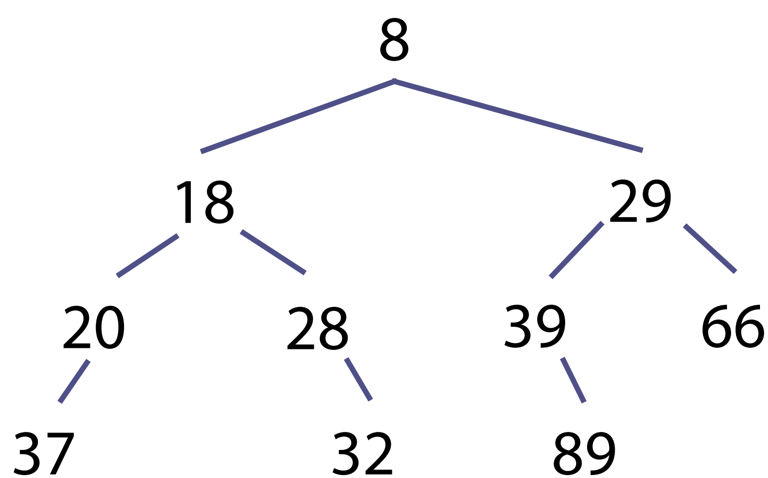
Skew heaps

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):



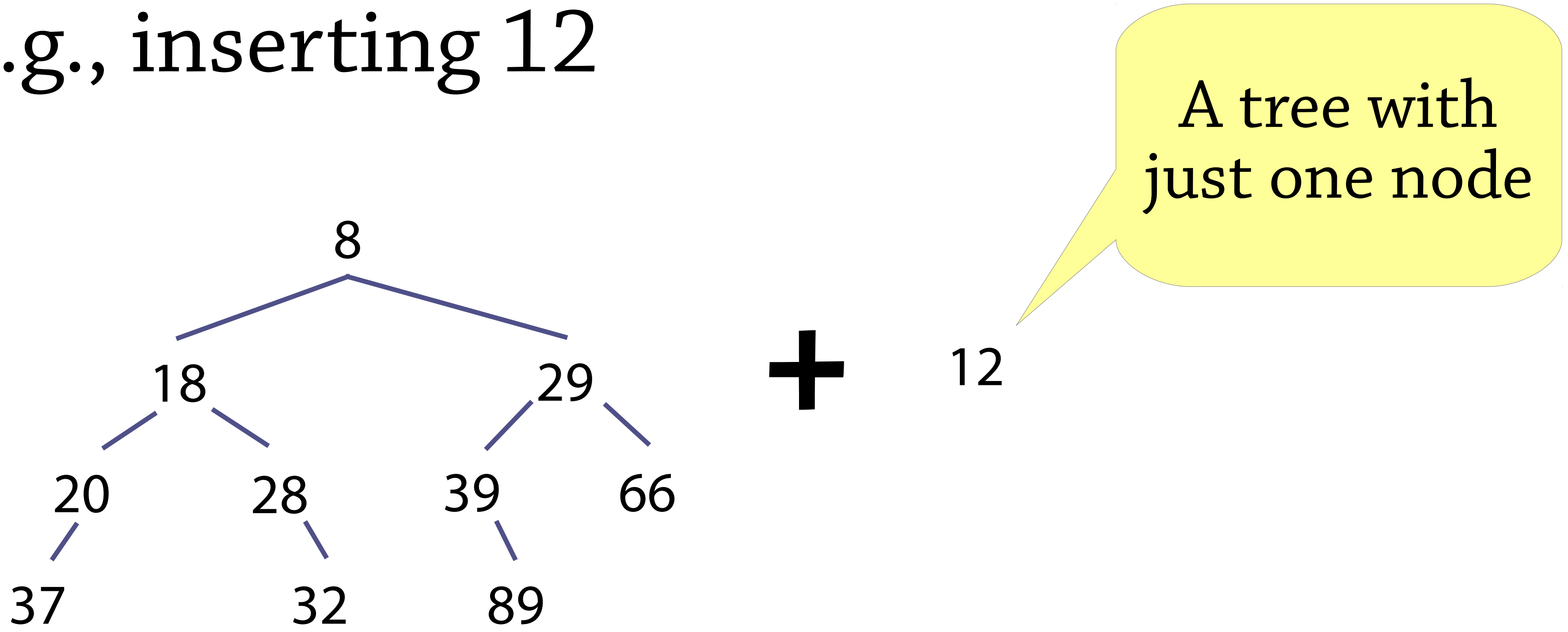
We can implement the other priority queue operations in terms of merging!

Insertion

To insert a single element:

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

E.g., inserting 12

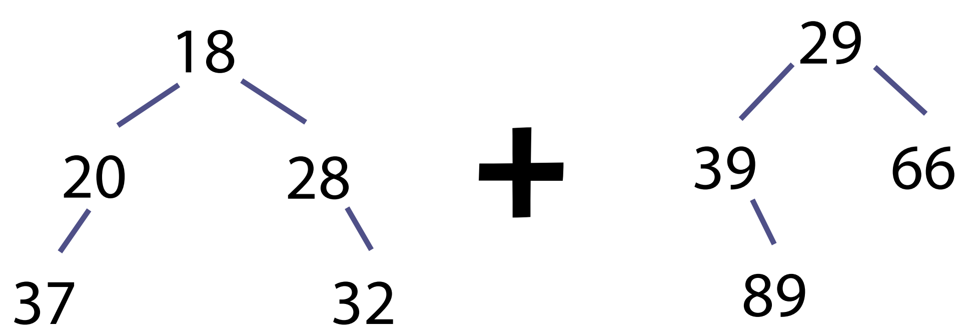


Delete minimum

To delete the minimum element:

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

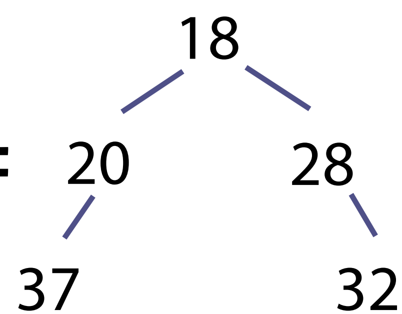
E.g., deleting 8 from the previous heap



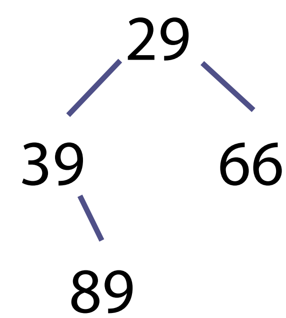
Naive merging

How to merge these two heaps?

A =



B =



Idea: root of resulting heap must be 18

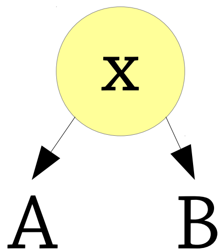
Take heap A. Pick one of its children. Recursively merge B into that child.

Let's use A's right child for no particular reason

Naive merging

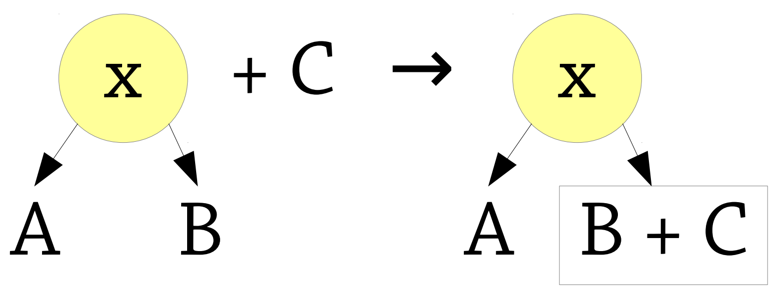
To merge two non-empty heaps:

Pick the heap with the smallest root:



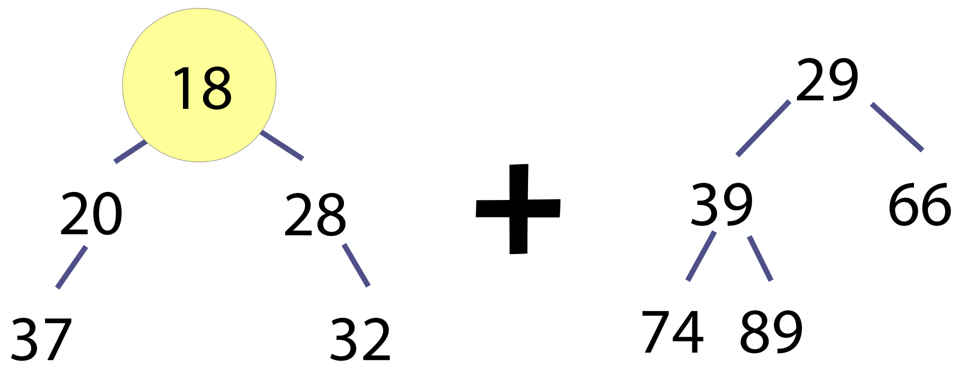
Let C be the other heap

Recursively merge B and C!



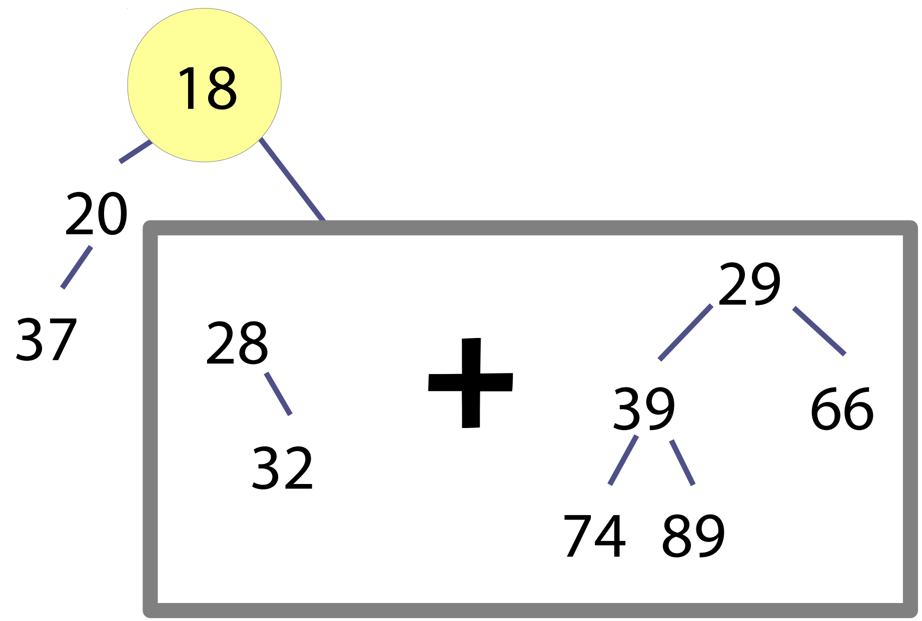
Example

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



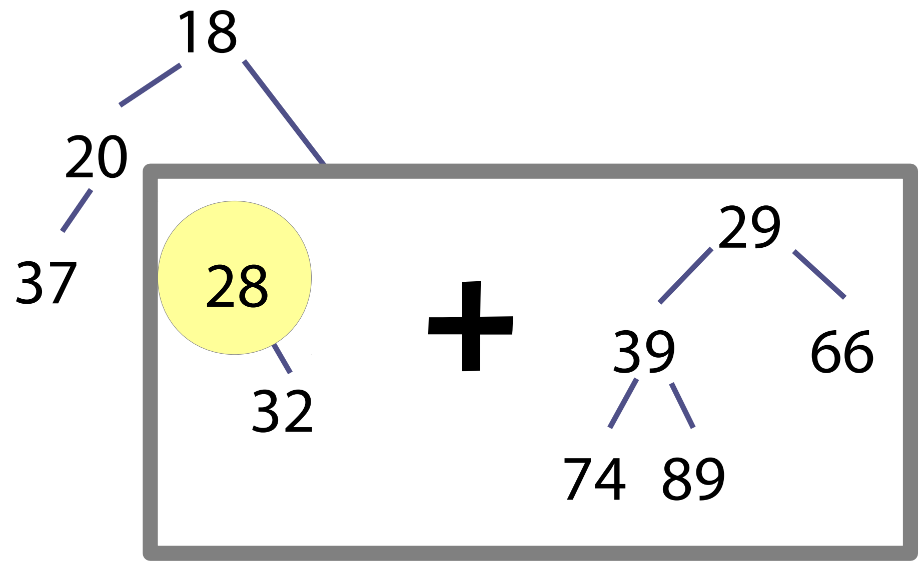
Naive merging

Recursively merge the right branch of 18 and the 29 tree



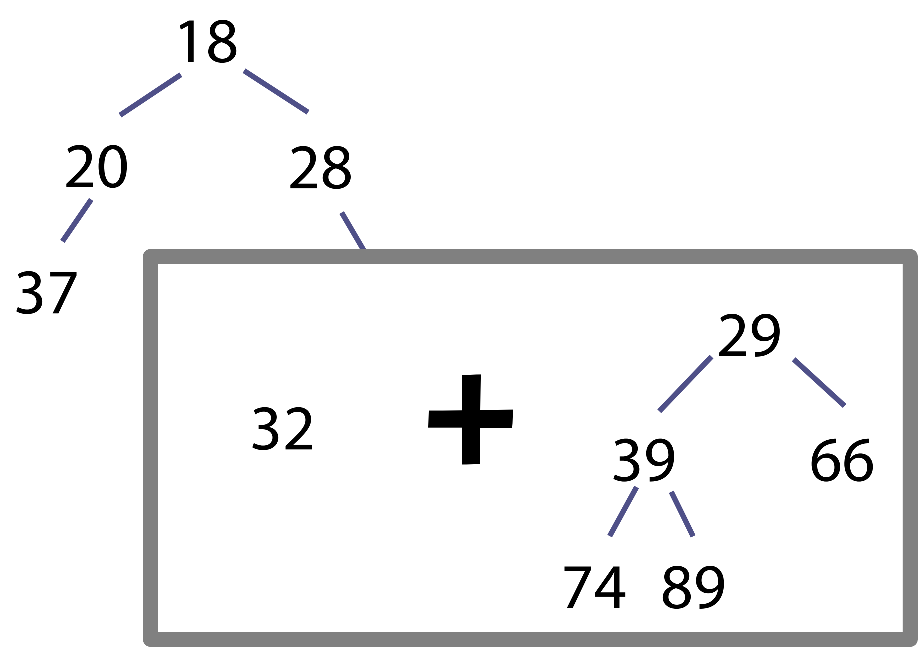
Naive merging

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



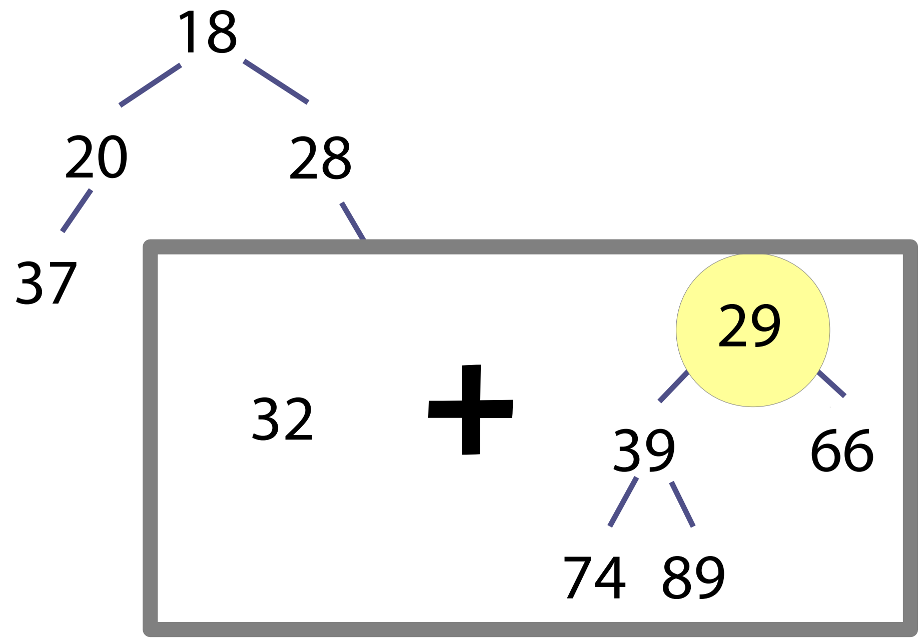
Naive merging

Recursively merge the right branch of 28 and the 29 tree



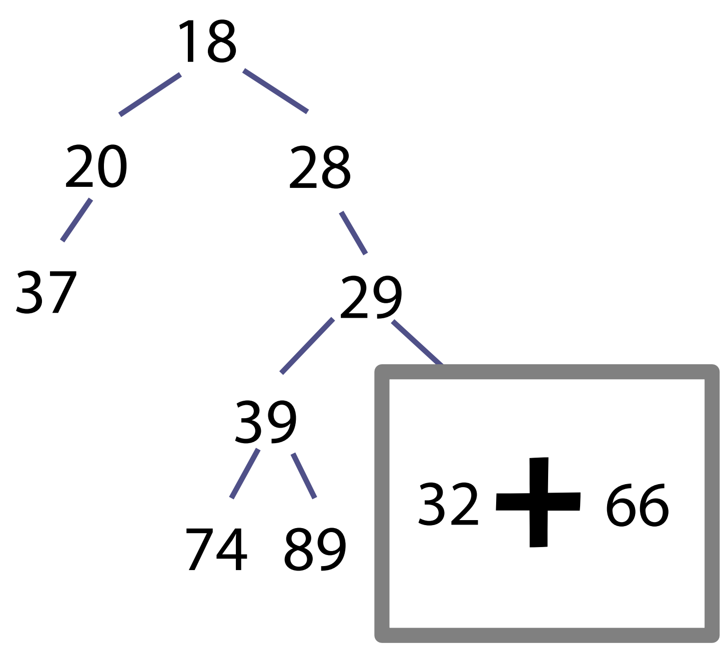
Naive merging

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



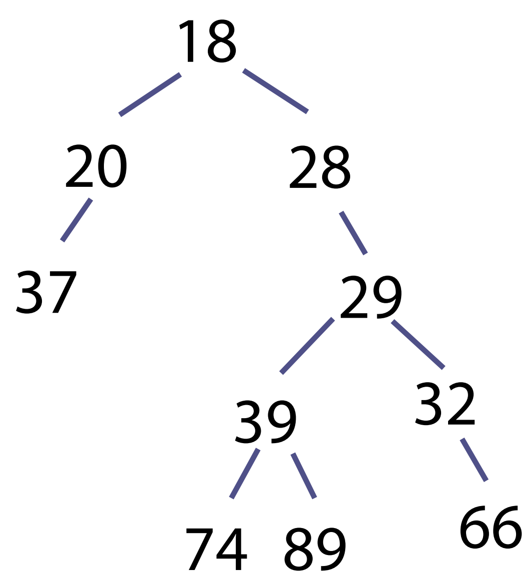
Naive merging

Recursively merge the right branch of 29 with 32



Naive merging

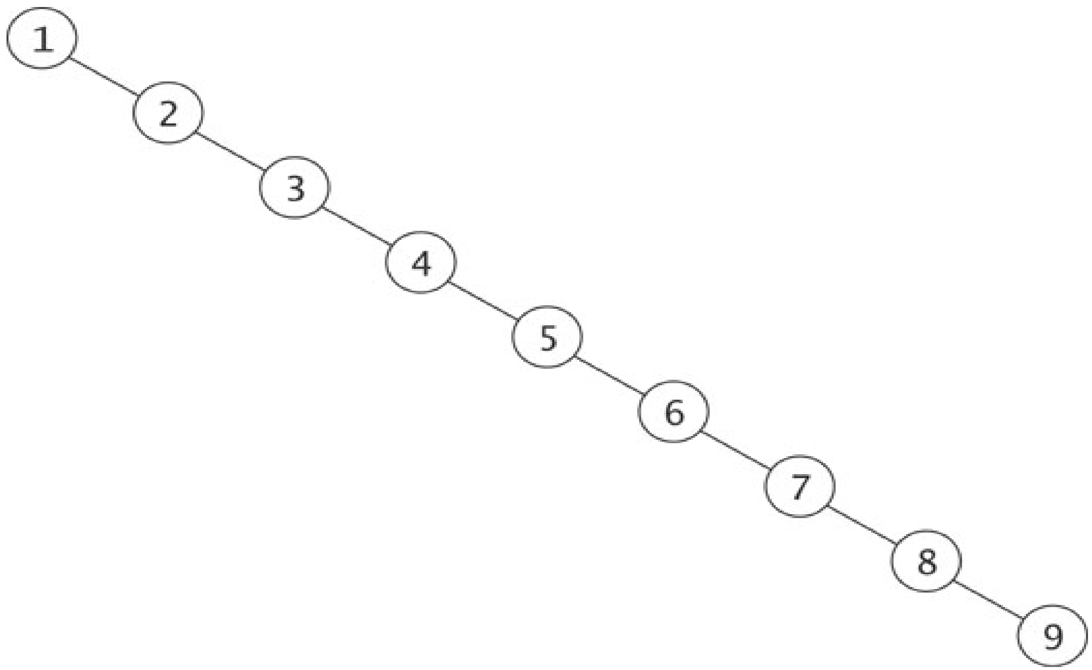
Base case: merge 66 with the empty tree



Notice that the tree looks pretty "right-heavy"

Worst case for naive merging

A right-heavy tree:

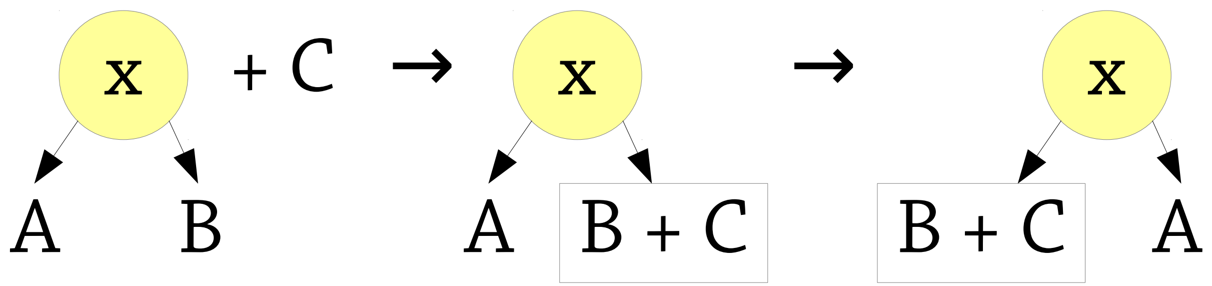


Unfortunately, you get this just by doing insertions! So insert takes O(n) time...

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

Skew merging

In a skew heap, after making a recursive call to merge, we *swap the two children*:



Amazingly, this small change completely fixes the performance of merge!

We never end up with right-heavy trees.

We get O(log n) amortised complexity.

Naive merging in code

```
data Heap a =
  Nil | Node a (Heap a) (Heap a)
root (Node x _ _) = x

merge x Nil = x
merge Nil x = x
merge x y
  | root x > root y = merge y x
merge (Node x a b) c =
  Node x a (merge b c)
```

Make sure that first argument has smallest root

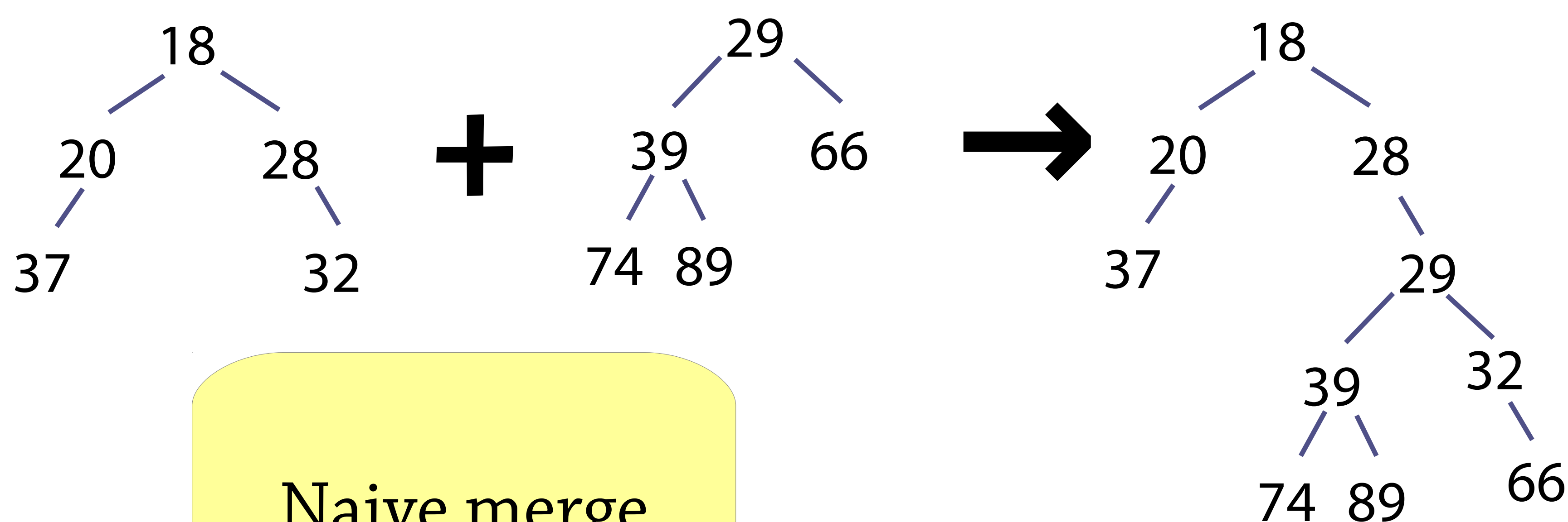
Skew merging in code

```
data Heap a =
  Nil | Node a (Heap a) (Heap a)
root (Node x _ _) = x

merge x Nil = x
merge Nil x = x
merge x y
  | root x > root y = merge y x
merge (Node x a b) c =
  Node x (merge b c) a
```

Example

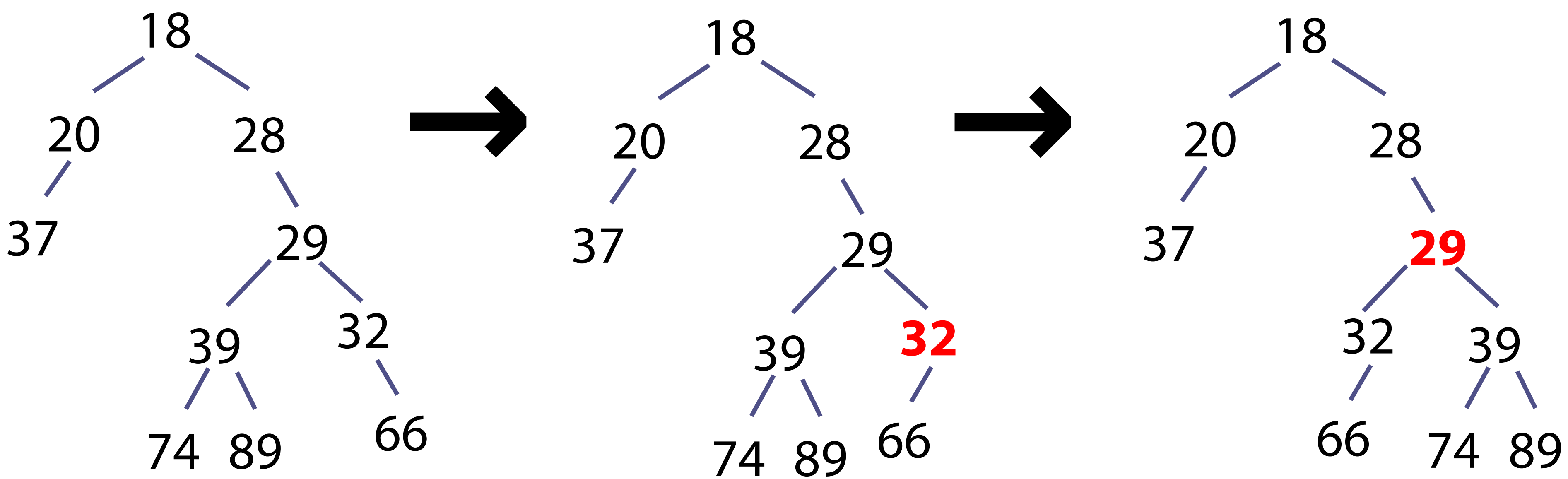
One way to do skew merge is to first do naive merge, then go up the tree swapping left and right children...



Naive merge

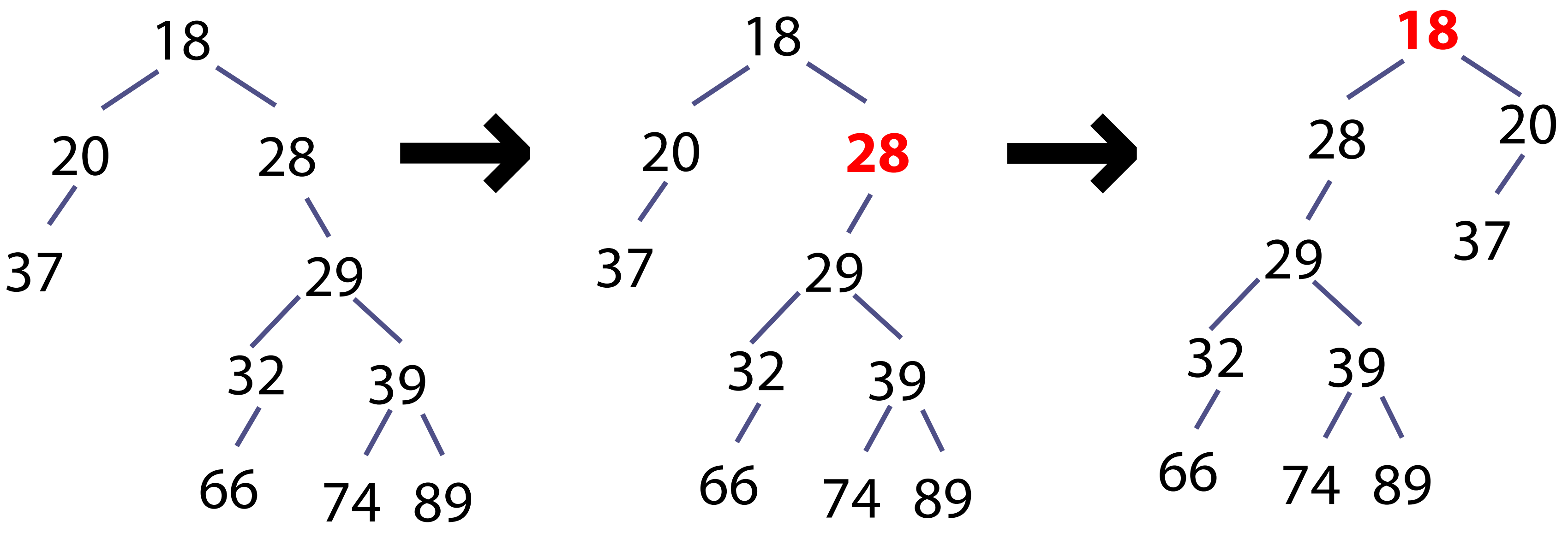
Example

...like this:



Example

...like this:



Skew heaps

Implementation of priority queues:

- binary trees with heap property
- skew merging avoids right-heavy trees, gives O(log n) amortised complexity
- other operations are based on merge

A good fit for functional languages:

- based on trees rather than arrays

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

- leftist heaps (swap children when needed)
- meldable heaps (swap children at random)

See webpage for link to visualisation site!