

# leftist heaps



data structure

# implemented as an immutable binary tree

```
(deftem 'node 'r nil 'x nil 'a nil 'b nil)

(def rank (h)
  (if h h!r 0))

(def maket (x a b)
  (if (>= rank.a rank.b) ; enforce leftist property. swap children if rank.b > rank.a
    (inst 'node 'r (+ 1 rank.b) 'x x 'a a 'b b)
    (inst 'node 'r (+ 1 rank.a) 'x x 'a b 'b a)))

(def isempty (h)
  (if (is (type h) 'tem) nil t))

(def merge (h1 h2)
  (if (no (and h1 h2)) (or h1 h2) ; if both aren't there, return whichever is
    (<= h1!x h2!x) (maket h1!x h1!a (merge h1!b h2)) ; smaller value moves to new root
    (maket h2!x h2!a (merge h1 h2!b)))))

(def insert (x h)
  (merge h (inst 'node 'r 1 'x x)))

(def findmin (h)
  (if (isempty h) (err "EMPTY") h!x))

(def deletemin (h)
  (if (isempty h) (err "EMPTY") (merge h!a h!b)))
```

# invariants

- each node  $\leq 2$  children (binary tree)
- the root is the minimum of the tree
- leftist property:
  - $(\text{rank left}) \geq (\text{rank right})$ 
    - rank is the distance to the nearest leaf
- every subtree is a leftist heap

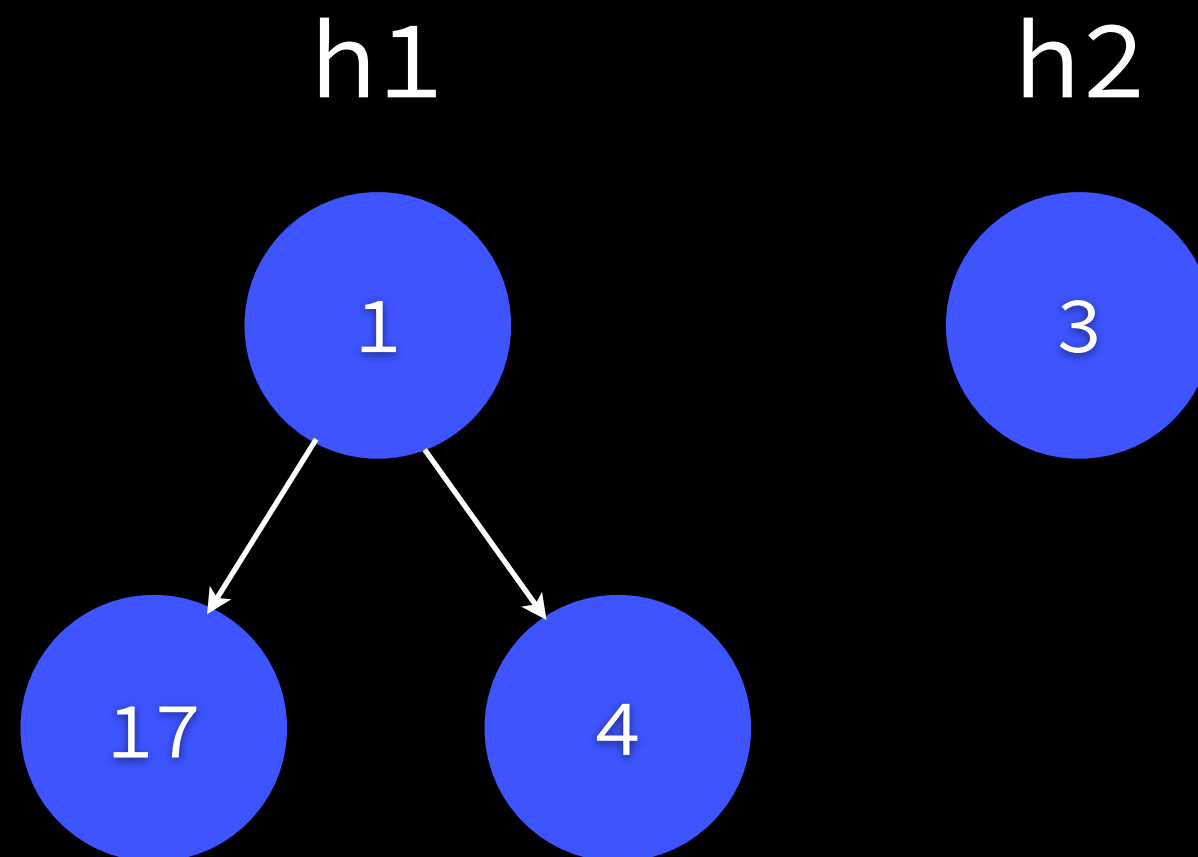
# why?

## $O(\log n)$ merging

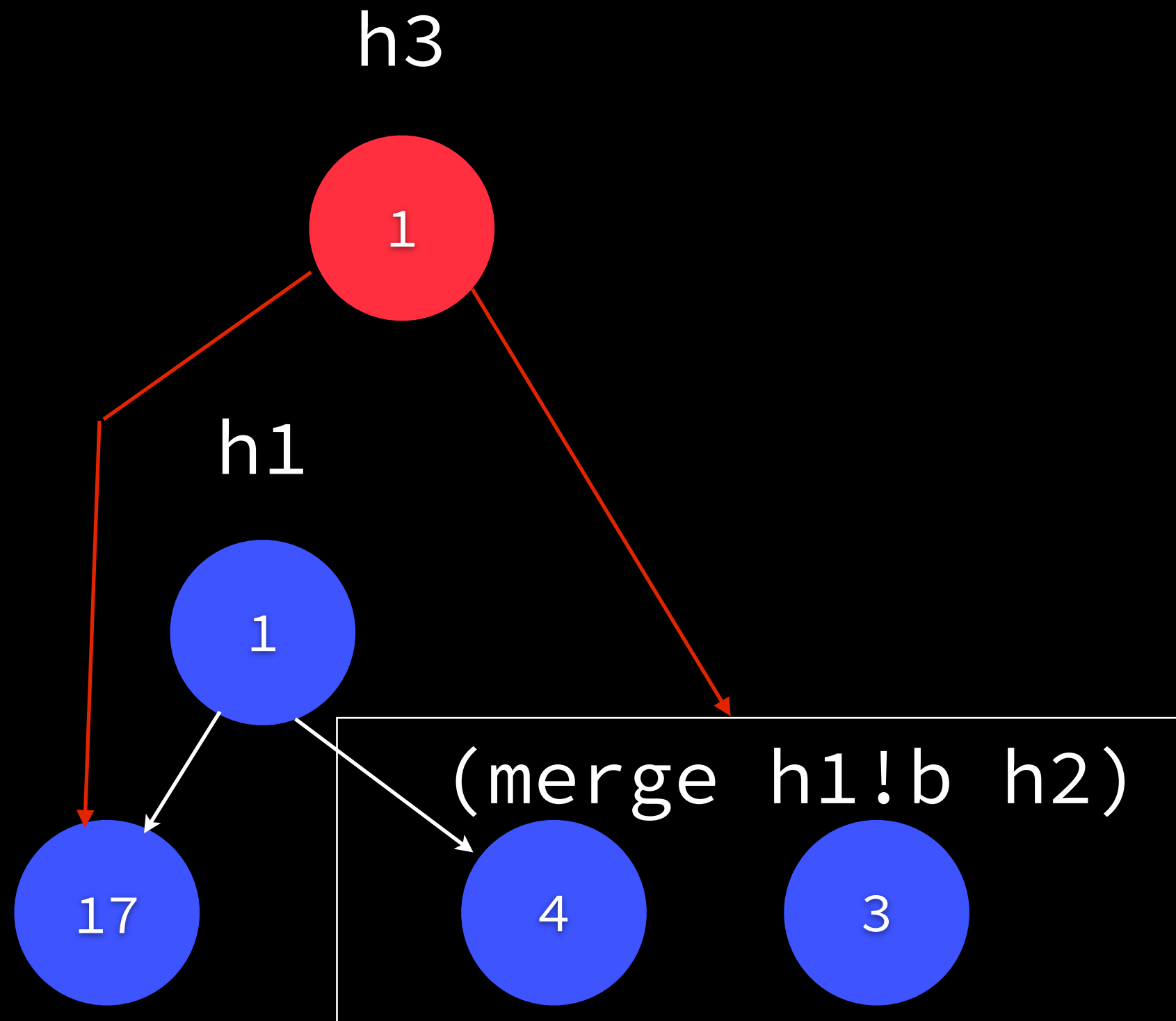
- cost? lose  $O(\log n)$  search
- benefit?  $O(\log n)$  merge  
 $O(1)$  findmin
- use case: priority queues

(= h3 (merge h1 h2))

(maket h1!x h1!a (merge h1!b h2))



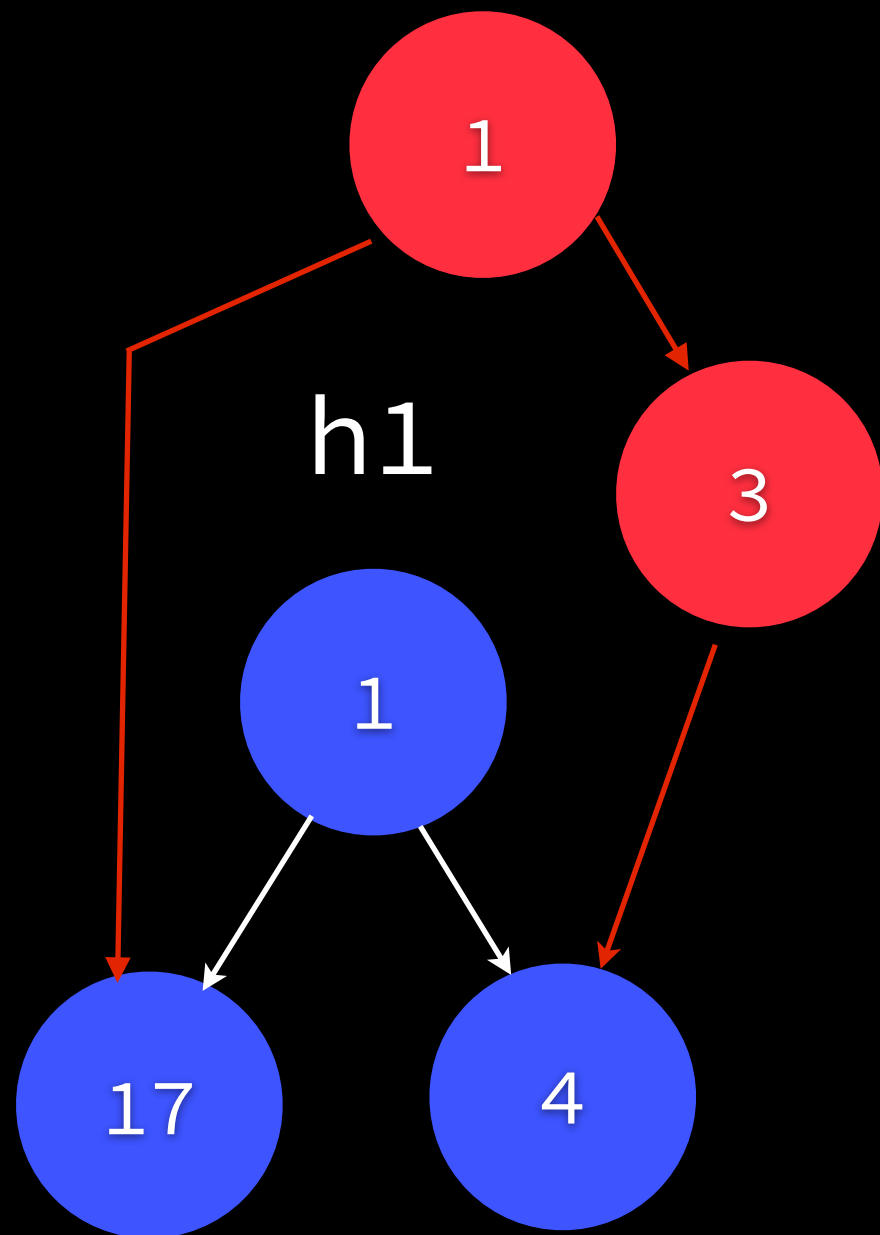
`(maket h1!x h1!a (merge h1!b h2))`





(merge h1!b h2)

h3



```
(maket h2!x h2!a (merge h1!b h2!b))
```

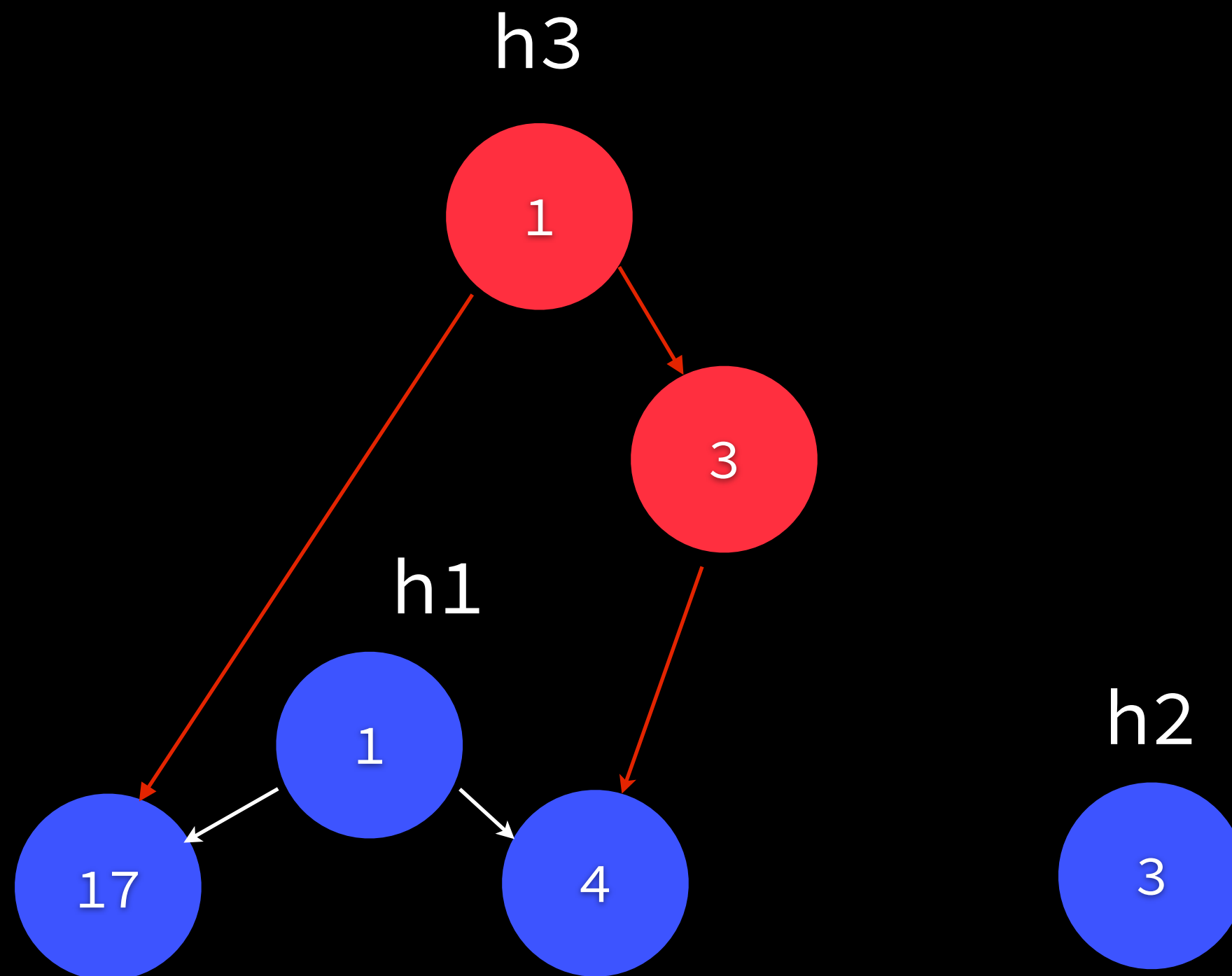
```
(maket h2!x nil (merge h1!b nil))
```

```
(maket h2!x nil h1!b)
```

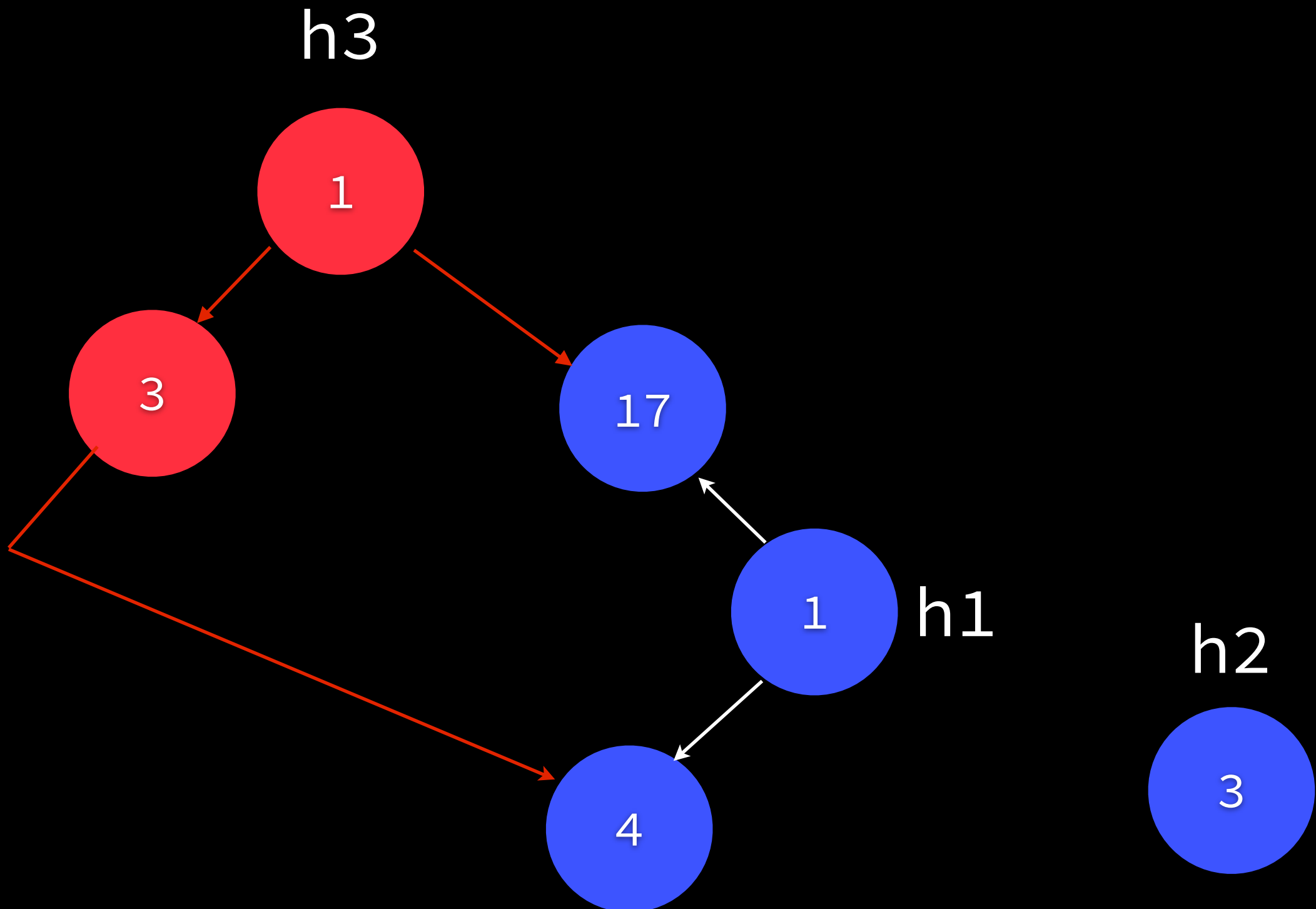
h2



are we done? no.



# enforce leftist property (flip it)



# Purely Functional Data Structures by Chris Okasaki

my solutions  
[github.com/brianru/pfds-arc](https://github.com/brianru/pfds-arc)