



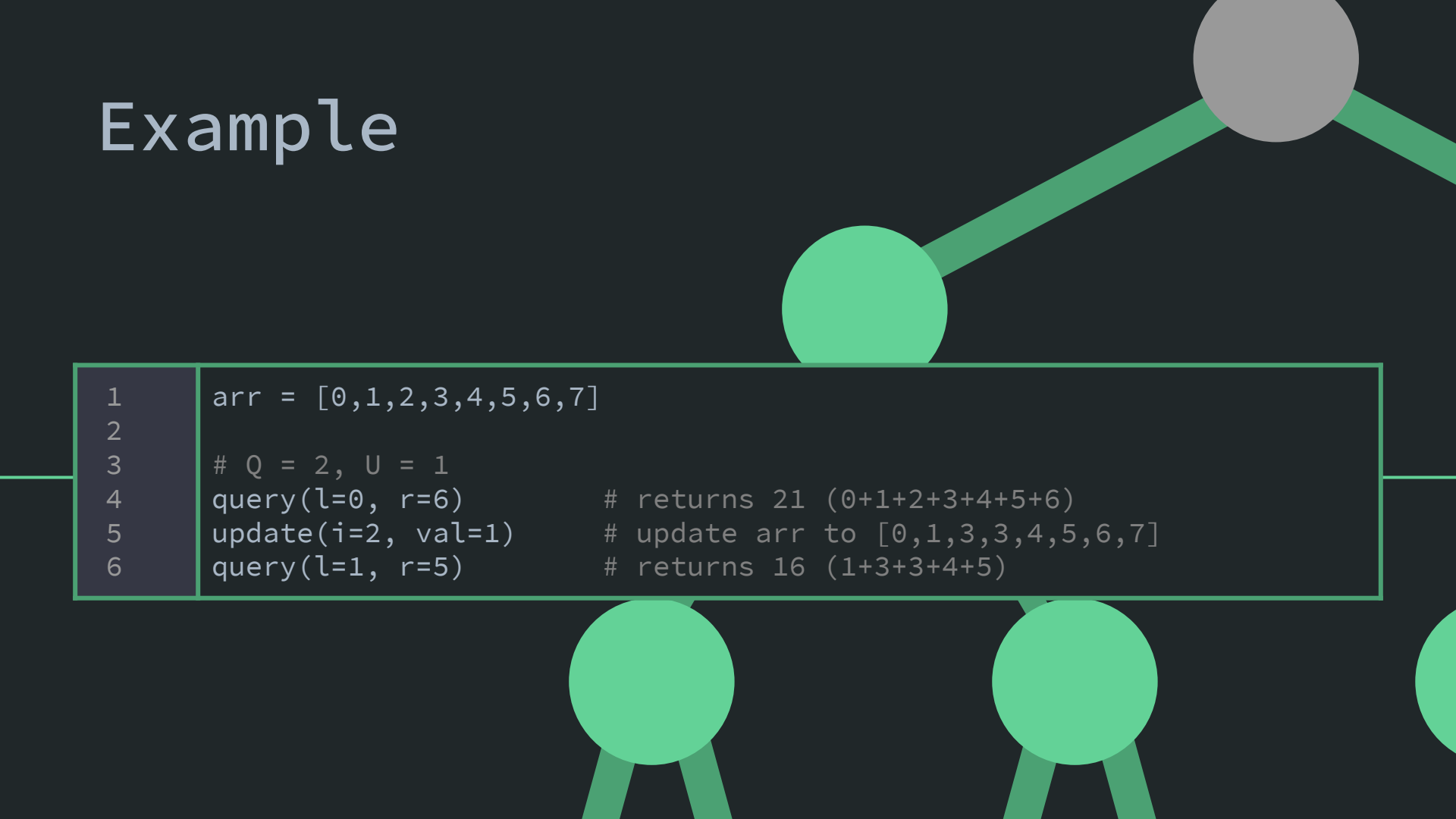
Let's consider a problem...

Given an
array
 $arr[0 \dots n-1]$,
perform

1. U # updates that increase the value at index i by val
2. Q # queries to find the sum over the interval $[l, r]$, where $l < r$ and are indices of arr

Queries and updates can be performed in any order

Example



```
1 arr = [0,1,2,3,4,5,6,7]
2
3 # Q = 2, U = 1
4 query(l=0, r=6)      # returns 21 (0+1+2+3+4+5+6)
5 update(i=2, val=1)   # update arr to [0,1,3,3,4,5,6,7]
6 query(l=1, r=5)      # returns 16 (1+3+3+4+5)
```

A Prefix Sum Array

1. For updates, apply

$$\text{arr}[i] \ += \ \text{val}$$

2. If a query comes after an update, process the array sum of *arr* so that
3. For each query, sum over the range $[l, r]$ could be found by

$$\text{arraySum}[r] - \text{arraySum}[l-1]$$

4. Optimal for large groups updates and queries

Implementation

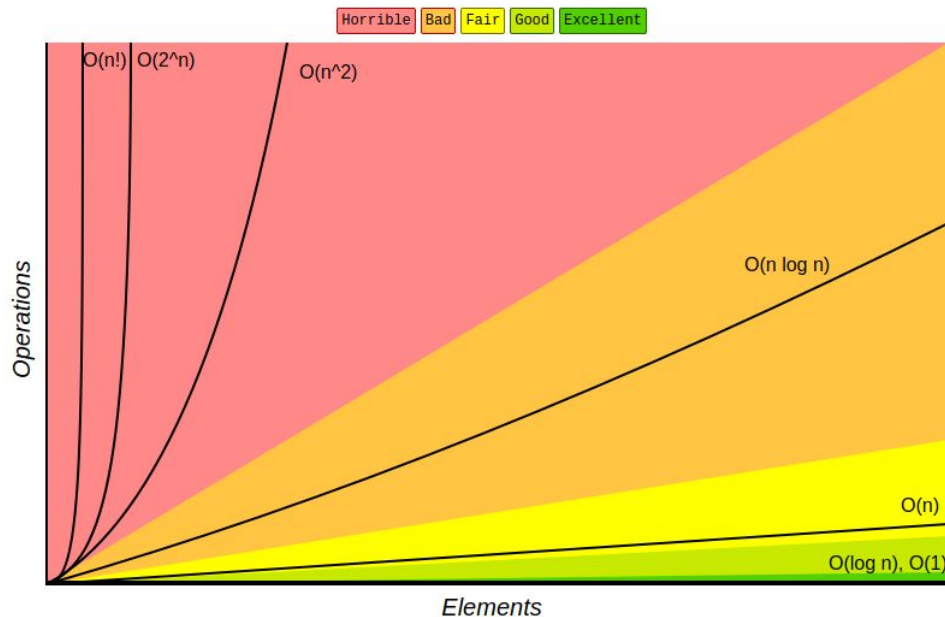
```
1  arr = [0,1,2,3,4,5,6,7]
2  arraySum = []*len(arr)
3
4  def update (i, val):
5      arr[i] += val;
6
7  # Called if a query comes after an update
8  def buildArraySum():
9      arraySum = arr[0]
10     for i in range(1, len(arr)):
11         arraySum = arraySum[i-1]+arr[i]
12
13     # Return the sum over the range
14     def query (l, r):
15         return arraySum[r]-arraySum[l-1]
16
```

The Problem

1. Slow with consecutive updates and queries
2. Processing array sum takes $O(n)$ time
3. Overall time complexity: $O(Q \cdot n)$

Large Array + Millions of Queries
= not good

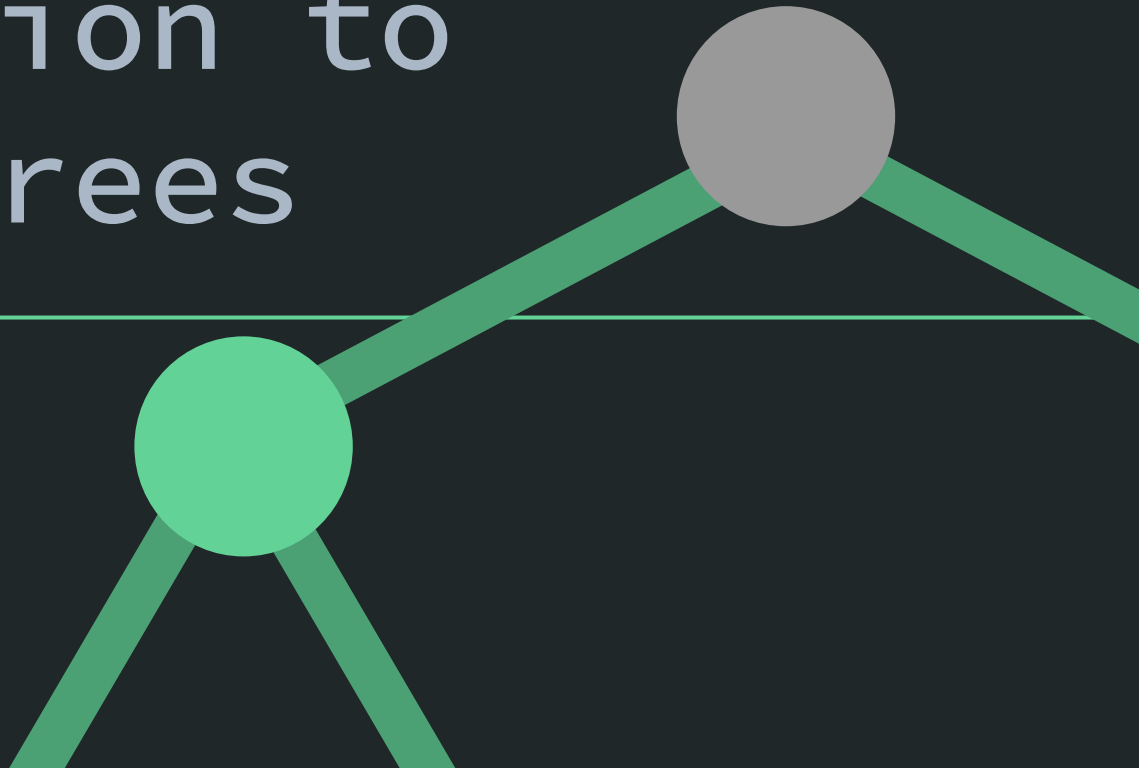
Can we do it better? Yes!



Introduction to Segment Trees

Felix Fong

Note: view slides in presentation mode



Segment Tree

- A data structure that stores information over segments of an array
- $O(\log_2 n)$ complexity for updates and range queries

Pros

- Same uses as sum arrays
- Supports irreversible operations
- Offers min/max, gcd/lcm, bitwise, etc. over $[l, r]$

Cons

- Takes $O(2n)$ memory
- Given large updates and few queries, sum arrays are faster
- Does not exist in the library

What is a Tree?



```
graph TD; A(( )) --- B(( )); B --- C(( )); B --- D(( ))
```

- A **hierarchical** data structure of nodes

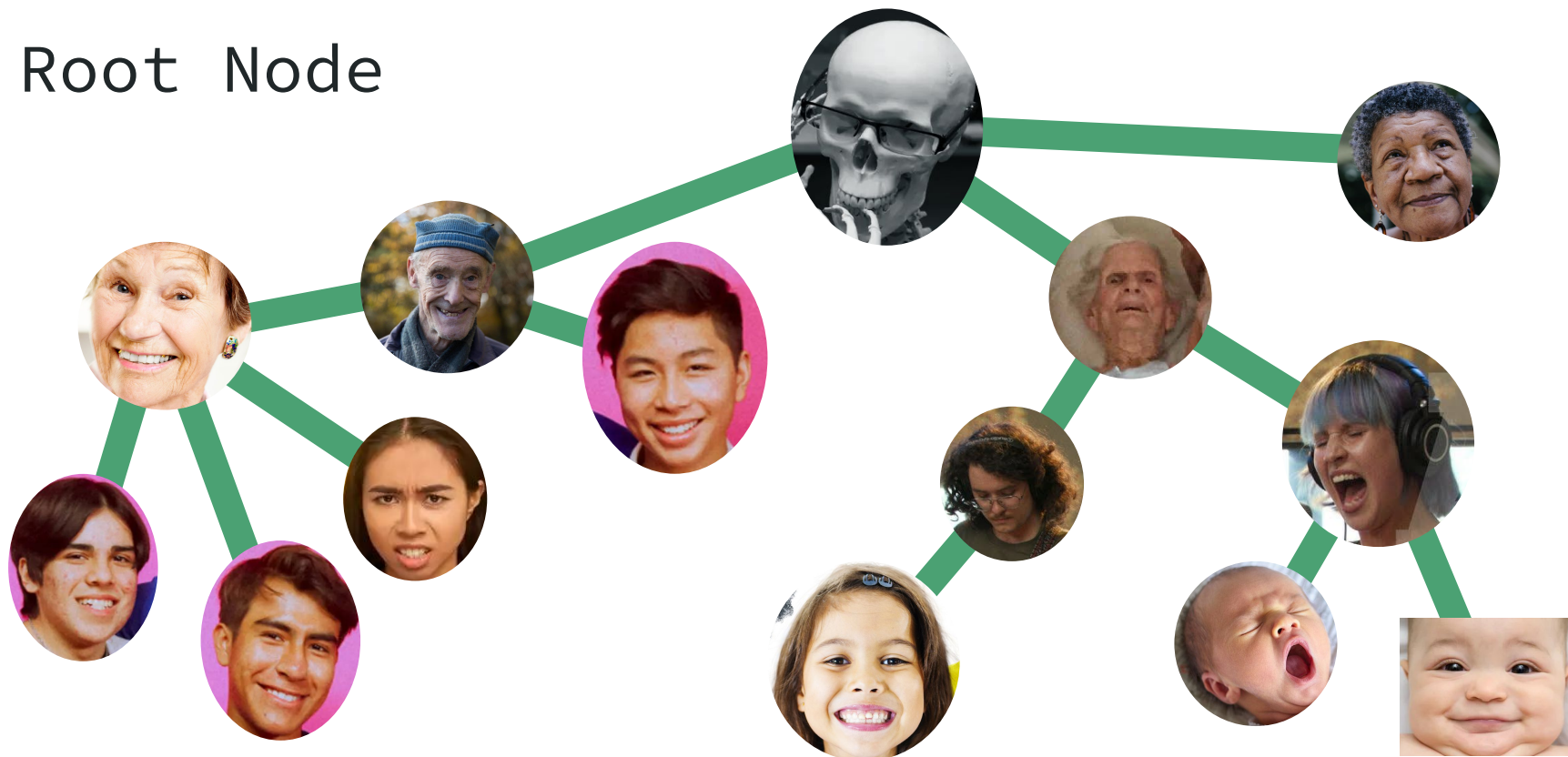
Think of a Family Tree...

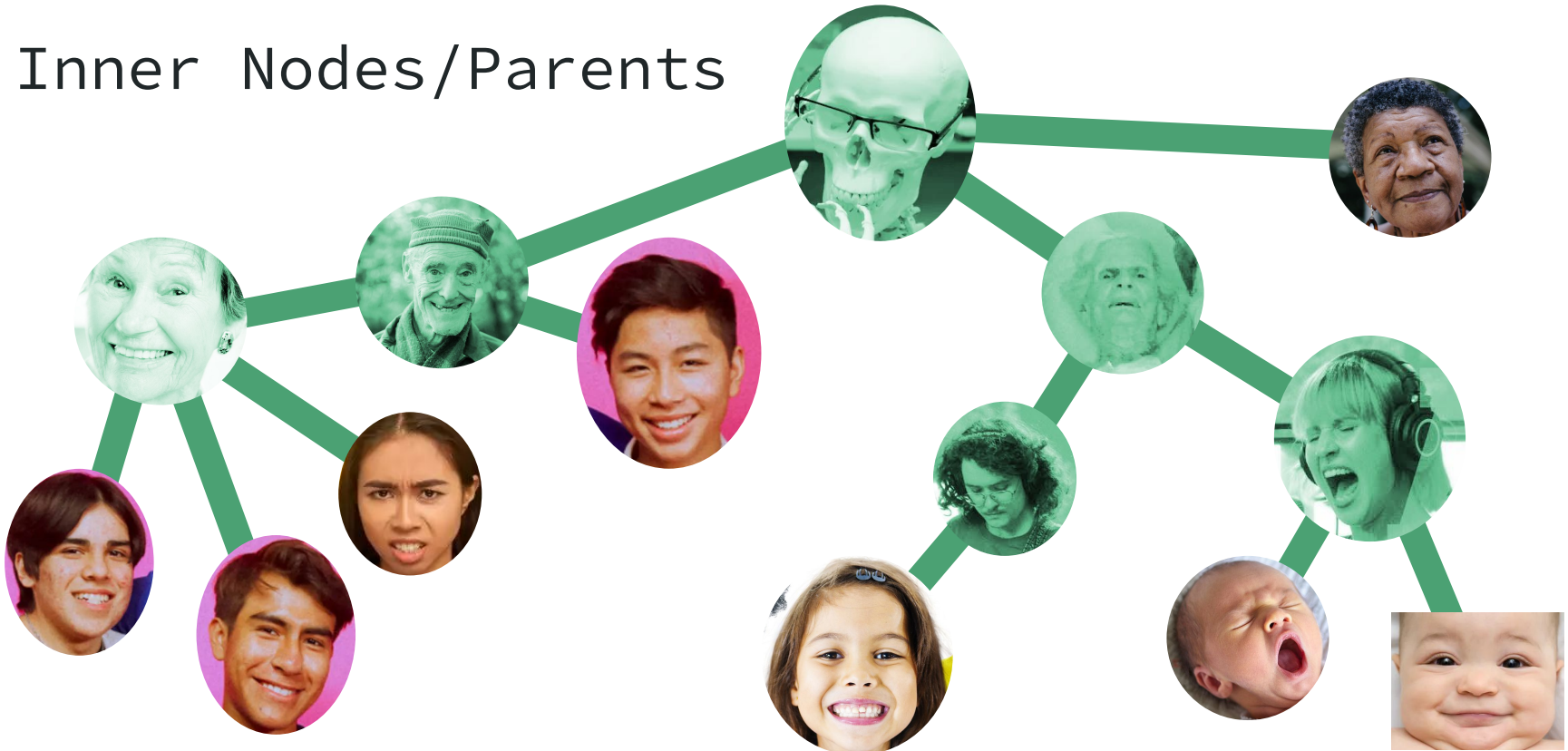
A stylized family tree diagram is visible in the background. It features a central green circle connected by thick green lines to other circles. One line goes up to a grey circle, and two lines go down to two more green circles. A horizontal green line is also present across the middle of the image.

Family Tree

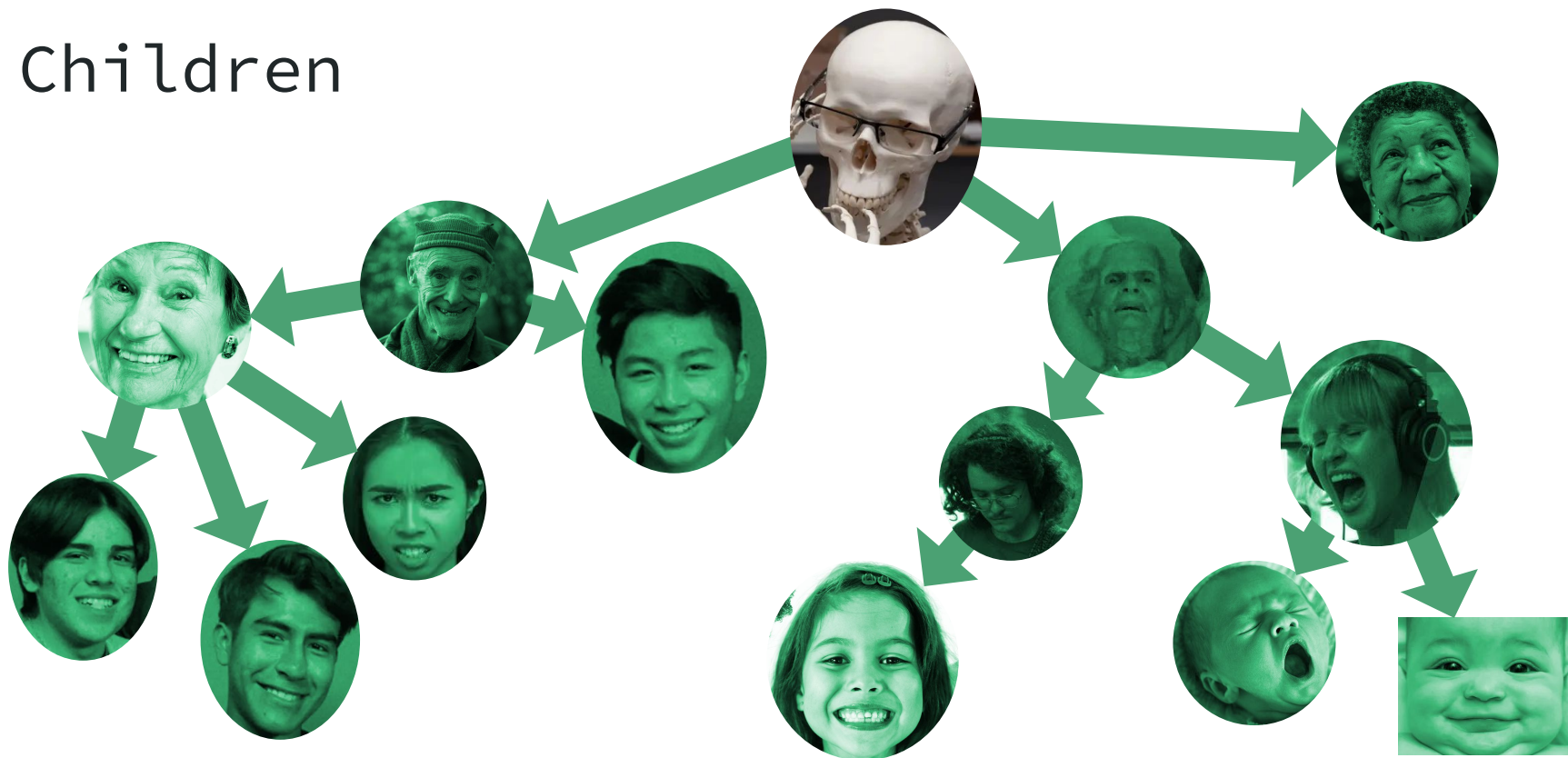


Root Node



[illegible]

Children



Leaf Nodes

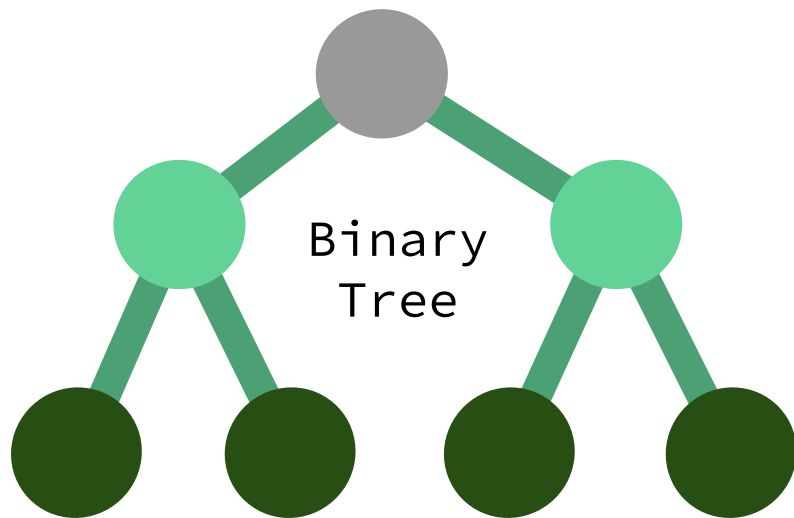




How do Segment Trees Work?

Details

- Segment Trees are **Binary Trees** (parents only allowed 2 children)
- $\# \text{ inner nodes} = \max \# \text{ leaf nodes} - 1$
- Leaf nodes contain the original array
- Each parent stores an operation's result between its children
 - Provides precomputed results over intervals of an array
- Can be stored in an array



Implementation

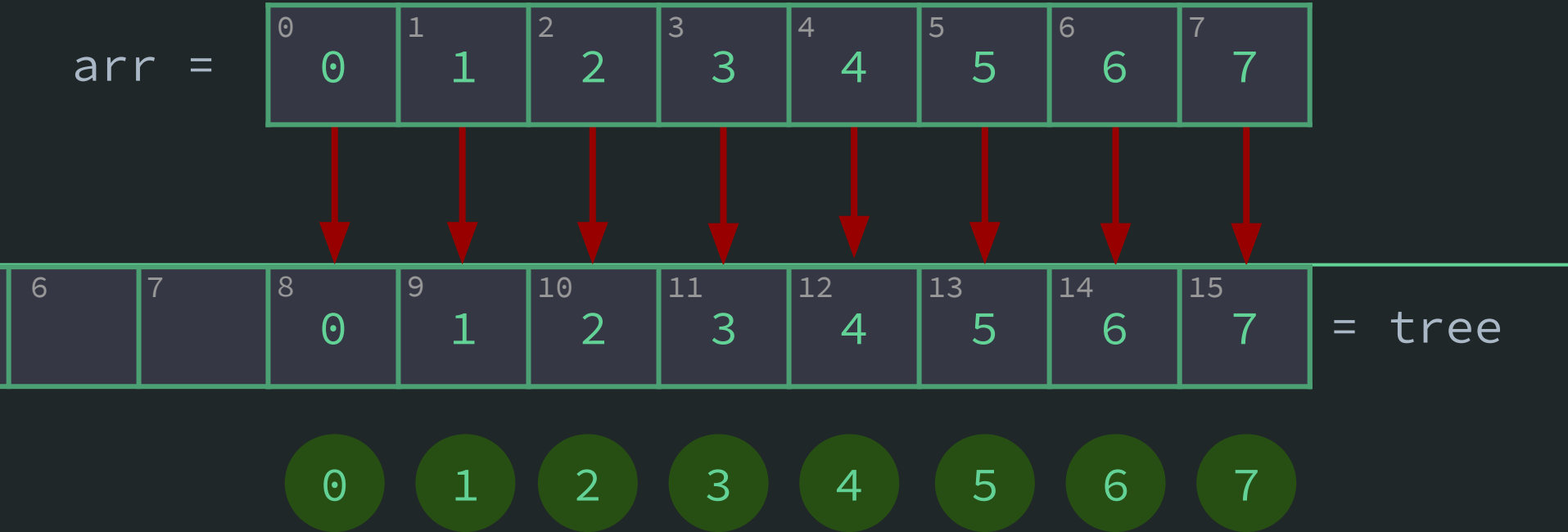


The diagram features a tree structure on a dark grey background. A horizontal green line spans the width of the image. To the left of this line, the word 'Implementation' is written in a light grey, monospace-style font. To the right, a tree structure is shown. It has a root node at the top right, which is a grey circle. This root node is connected by a thick green line to a child node below it, which is a bright green circle. This child node is then connected by two thick green lines to two more child nodes below it, both of which are bright green circles. The tree structure continues downwards, with more green nodes and lines visible at the bottom of the frame.

Let's Return to This

```
1 arr = [0,1,2,3,4,5,6,7]
2
3 # Q = 2, U = 1
4 query(l=0, r=6)      # returns 21 (0+1+2+3+4+5+6)
5 update(i=2, val=1)   # update arr to [0,1,3,3,4,5,6,7]
6 query(l=1, r=5)      # returns 16 (1+3+3+4+5)
```

Construction - Leaf Nodes



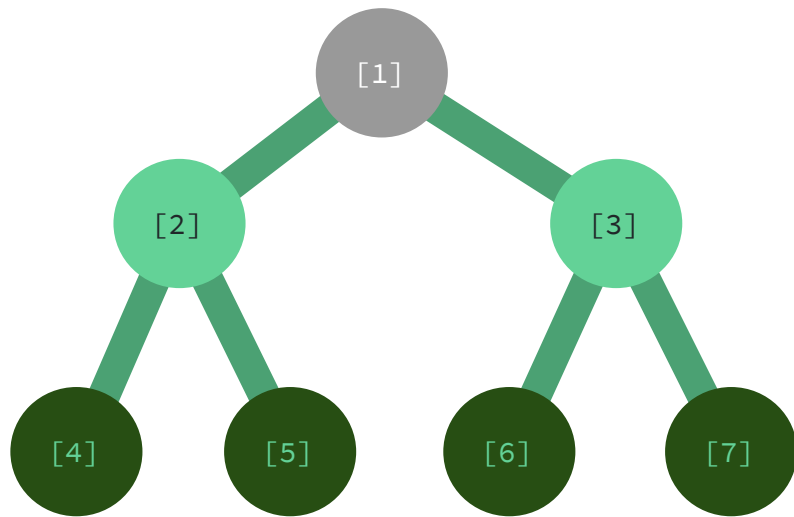
Implementation



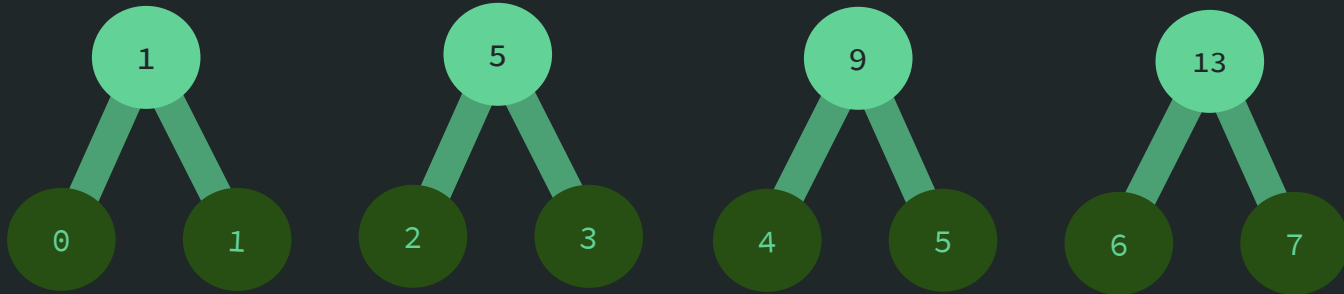
```
1 // Copy arr into the leaf nodes
2 for (int i = 0; i<arrLen; ++i) {
3     tree[i+arrLen] = arr[i];
4 }
5
6 // Alternative
7 // System.arraycopy(arr, 0, tree, arrlen, arrlen);
```

How do We Find a Node's Children and Parent?

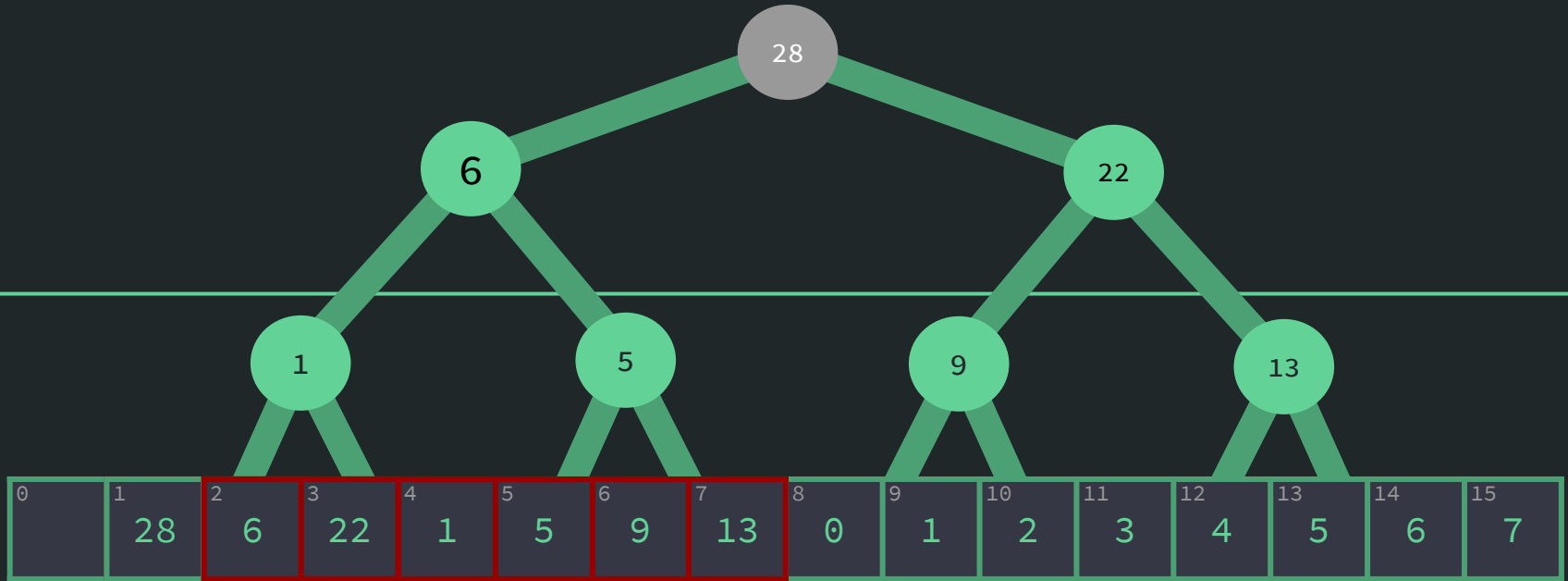
- Let i be the current node
- i 's children:
 - Left child: $2*i$ (always even)
 - Right child: $2*i+1$ (always odd)
- i 's parent:
 - $i/2$ (java truncates decimals
-> same index for left and right child)

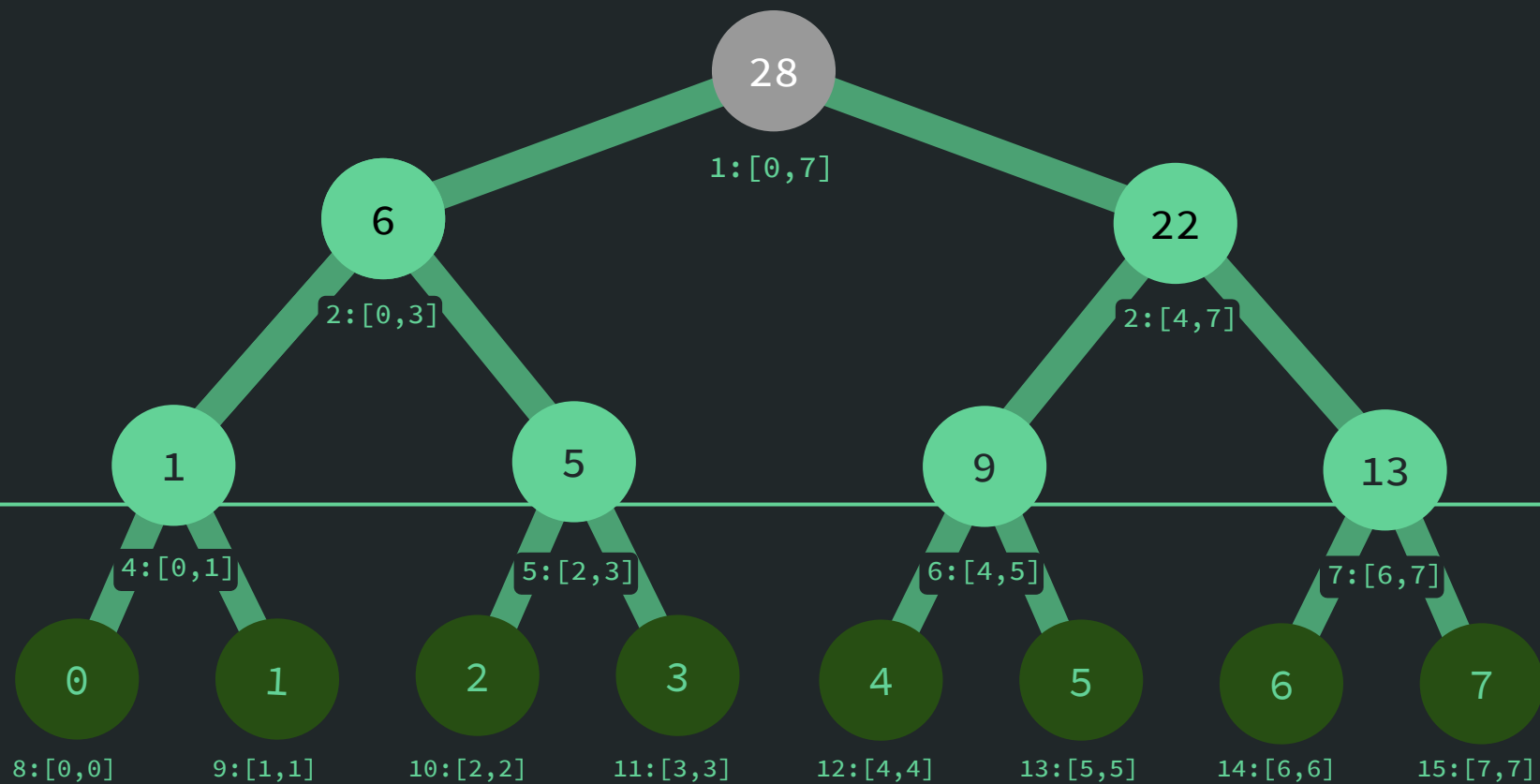


Construction – Inner Nodes



Construction – Inner Nodes





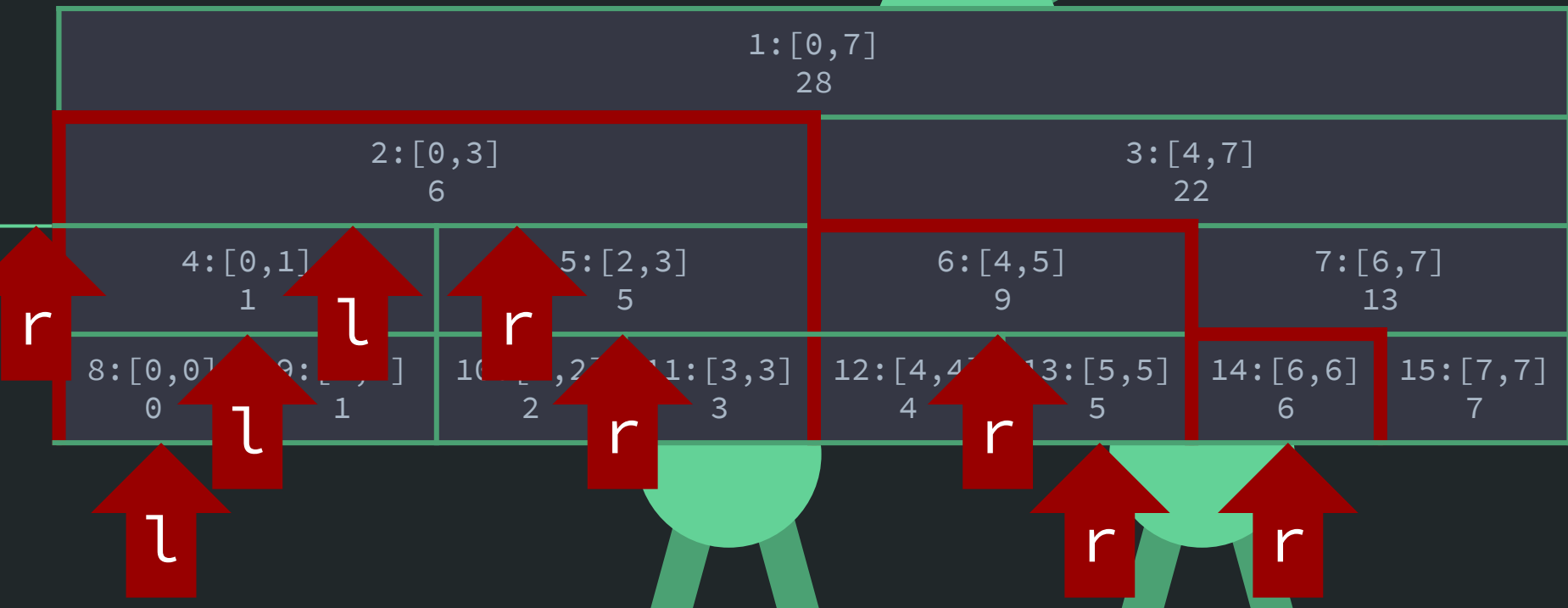
Implementation

```
1 void build {  
2  
3     // Start at the last parent  
4     for (int i = arrLen-1; i>=1; --i) {  
5  
6         // Parents are the sum of their children  
7         tree[i] = tree[2*i]+tree[2*i+1];  
8  
9     }  
10  
11 }
```

Queries Part 1

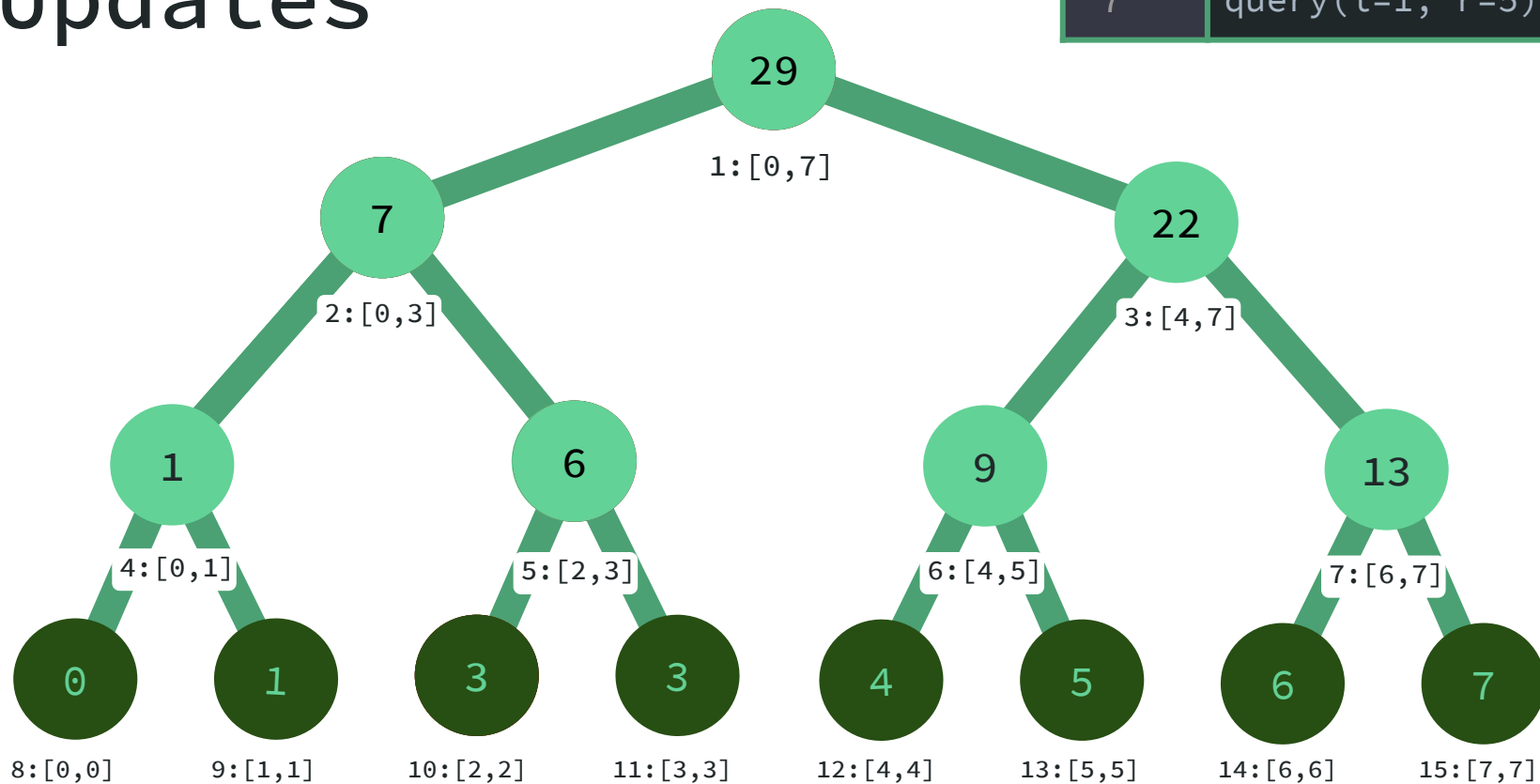
5	query(l=0, r=6)
6	update(i=2, val=1)
7	query(l=1, r=5)

sum = 05



Updates

5	<code>query(l=0, r=6)</code>
6	<code>update(i=2, val=1)</code>
7	<code>query(l=1, r=5)</code>



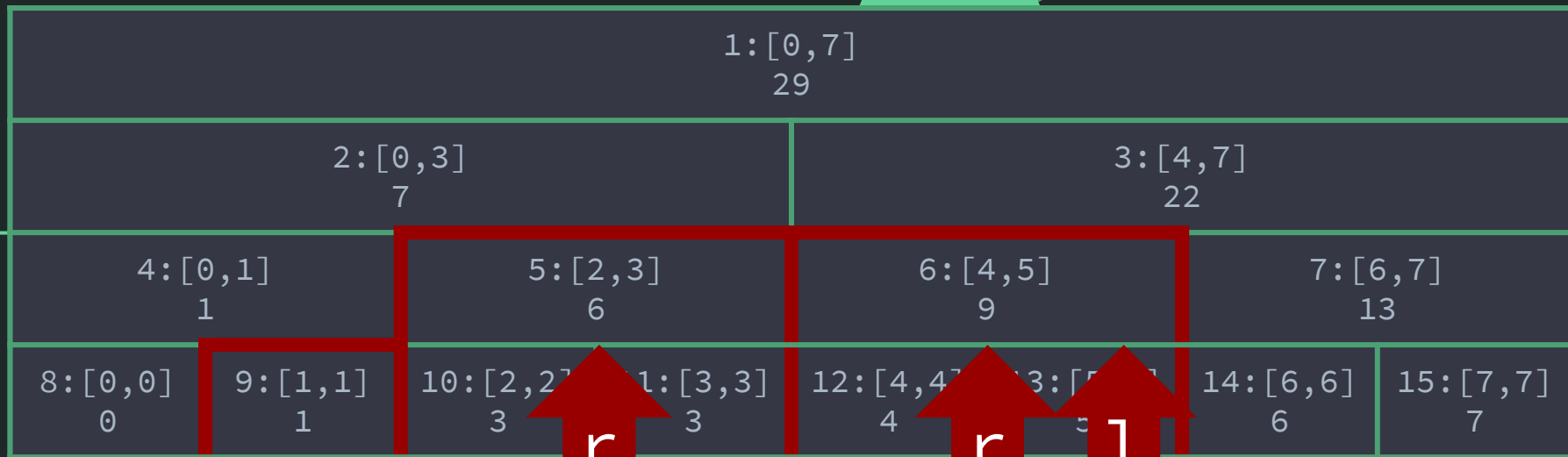
Implementation

```
1 void update (int i, int val) {
2
3     i += arrLen;          // Increment i to its leaf node counterpart
4     tree[i] += val;       // Update the leaf node
5     i /= 2;              // Go to i's parent
6
7     // Bubble up to i's parents until the root node
8     for (; i>=1; i /= 2) {
9         // Update the parents
10        tree[i] = tree[2*i]+tree[2*i+1];
11    }
12
13 }
```

Queries Part 2

5	query(l=0, r=6)
6	update(i=2, val=1)
7	query(l=1, r=5)

sum = 16



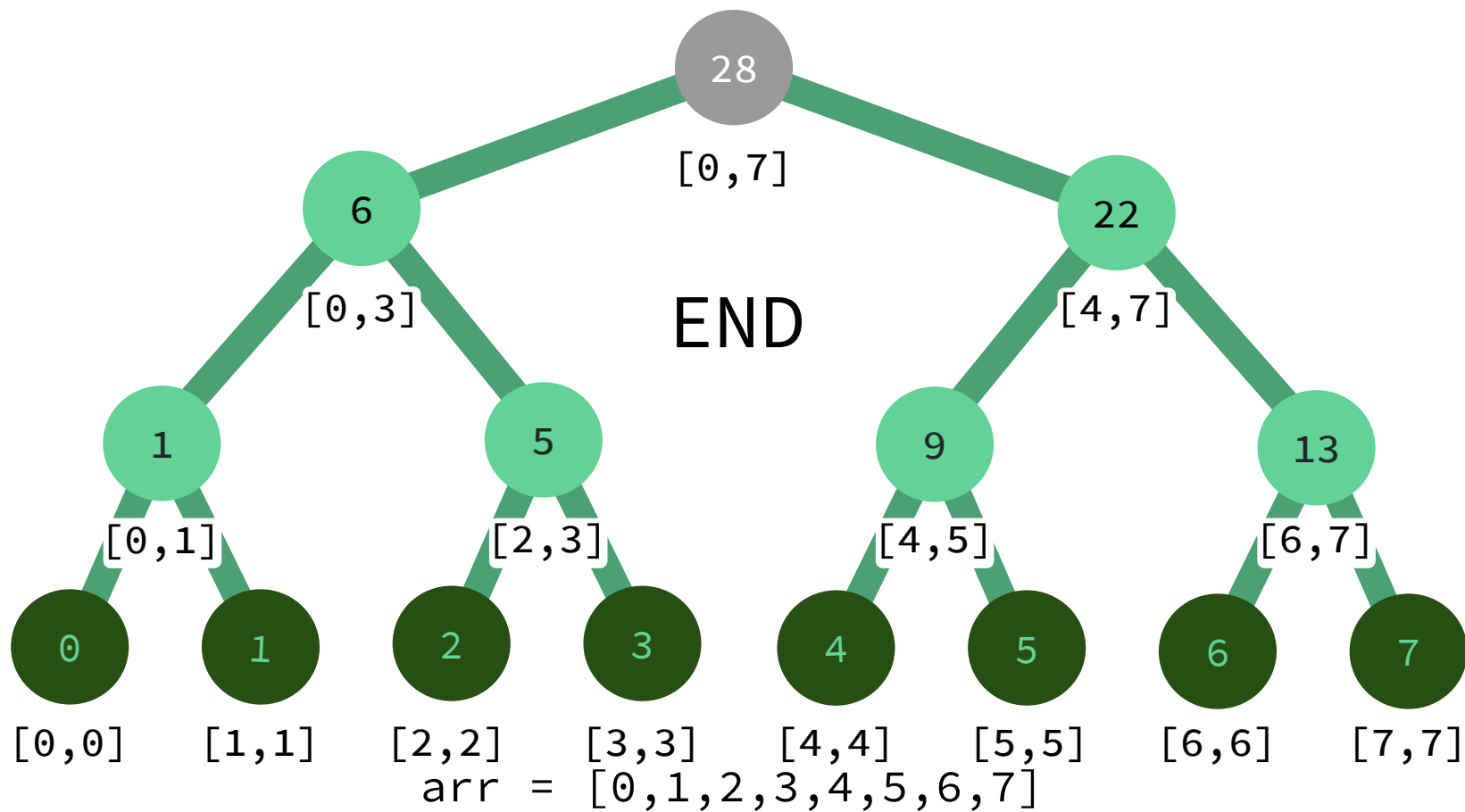
Implementation Part 1



```
1 int getSum (int l, int r) {  
2  
3     // Increment l and r to their leaf node counterparts  
4     l += arrLen;  
5     r += arrLen;  
6  
7     int sum = 0;
```

Implementation Part 2

```
8      // While l and r are still left and right
9      while (l<=r) {
10
11          if (l%2==1) {                // If l is the right child of the parent
12              sum += tree[l];          // Add the right child to the sum
13              ++l;                     // Move l to the left child of the next branch
14          }
15          if (r%2==0) {                // If r is the left child of its parent
16              sum += tree[r];          // Add the left child to the sum
17              --r;                     // Move r to the right child of the next branch
18          }
19          l /= 2, r/=2;                // Traverse to their parents
20
21      }
22
23      return sum;
24
25  } // End of function
```

Further Reading

[Iterative Implementation](#)

[Recursive Implementation](#)

[Lazy Propagation \(Range Updates\)](#)

Get Grepper rn

[Grepper - Chrome Web Store \(google.com\)](#)

Problems

[ICHB Selection Contest '17 Problem 3 - Parallel Universe](#)

[Dynamic Range Minimum Test](#)

[Mock CCO '18 Contest 3 Problem 4 - Roger Solves A Classic Rage Tree Problem](#)

Challenge:

[DMOPC '19 Contest 2 P3 - Selection](#)

CCC '21 S5 - Math Homework