaf
y and z are children surronding k









Case 3: Node we want to go to next has only t-1 keys
z is the node you want to go to next in the search, and y is a neighboring sibling
(y could be right of z, and y and z could both be leaves)