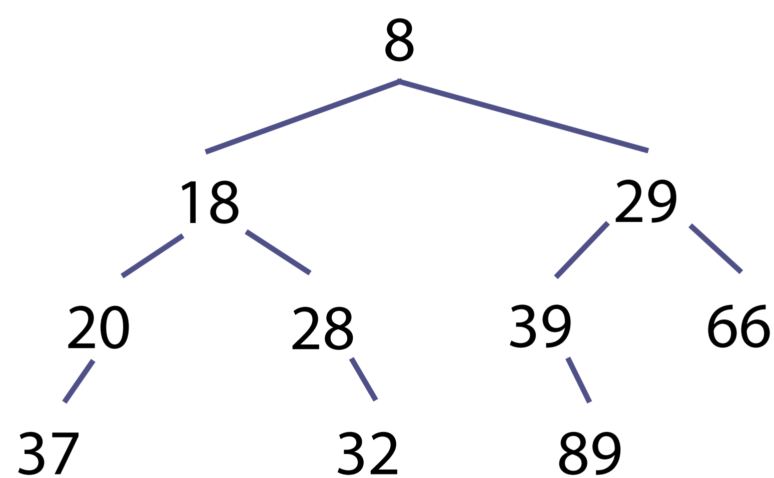
Leftist heaps
(Weiss 6.6)

Heaps with merging

Another useful operation is *merging two heaps into one*

To do this, let's go back to *binary trees with the heap property* (no completeness):



We can implement the other priority queue operations in terms of merging!

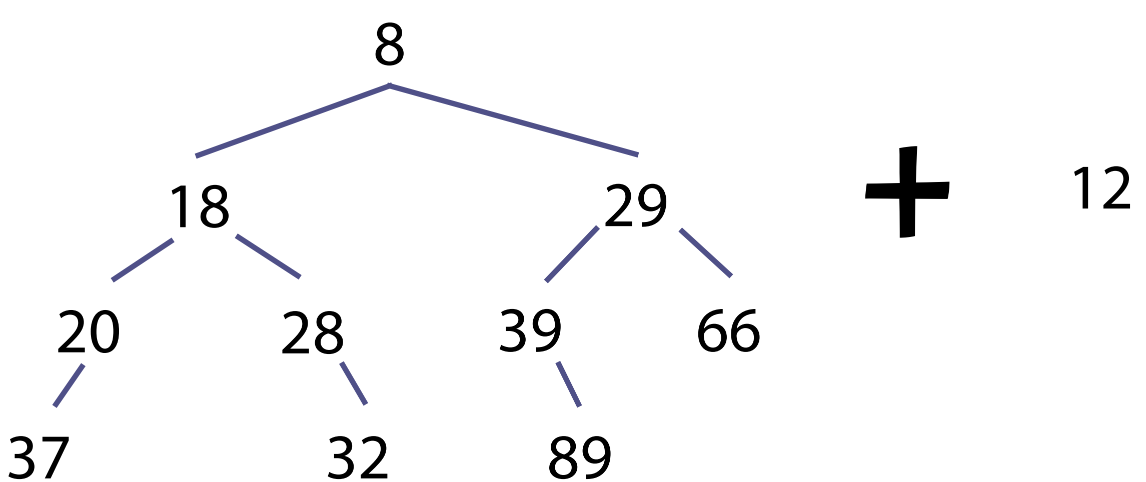
Insertion

To insert a single element:

- build a heap containing just that one element
- merge it into the existing heap!

E.g., inserting 12

A tree with just one node

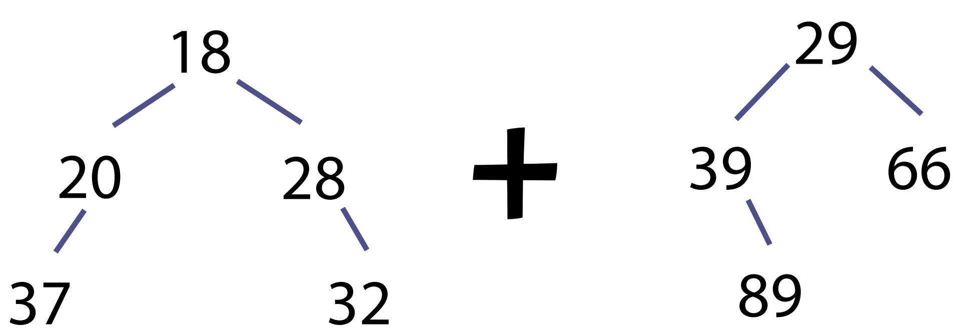


Delete minimum

To delete the minimum element:

- take the left and right branches of the tree
- these contain every element except the smallest
- merge them!

E.g., deleting 8 from the previous heap



Heaps with merging

Using merge, we can efficiently implement:

- insertion
- delete minimum

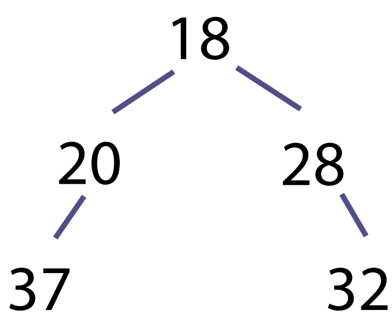
Only question is, how to implement merge?

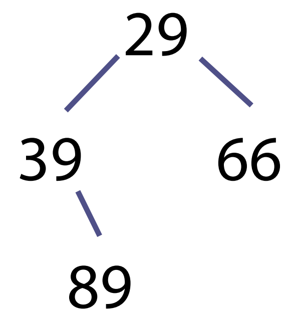
- Should take O(log n) time

We'll start with a bad merge algorithm, and then fix it

Naive merging

How to merge these two heaps?

A = 

B = 

Idea: root of resulting heap must be 18

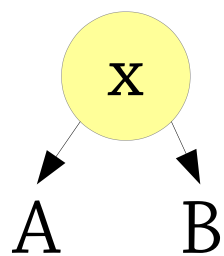
Take heap A, it has the smallest root.
Pick one of its children. Recursively merge B into that child.

Which child should we pick? Let's pick the right child for no particular reason

Naive merging

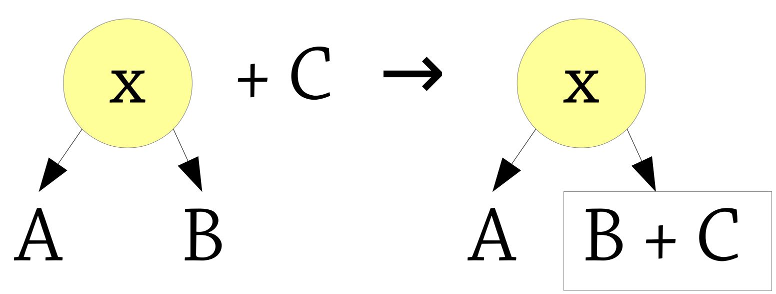
To merge two non-empty heaps:

Pick the heap with the smallest root:



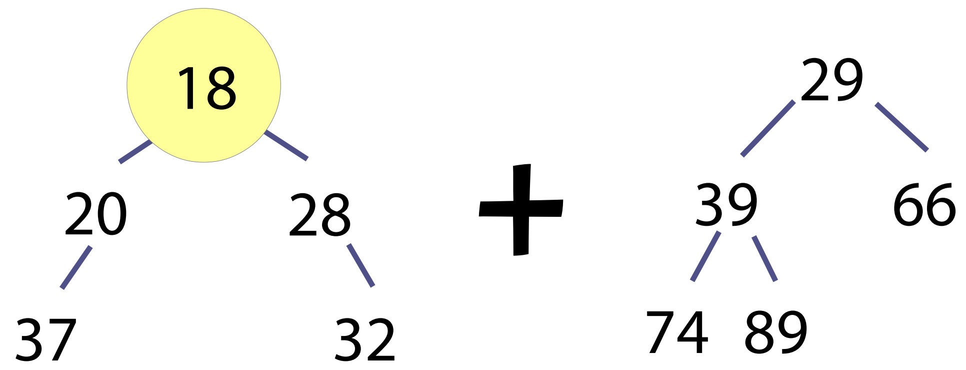
Let C be the other heap

Recursively merge B and C!



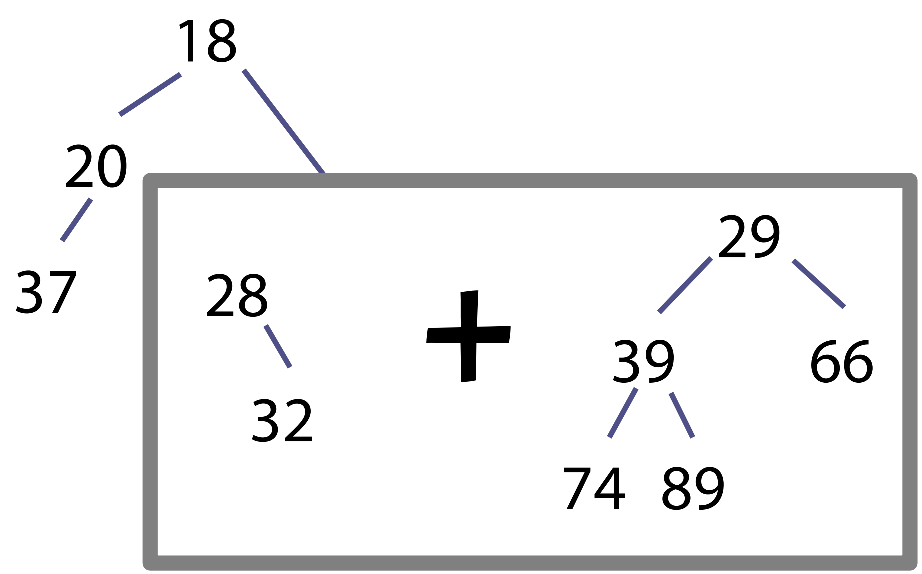
Example

18 < 29 so pick 18 as the root of the merged tree



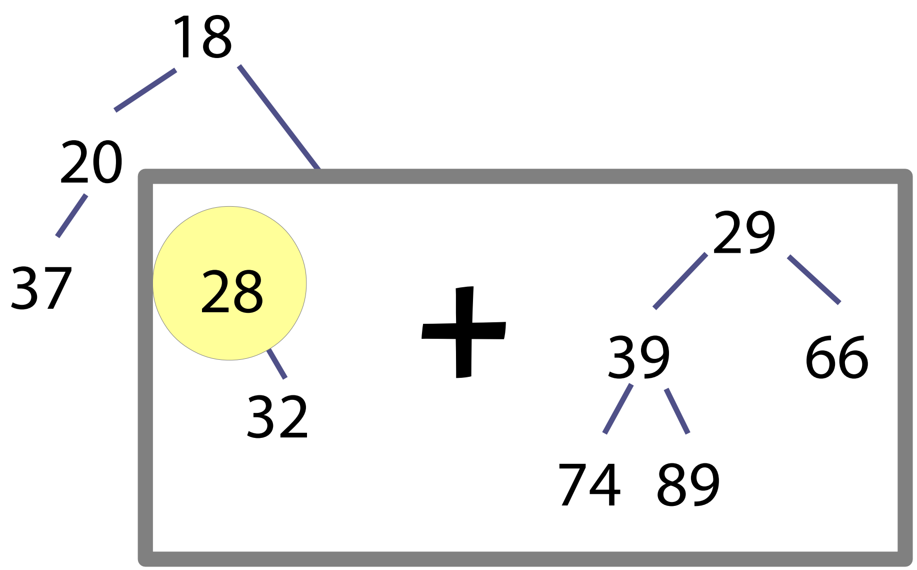
Naive merging

Recursively merge the right branch of 18 and the 29 tree



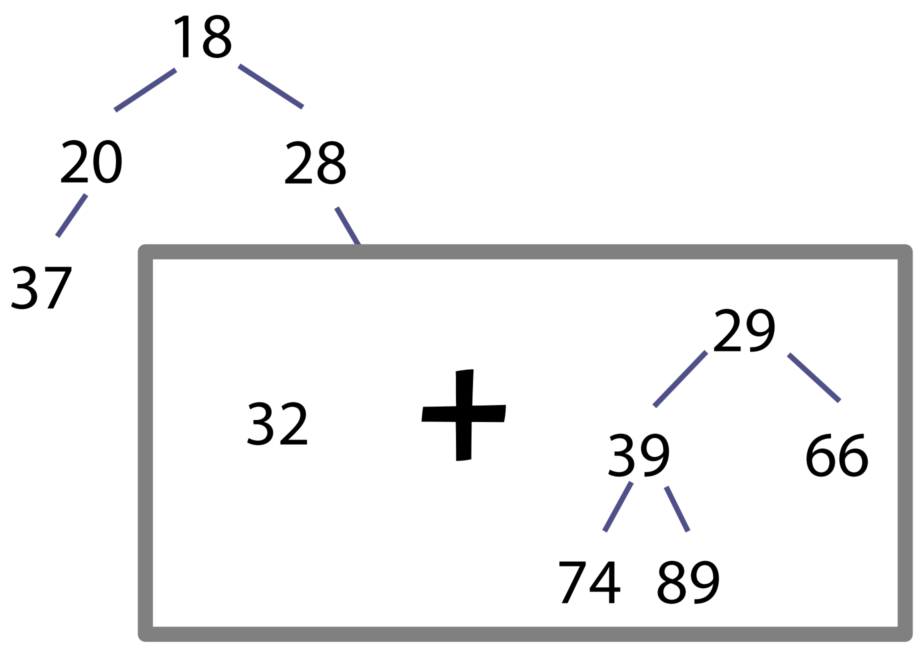
Naive merging

28 < 29 so pick 28 as the root of the merged tree



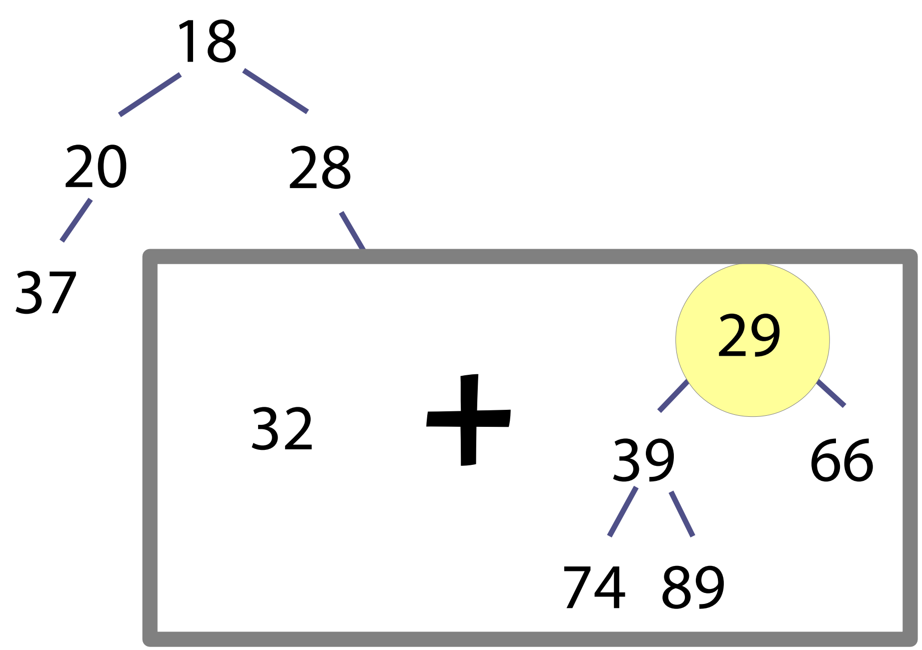
Naive merging

Recursively merge the right branch of 28 and the 29 tree



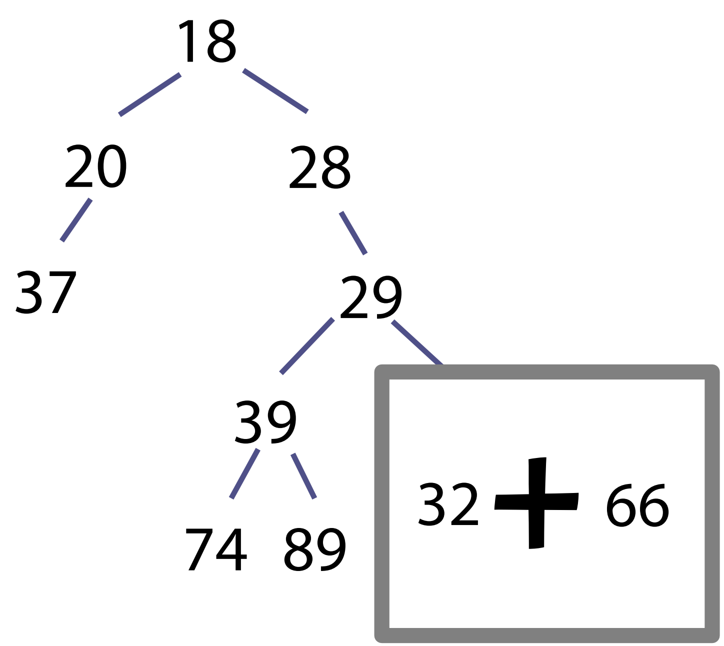
Naive merging

29 < 32 so pick 29 as the root of the merged tree



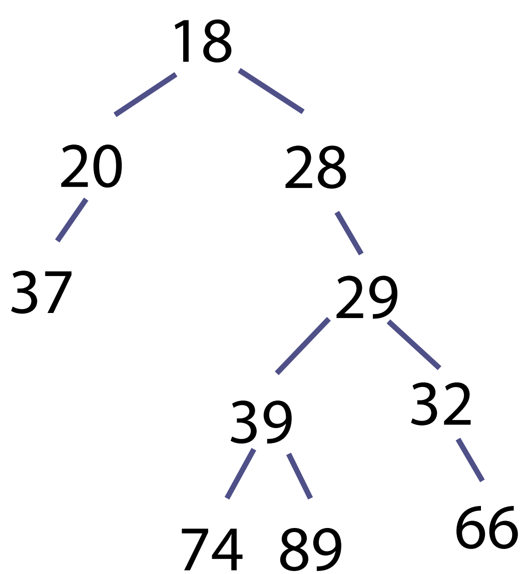
Naive merging

Recursively merge the right branch of 29 with 32



Naive merging

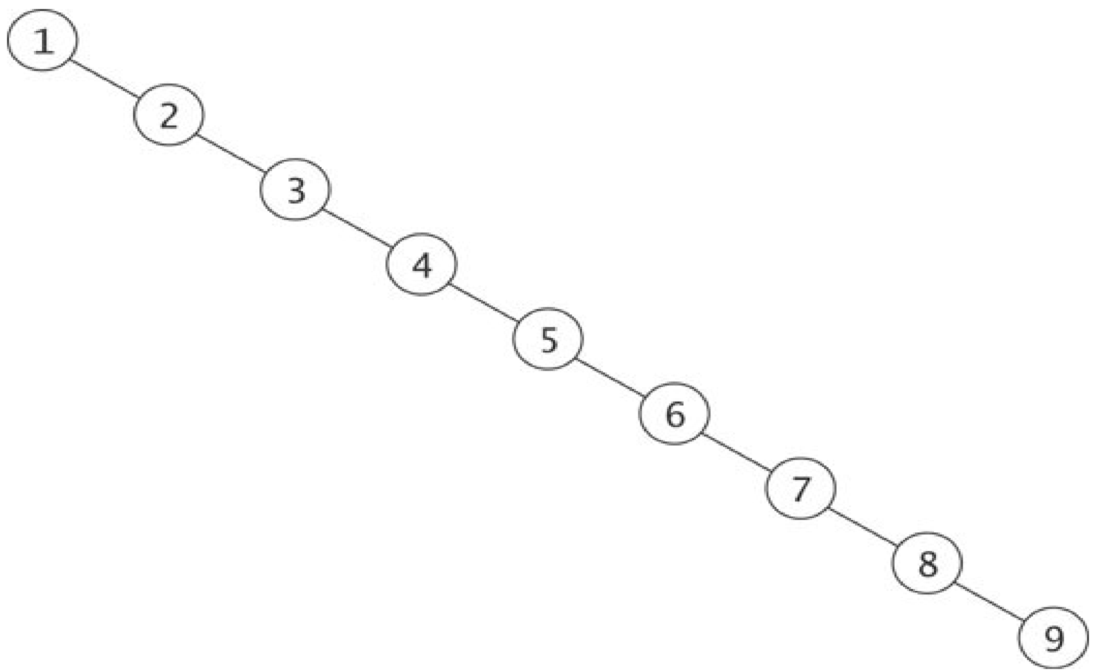
Base case: merge 66 with the empty tree



Notice that the tree looks pretty "right-heavy"

Worst case for naive merging

A right-heavy tree:

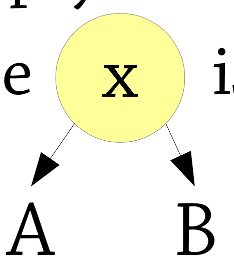


Unfortunately, you get this just by doing insertions! So insert takes O(n) time...

How can we stop the tree from becoming right-heavy?

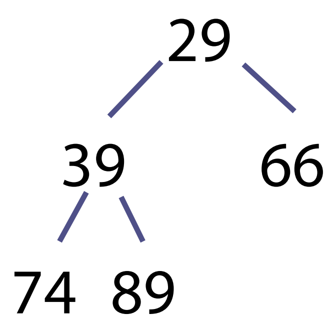
Null path length

Define the *null path length* (npl) of a binary tree as follows:

- The npl of the empty tree is 0
- The npl of the tree 

is 1 + min(npl(A), npl(B))

npl = 2

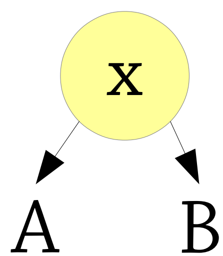


Observation: npl is at most O(log n), where n = number of nodes (if npl is k, then first k levels of tree are "full")

Leftist heaps

A *leftist heap* is a binary tree satisfying the heap property and the following invariant:

For any node in the tree…



…we must have npl(A) ≥ npl(B).

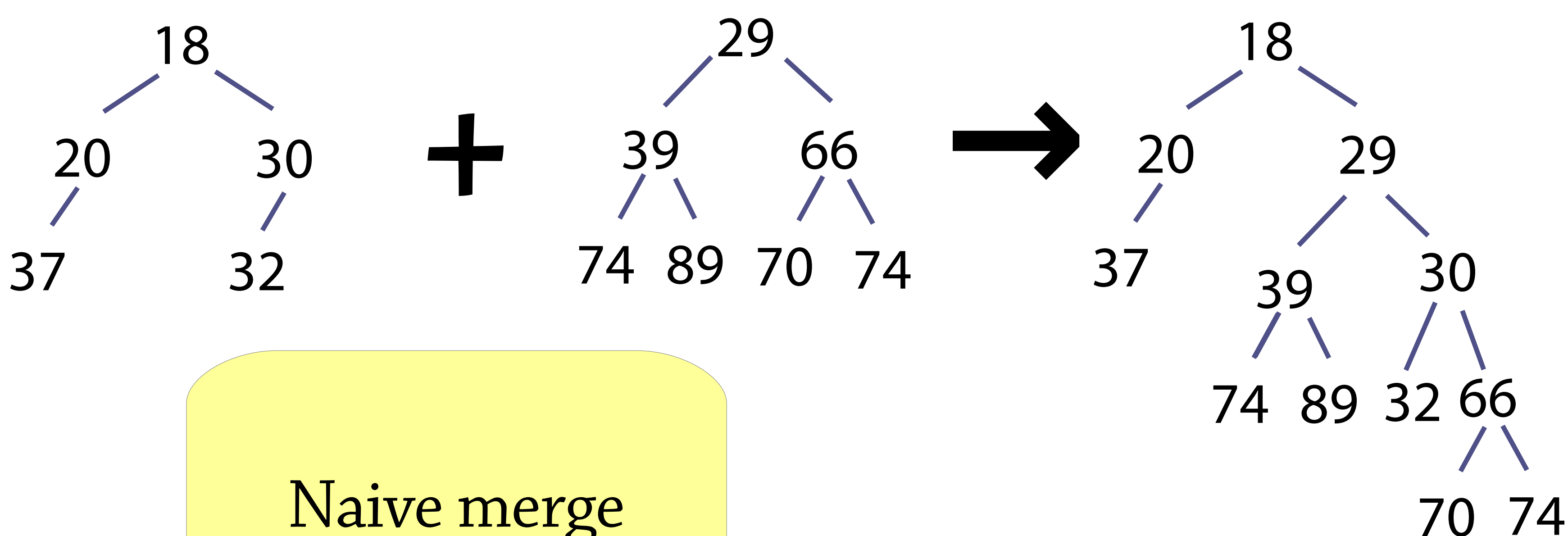
This means the tree is not right-heavy

If this invariant is violated, we can repair it by swapping A and B!

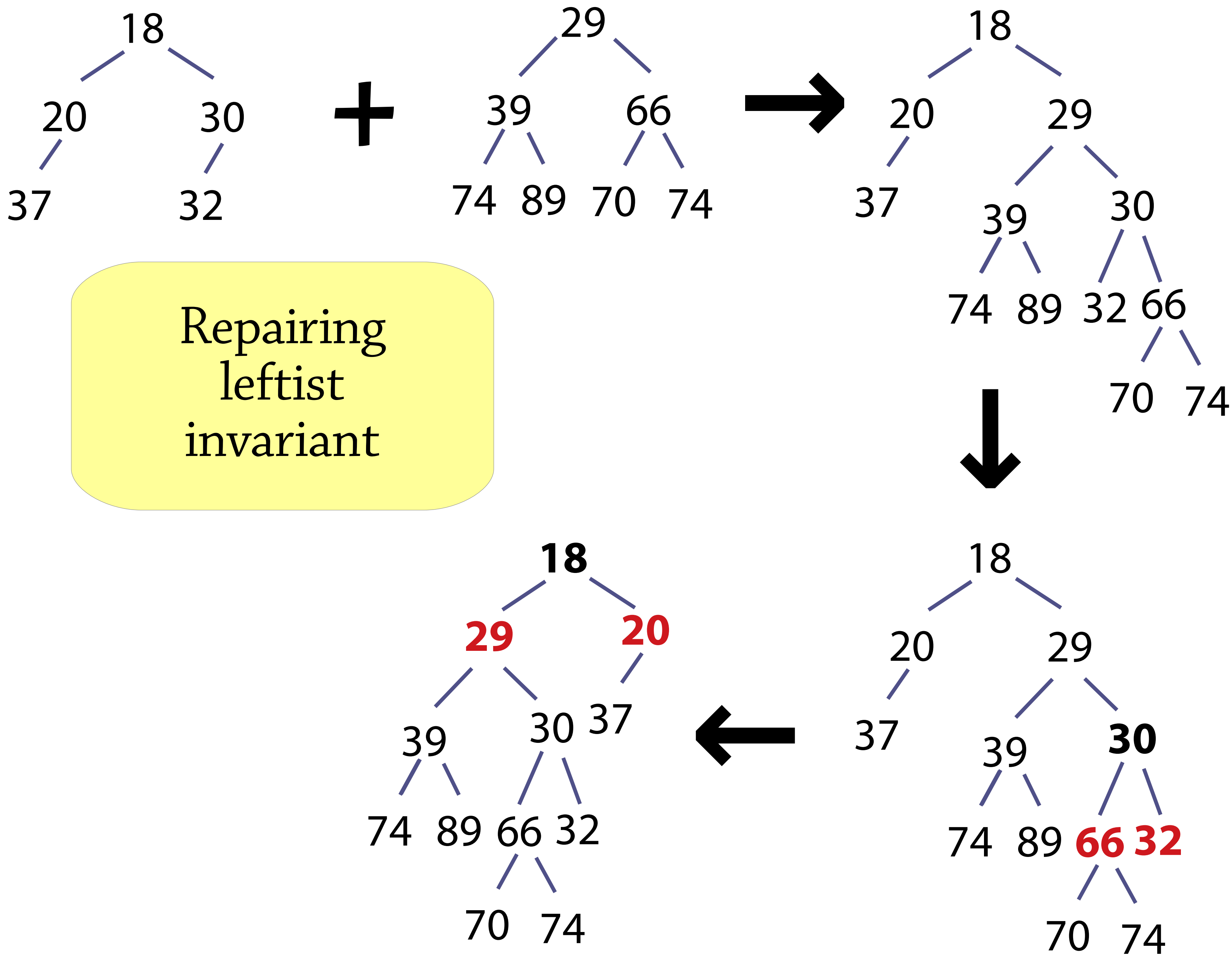
Note: we must have npl(x) = 1 + npl(B). This means that naive merging will take logarithmic time!

Example

One way to do leftist merge is to first do naive merge, then go up the tree swapping left and right children when necessary...



Naive merge


Repairing leftist invariant

Leftist heaps – implementation

Add a field

```
int npl;
```

to each node, which records the null path length

- Invariant: x.npl == null path length of x

In the recursive case of merge:

- Merge the other tree into the right child of this node
- Then swap the left and right children if left.npl < right.npl
- Then update the npl of this node (npl = right.npl + 1)

Leftist heaps

Implementation of priority queues:

- binary trees with heap property and leftist invariant, which avoids right-heavy trees
- other operations are based on merge

A good fit for functional languages:

- based on trees rather than arrays, tiny implementation!

Other data structures based on naive merging + avoiding right heavy trees:

- skew heaps (always swap left and right child)
- meldable heaps (swap children at random)

See webpage for link to visualisation site!