Segment Trees

Legit Dark Magic

Range Sum

Index	0	1	2	3	4	5	6	7
Value	8	14	23	5	14	10	5	20

- We want to calculate the sum of a range of values
- RangeSum(1,5) = 14 + 23 + 5 + 14 + 10
- Complexity?

Range Sum

Index	0	1	2	3	4	5	6	7
Value	8	14	23	5	14	10	5	20
Cumulative Sum	8	22	45	50	64	74	79	99

Sum(i,j) = cumulative sum[j] – cumulative sum[i-1]

But what if we want to change the one of the values?

What’s the complexity to update it?

Complexities

Type	Range Sum	Point update	Range Update
No pre-processing	O(n)	O(1)	O(n)
Store cumulative	O(1)	O(n)	O(n)

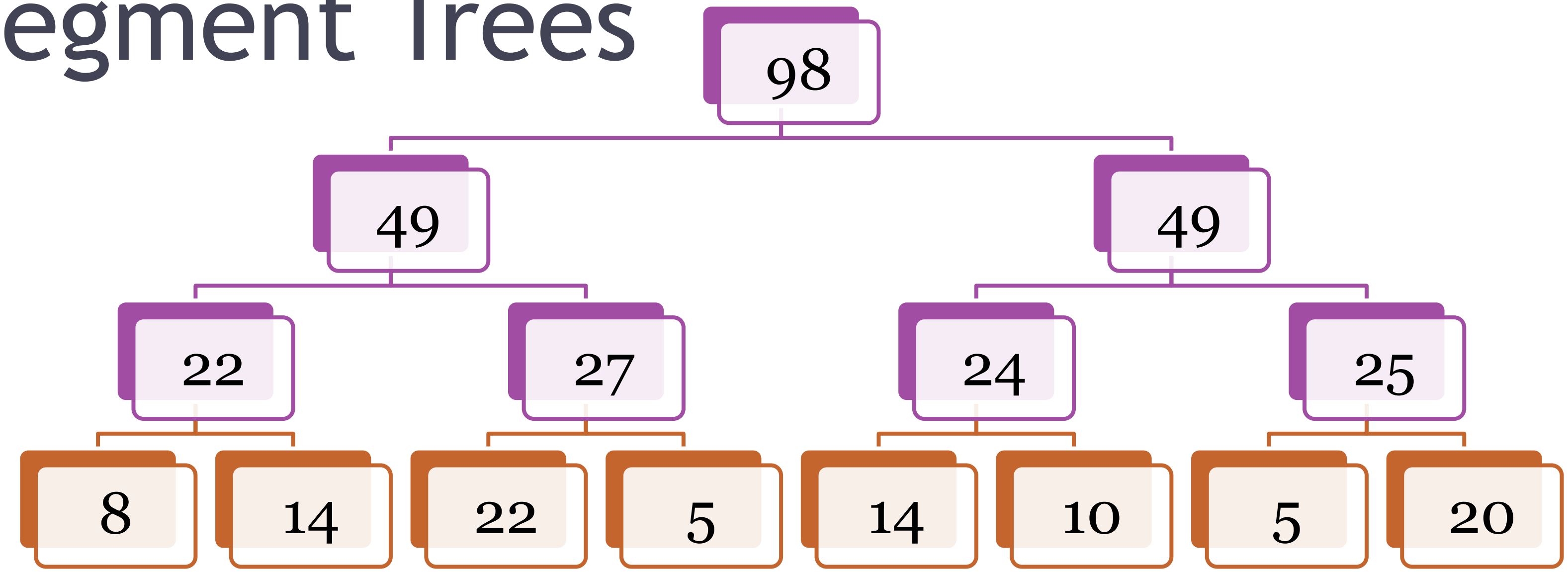
Complexities

Type	Range Sum	Point update	Range Update
No pre-processing	O(n)	O(1)	O(n)
Store cumulative	O(1)	O(n)	O(n)
Segment trees	O(log(n))	O(log(n))	O(log(n))

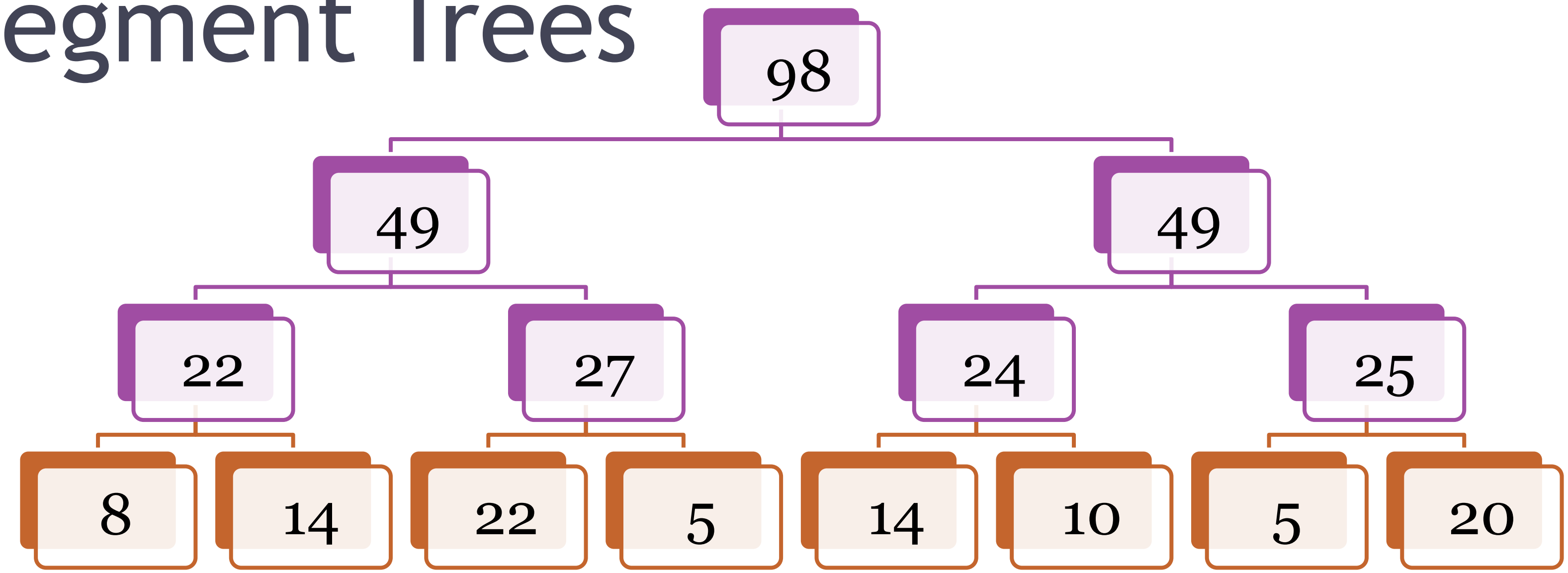
Segment Tree Operations

- Range Query (sum, min, max, gcd, etc)
- Point Update
- Range Update

Segment Trees



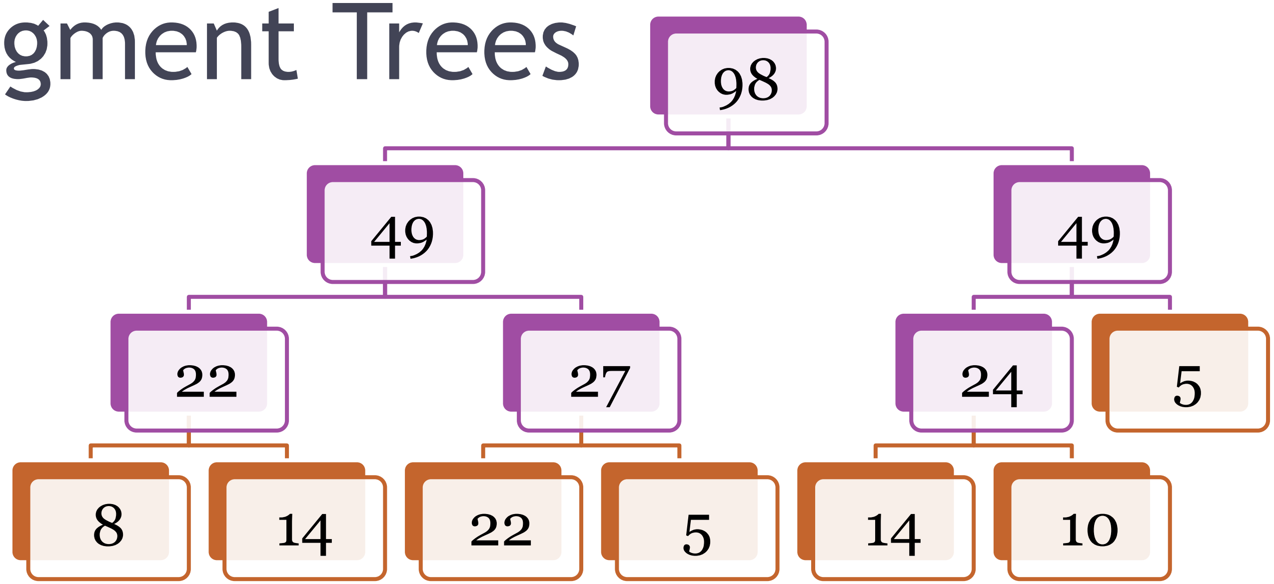
Segment Trees



Binary Heap array implementation
Binary heap = binary tree filled layer by layer

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	98	49	49	22	27	24	25	8	14	22	5	14	10	5	20

Segment Trees

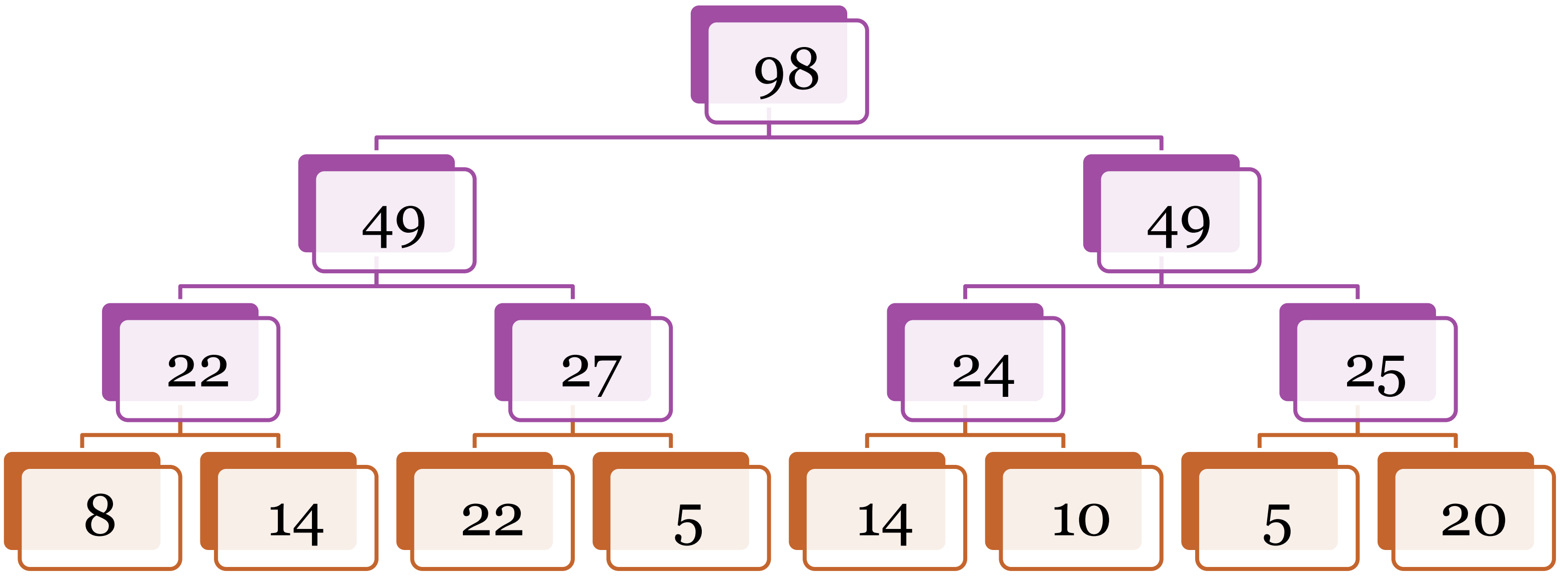


Binary Heap array implementation
Binary heap = binary tree filled layer by layer

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	98	49	49	22	27	24	25	8	14	22	5	14	10	5

Building

- Complexity?
- Easiest to do with a recursive function



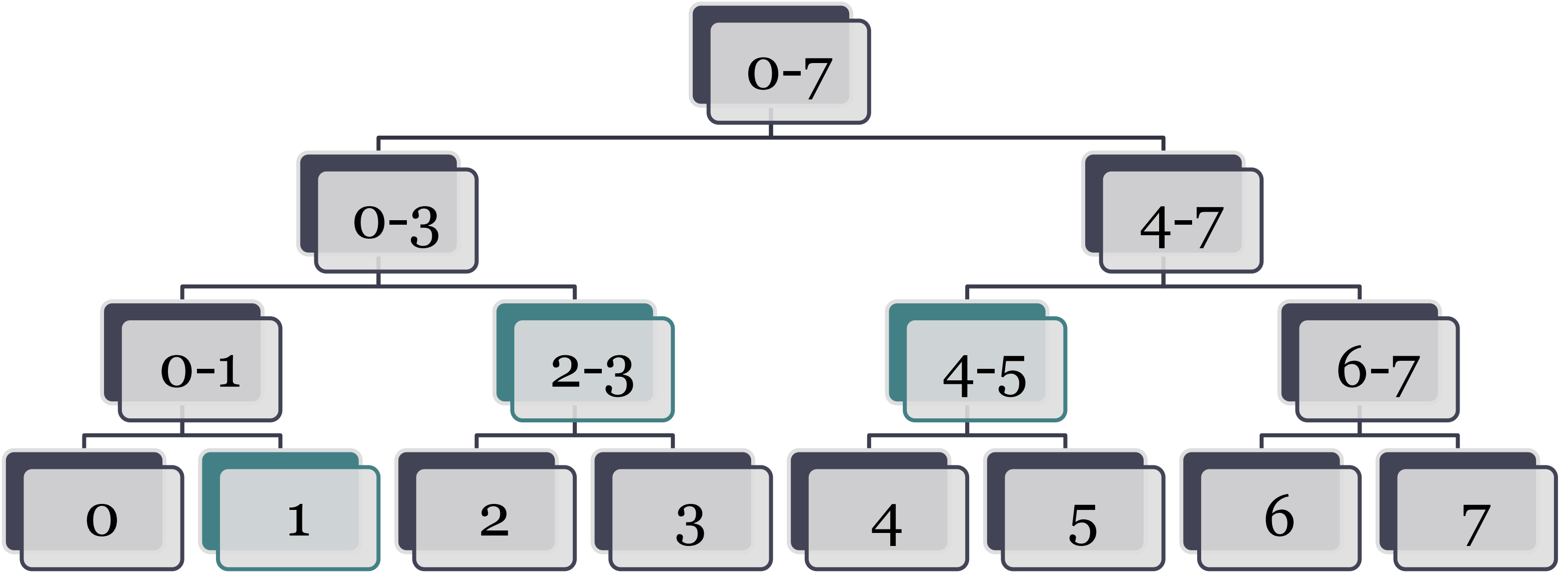
Building

```
Build(node):
      if node is a leaf:
            value of node = value in array cell
      else:
            Build(node's left child);
            Build(node's right child);
            value of node = value of left child +
                              value of right child
```

Visualgo time

Querying

- Just use the minimal set of nodes which cover the range
- IE querying Sum(1,5)



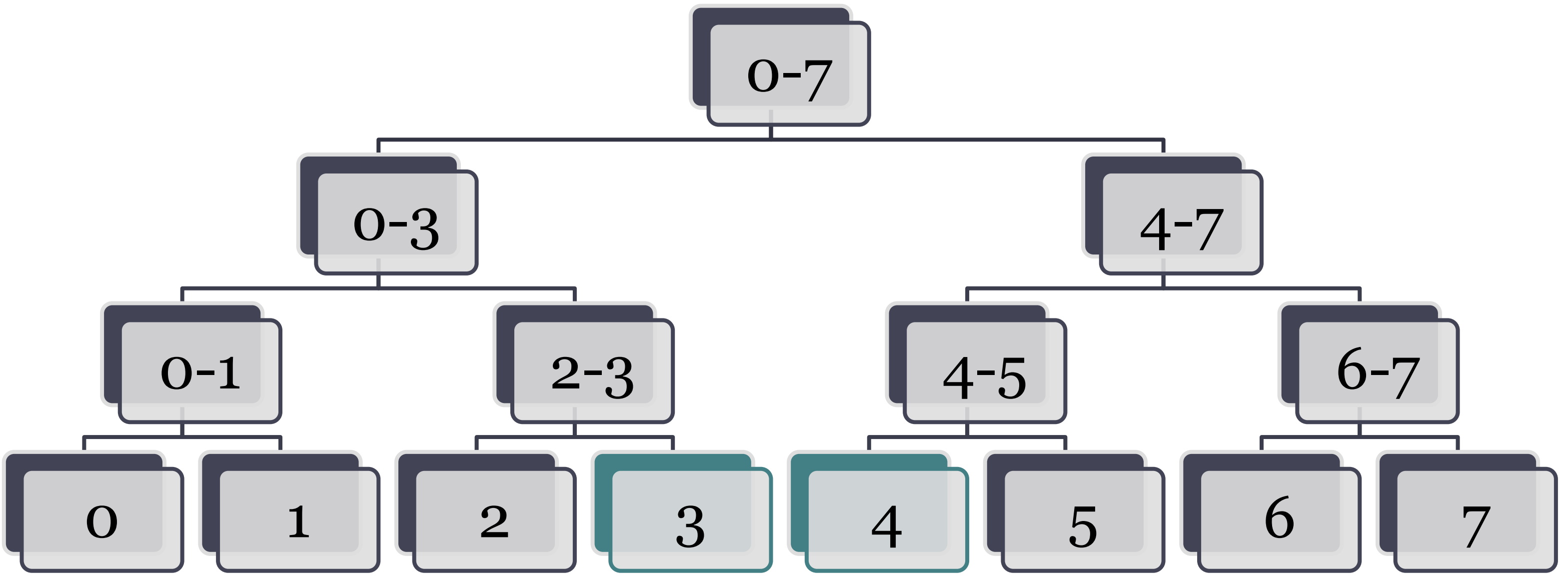
Querying

```
// node is the current node, left and right is the range of
// the query.
Query(node, left, right):
    if the node's range has no overlap with the query range:
        return 0
    else if the node's range is within the query range:
        return value of node
    else
        leftSeg = query(left child, left, right)
        rightSeg = query(right child, left, right)
    return leftSeg + rightSeg
```

Visualgo time!

Querying – Worst case

- Query sum(3,4)
- O(log(n)) worst case

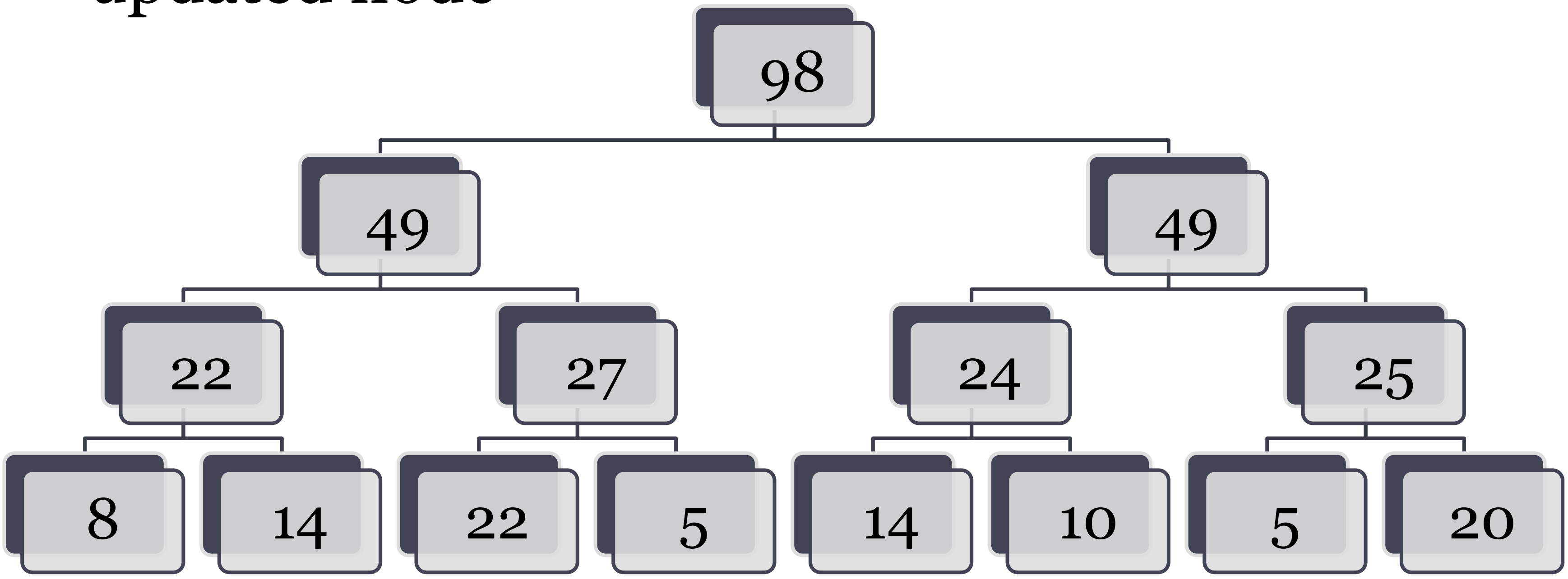


Updates

- Point Updates
- Range Updates

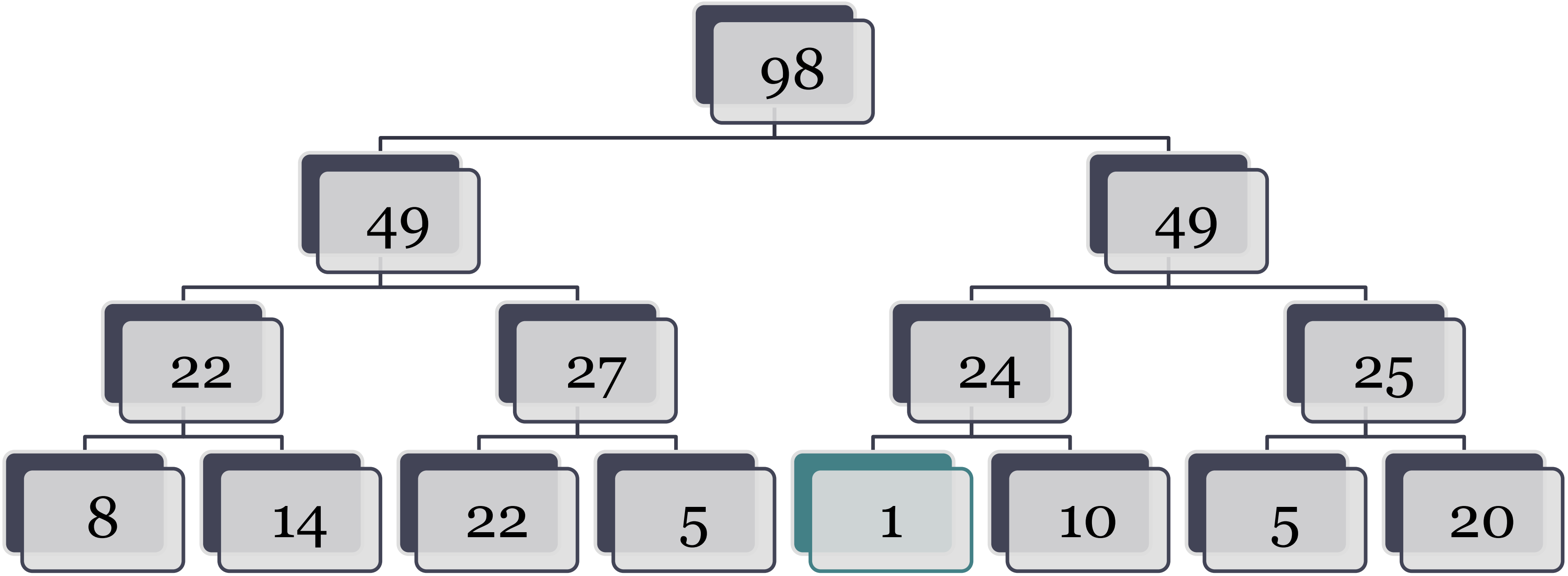
Point Updates

- The only affected nodes are the ancestors of the updated node



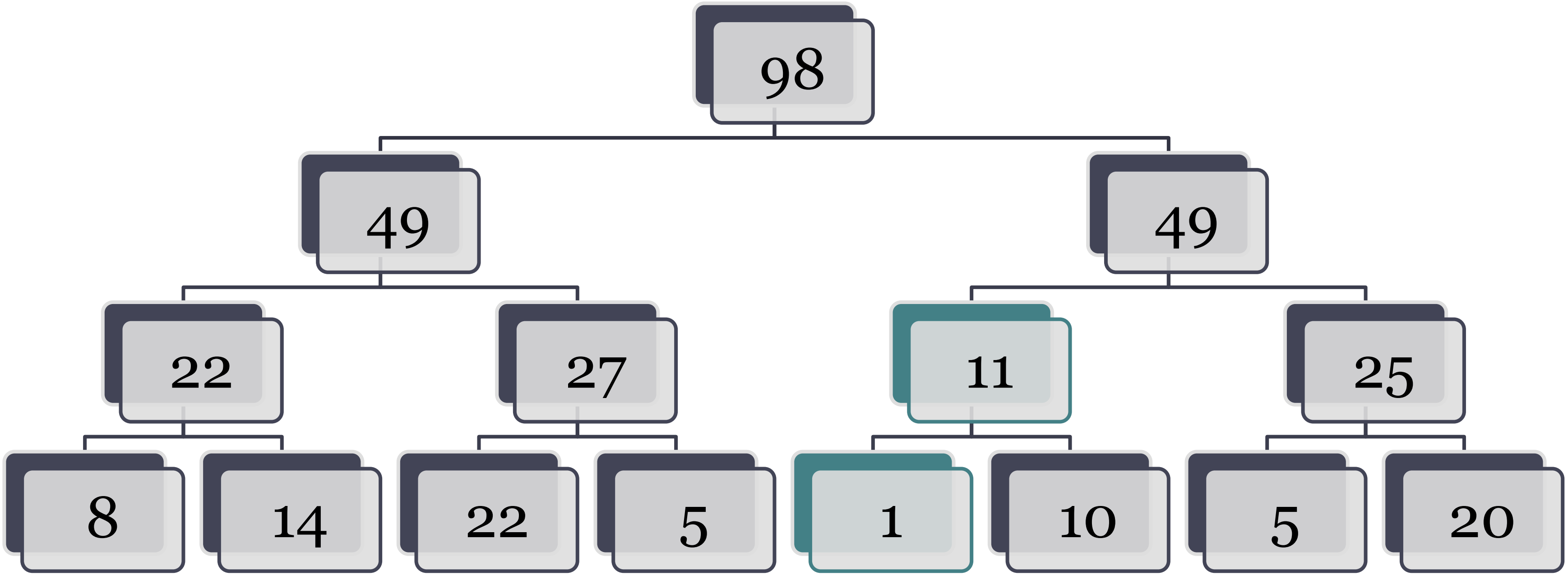
Point Updates

- Change 14 to 1



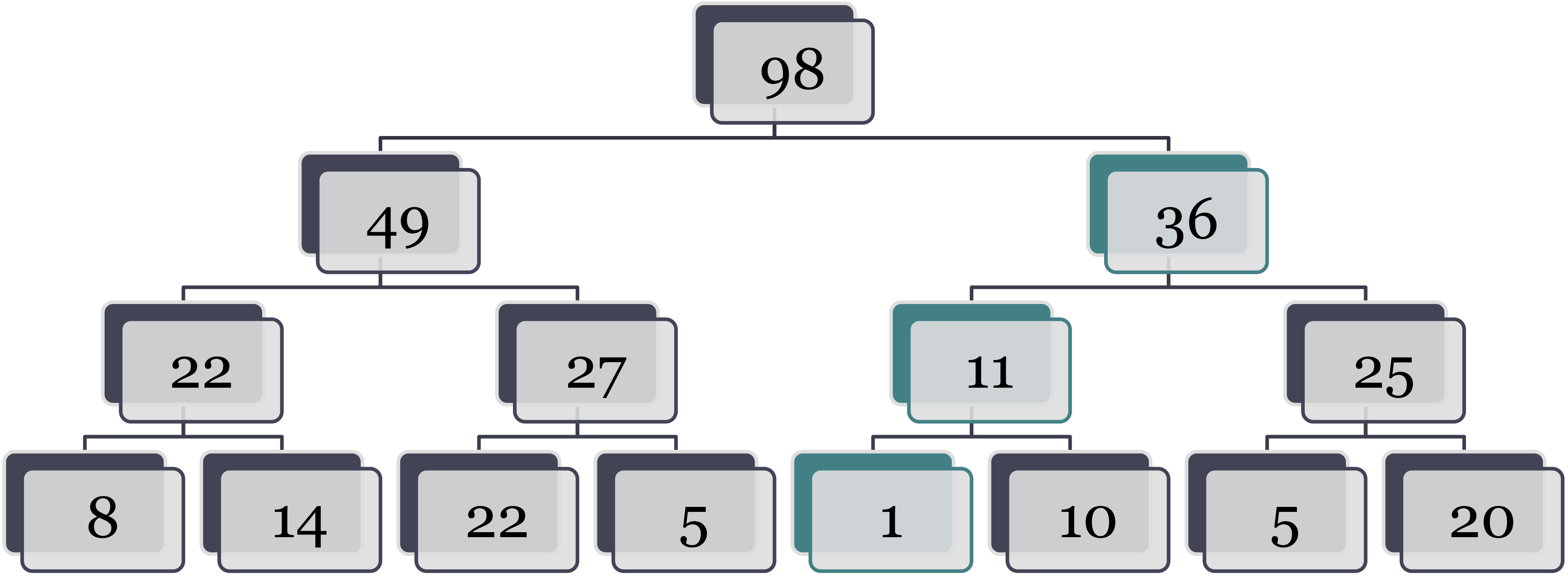
Point Updates

- 24 to 11



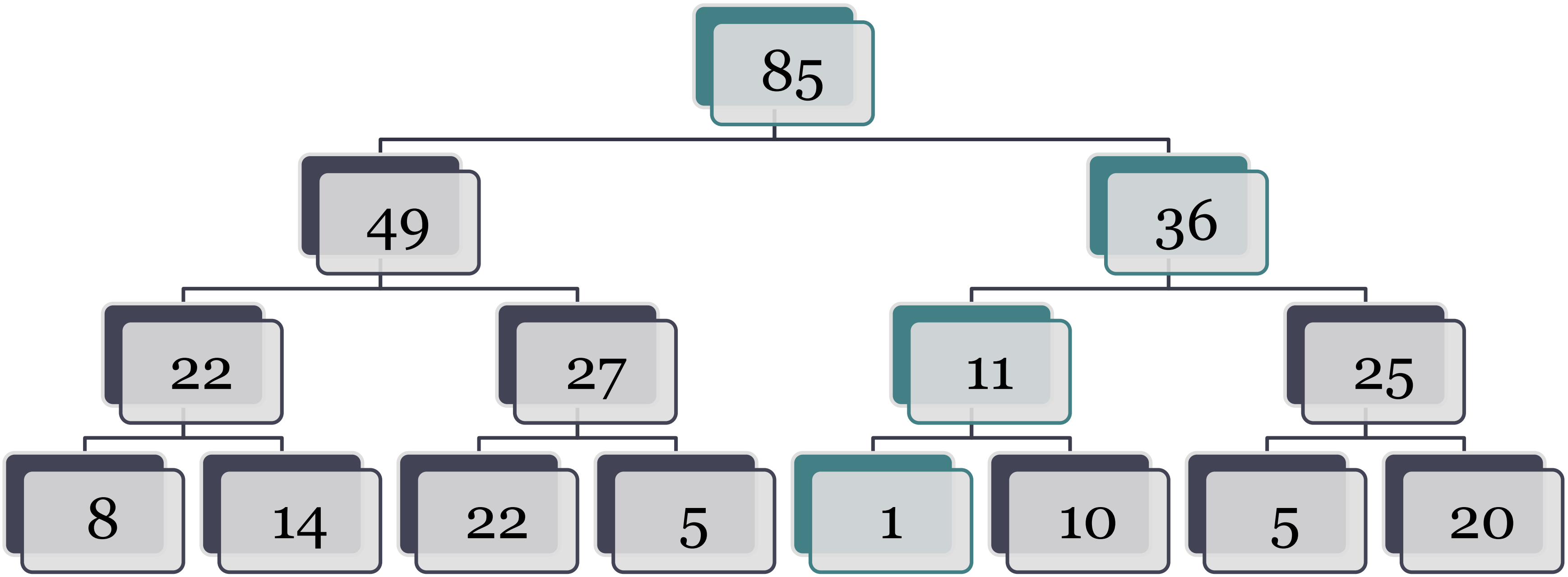
Point Updates

- 49 to 36



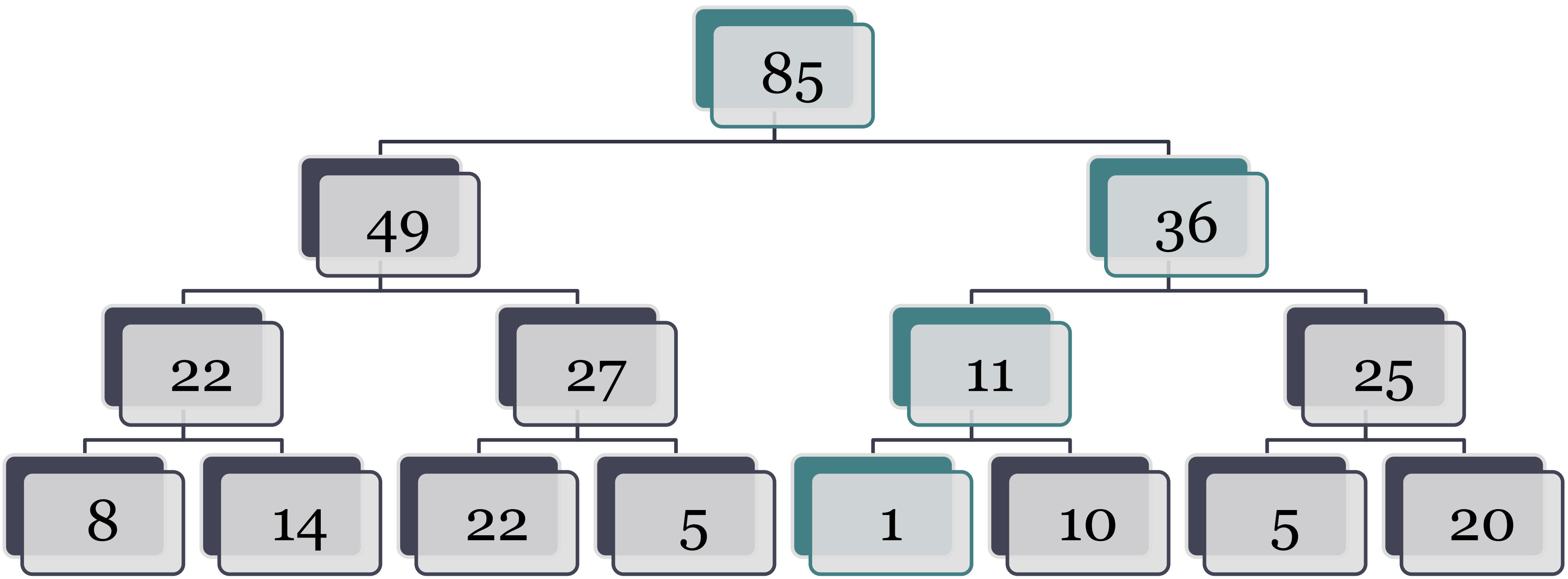
Point Updates

- 98 to 85



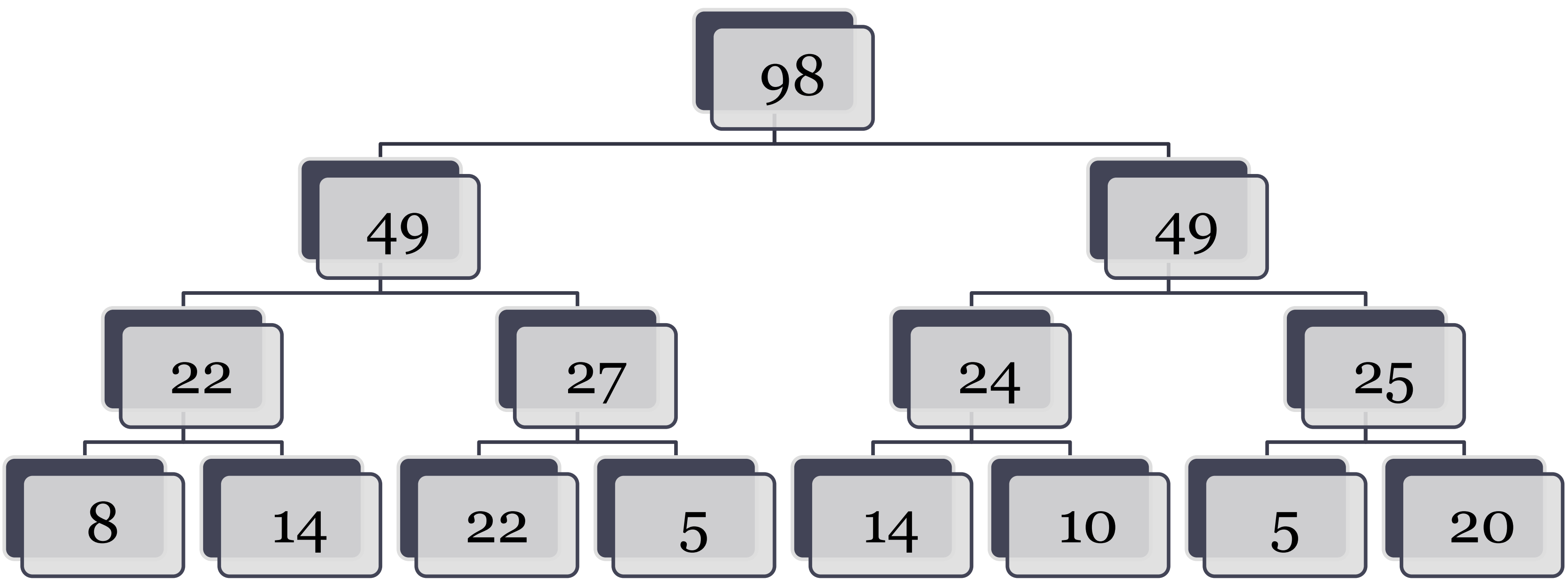
Point Updates

- Complexity?



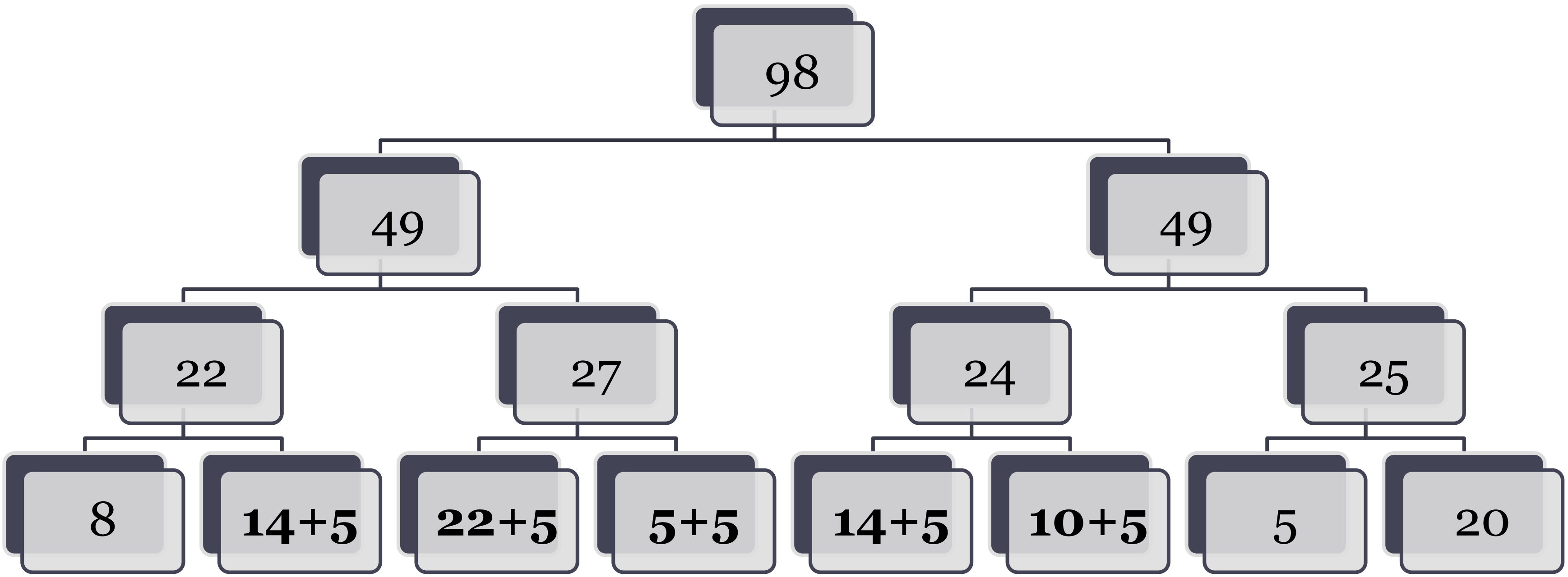
Range Updates

- We want to modify a range of numbers quickly
- Modify(delta, left, right)



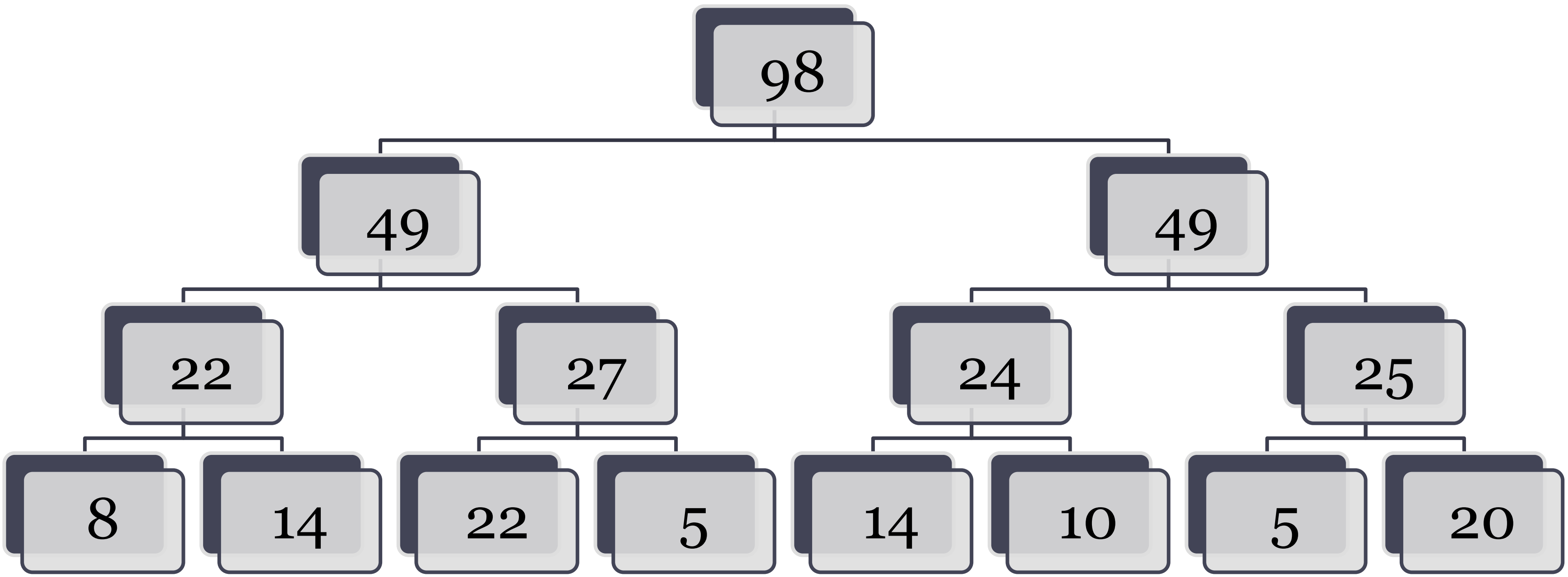
Range Updates

- We want to modify a range of numbers quickly
- Modify(5, 1, 5)



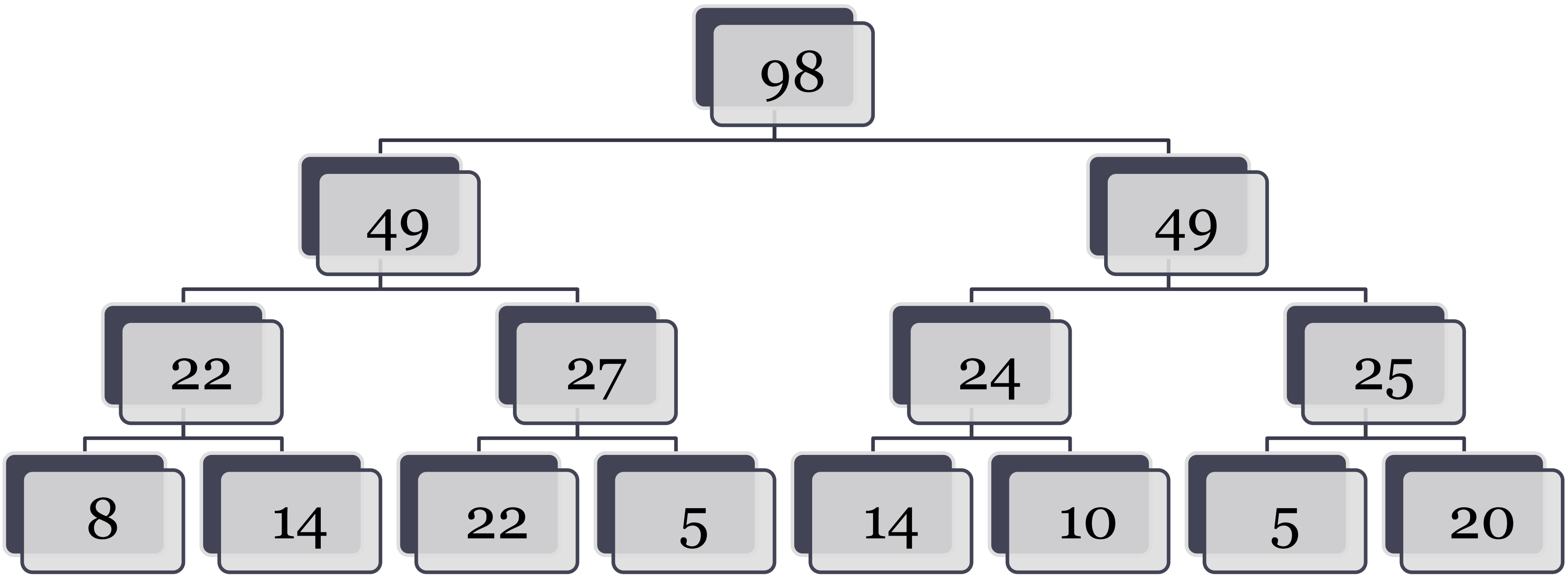
Range Updates

- Naïve method
- Complexity?



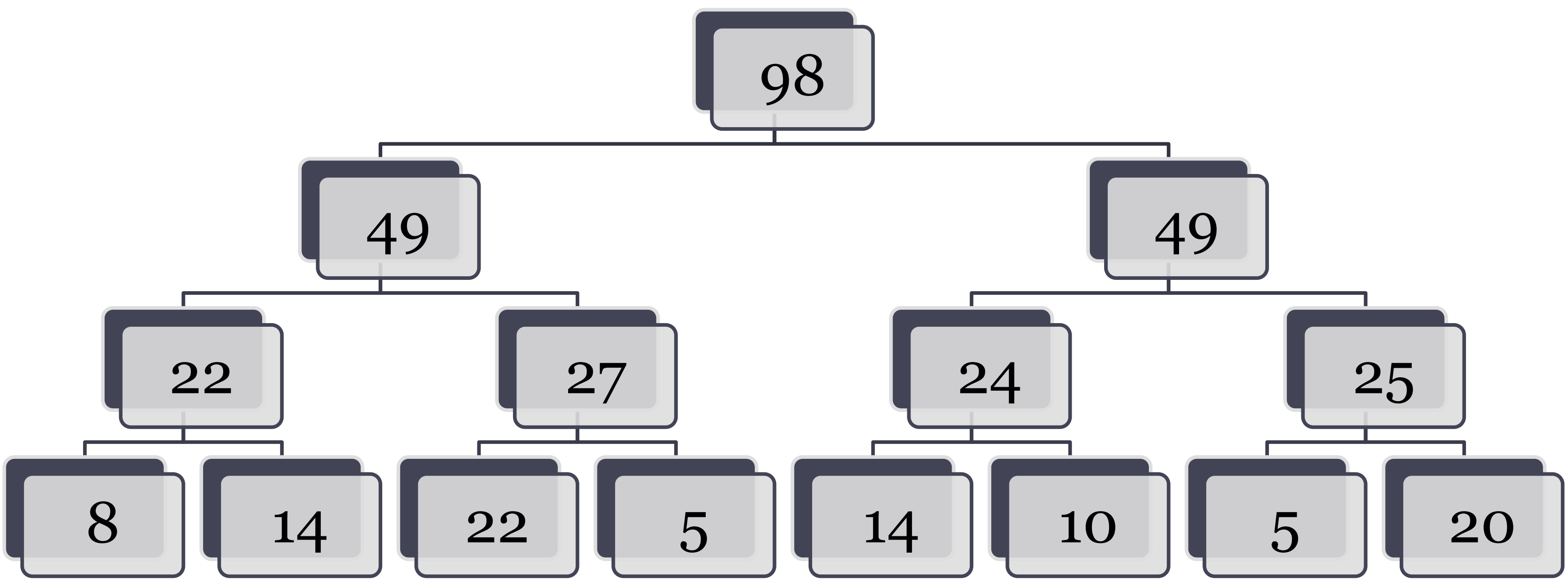
Range Updates

- Be LAZY! Update minimum nodes to represent the range



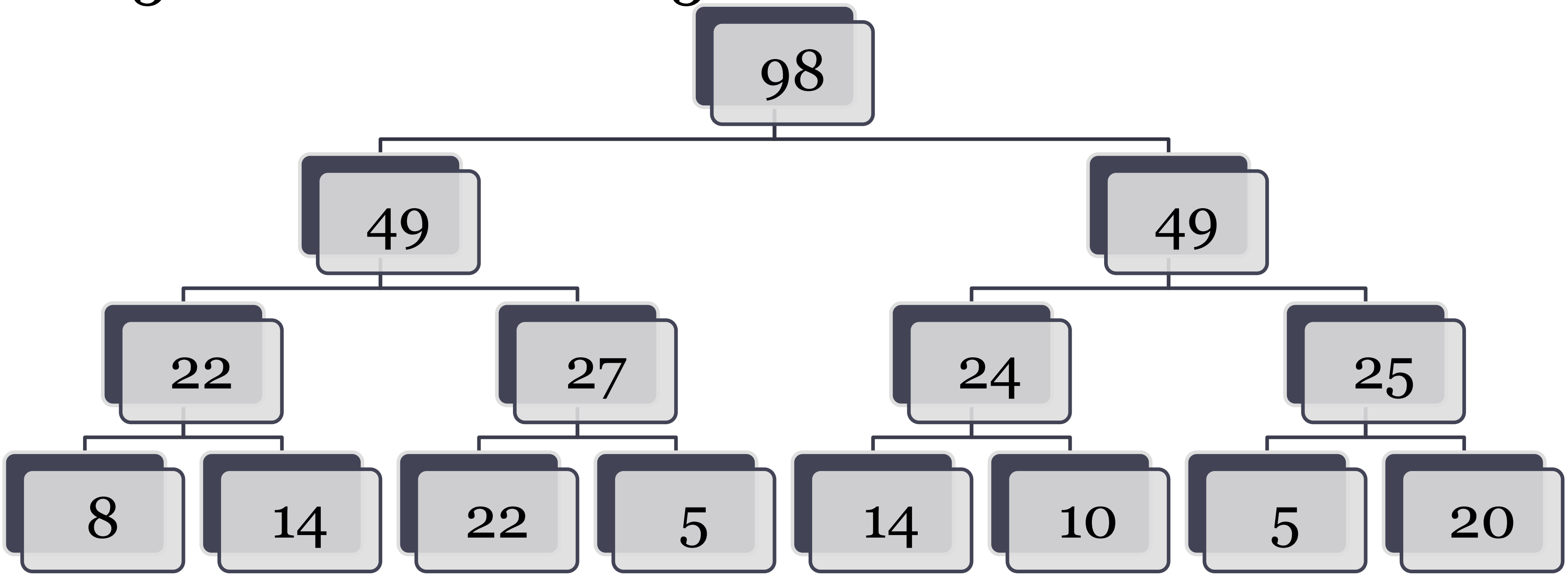
Range Updates

- Defer updating nodes underneath till when you need them



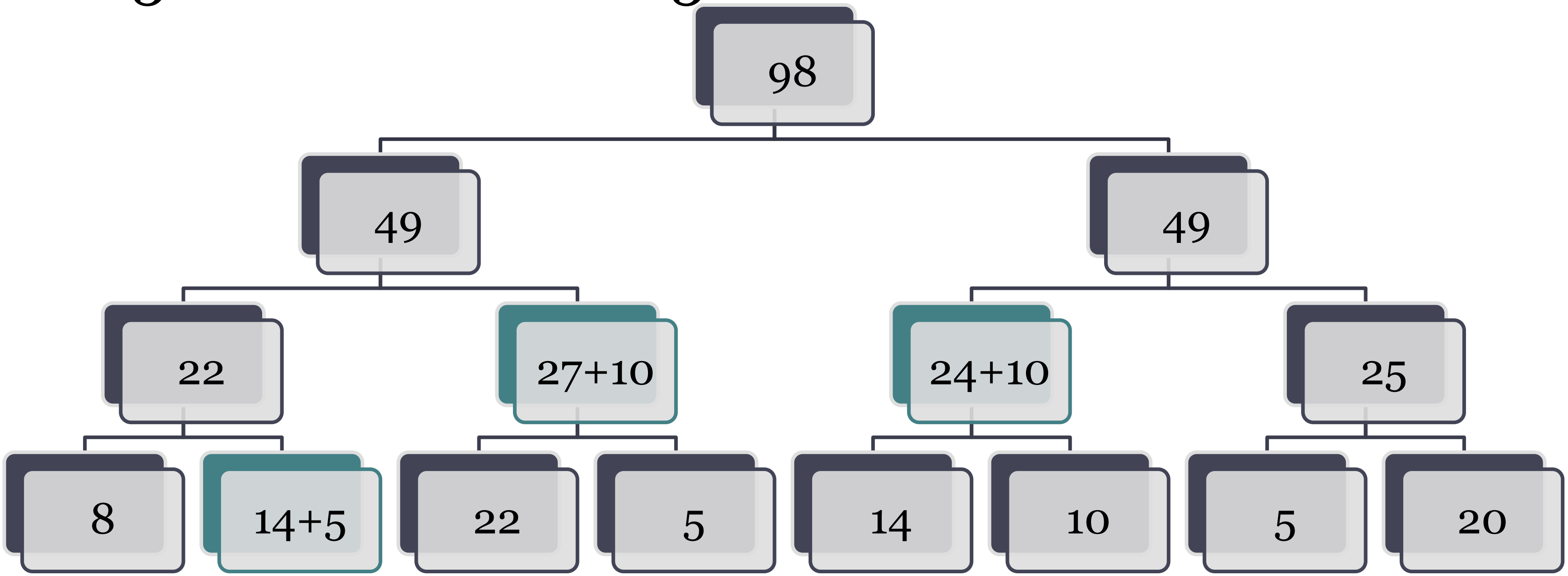
Range Updates

- Only update the set of nodes you would use in a query
- +5 to all nodes in 1-5



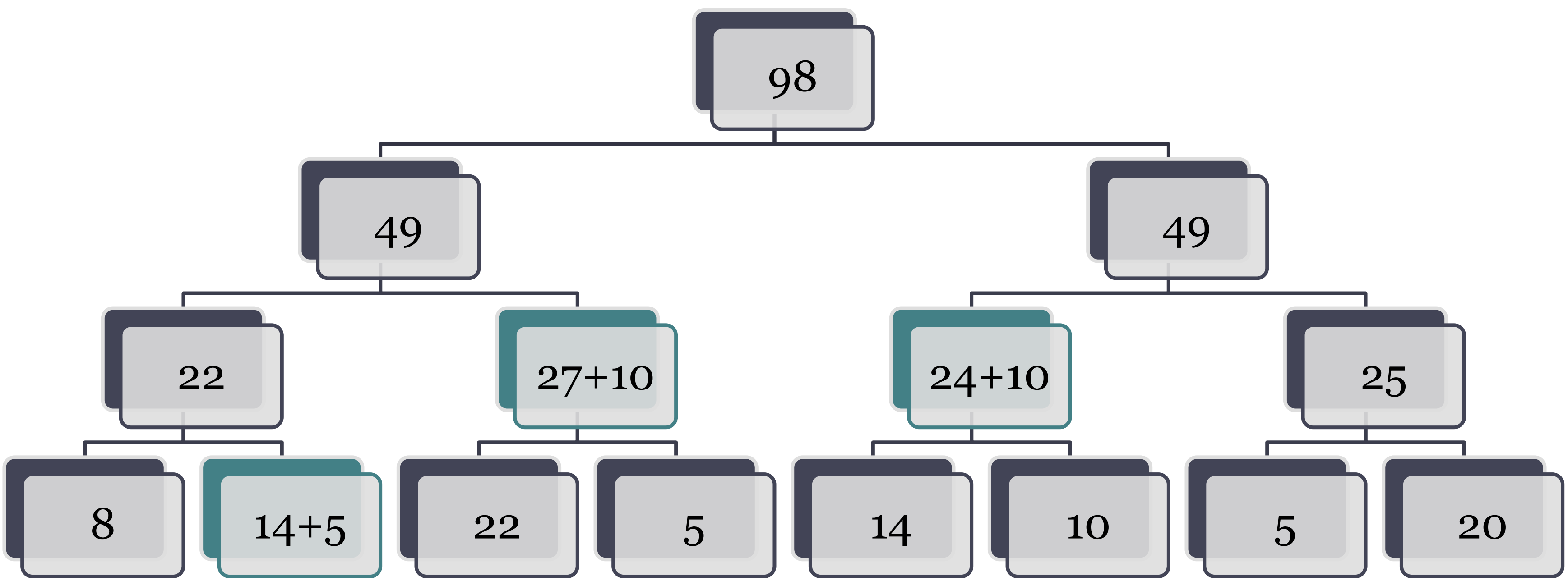
Range Updates

- Only update the set of nodes you would use in a query
- +5 to all nodes in 1-5



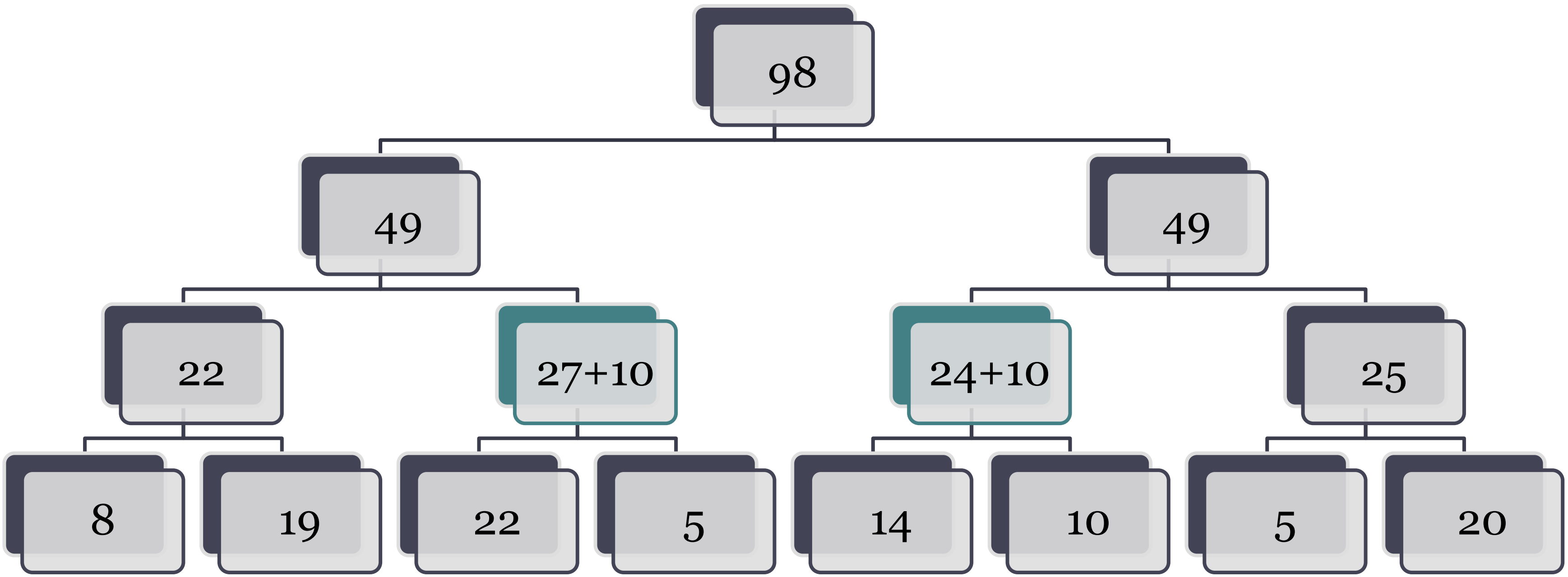
Range Updates

- To update a leaf node, just change the value.



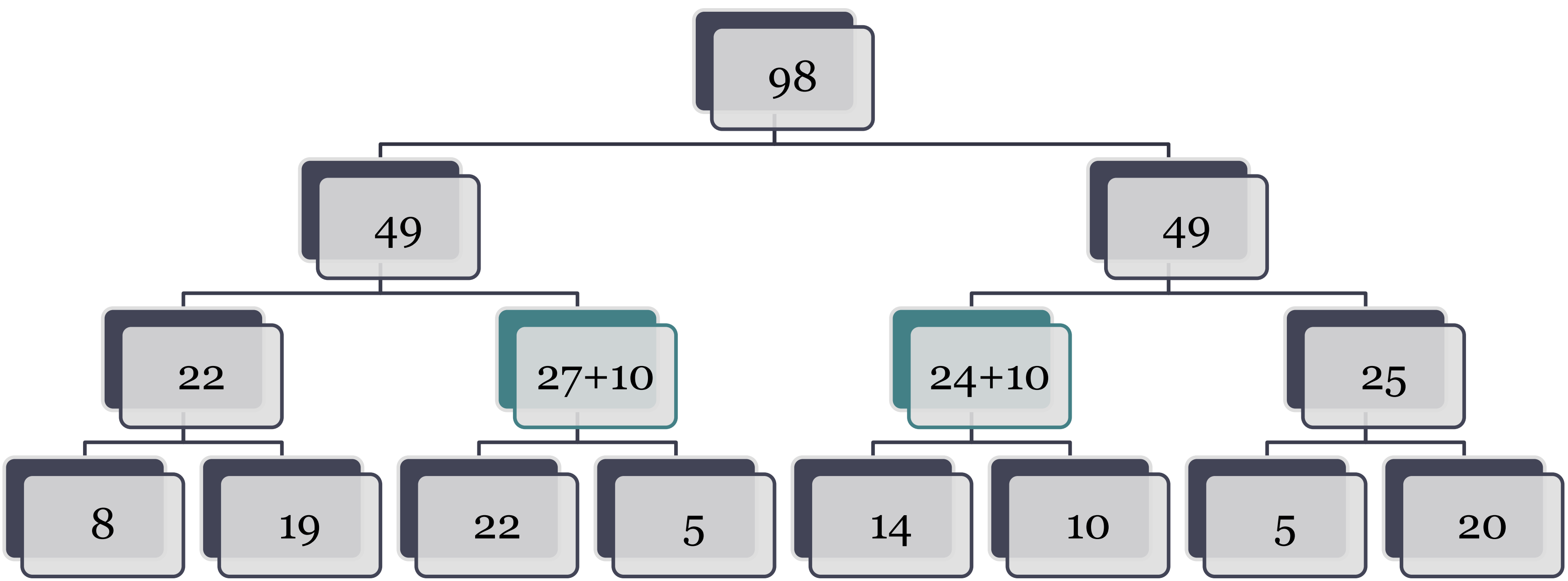
Range Updates

- To update a leaf node, just change the value.



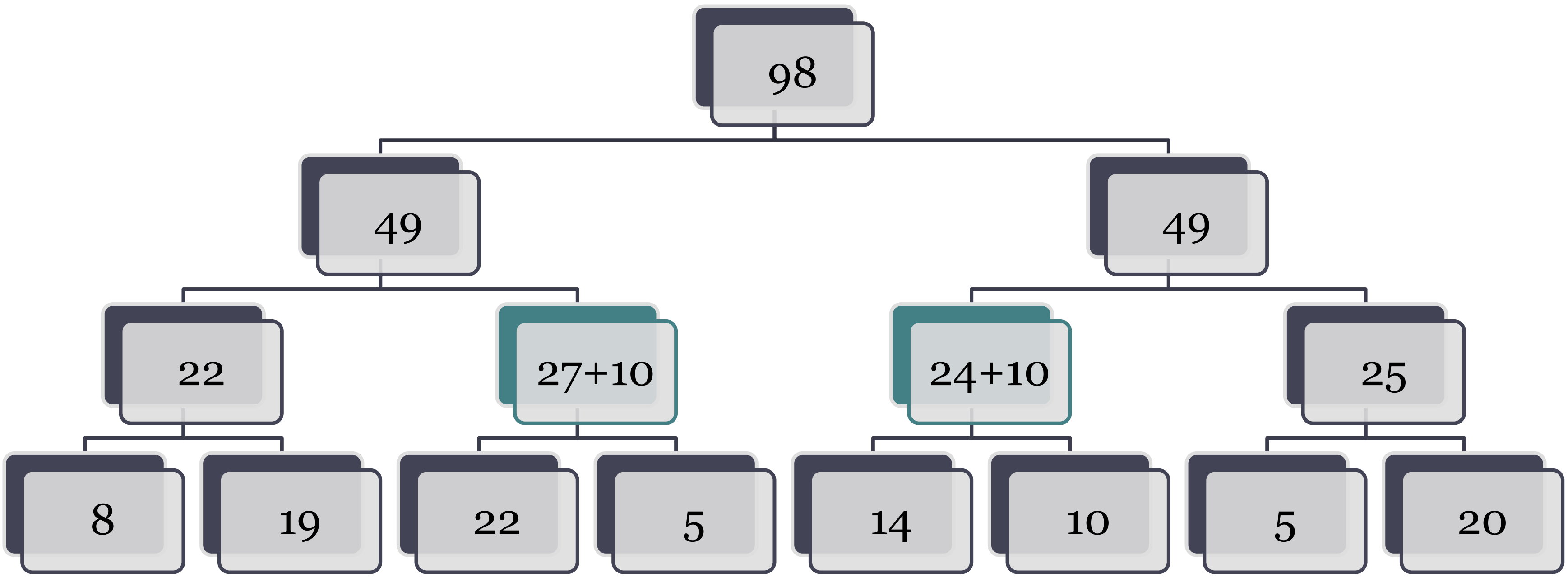
Range Updates

- Otherwise, mark it as lazy, and store the difference in an array



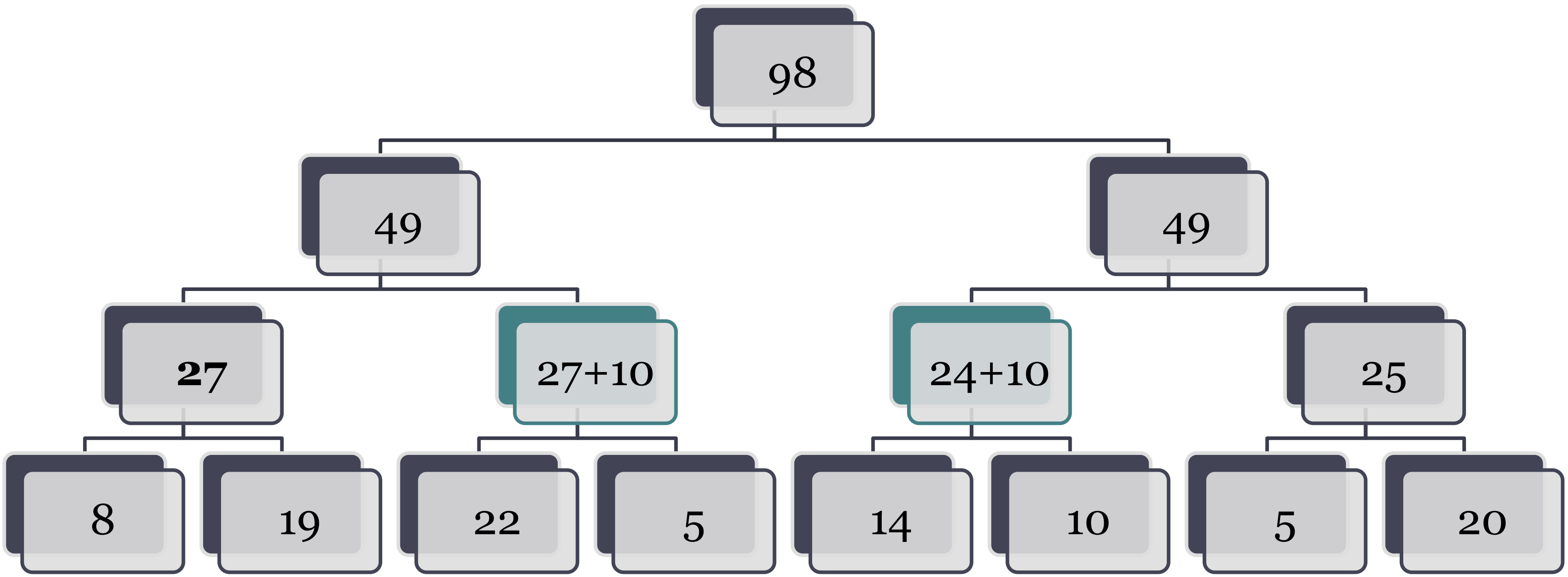
Range Updates

- Of course, update the parents as usual



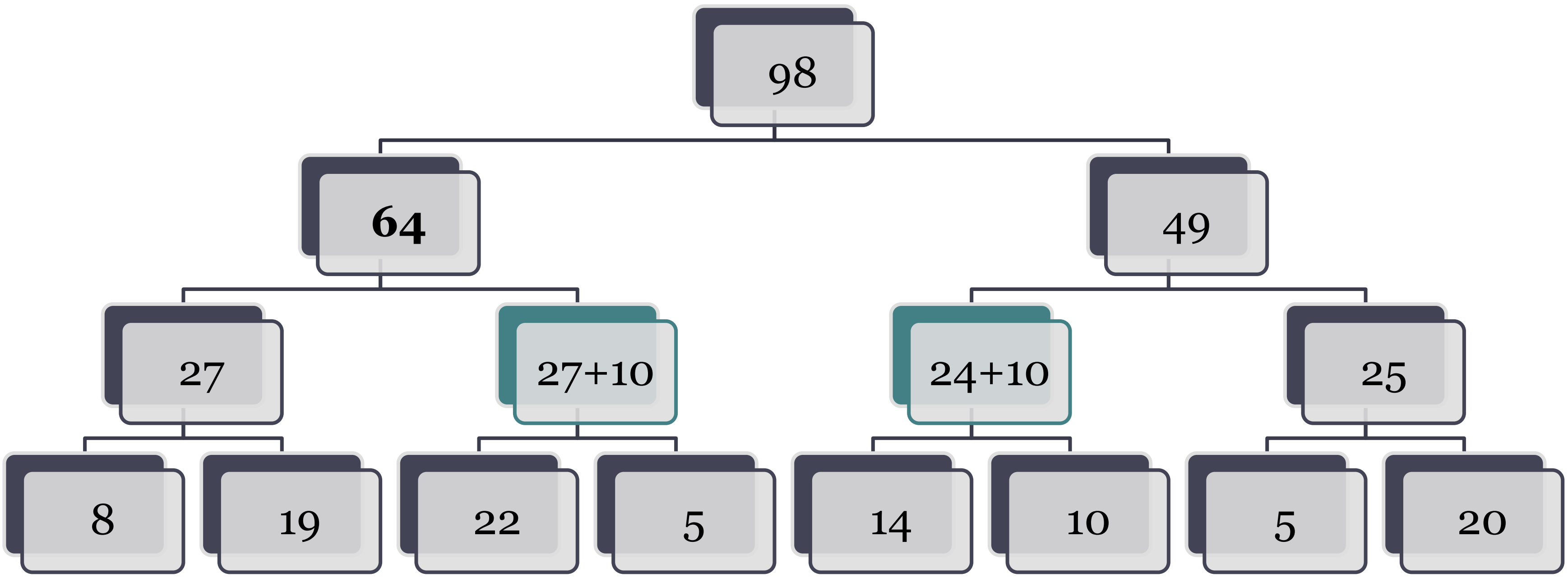
Range Updates

- Of course, update the parents as usual



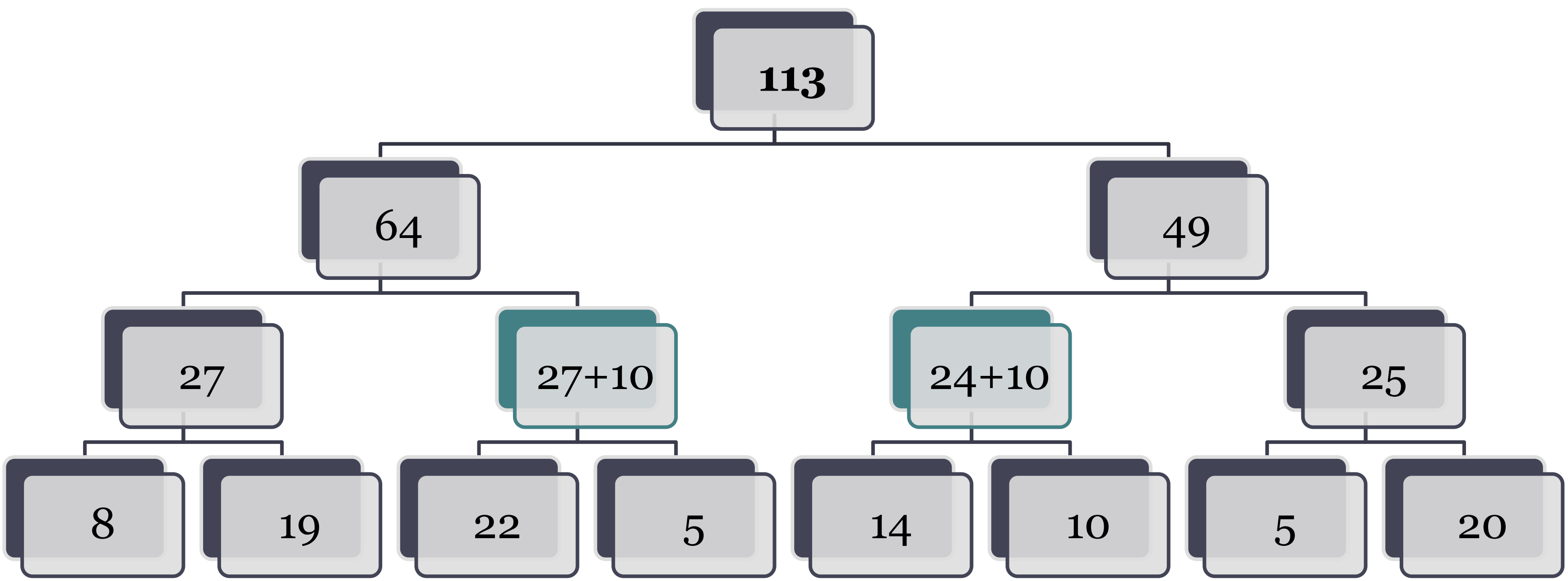
Range Updates

- Of course, update the parents as usual



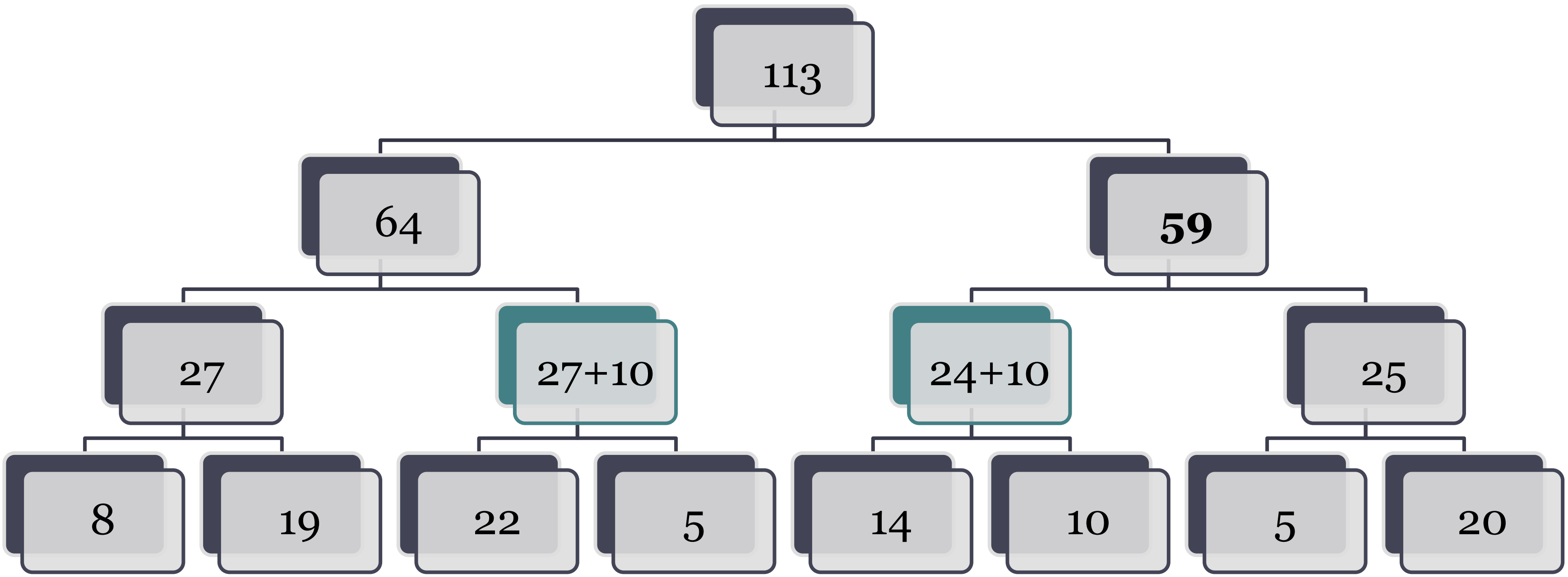
Range Updates

- Of course, update the parents as usual



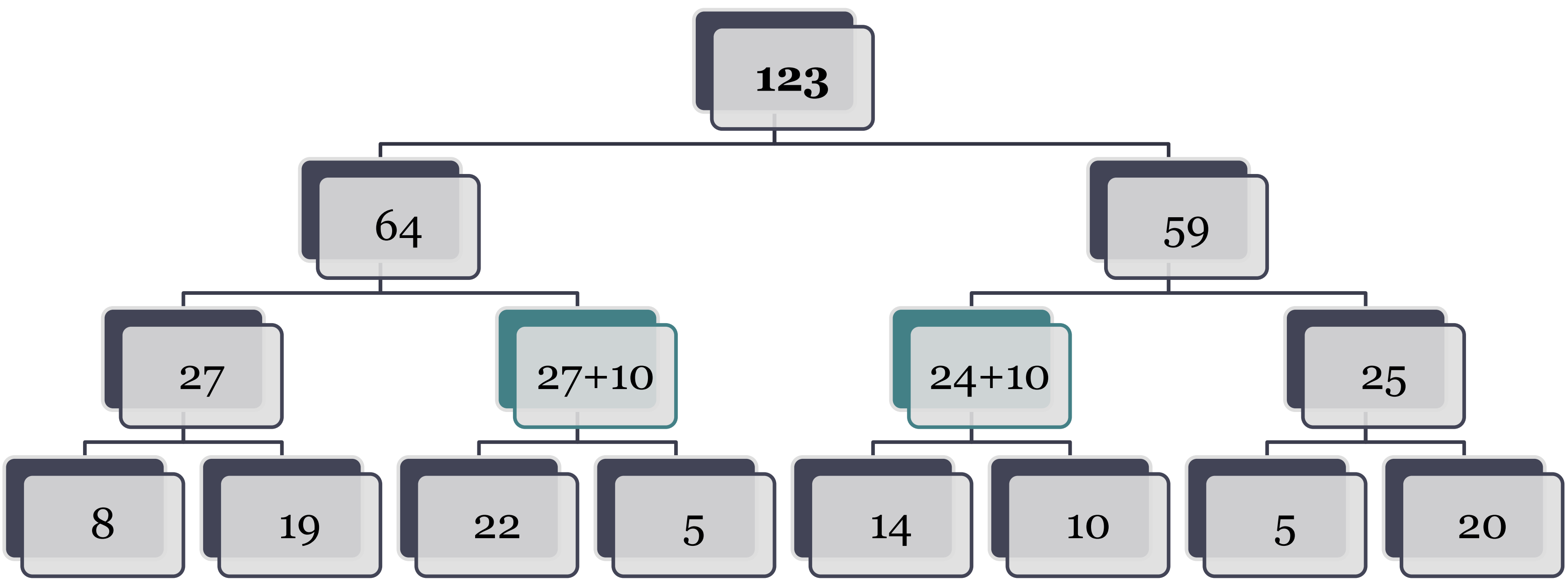
Range Updates

- Of course, update the parents as usual



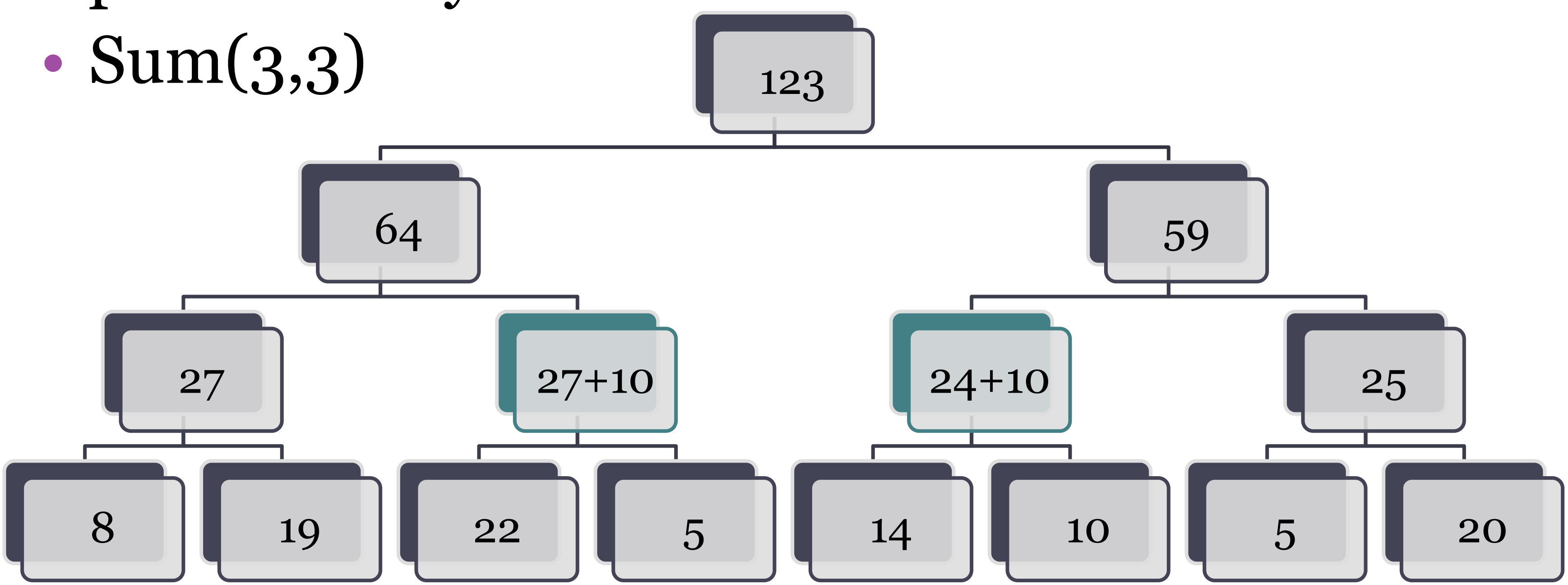
Range Updates

- Of course, update the parents as usual



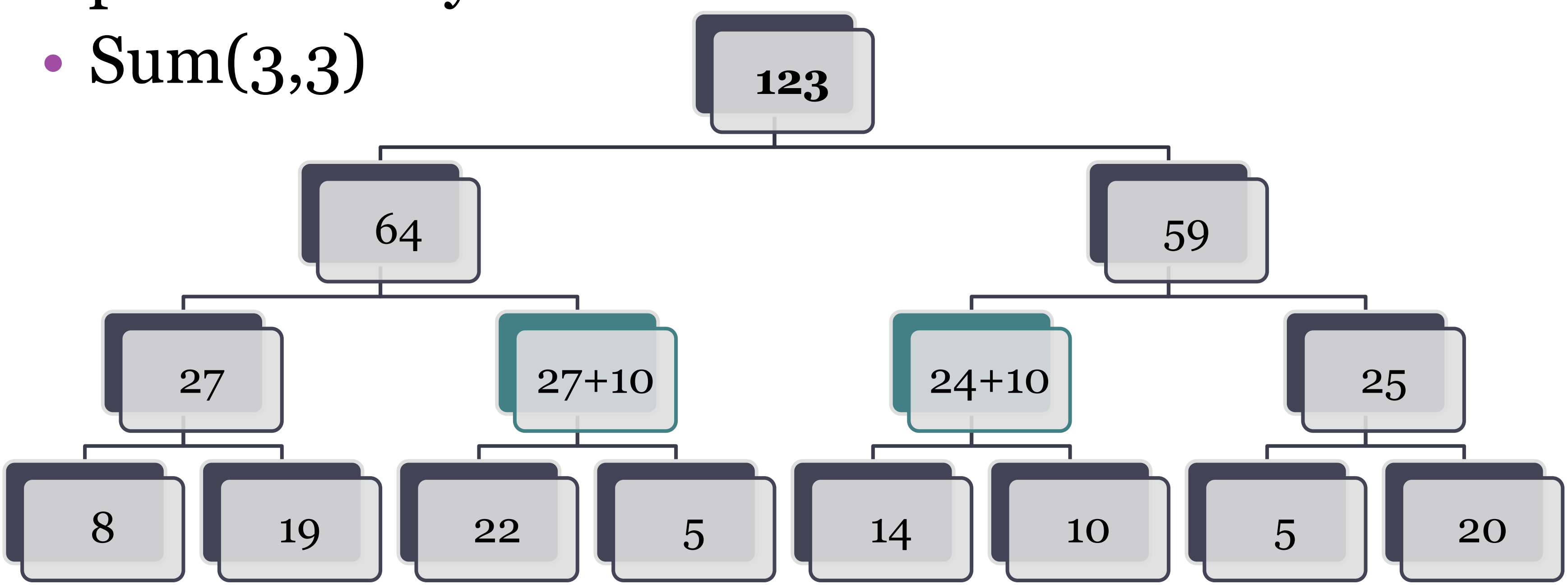
Range Updates

- Any time we need to access the child of a lazily updated node, you update the node properly and push the lazy down.
- Sum(3,3)



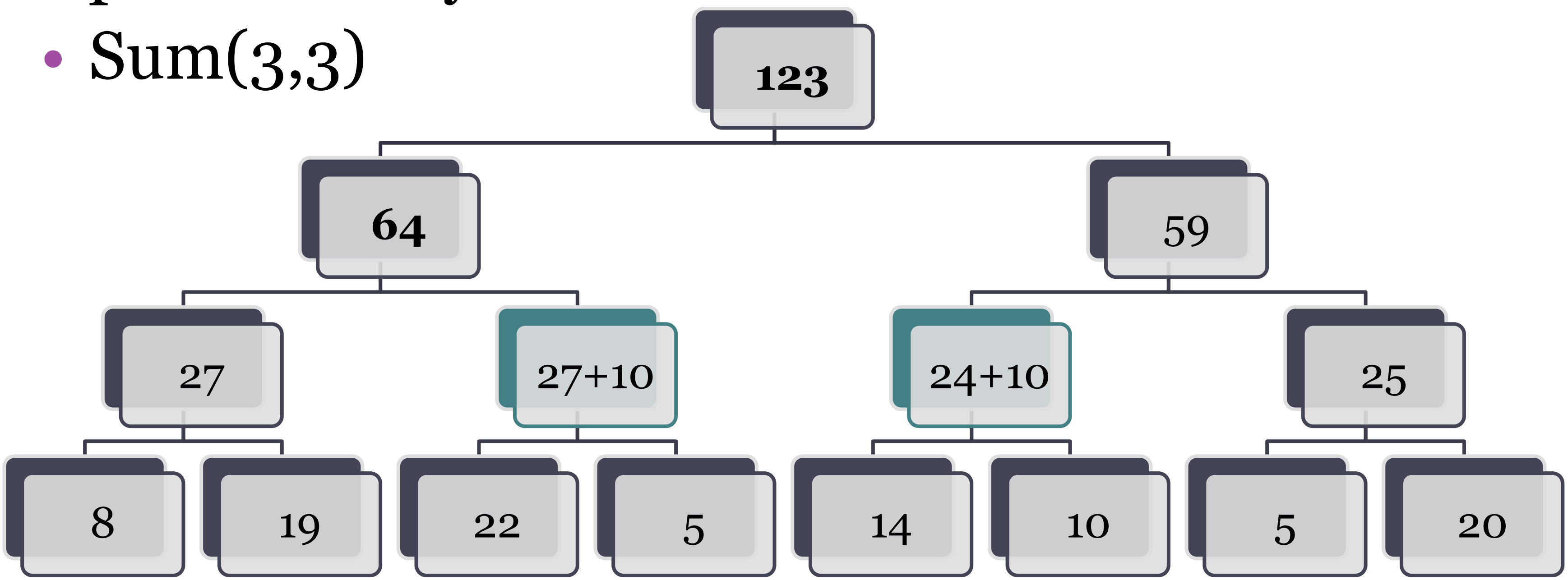
Range Updates

- Any time we need to access the child of a lazily updated node, you update the node properly and push the lazy down.
- Sum(3,3)



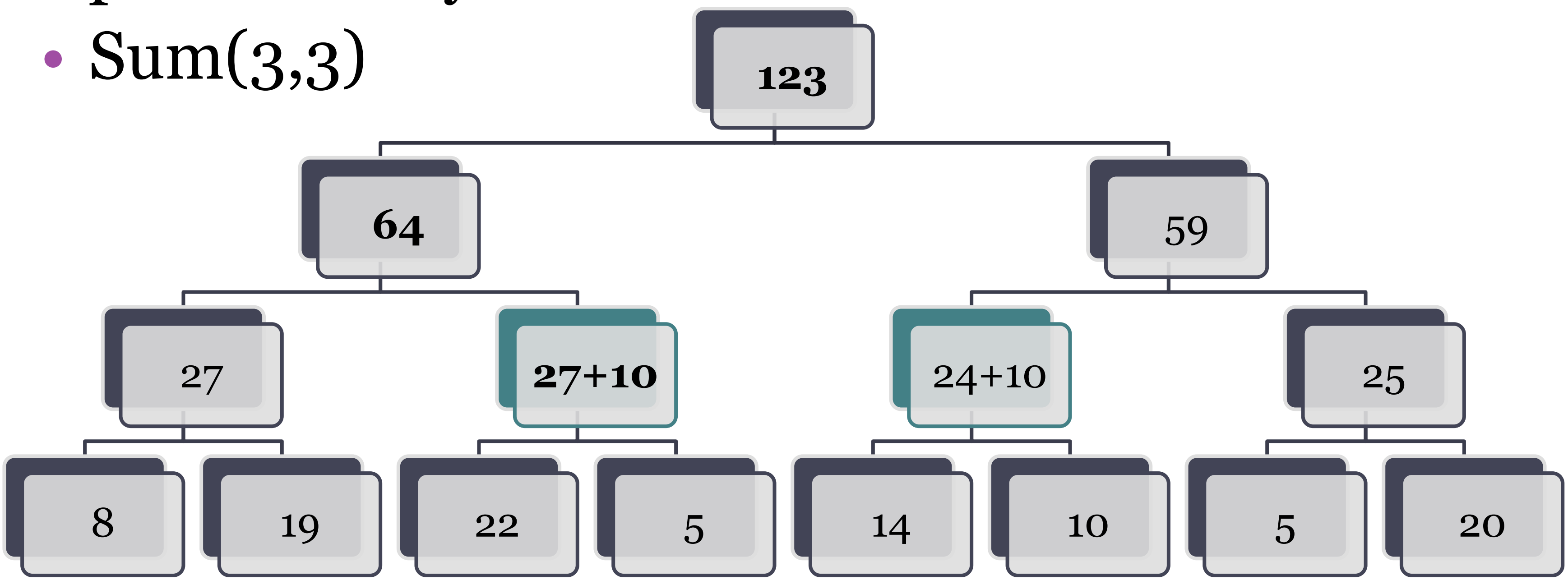
Range Updates

- Any time we need to access the child of a lazily updated node, you update the node properly and push the lazy down.
- Sum(3,3)



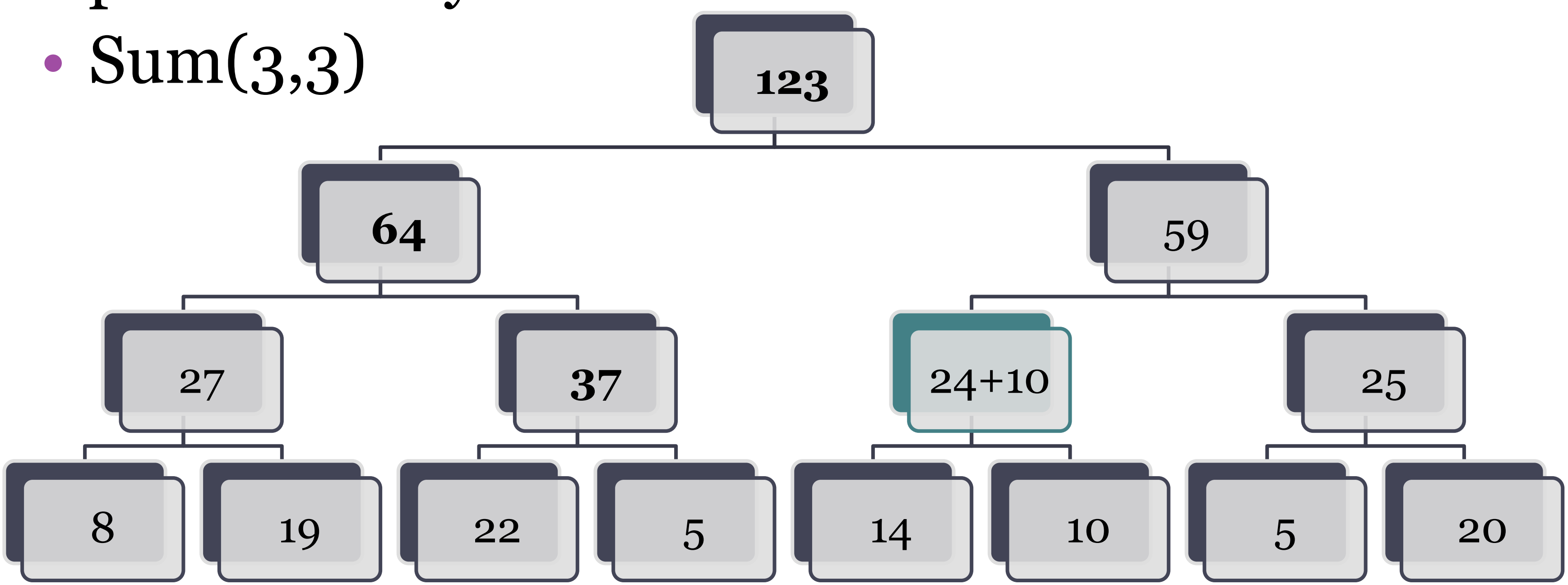
Range Updates

- Any time we need to access the child of a lazily updated node, you update the node properly and push the lazy down.
- Sum(3,3)



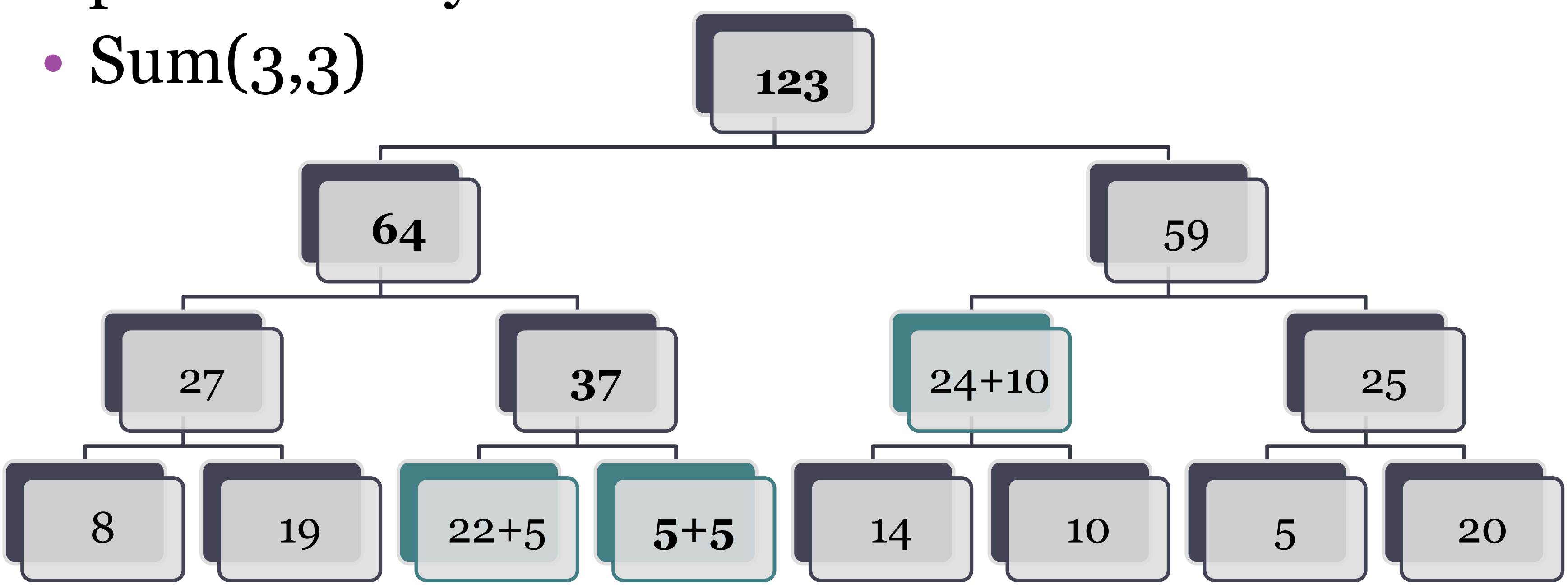
Range Updates

- Any time we need to access the child of a lazily updated node, you update the node properly and push the lazy down.
- Sum(3,3)



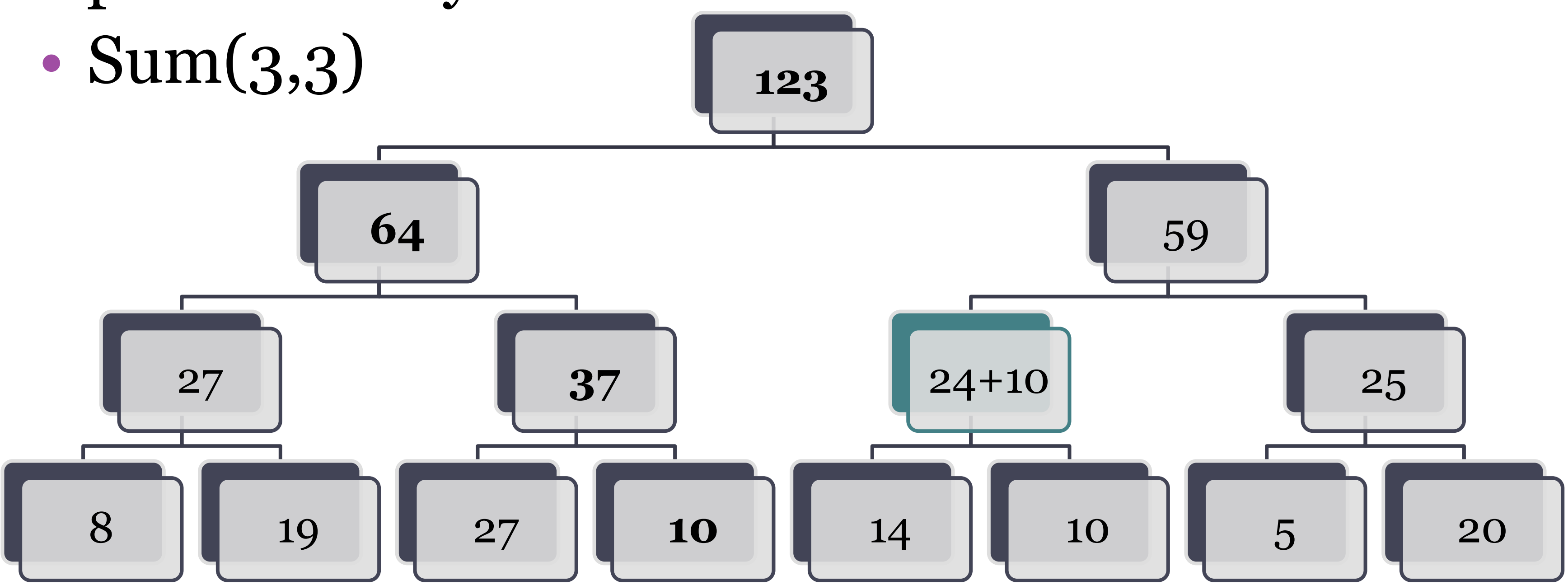
Range Updates

- Any time we need to access the child of a lazily updated node, you update the node properly and push the lazy down.
- Sum(3,3)



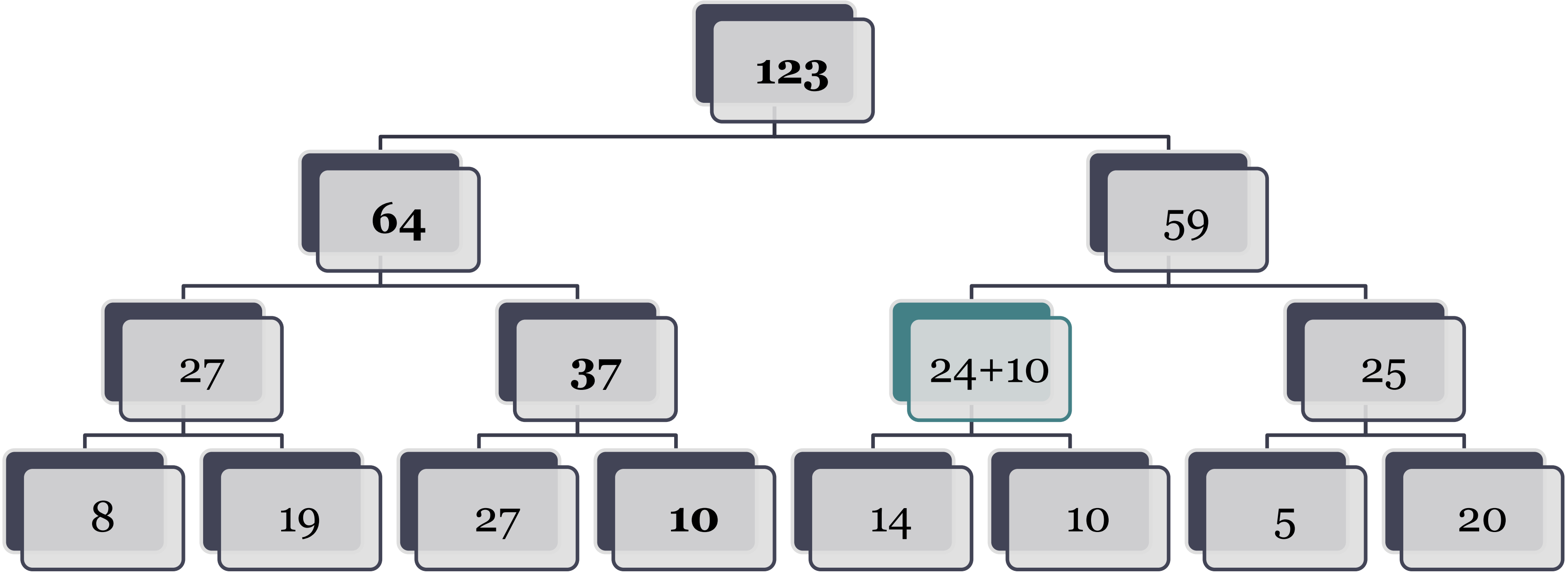
Range Updates

- Any time we need to access the child of a lazily updated node, you update the node properly and push the lazy down.
- Sum(3,3)



Range Updates

- Complexity of query with lazy updates?

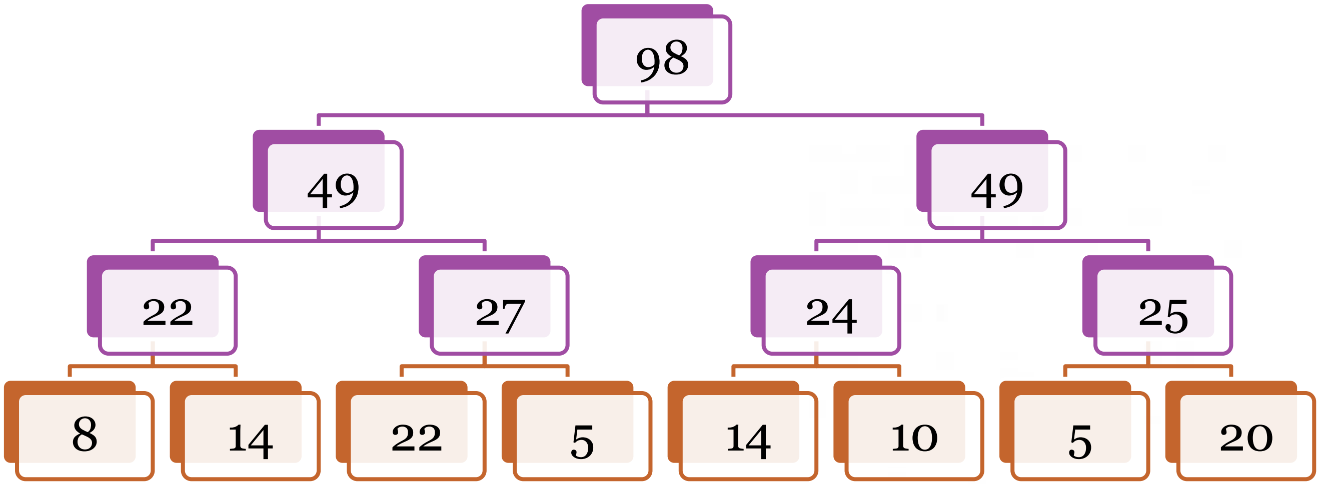


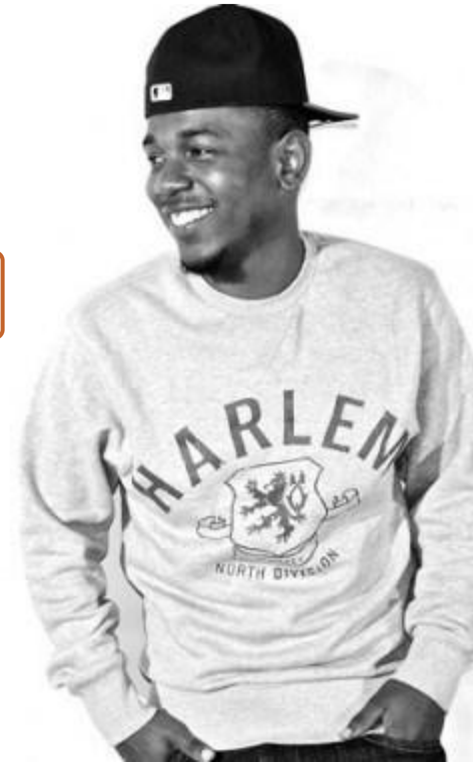
Conclusion

- Build time: O(n)
- Range Query: **O(log(n))**
- Point Update: **O(log(n))**
- Range Update : **O(log(n))**

Conclusion

- Segment trees are the perfect place for shade.





Bibliography

- Lazy updates: http://wcipeg.com/wiki/Segment_tree
- The rest of it: Competitive Programming 3