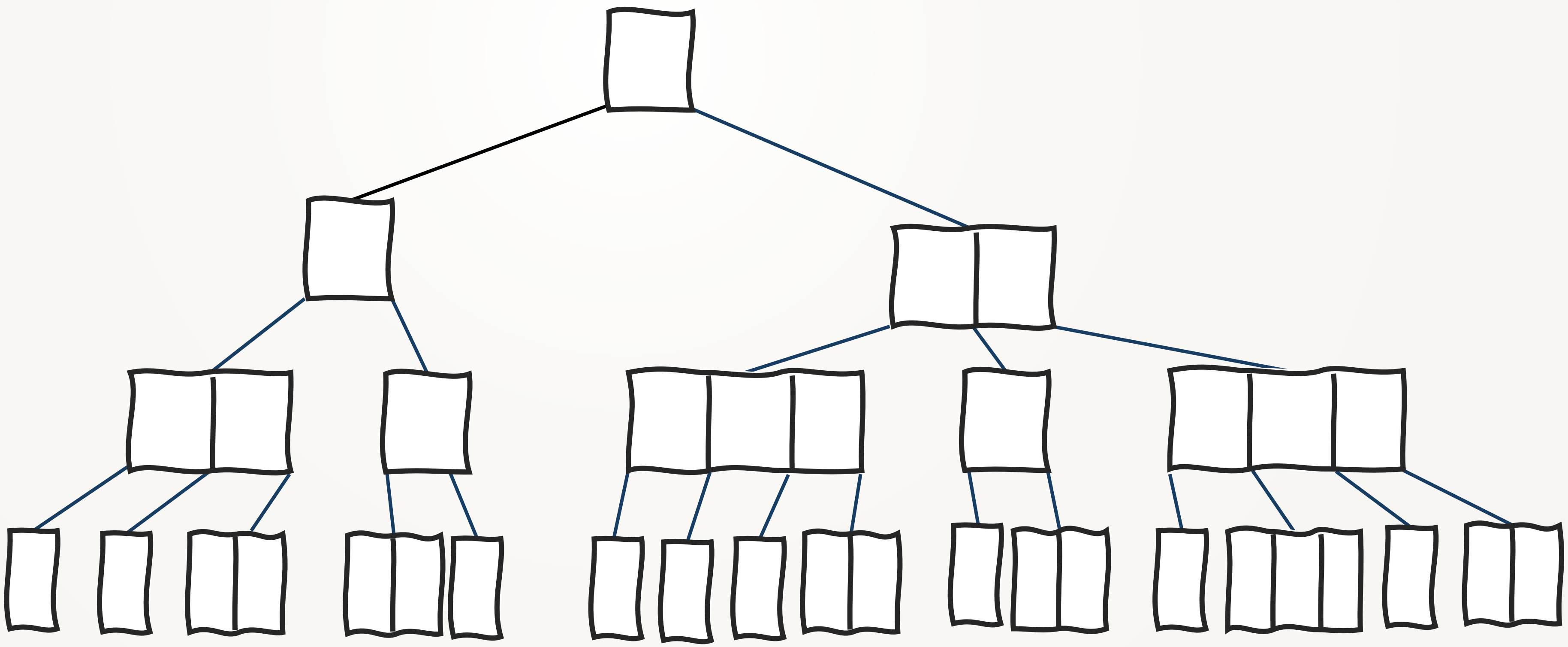
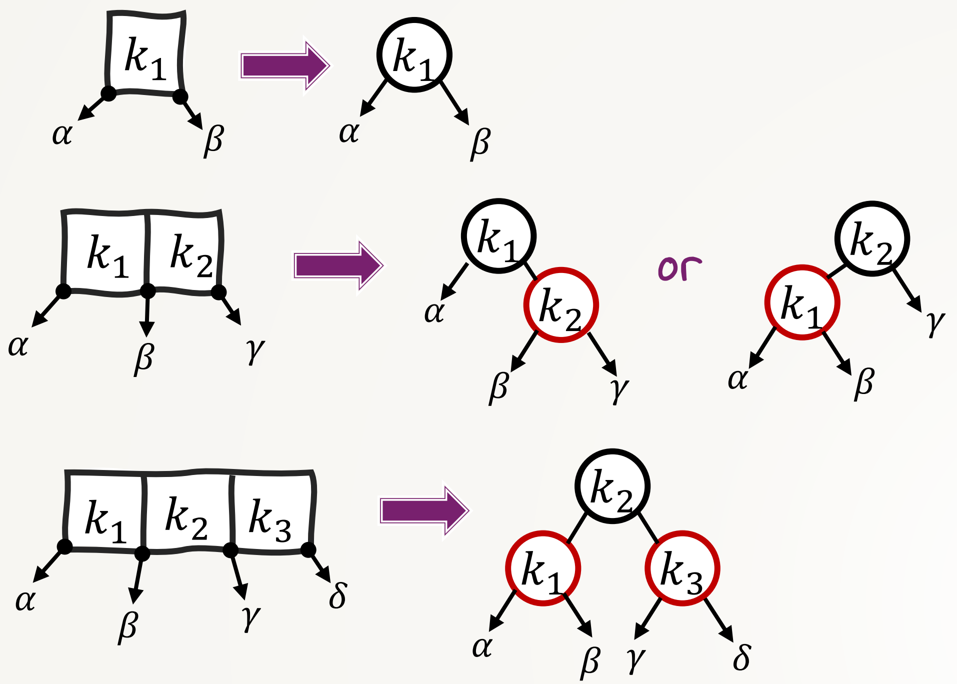
We've designed a beautiful search-tree data structure that keeps itself roughly balanced. Now, let's translate this design back into a simple binary search tree.

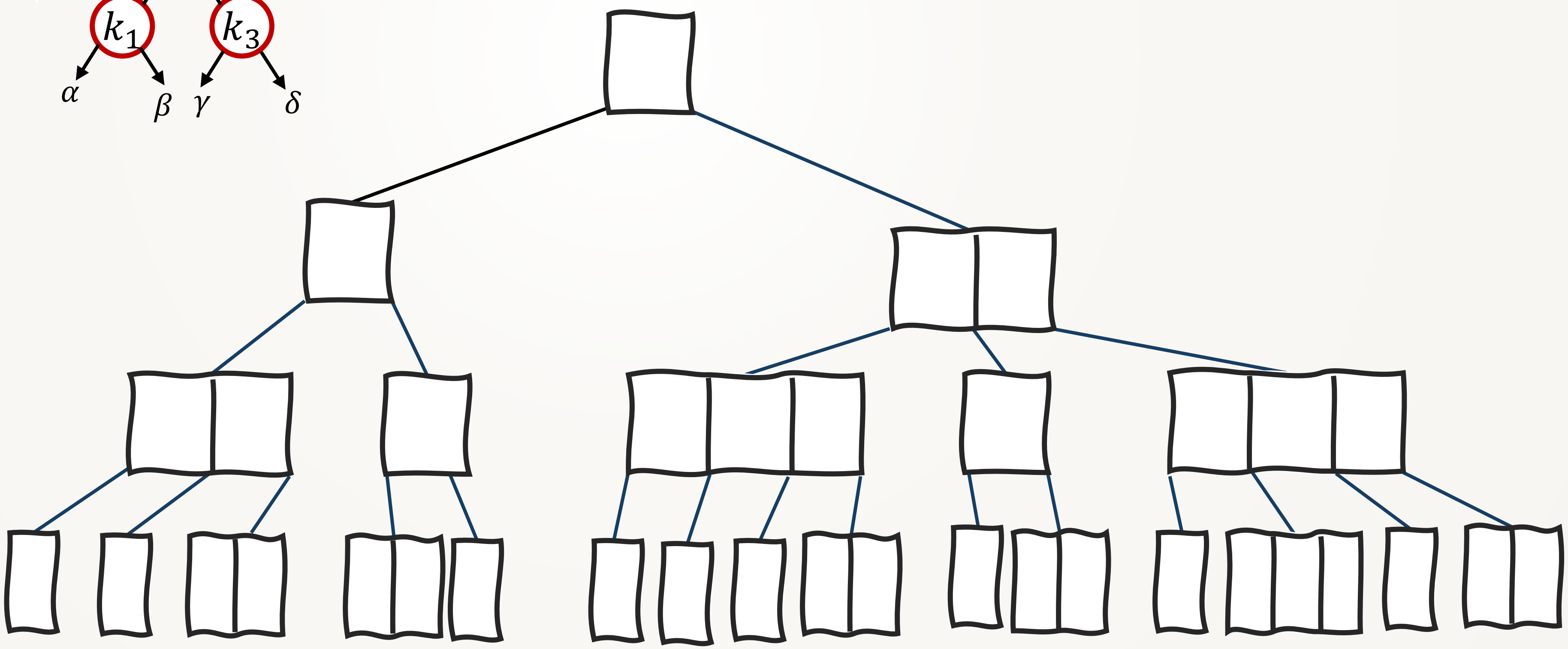
Why?

- It's easier to code a BST than a B-tree
- It clarifies the logic we're applying inside each node
- It may give us ideas for other self-balancing designs

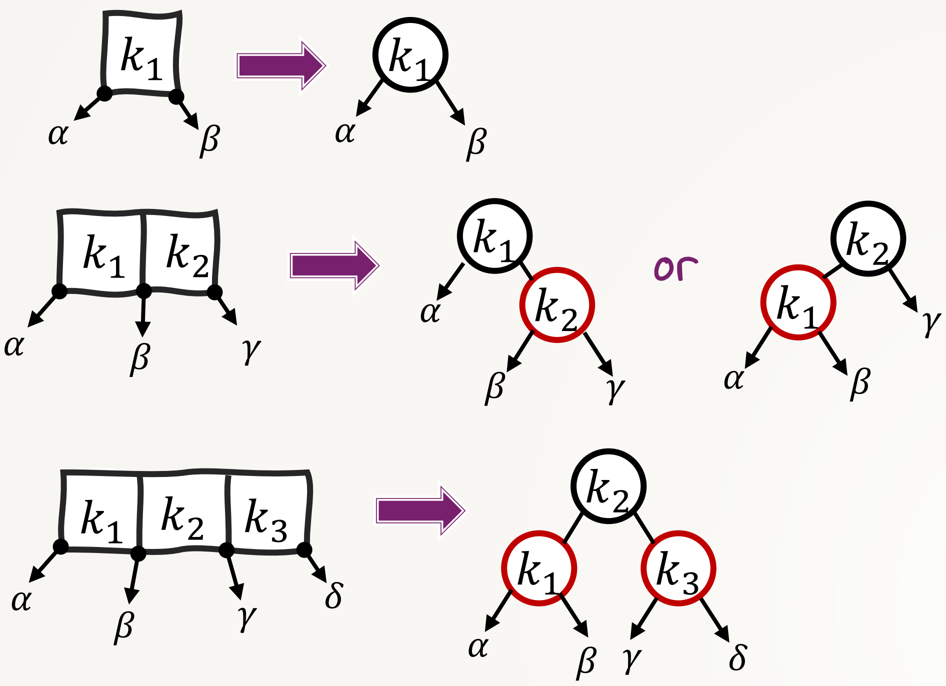


Let's look at 2-3-4 trees, i.e. #keys ∈ {1,2,3} at each node. Let's translate as follows:



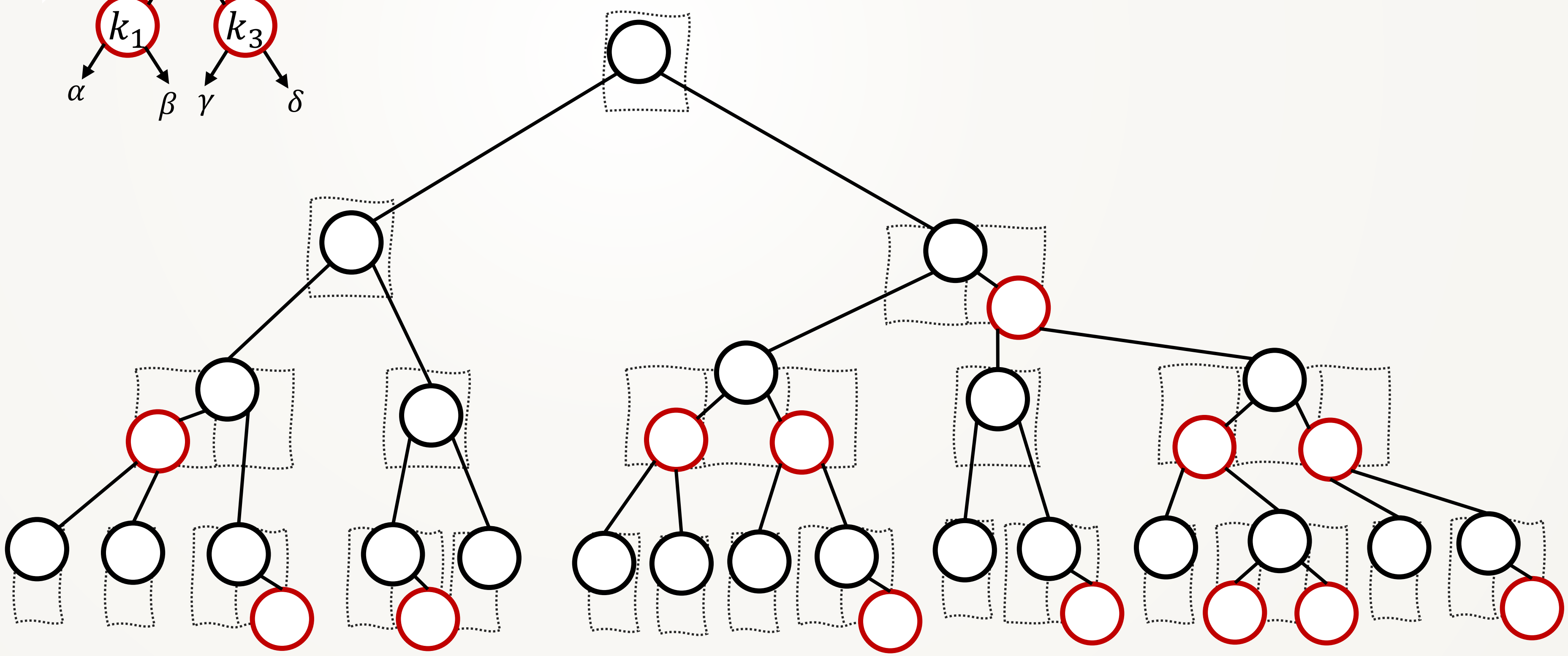


Let’s look at 2-3-4 trees, i.e. #keys ∈ {1,2,3} at each node. Let’s translate as follows:



"top" of a node in 2-3-4 tree ≡ black node in BST

"rest" of a node in 2-3-4 tree ≡ red nodes in BST



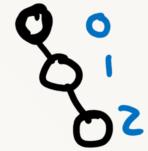
A 2-3-4 tree corresponds to a roughly-balanced BST, called a *red-black tree*.

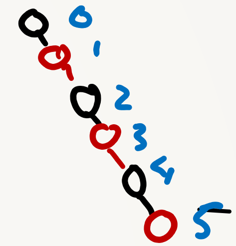
greatest depth ≈ 2 × shallowest depth

QUESTION. If a 2-3-4 tree has height h, what's the minimum and maximum depth of a node in its corresponding red-black tree?

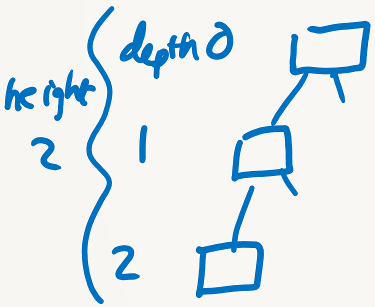
minimum = h

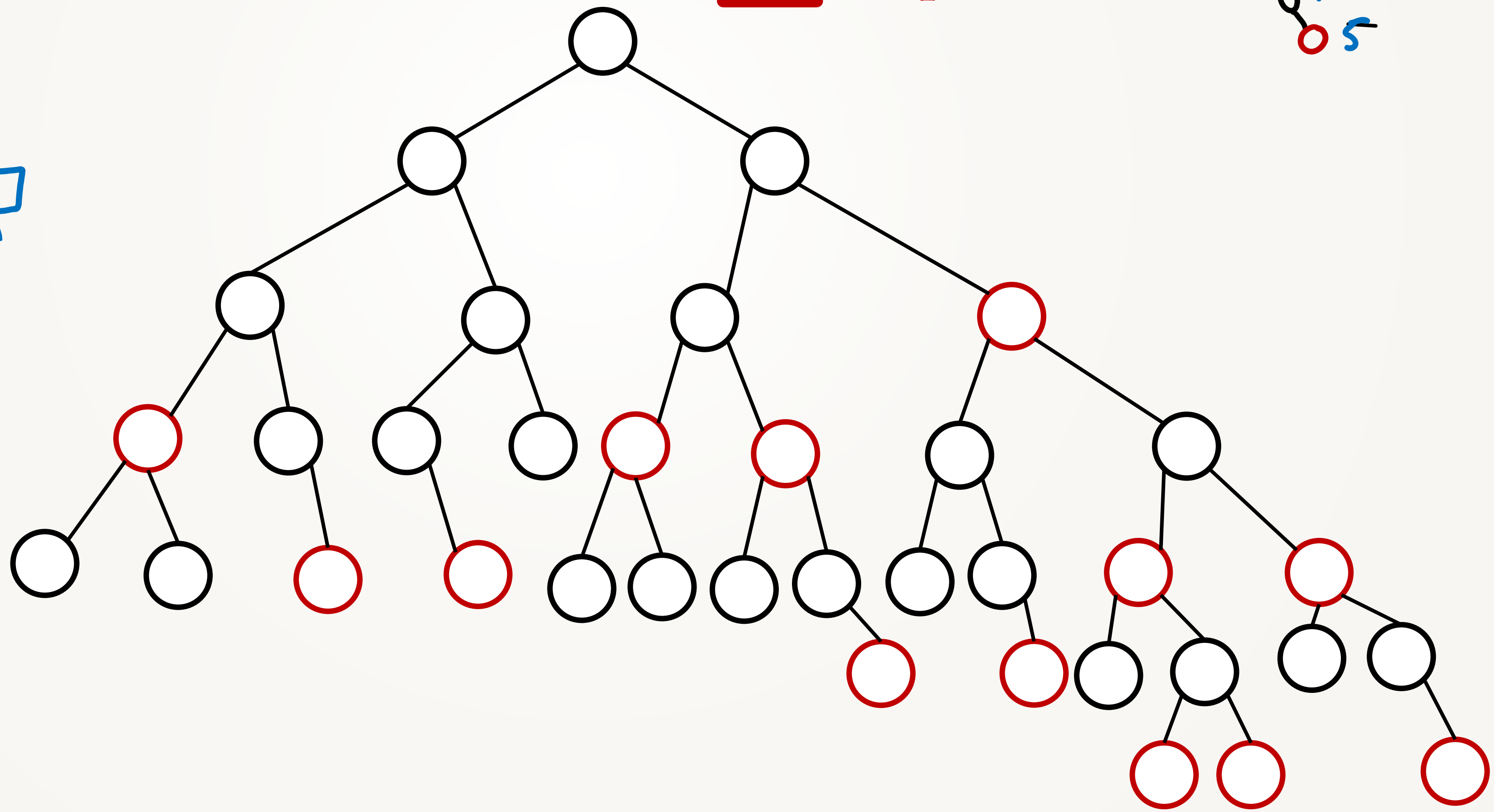
maximum = $2h+1$





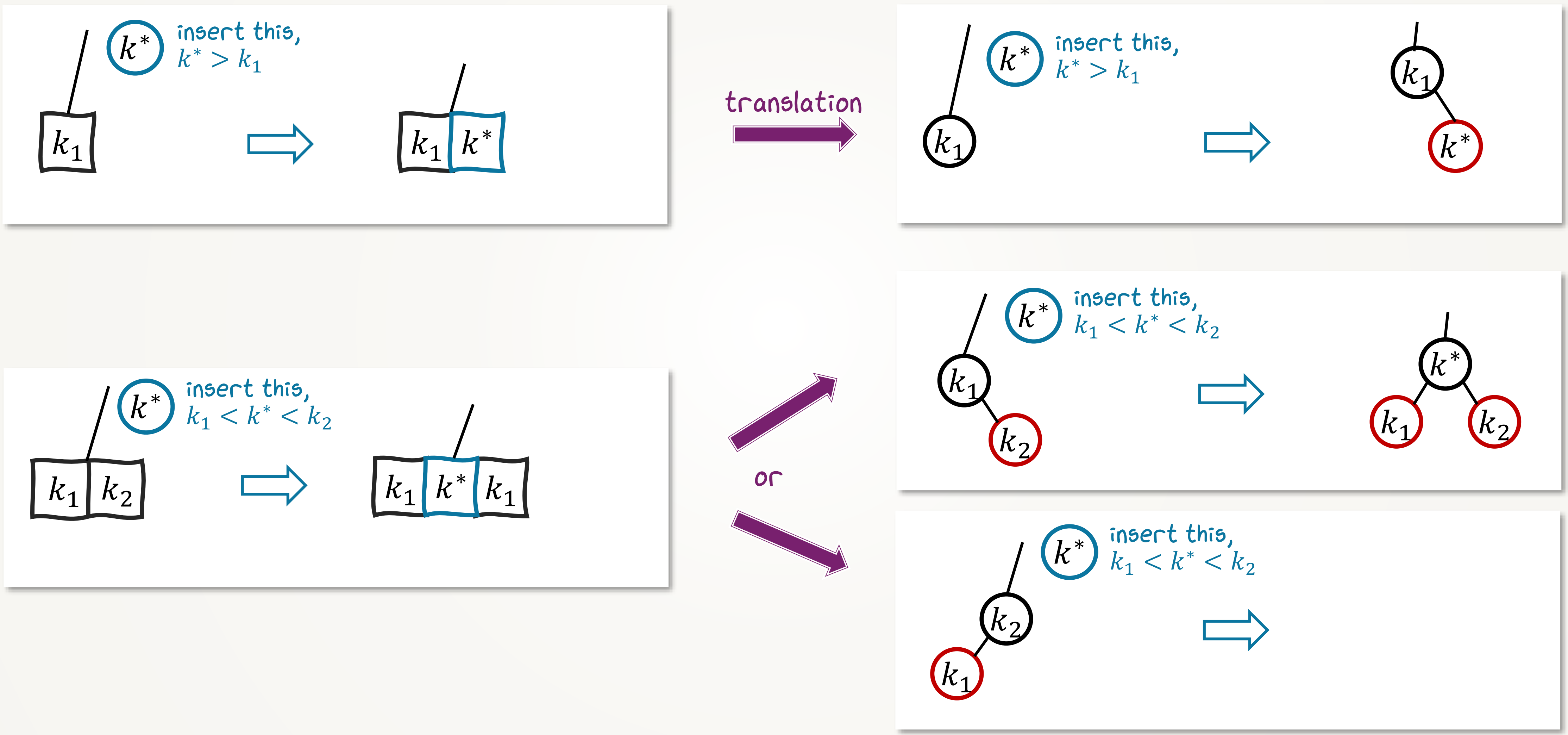
In lectures I said 2h−1





Let's translate 2-3-4 tree operations into red-black operations.

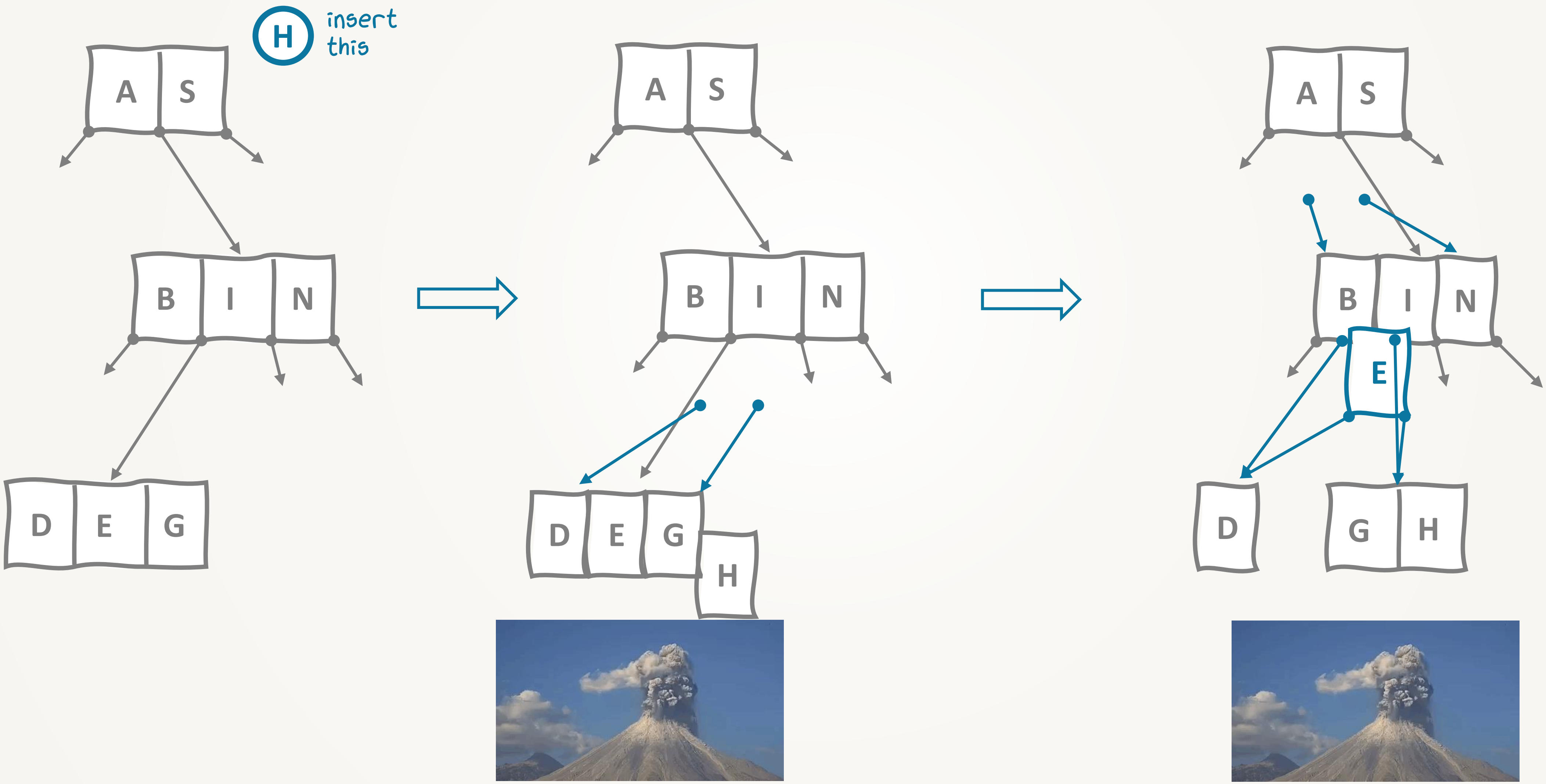
insert(k^*,v^*)



There are lots of cases to work through.
And we still have to deal with the harder "welling up" cases!

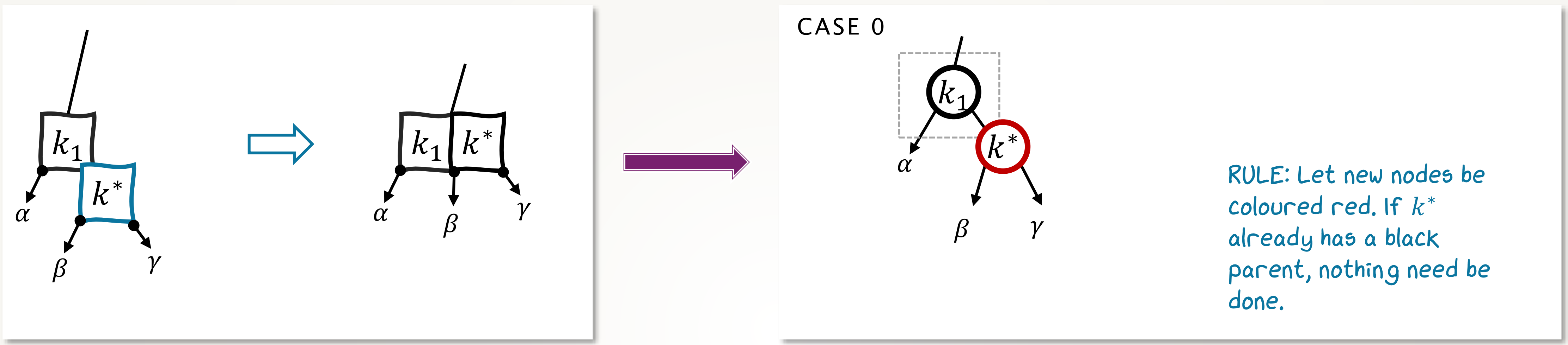
For the “welling up” cases, let’s insert the new key at the bottom, and let the impact “well up” towards the root.

The general case: a node receives a key from below, and perhaps sends one of its keys up.

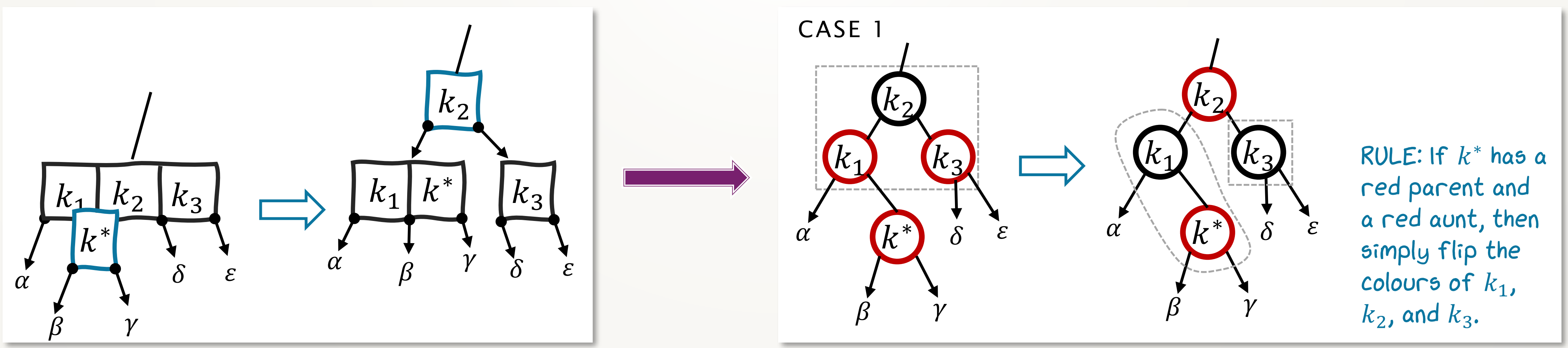


For the "welling up" cases, let's insert the new key at the bottom, and let the impact "well up" towards the root.

The general case: a node receives a key from below, and perhaps sends one of its keys up.

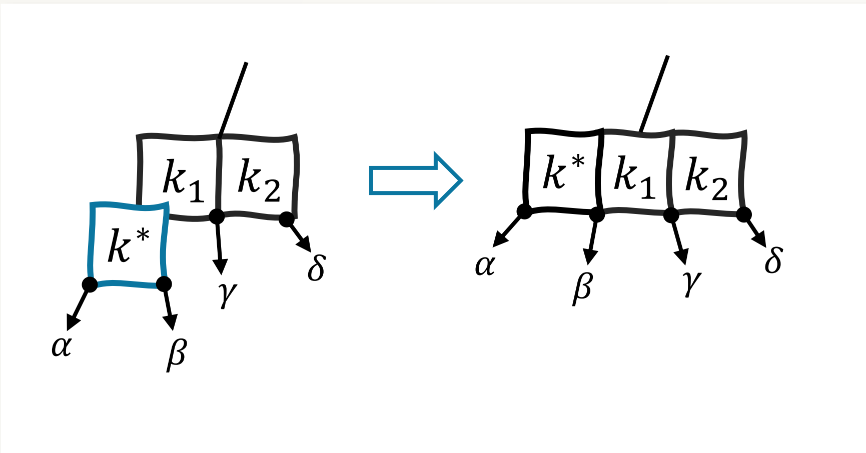


α, β, and γ are all either empty,
or subtrees with a black node on top.

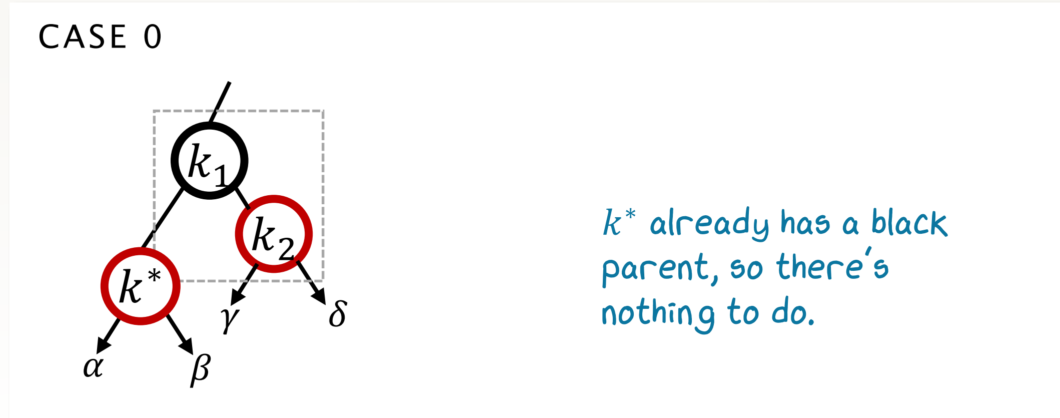


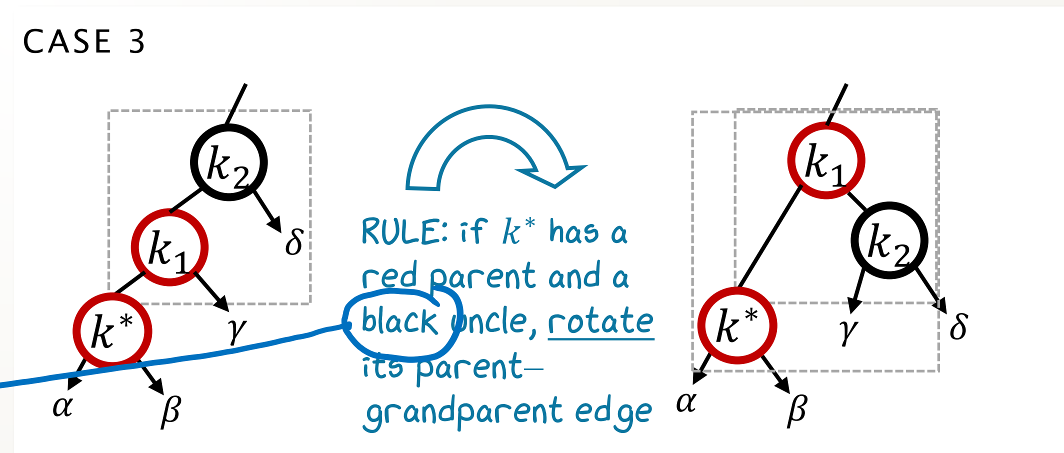
For the "welling up" cases, let's insert the new key at the bottom, and let the impact "well up" towards the root.

The general case: a node receives a key from below, and perhaps sends one of its keys up.



or





$\alpha, \beta, \gamma, \delta$ refer to subtrees (possibly empty) in the 2-3-4 tree.

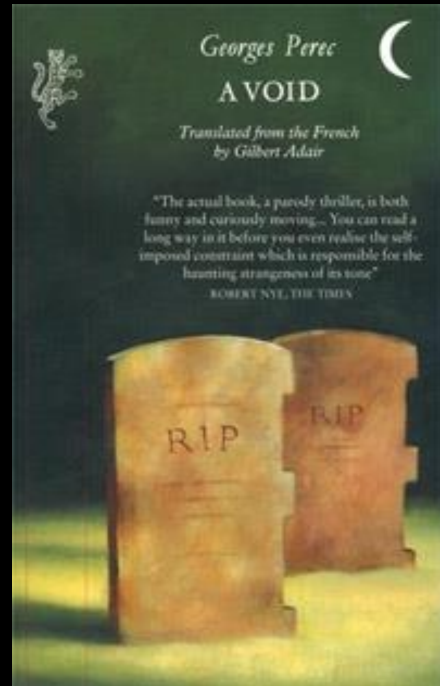
In BST language, $\alpha, \beta, \gamma, \delta$ refer to subtrees (possibly empty) WITH A BLACK ROOT (if non-empty).

δ is a subtree with a black root (or empty).

This "rotation" trick is behind a host of other self-balancing tree data structures.

Georges Perec

Born	7 March 1936 Paris, France
Died	3 March 1982 (aged 45) Ivry-sur-Seine, France
Occupation	Novelist, filmmaker, essayist
Language	French
Spouse	Paulette Petras



A Void, translated from the original French ***La Disparition***, is a 300-page French novel by Georges Perec, entirely without using the letter e, following Oulipo constraints.

Oulipo, short for *Ouvroir de littérature potentielle*, is a loose gathering of French-speaking writers and mathematicians who seek to create works using constrained writing techniques.

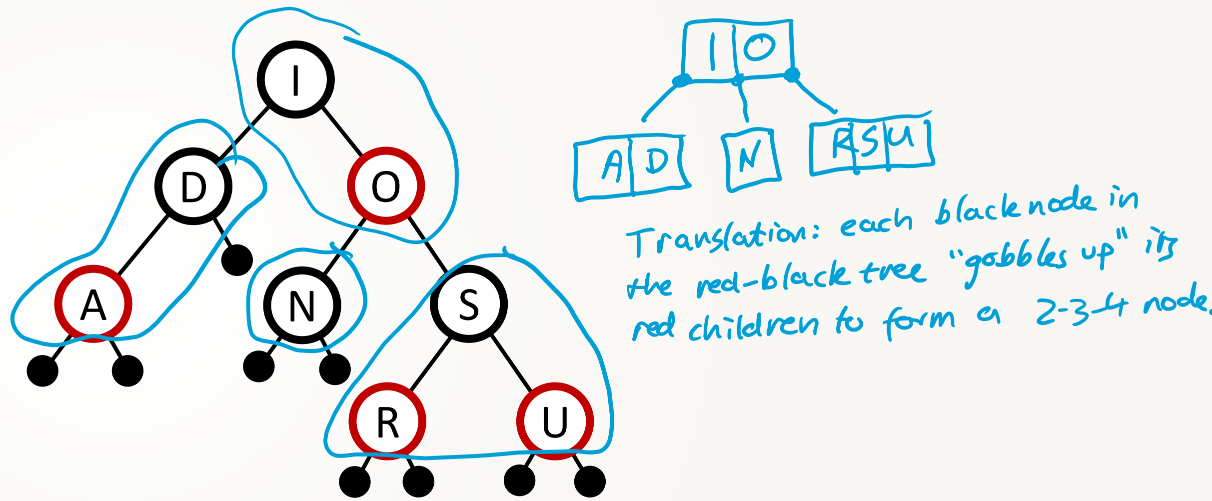
Can we characterize valid red-black trees, without reference to 2-3-4 trees?

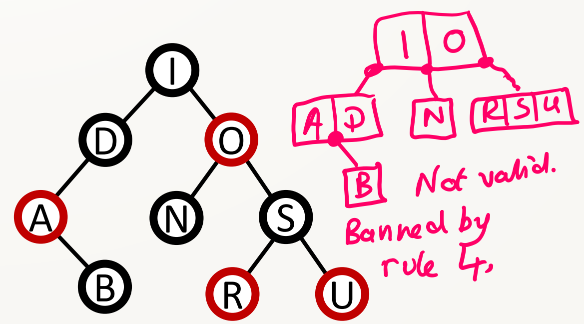
Why?

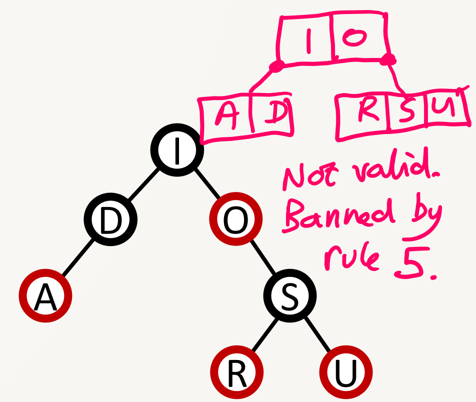
- To debug our insert/delete code, it's useful to have a formal test "is this a valid red-black tree"?
- A perfectly satisfactory test is "does it translate to a valid 2-3-4 tree?"
- Let's be Oulipo coders, and come up with a characterization that doesn't mention 2-3-4 trees.

A red-black tree is a binary search tree that satisfies the following properties:

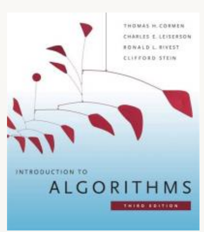
1. Every node is either black or red
2. The root is black
3. A red node's children are black
4. All paths from the root to the bottom of the tree have the same number of black nodes
5. All nodes have 2 children, apart from the leaves, which are keyless childless black nodes







We could then analyse all our cases for insert / delete and prove that they maintain these properties. (If we had no taste.)

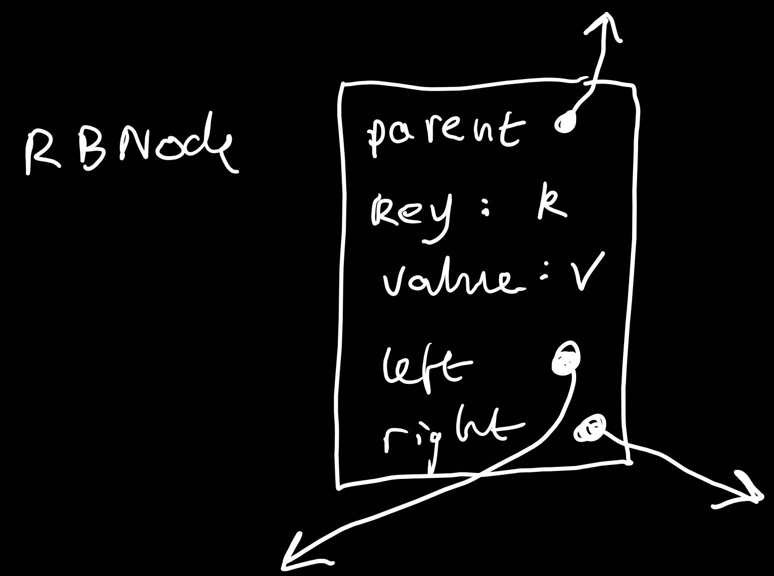


A red-black tree is a binary search tree that satisfies the following properties:

1. Every node is either black or red
2. The root is black
3. A red node's children are black
4. All paths from the root to the bottom of the tree have the same number of black nodes
5. All nodes have 2 children, apart from the leaves, which are keyless childless black nodes

Straightforward implementation:

each node in the red-black BST is a record



FINAL LECTURE

A correctness proof for bfs-all

Algorithms assignment grade-chatgpt: Grading ChatGPT's proof

Can ChatGPT be persuaded to give a proper proof of correctness of an algorithm? Here are three attempts, for an algorithm that solves the bfs-all tick:

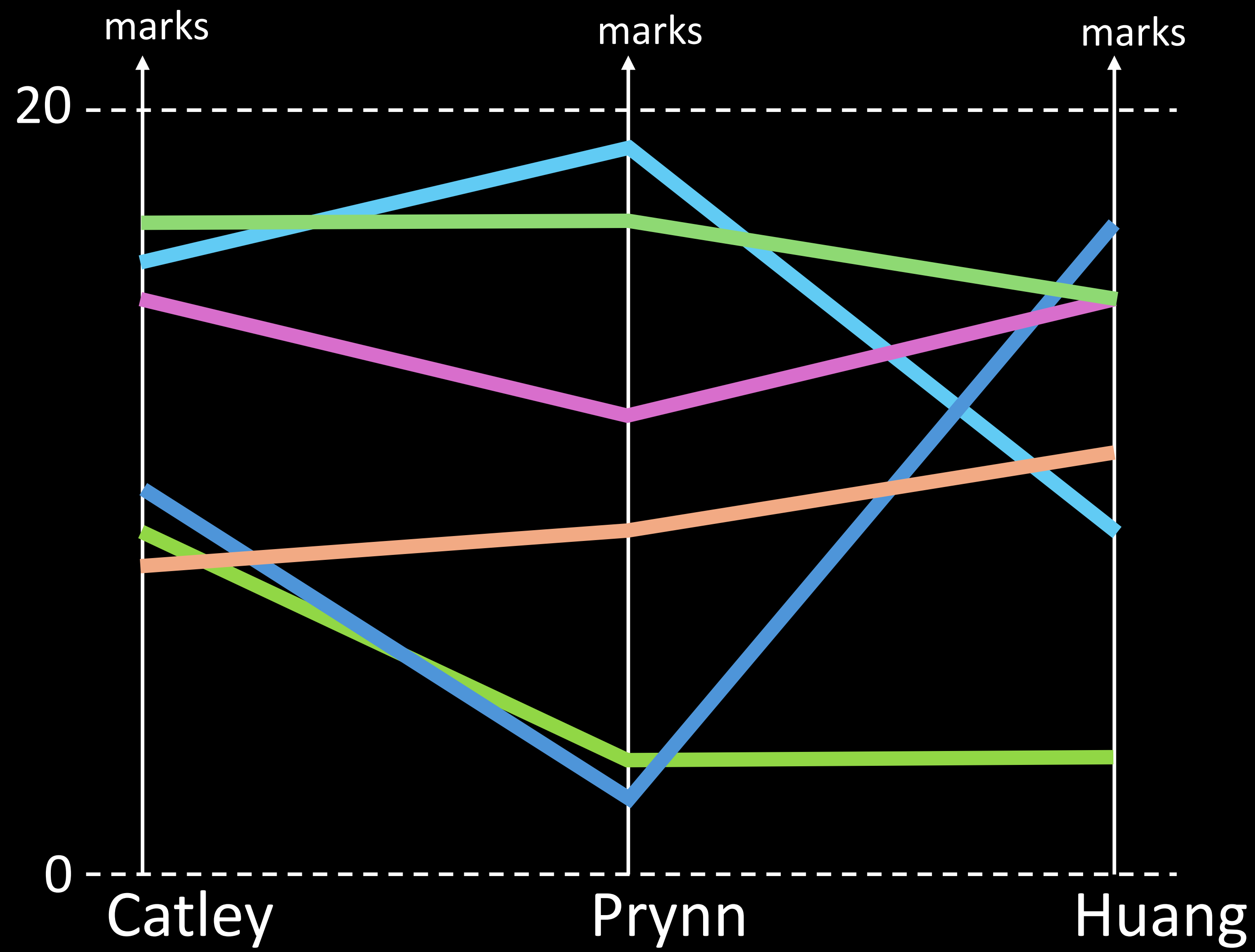
- Catley, Prynn, and Huang.

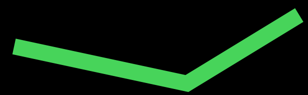
Please mark these attempts, on a scale of 0–20. Your mark should be for the final proof, not for how well it was elicited. **Please submit your grades on Moodle.** I'll pick the most controversially-marked answer and go through it in lectures. Please use the following marking scheme:

mark	meaning
5	Coherent fragments
9	Coherent in parts, but with serious gaps
13	A basically correct argument but with some signs of confusion
17	Essentially correct, but not fully rigorous
19	Nearly all correct, only minor technical holes

How well does ChatGPT generate algorithms? For interest, here are the attempts to get ChatGPT to design the algorithm:

- Catley, Shen, Chen, Prynne, and Huang.





Each coloured line represents a different assessor's marks

TODO: Please review ~~Huang~~ Catley's code and proof before the final lecture.