Segment Trees

HKOI Training 2012

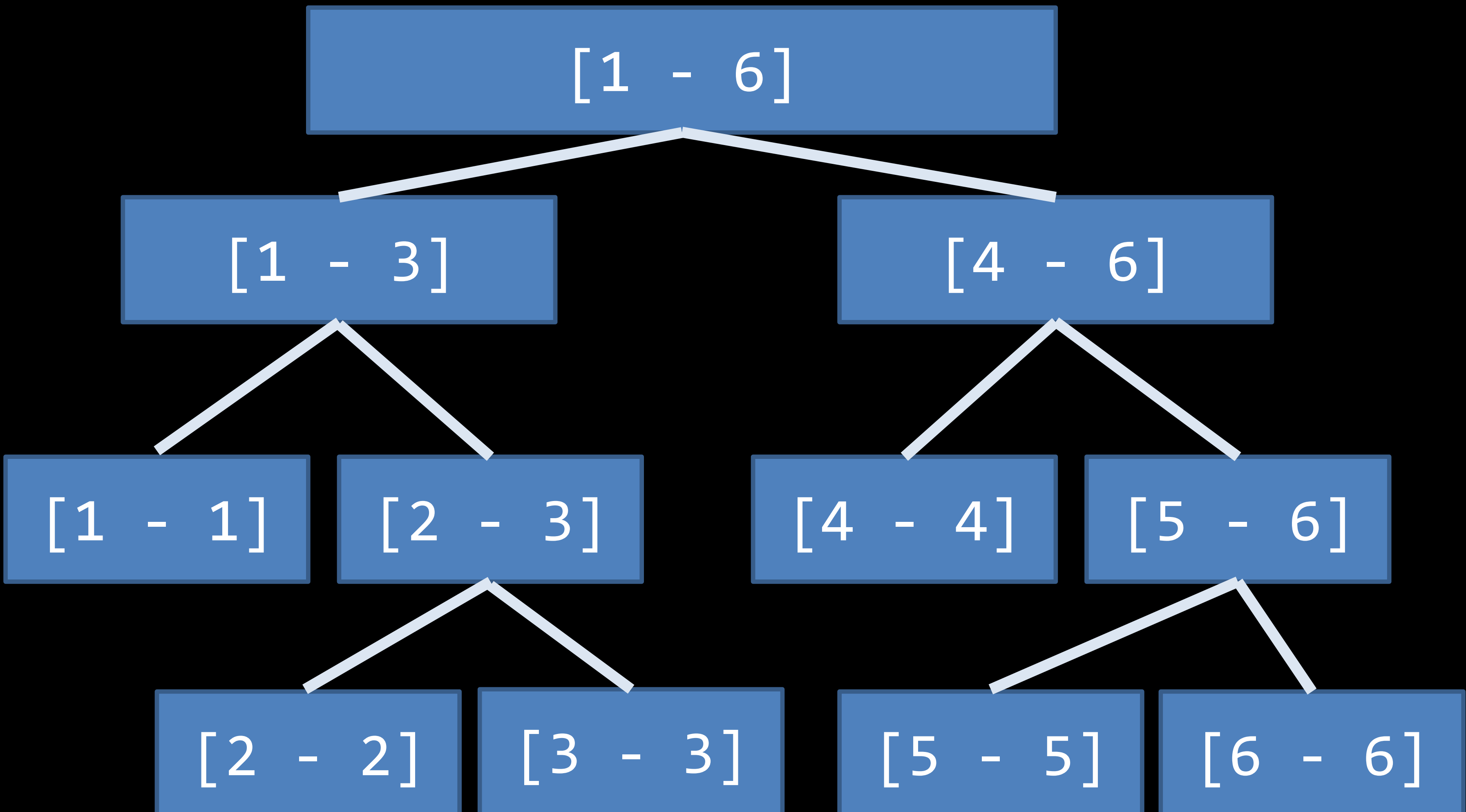
Overview

- Segment tree is a data structure that supports operations on ‘segments’
- Usually, operations runs in O(lg N)
- Often appeared in IOI

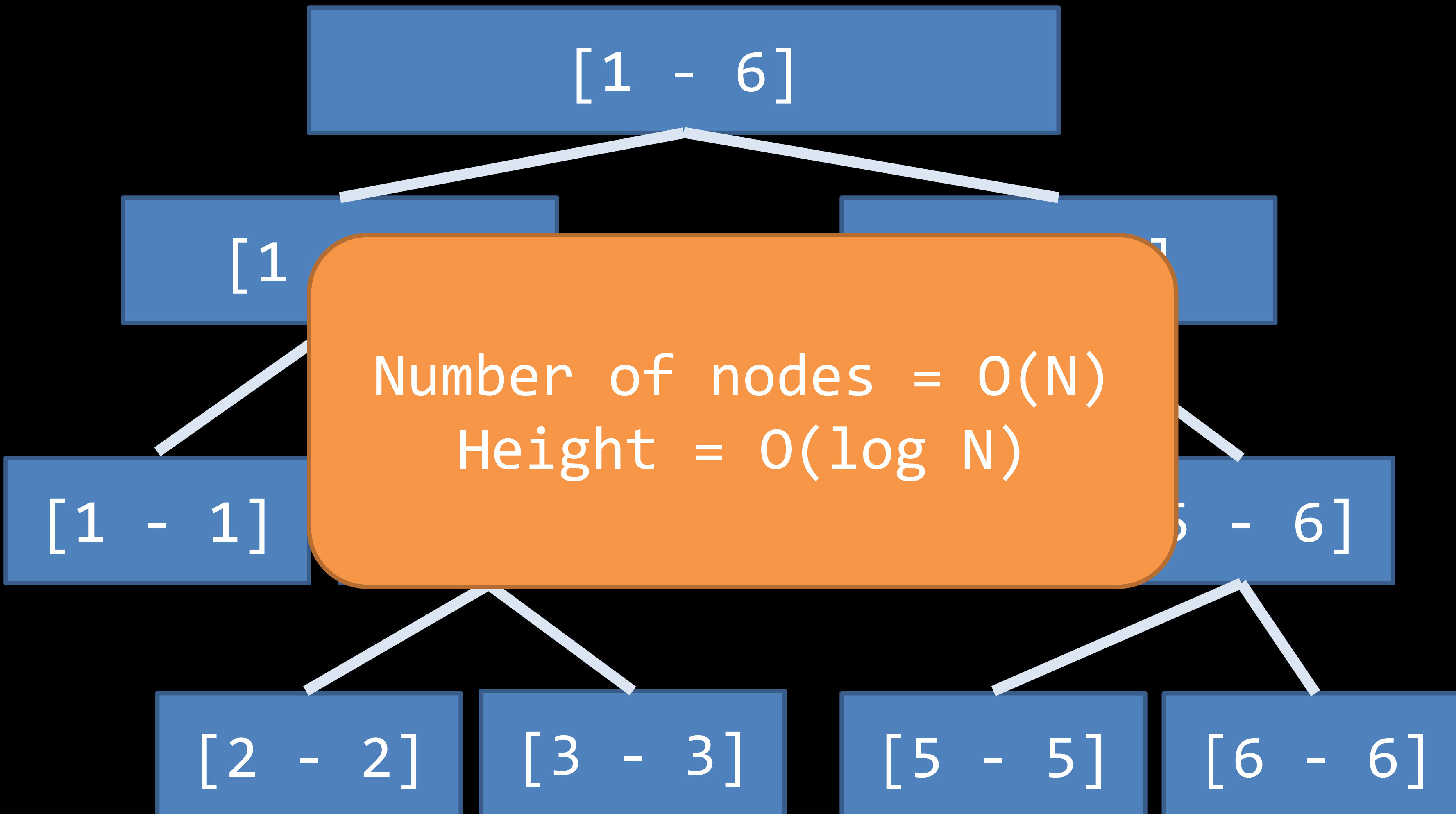
Structure

- Segment tree is a **full** binary tree i.e. every node has 0 or 2 children
- The root represent the whole segment
- The segment is break into two children

Structure



Structure

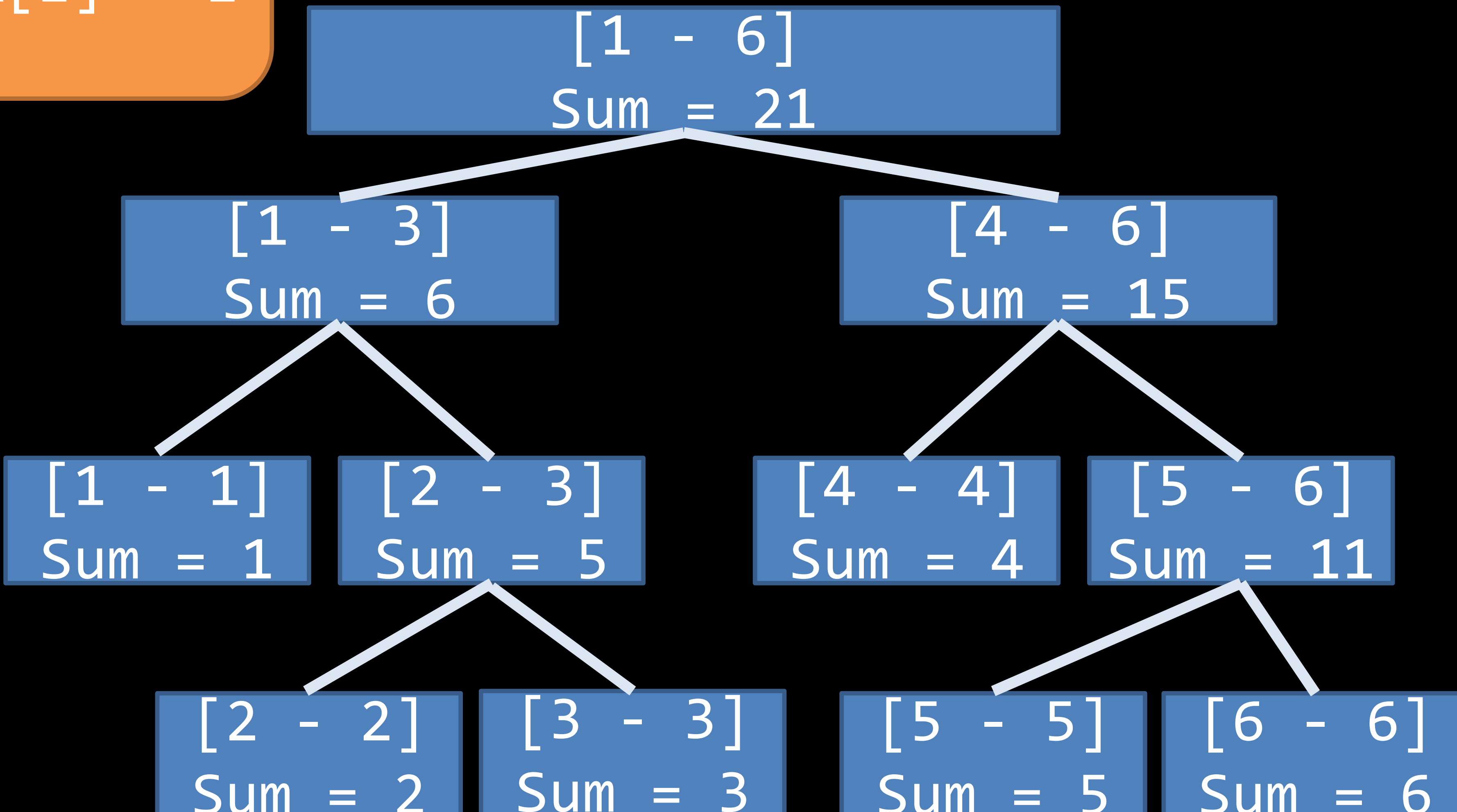


Example 1

- Given an array A[1..N]
- There are 2 types of operations
  - A[x] := v
  - What is A[l] + A[l+1] + … + A[r]? (Sum of A[l..r])

Initial
A[i] = i

Initialize

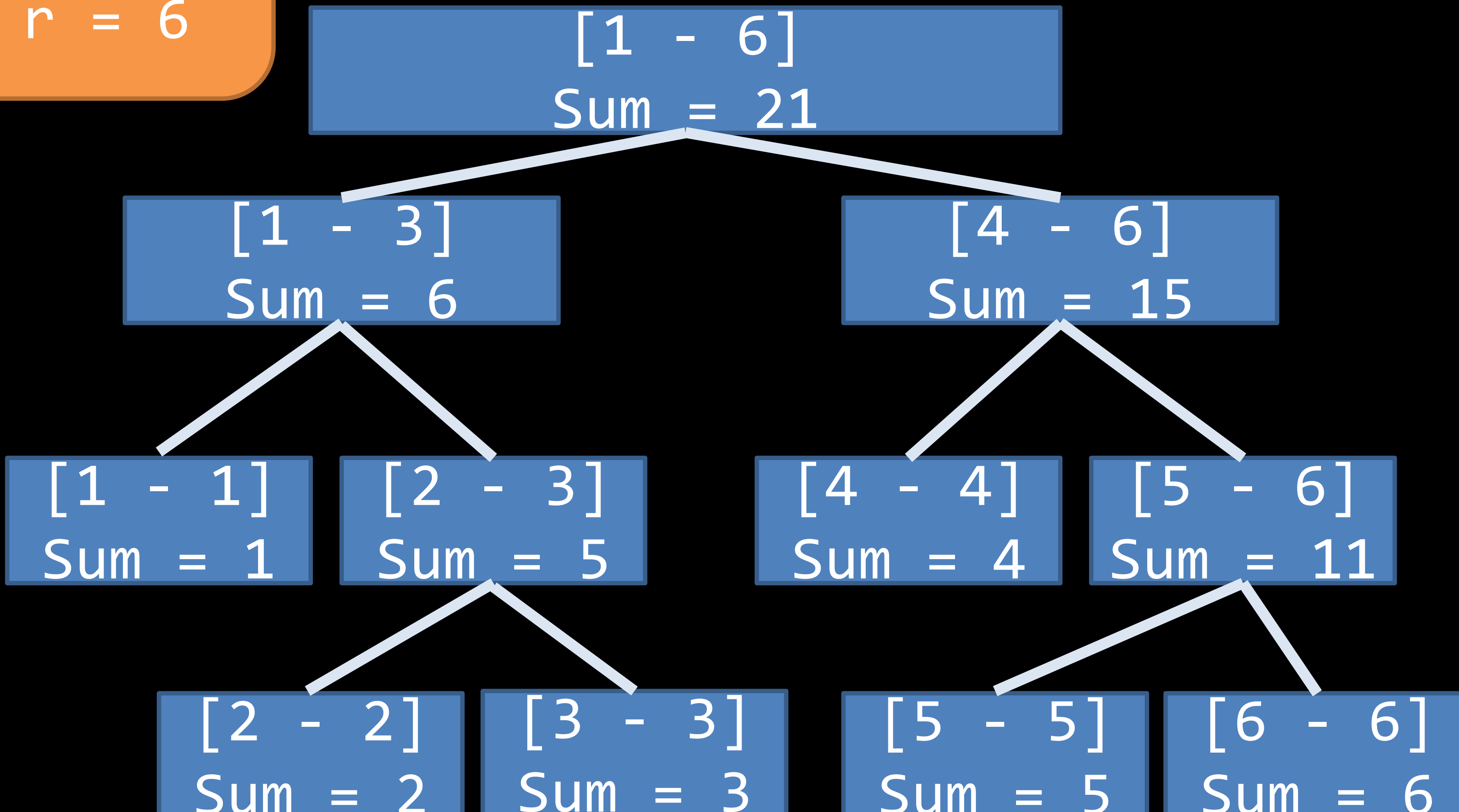


Initialize

- There are O(N) nodes
- Build the tree bottom-up
- Time complexity: O(N)

Query

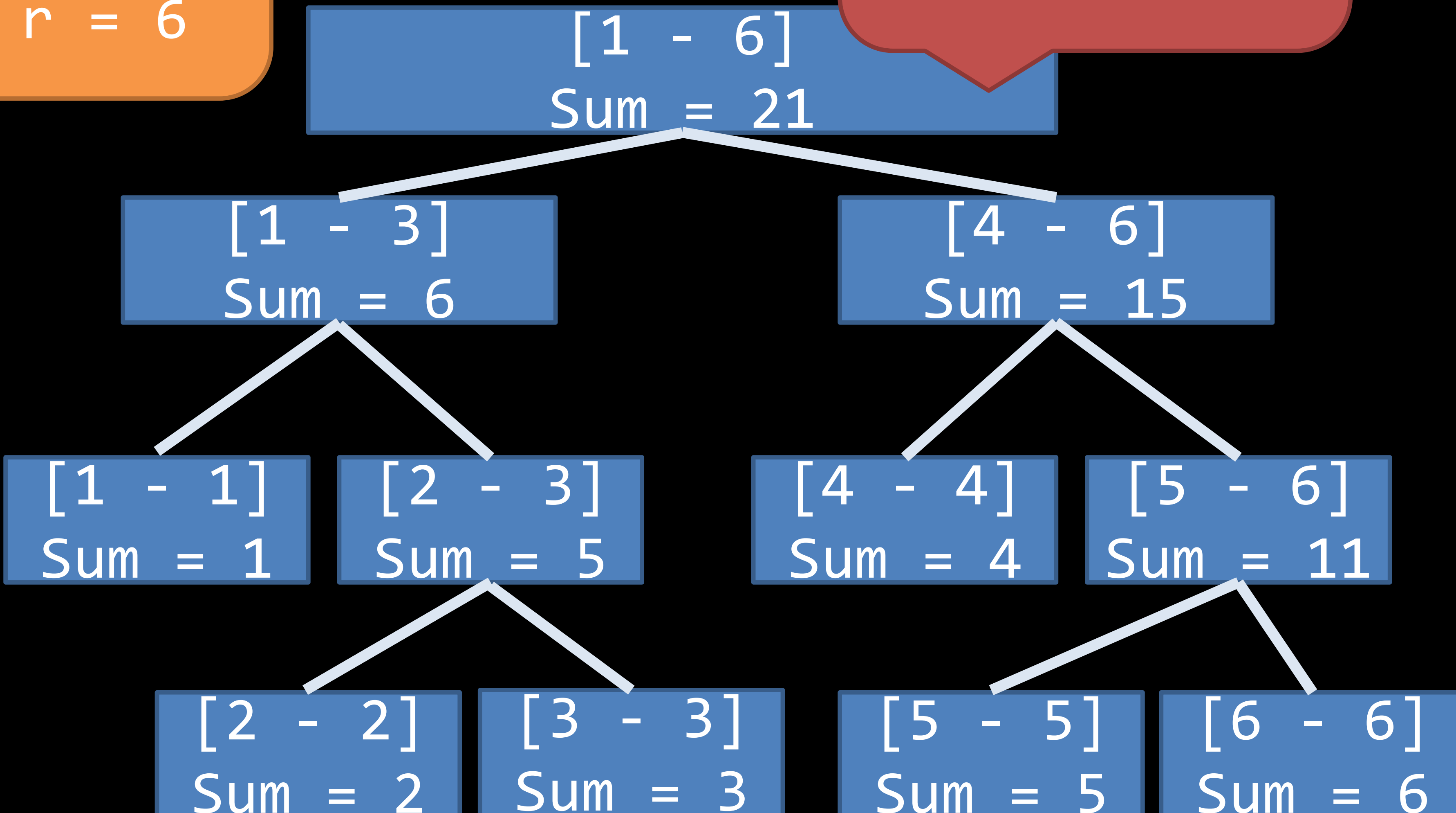
Query
l = 4
r = 6

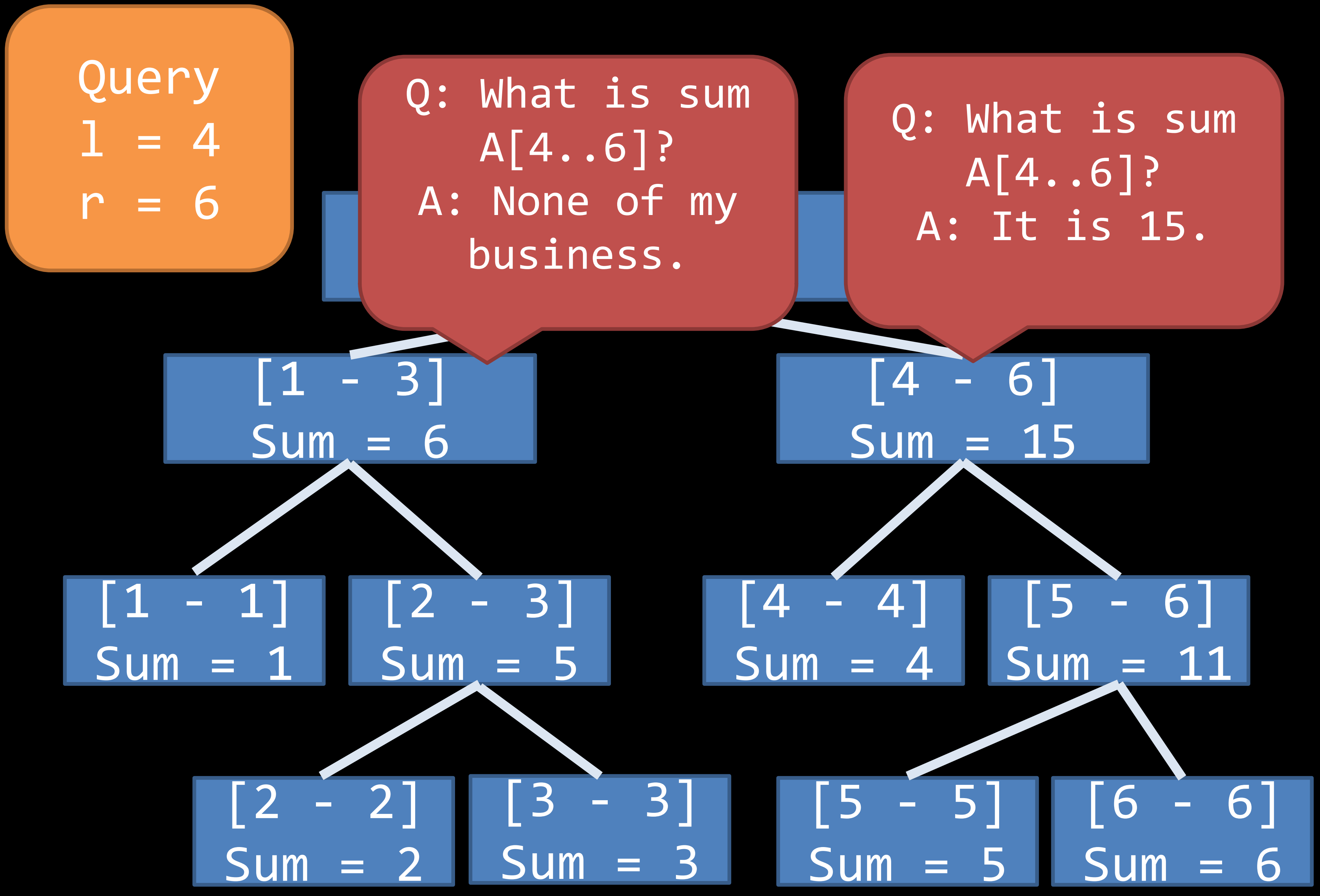


Query

Query
l = 4
r = 6

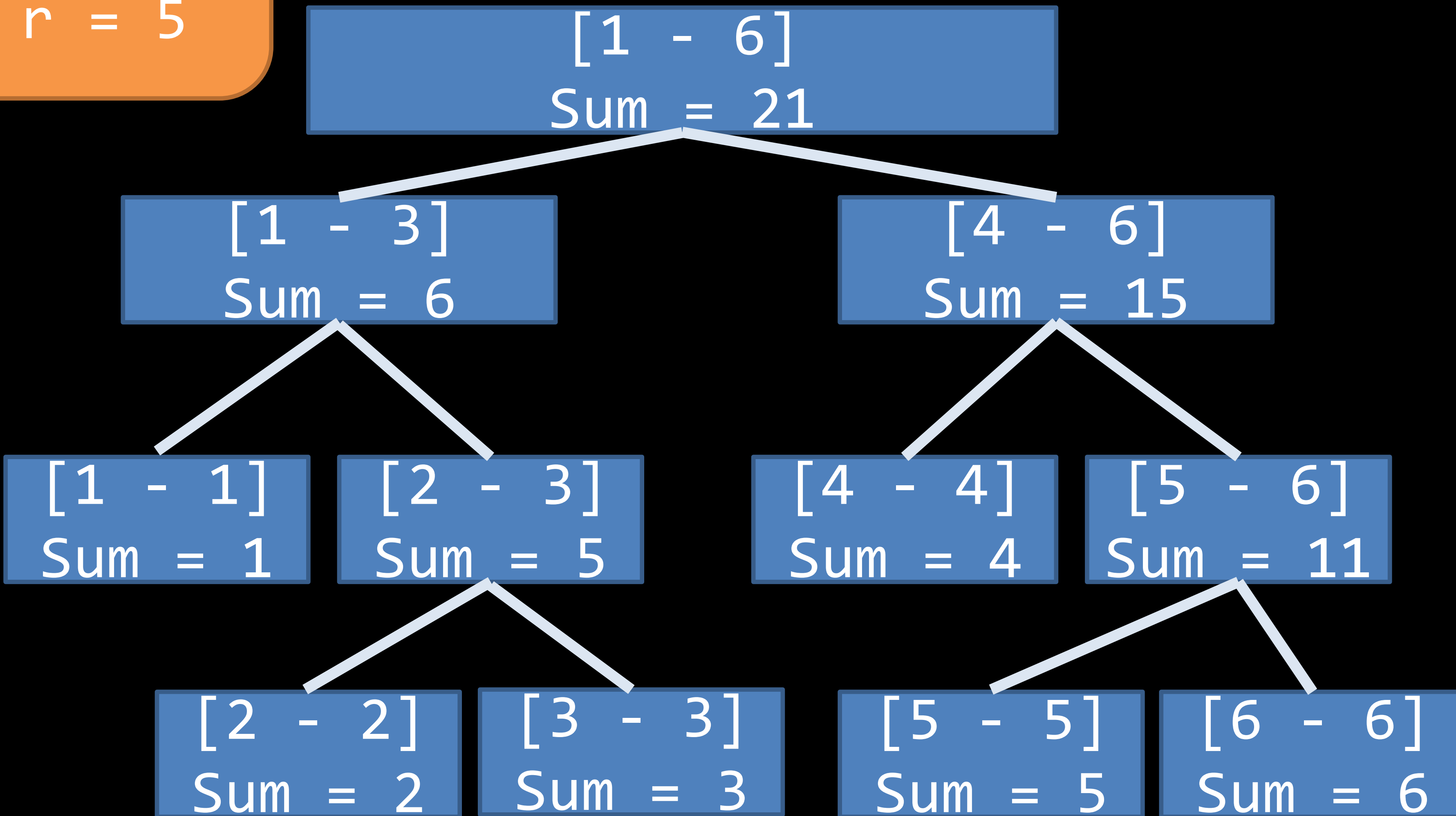
Q: What is sum A[4..6]?
A: I don’t know. Ask my children.




Query
l = 4
r = 6
Q: What is sum A[4..6]?
A: None of my business.
Q: What is sum A[4..6]?
A: It is 15.
[1 - 3]
Sum = 6
[4 - 6]
Sum = 15
[1 - 1]
Sum = 1
[2 - 3]
Sum = 5
[4 - 4]
Sum = 4
[5 - 6]
Sum = 11
[2 - 2]
Sum = 2
[3 - 3]
Sum = 3
[5 - 5]
Sum = 5
[6 - 6]
Sum = 6

Query
l = 3
r = 5

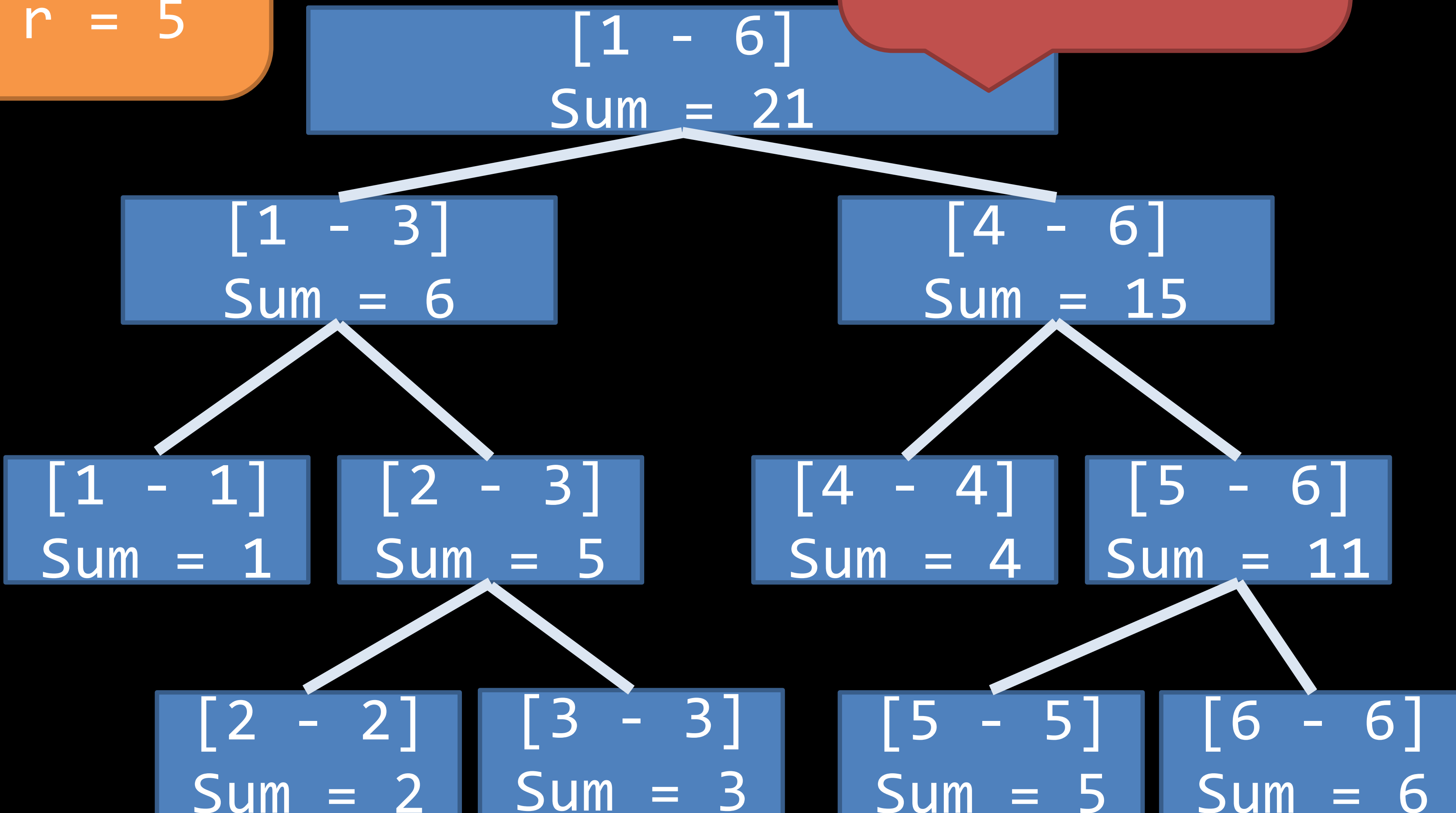
Query



Query
l = 3
r = 5

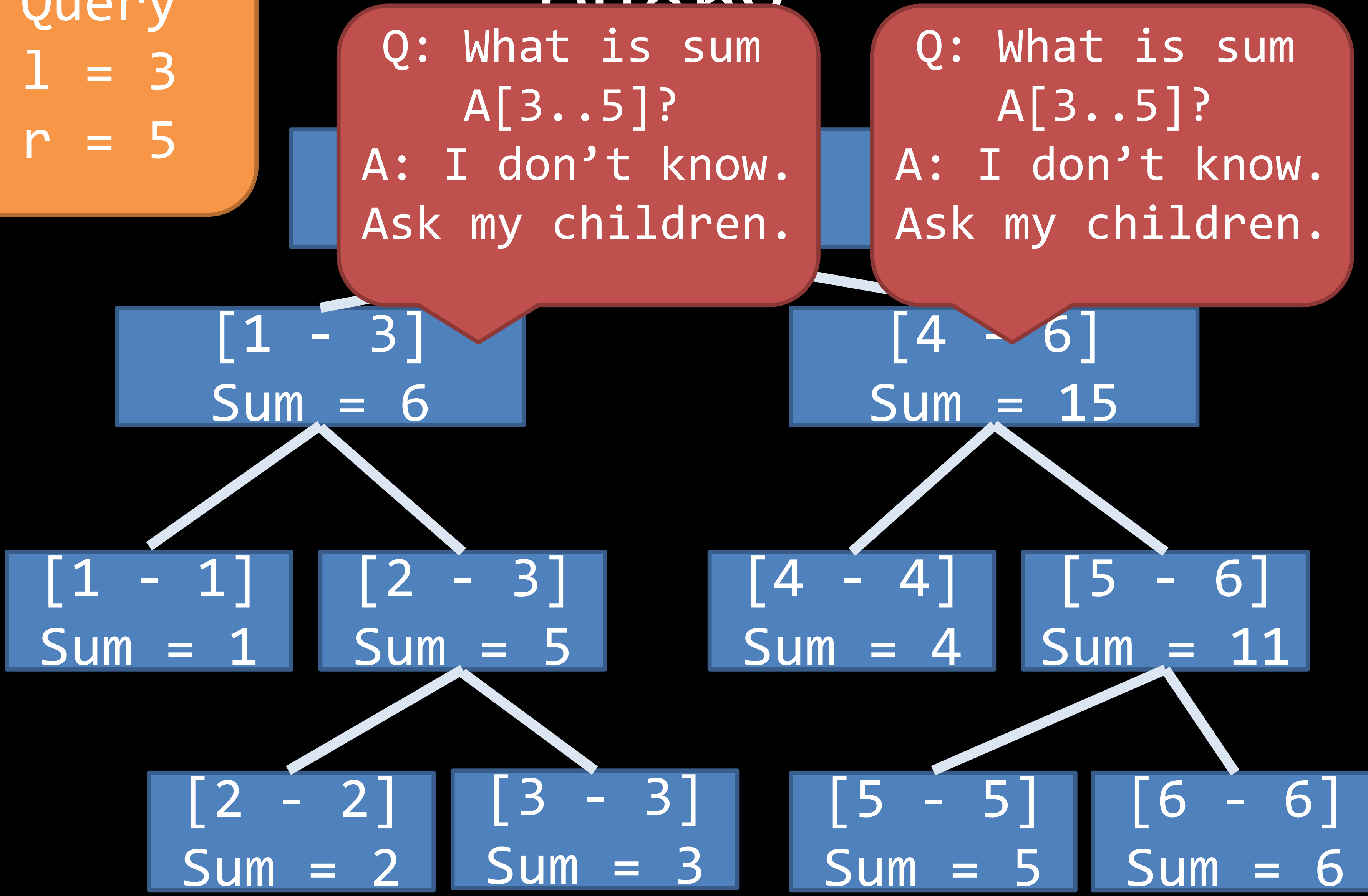
Query

Q: What is sum A[3..5]?
A: I don’t know. Ask my children.



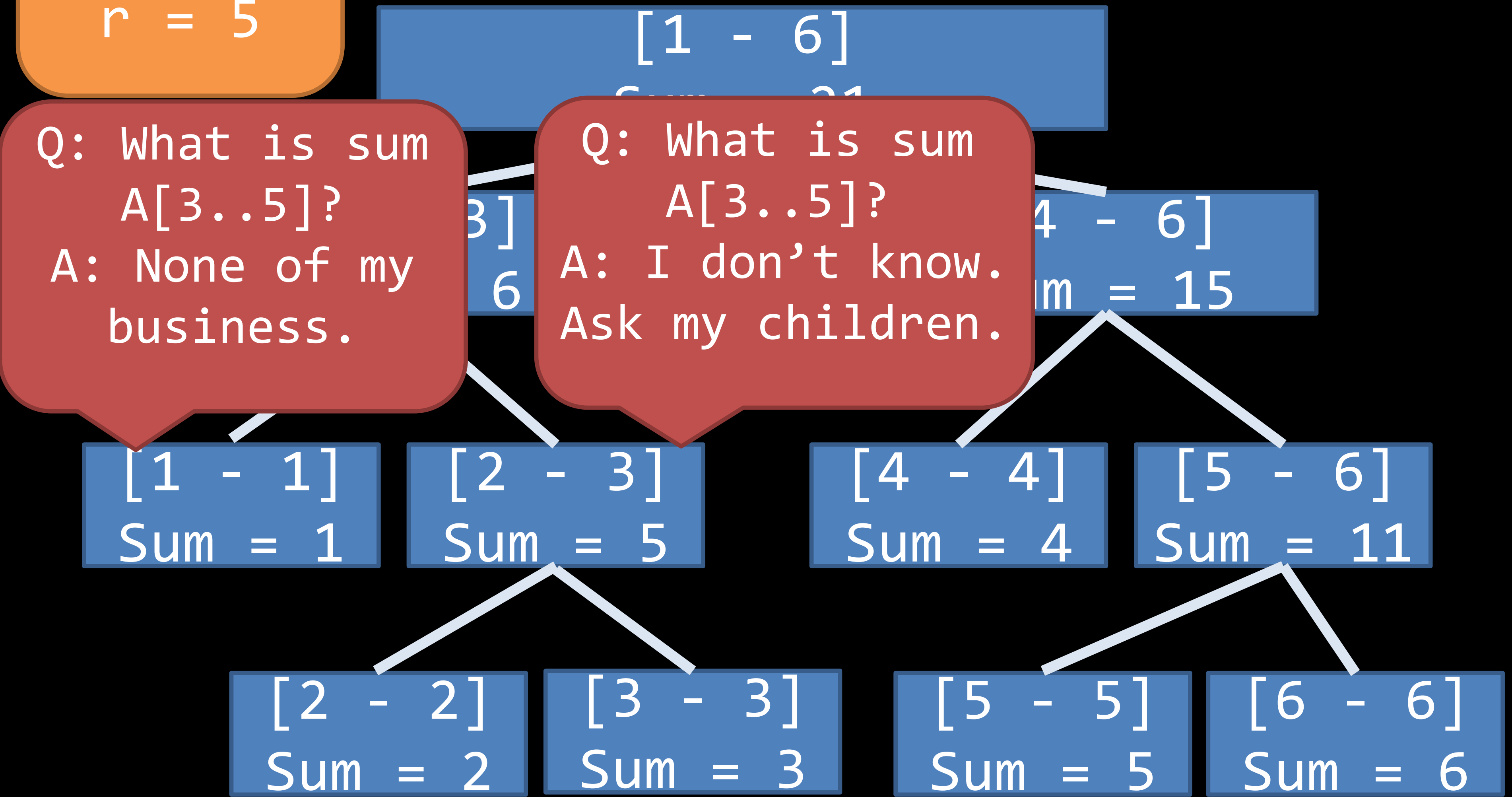
Query

Query
l = 3
r = 5



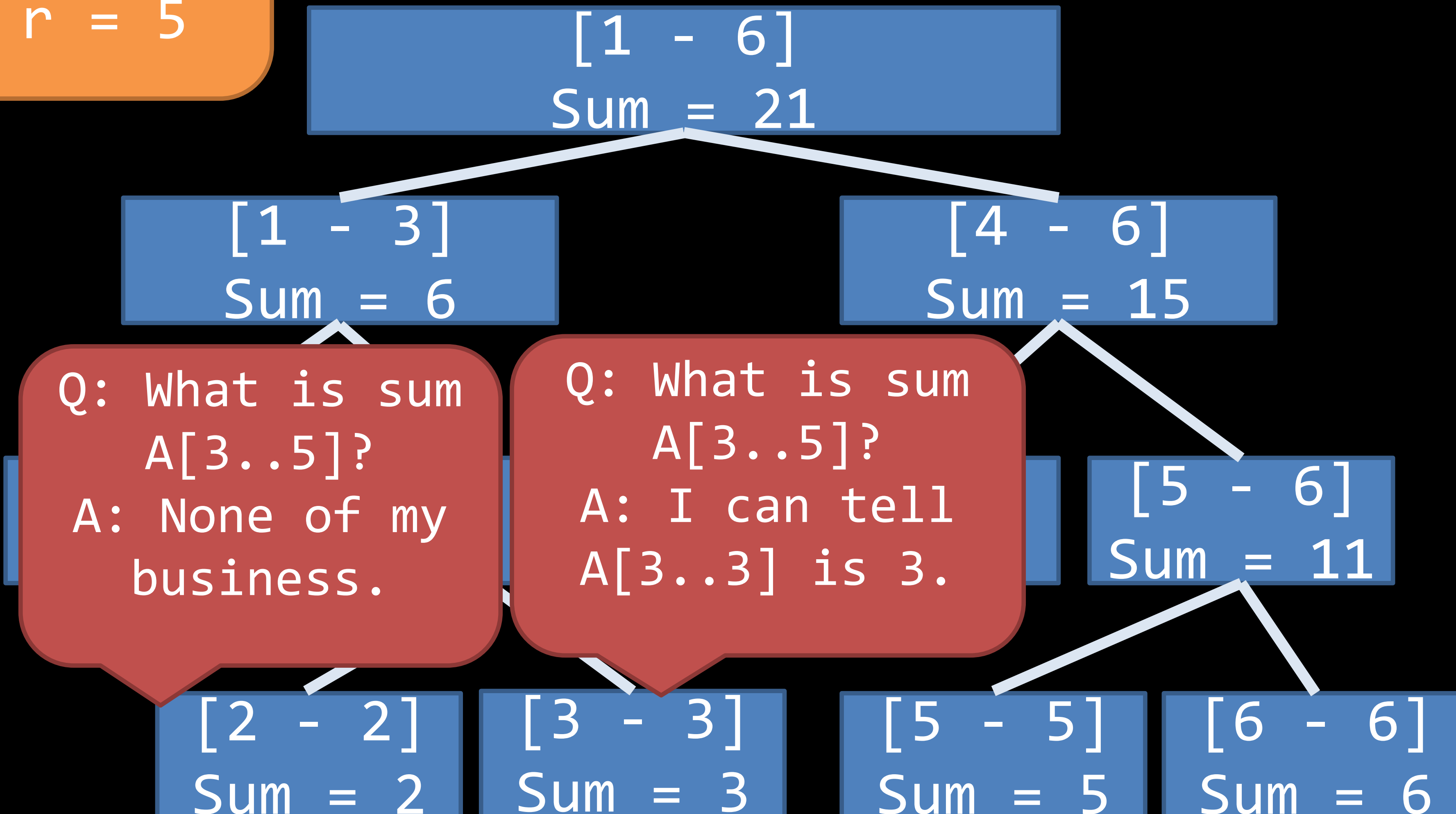
Query

Query
l = 3
r = 5



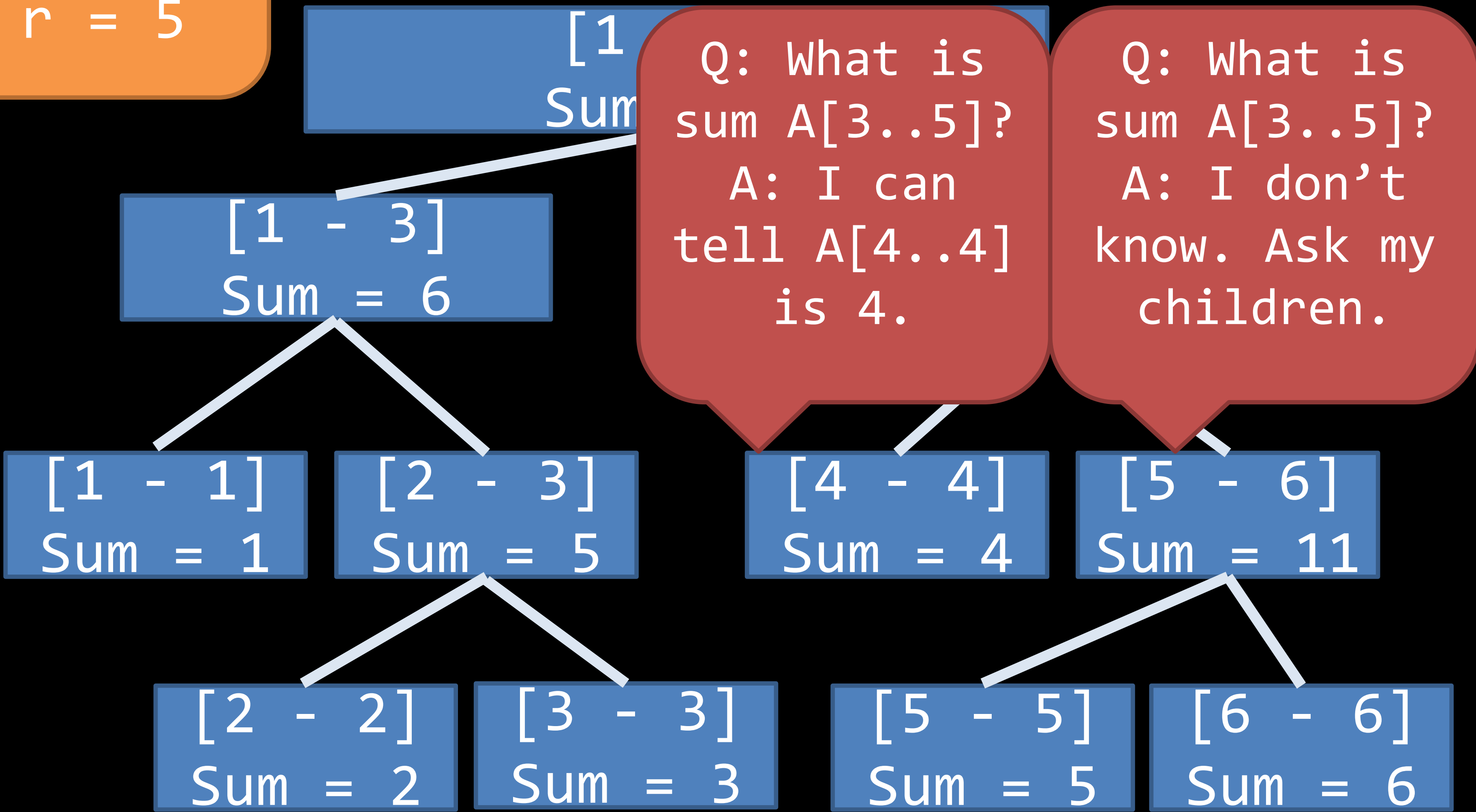
Query

Query
l = 3
r = 5



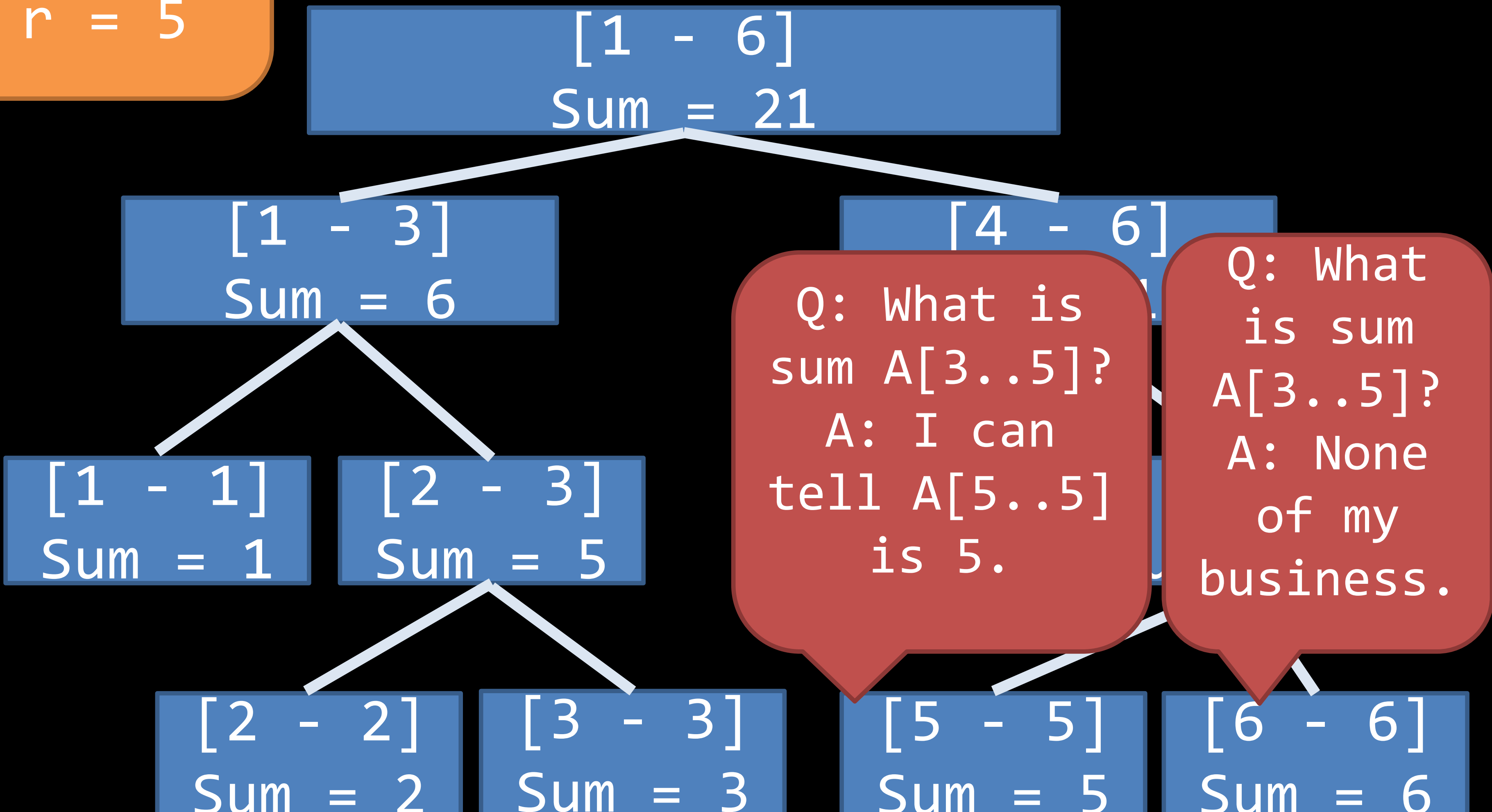
Query
l = 3
r = 5

Query



Query

Query
l = 3
r = 5

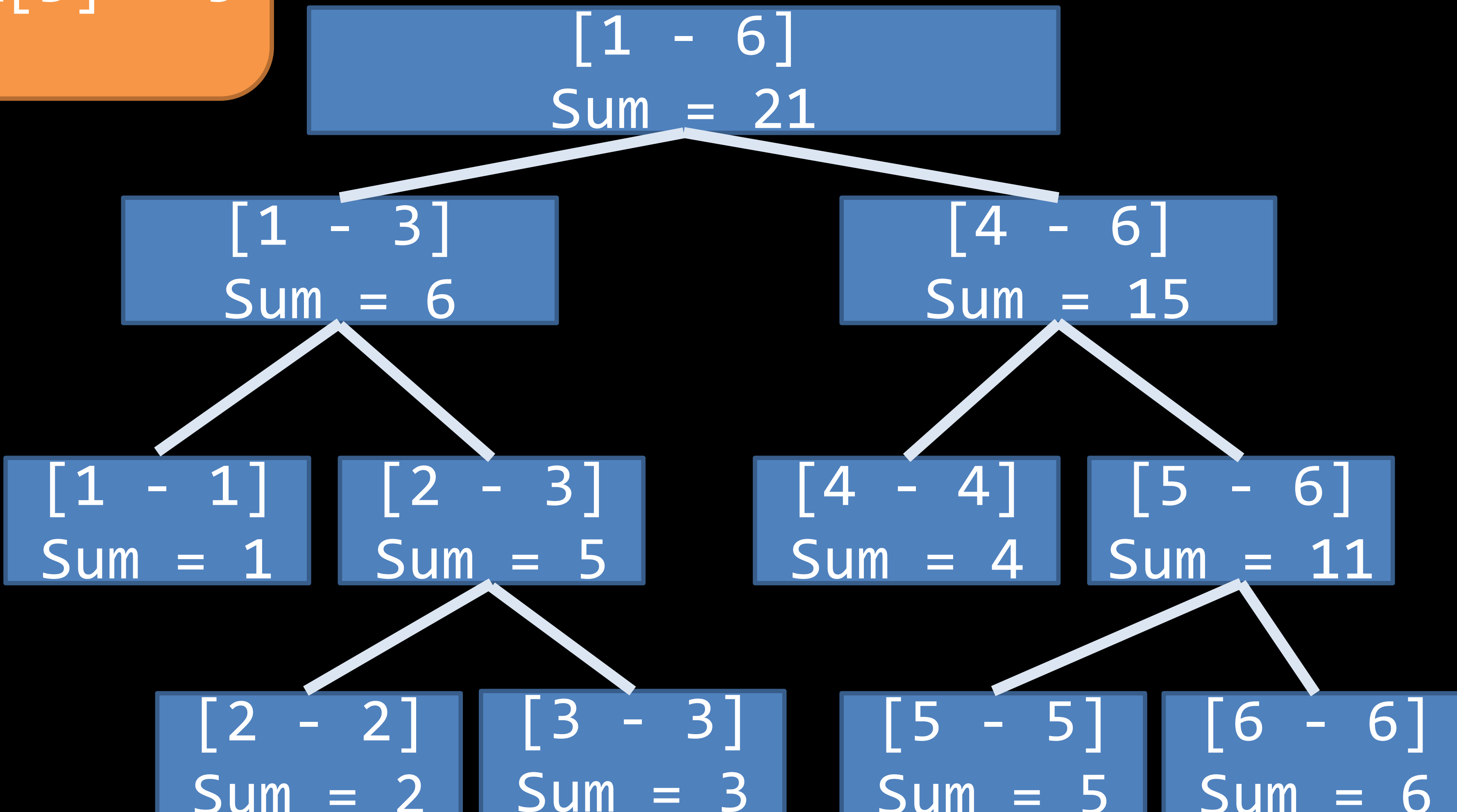


Query

- Summary
  - If the node lies in the query interval, add it to answer
  - If the node intersect with the query interval, go down
- We visit at most 4 nodes at each level (exclude those ‘none of my business’)
- Height of tree = O(log N)
- Time complexity = O(log N)

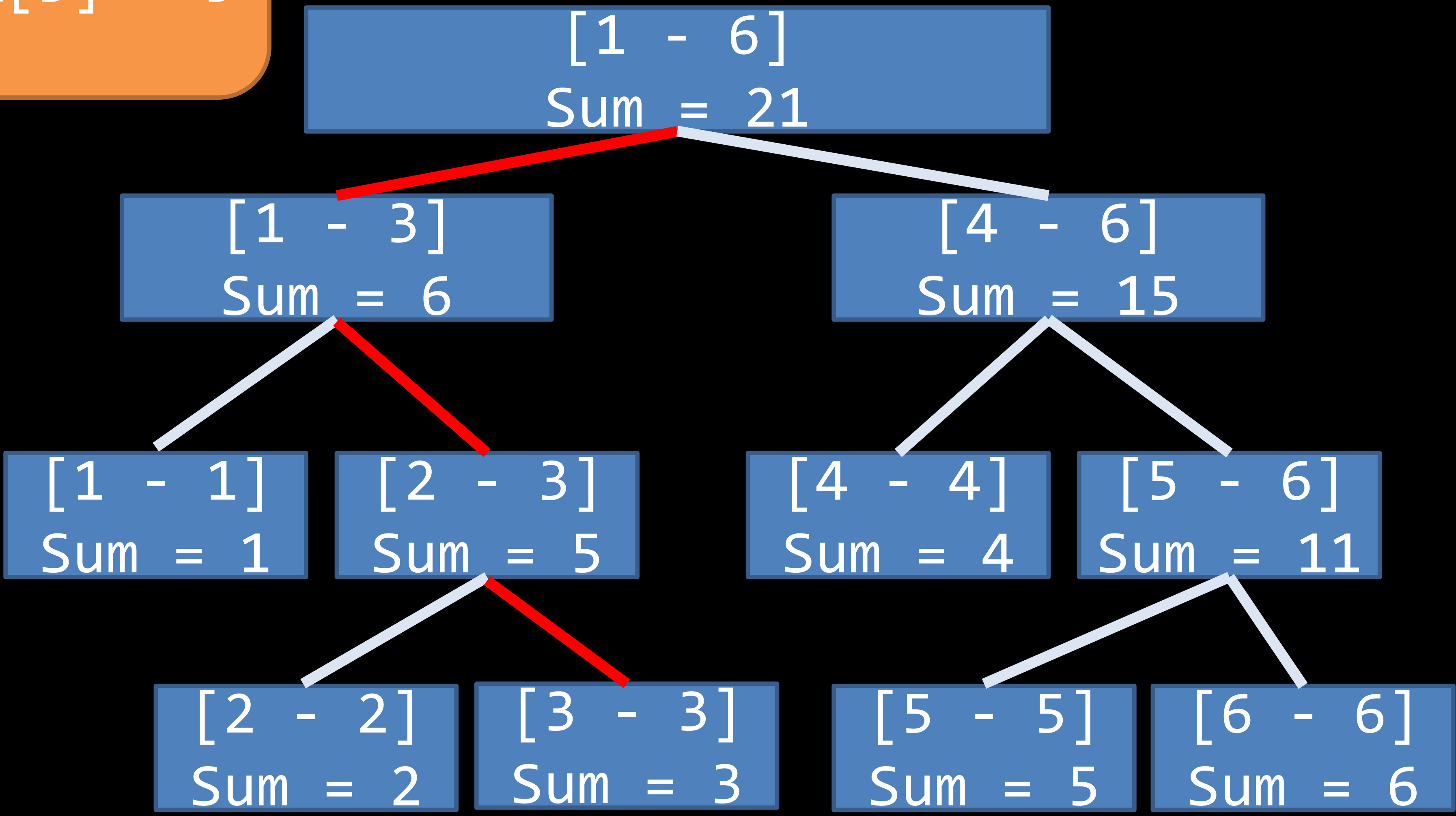
Update

Update
A[3] = 5



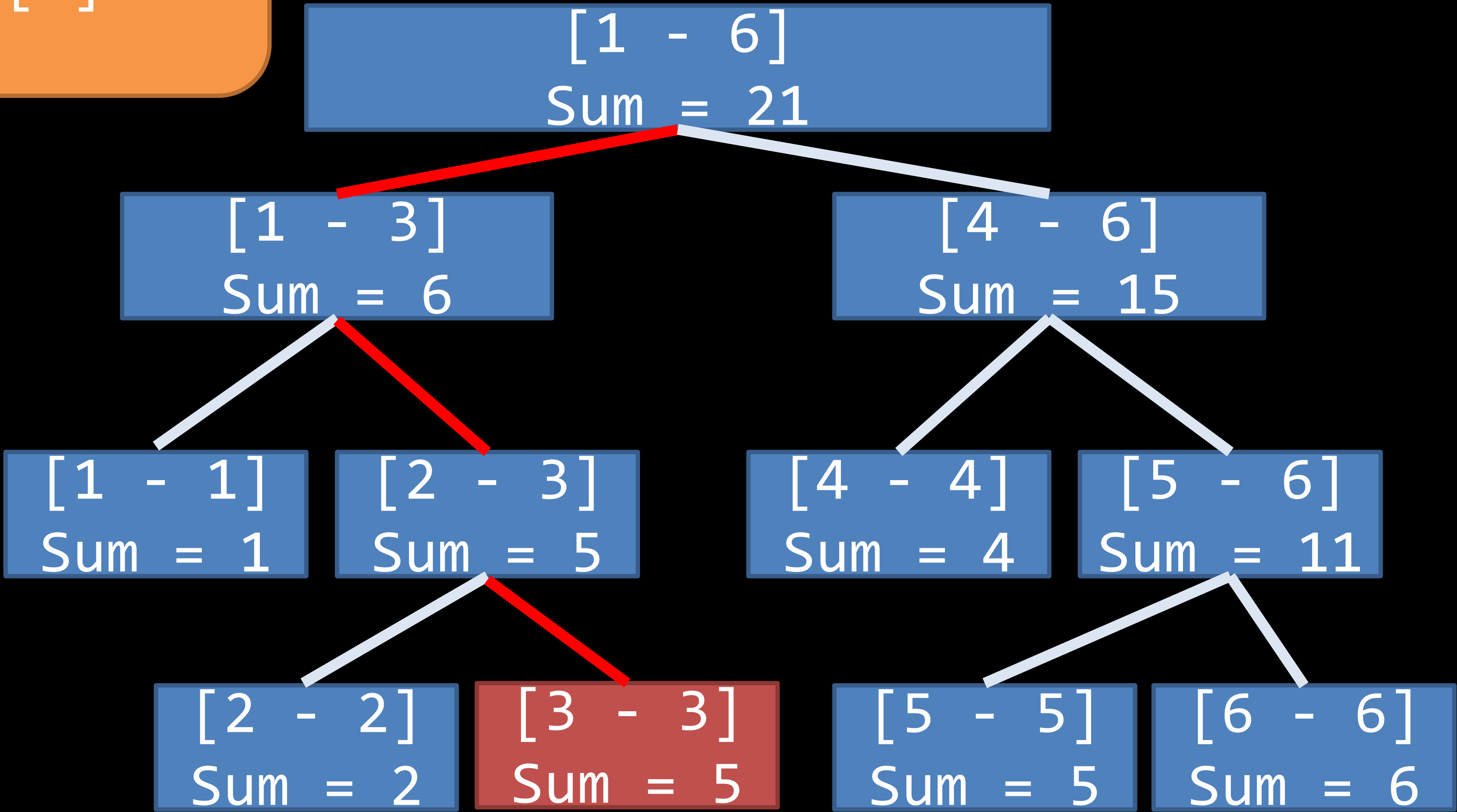
Update

Update
A[3] = 5

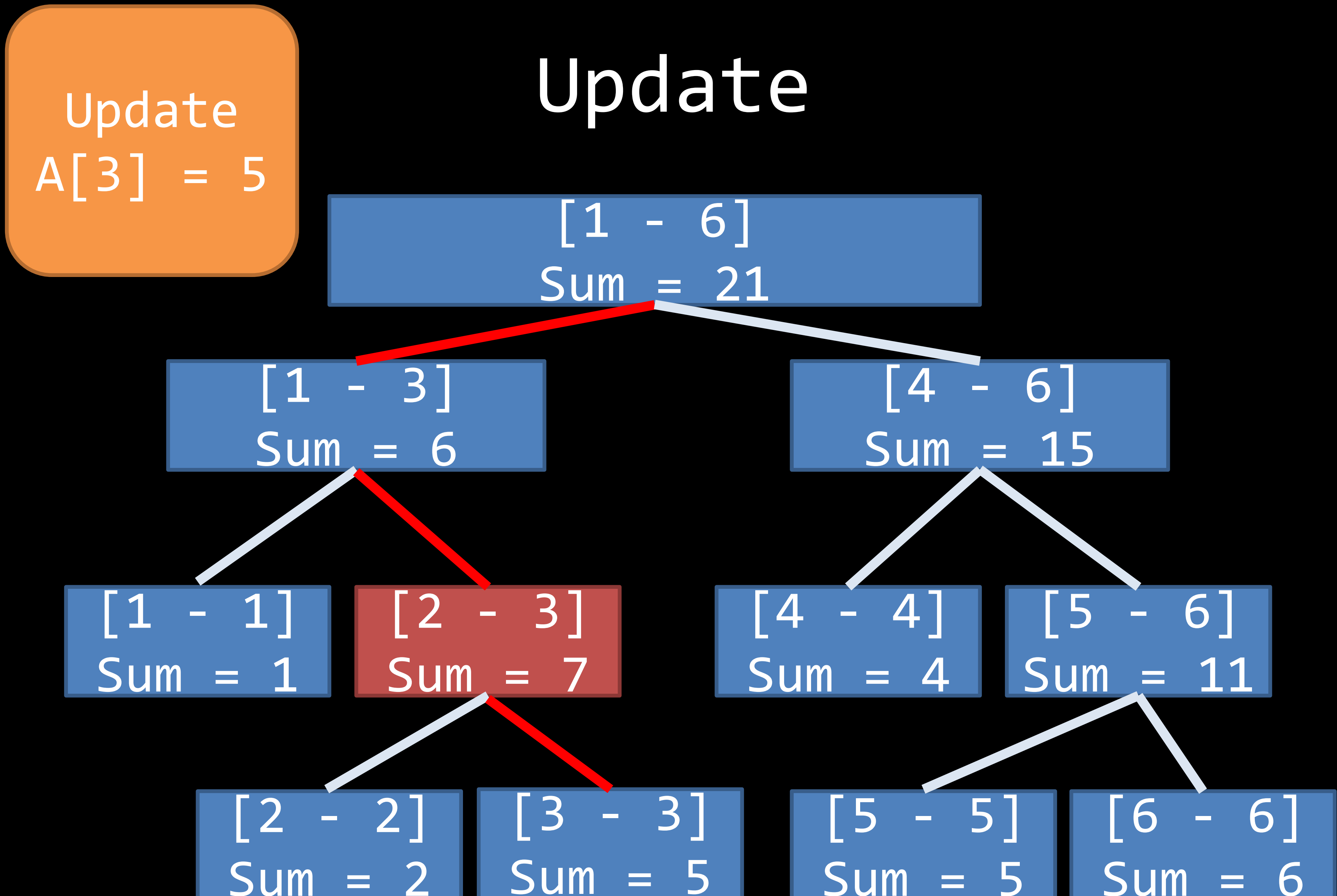


Update

Update
A[3] = 1

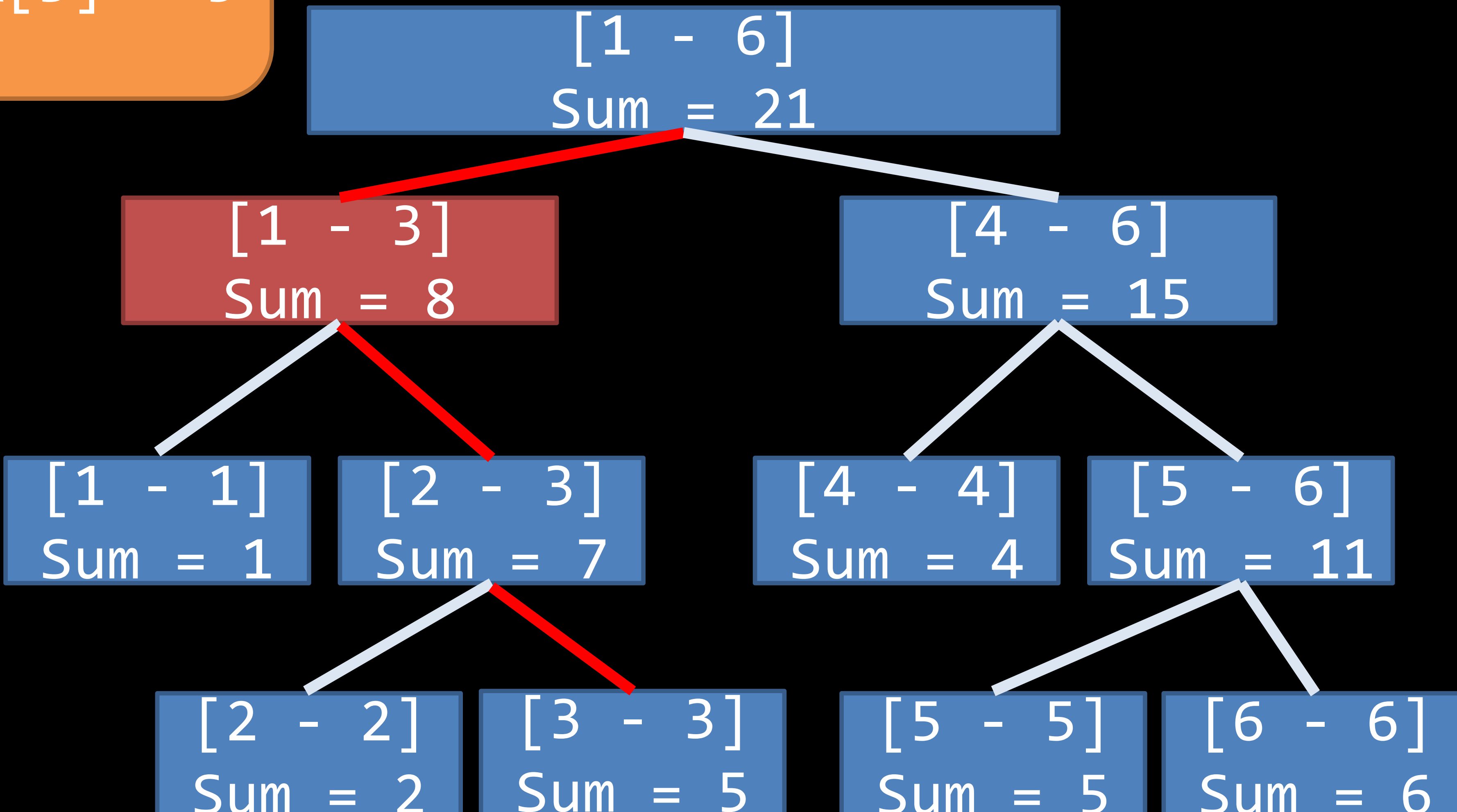


Update

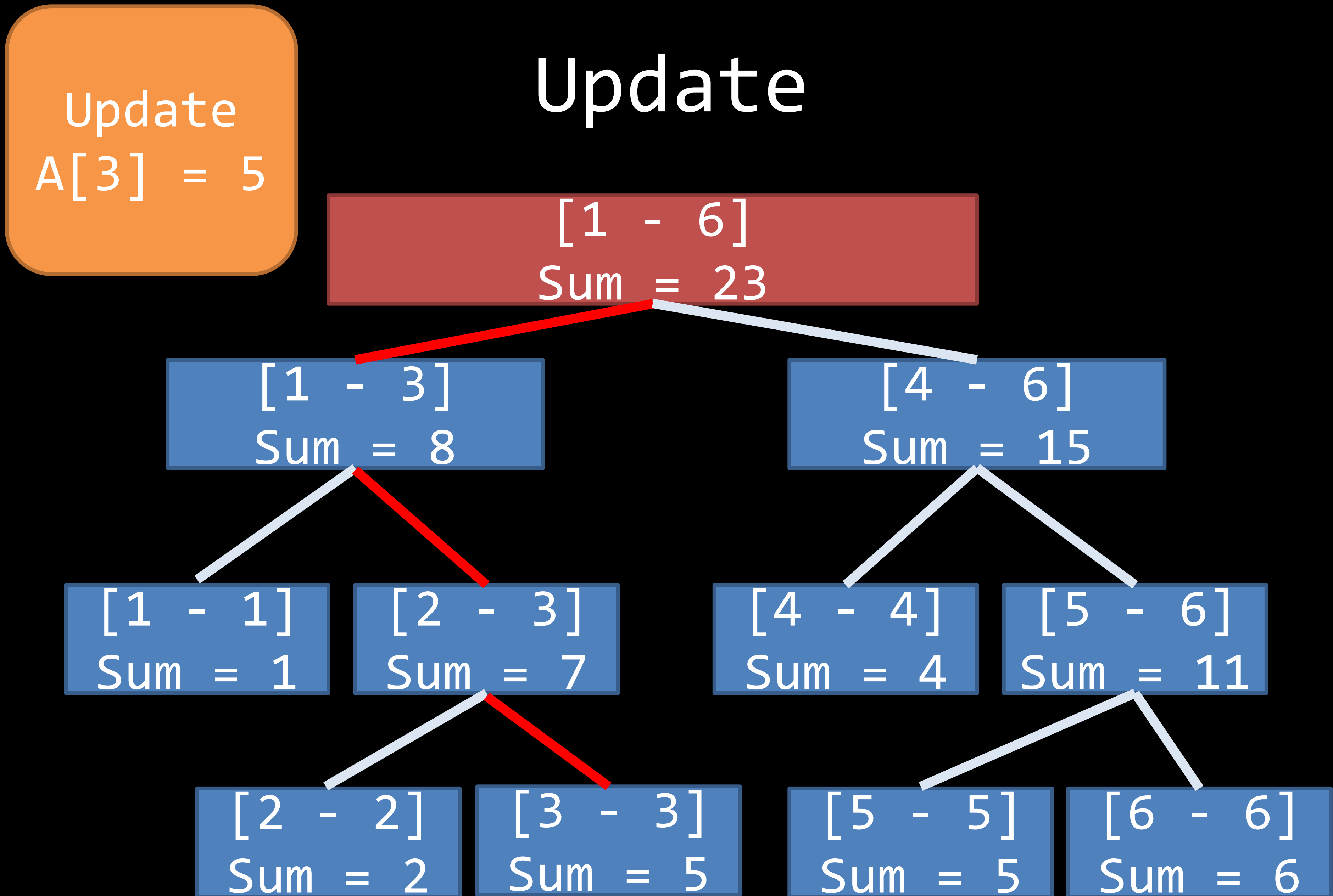

Update
A[3] = 5
[1 - 6]
Sum = 21
[1 - 3]
Sum = 6
[4 - 6]
Sum = 15
[1 - 1]
Sum = 1
[2 - 3]
Sum = 7
[4 - 4]
Sum = 4
[5 - 6]
Sum = 11
[2 - 2]
Sum = 2
[3 - 3]
Sum = 5
[5 - 5]
Sum = 5
[6 - 6]
Sum = 6

Update

Update
A[3] = 5



Update


Update
A[3] = 5
[1 - 6]
Sum = 23
[1 - 3]
Sum = 8
[4 - 6]
Sum = 15
[1 - 1]
Sum = 1
[2 - 3]
Sum = 7
[4 - 4]
Sum = 4
[5 - 6]
Sum = 11
[2 - 2]
Sum = 2
[3 - 3]
Sum = 5
[5 - 5]
Sum = 5
[6 - 6]
Sum = 6

Update

- We visit 1 node at each level
- Height of tree = O(log N)
- Time complexity = O(log N)

Example 2

- We draw N lines one by one on the x-axis [1, N]. Each line has integer end points.
- For each line, find the length that is visible from top

Example 2

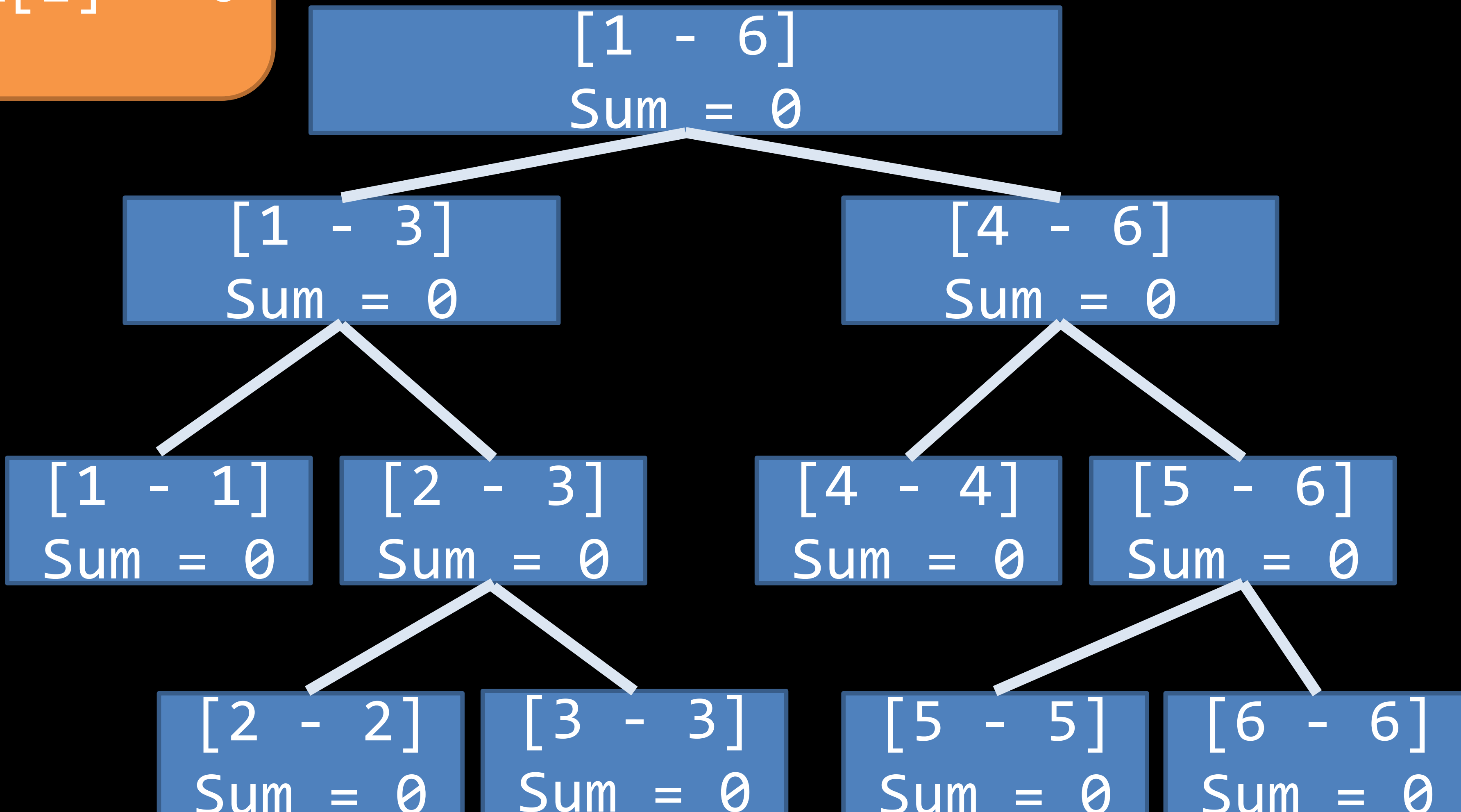
- Process the lines from newest to oldest
- For a line [l, r], we want to know the length that is covered by newer lines

Example 2

- The problem reduce to: at the begin, A[1] = A[2] = … = 0
- There are 2 type of operations
  - A[l] = A[l+1] = … = A[r] = 1
  - A[l] + A[l+1] + … + A[r] = ?

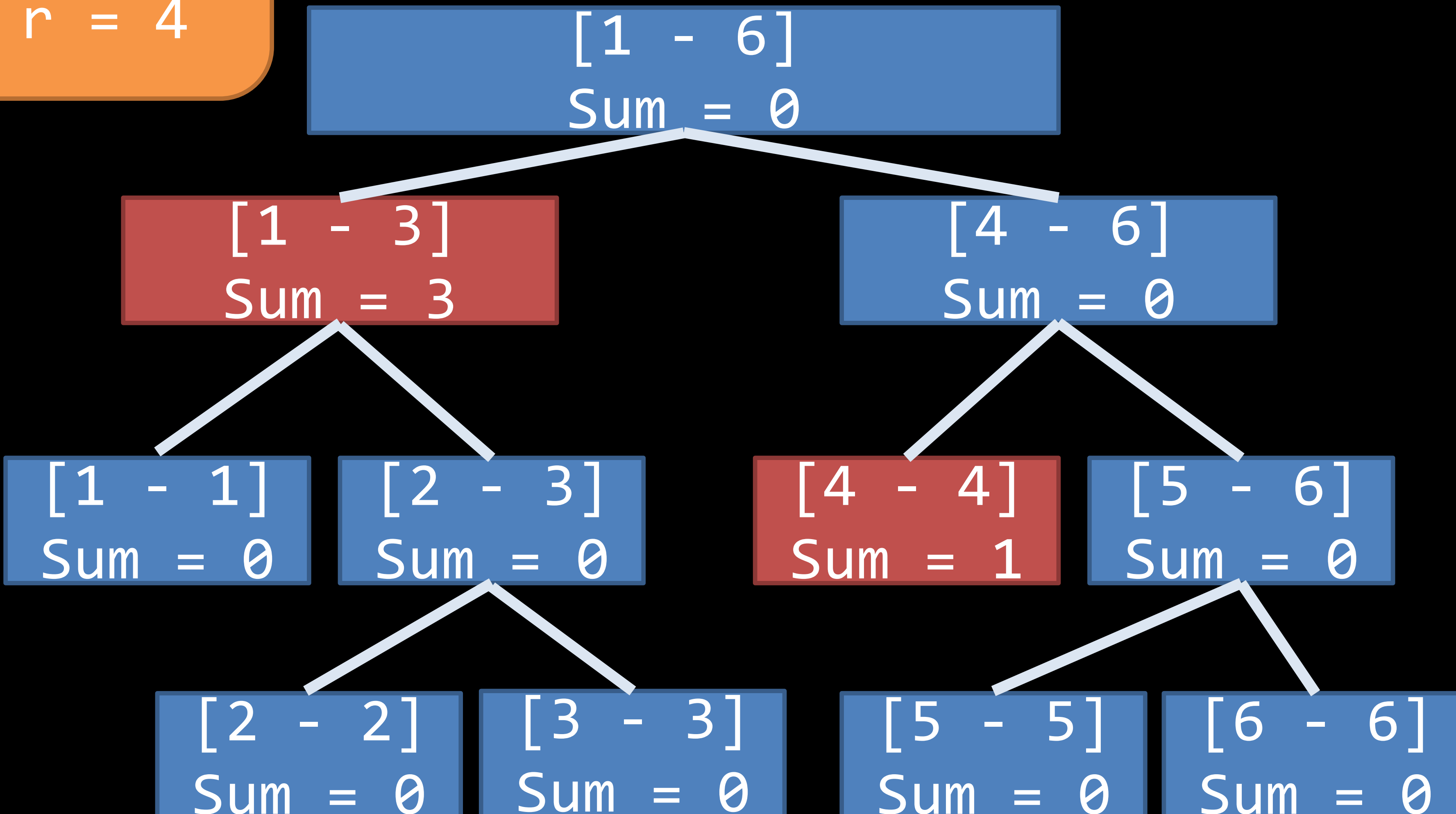
Initialize

Initial
A[i] = 0

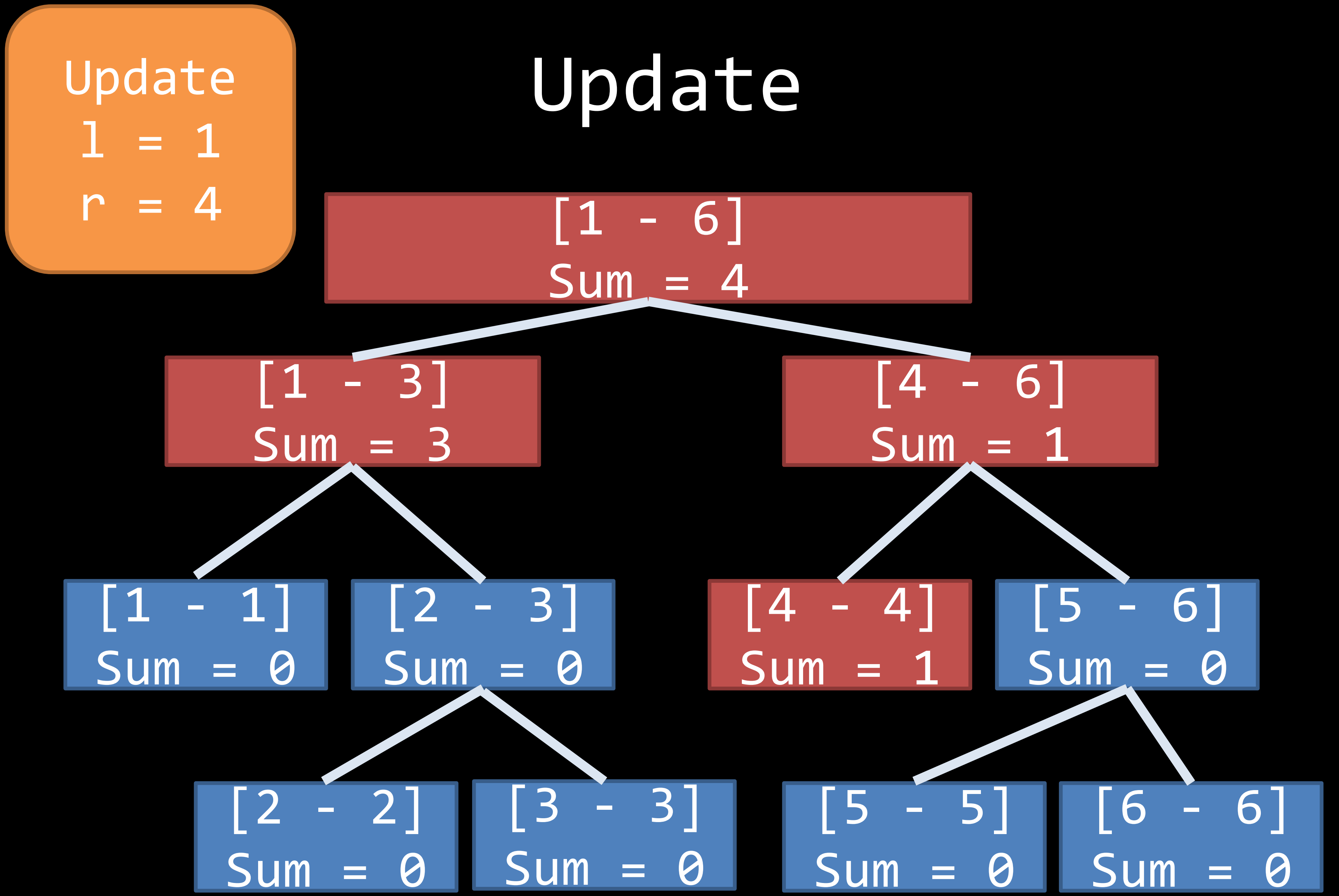


Update
l = 1
r = 4

Update

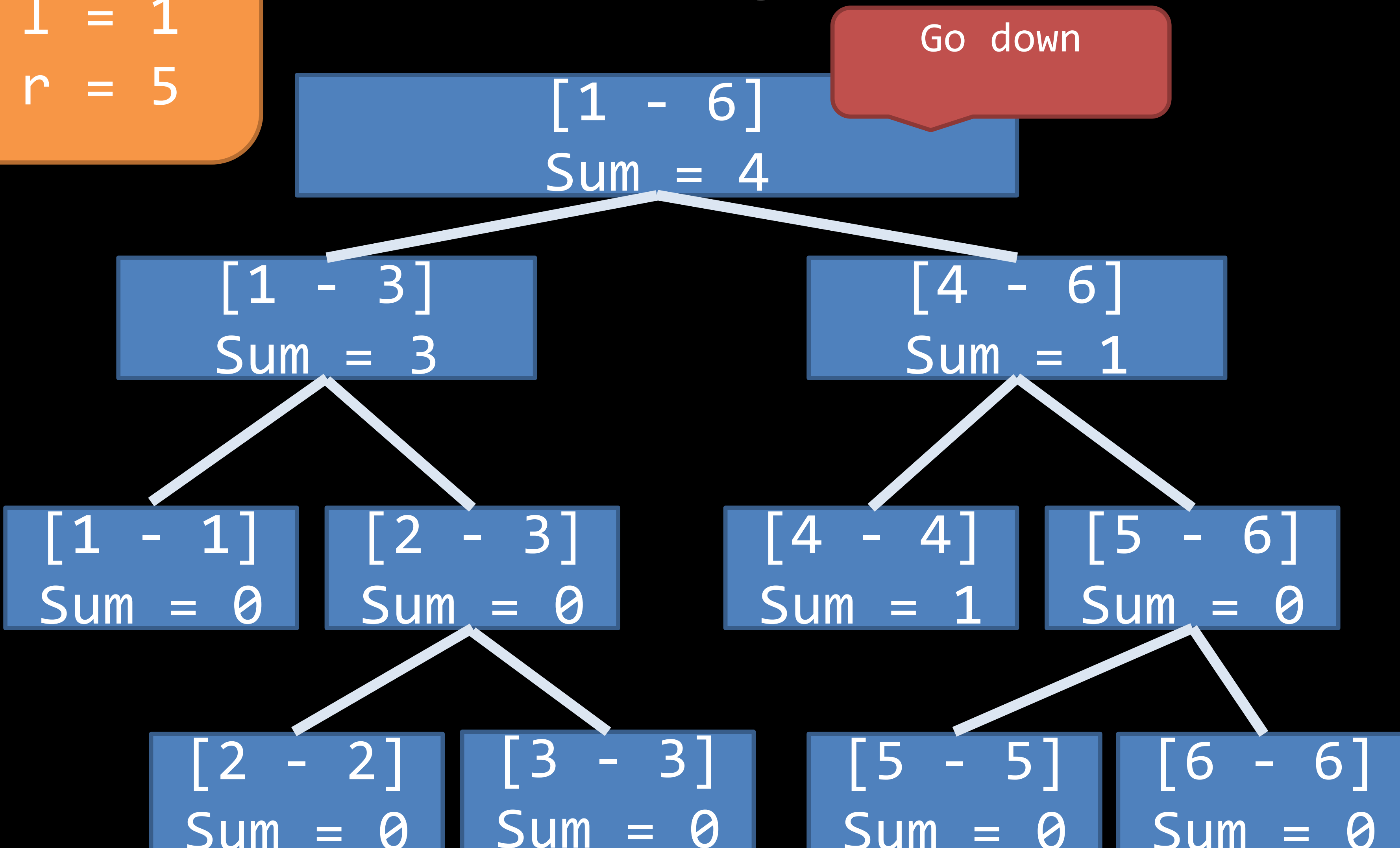

[1 - 6]
Sum = 0
[1 - 3]
Sum = 3
[4 - 6]
Sum = 0
[1 - 1]
Sum = 0
[2 - 3]
Sum = 0
[4 - 4]
Sum = 1
[5 - 6]
Sum = 0
[2 - 2]
Sum = 0
[3 - 3]
Sum = 0
[5 - 5]
Sum = 0
[6 - 6]
Sum = 0

Update


Update
l = 1
r = 4
[1 - 6]
Sum = 4
[1 - 3]
Sum = 3
[4 - 6]
Sum = 1
[1 - 1]
Sum = 0
[2 - 3]
Sum = 0
[4 - 4]
Sum = 1
[5 - 6]
Sum = 0
[2 - 2]
Sum = 0
[3 - 3]
Sum = 0
[5 - 5]
Sum = 0
[6 - 6]
Sum = 0

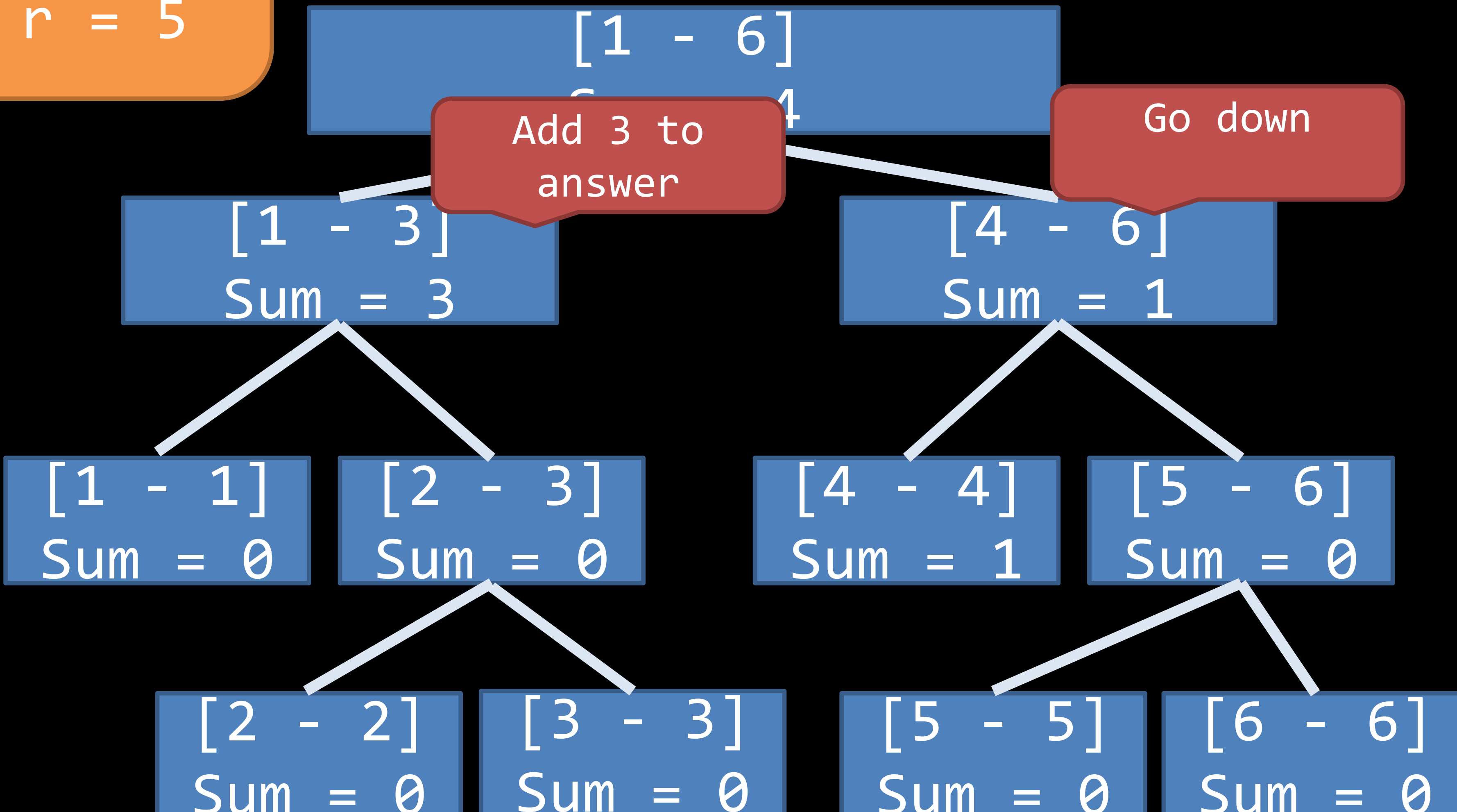
Query

Query
l = 1
r = 5

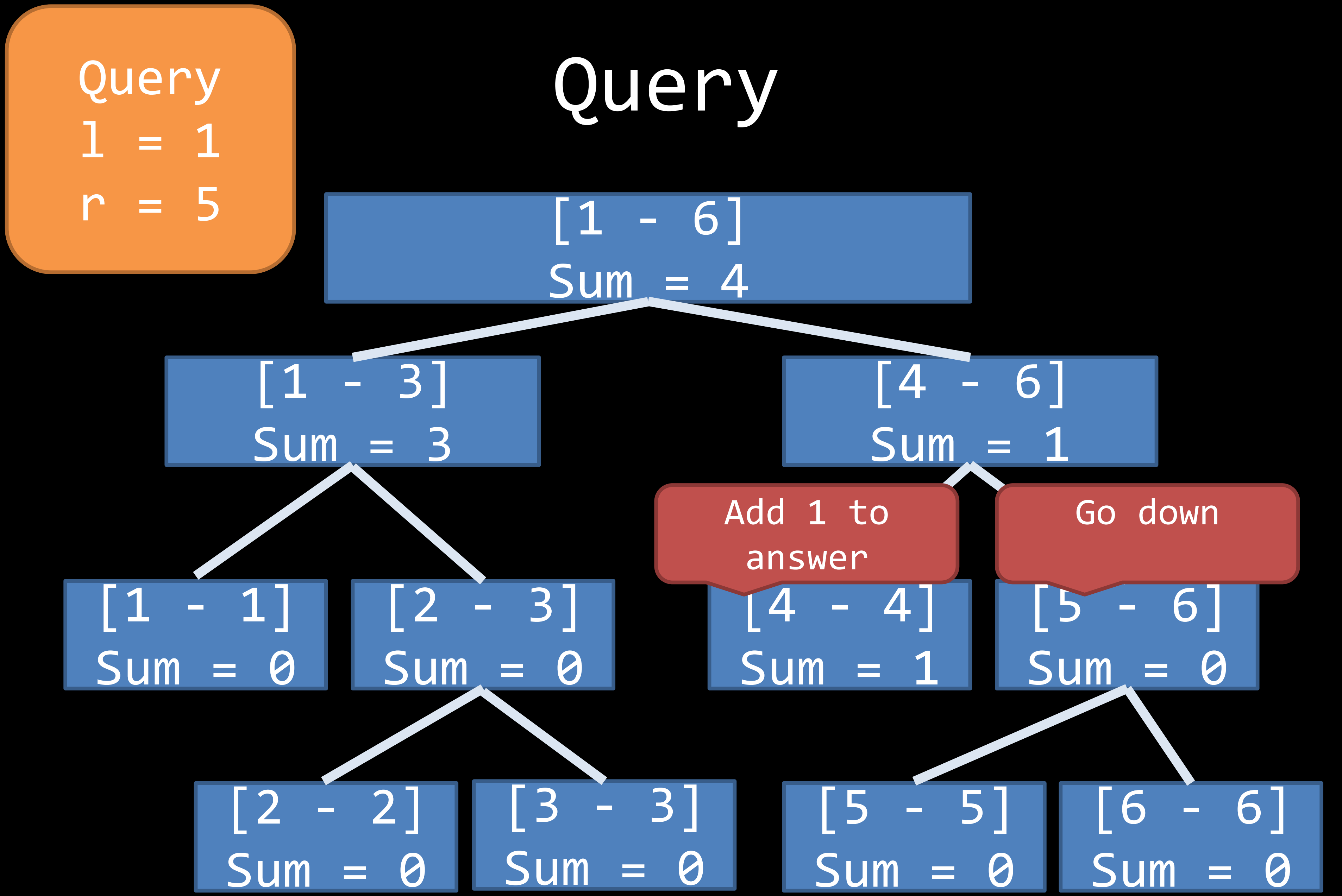


Query

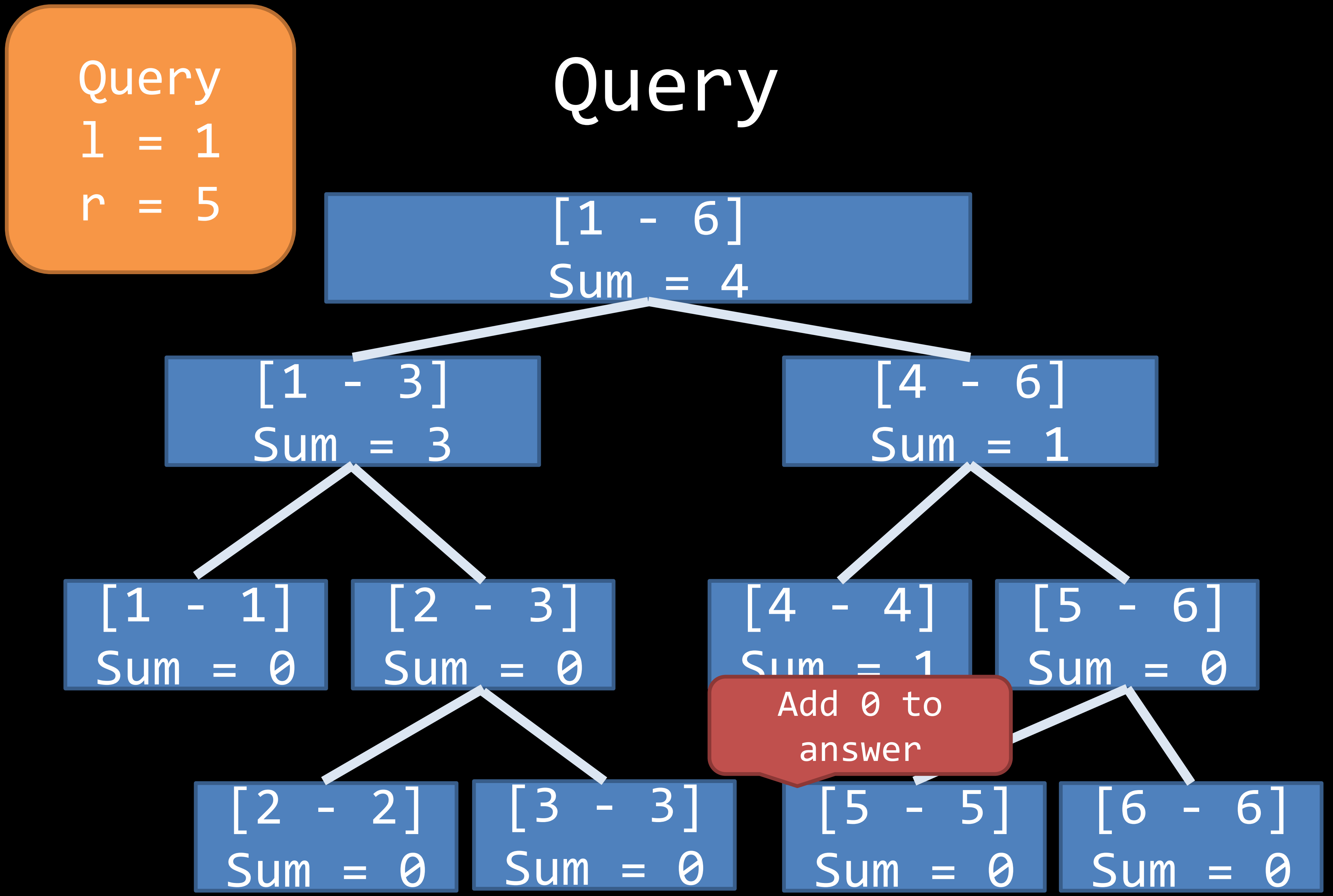
Query
l = 1
r = 5



Query

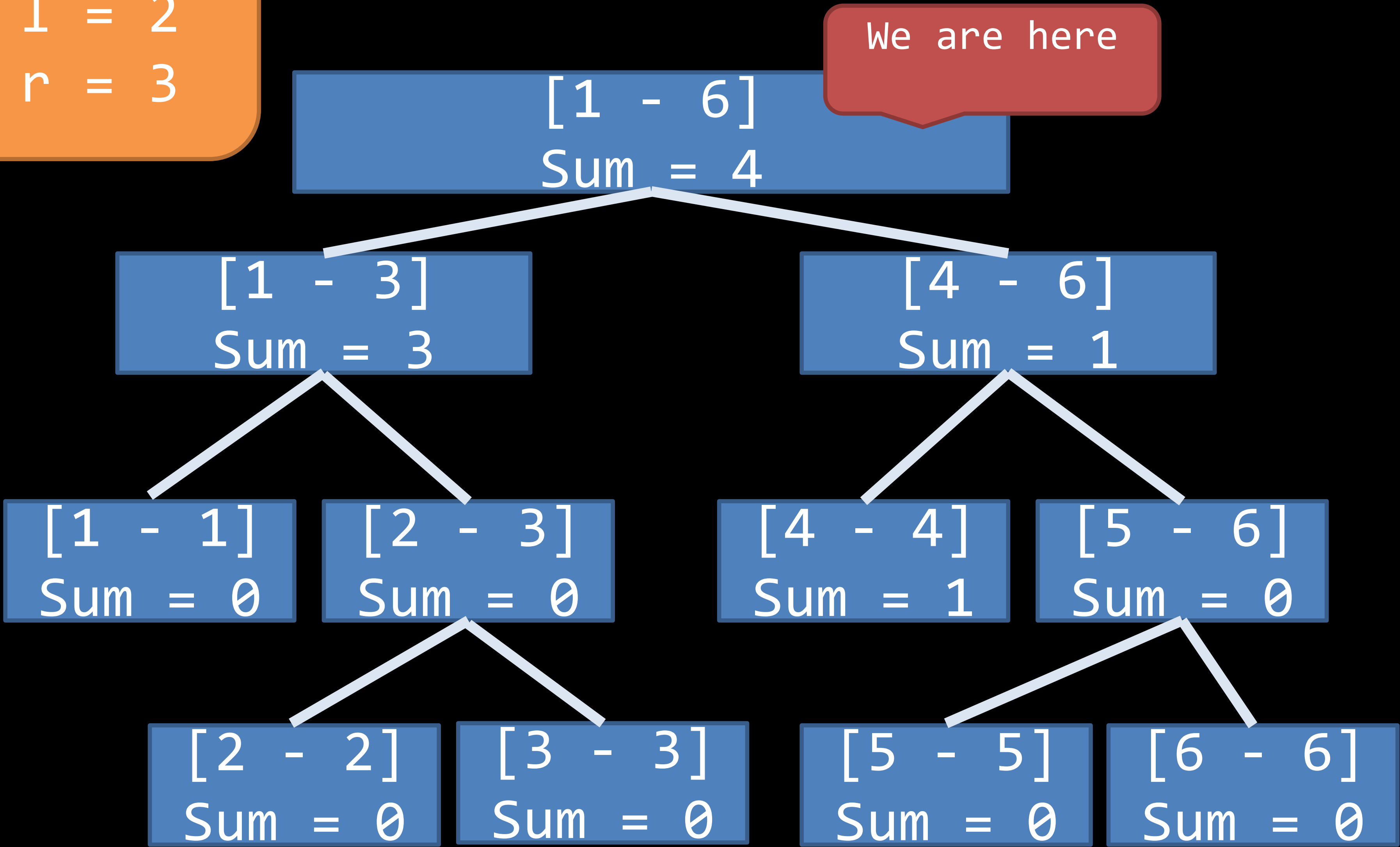

Query
l = 1
r = 5
[1 - 6]
Sum = 4
[1 - 3]
Sum = 3
[4 - 6]
Sum = 1
Add 1 to answer
Go down
[1 - 1]
Sum = 0
[2 - 3]
Sum = 0
[4 - 4]
Sum = 1
[5 - 6]
Sum = 0
[2 - 2]
Sum = 0
[3 - 3]
Sum = 0
[5 - 5]
Sum = 0
[6 - 6]
Sum = 0

Query


Query
l = 1
r = 5
[1 - 6]
Sum = 4
[1 - 3]
Sum = 3
[4 - 6]
Sum = 1
[1 - 1]
Sum = 0
[2 - 3]
Sum = 0
[4 - 4]
Sum = 1
[5 - 6]
Sum = 0
Add 0 to answer
[2 - 2]
Sum = 0
[3 - 3]
Sum = 0
[5 - 5]
Sum = 0
[6 - 6]
Sum = 0

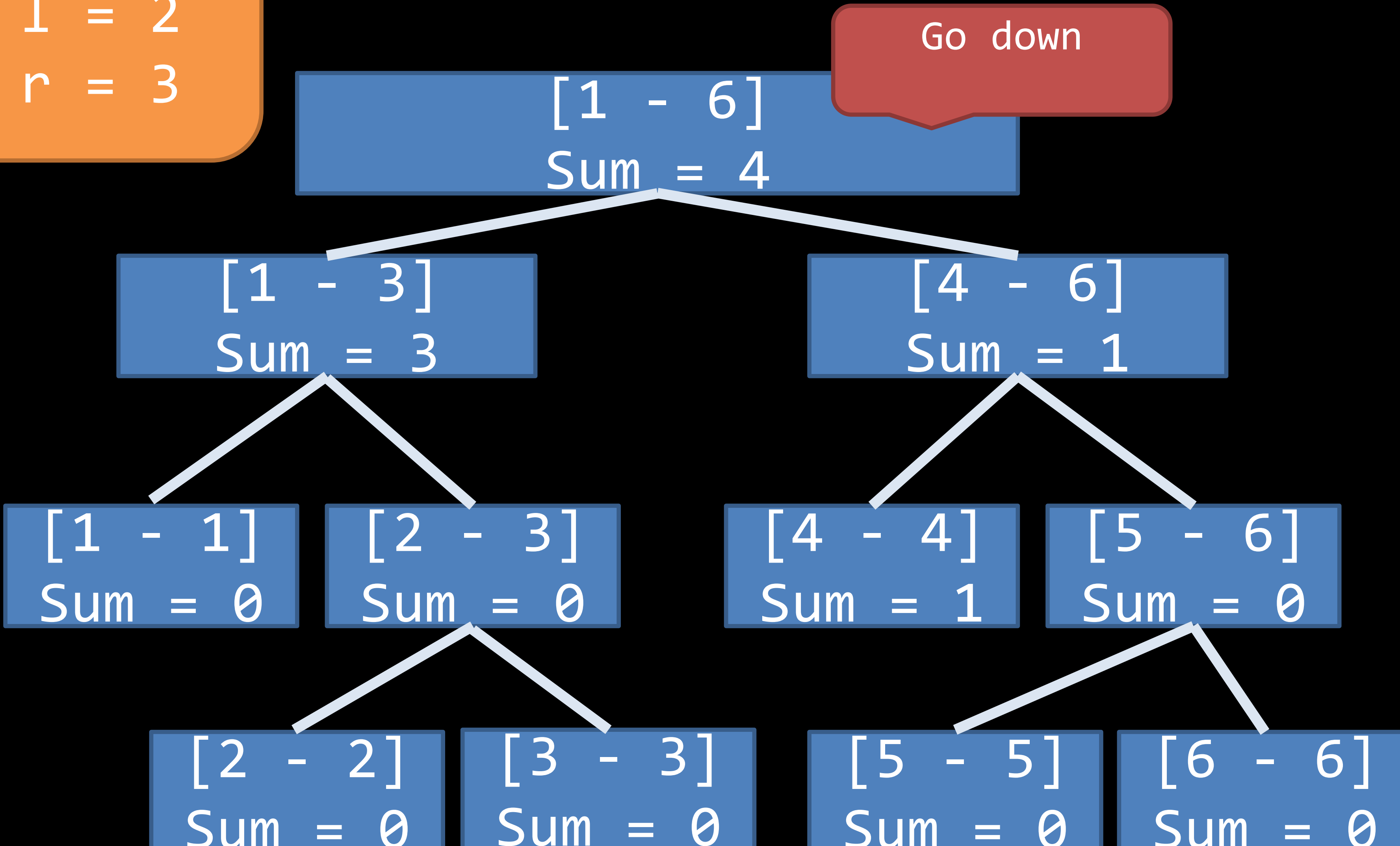
Query

Query
l = 2
r = 3



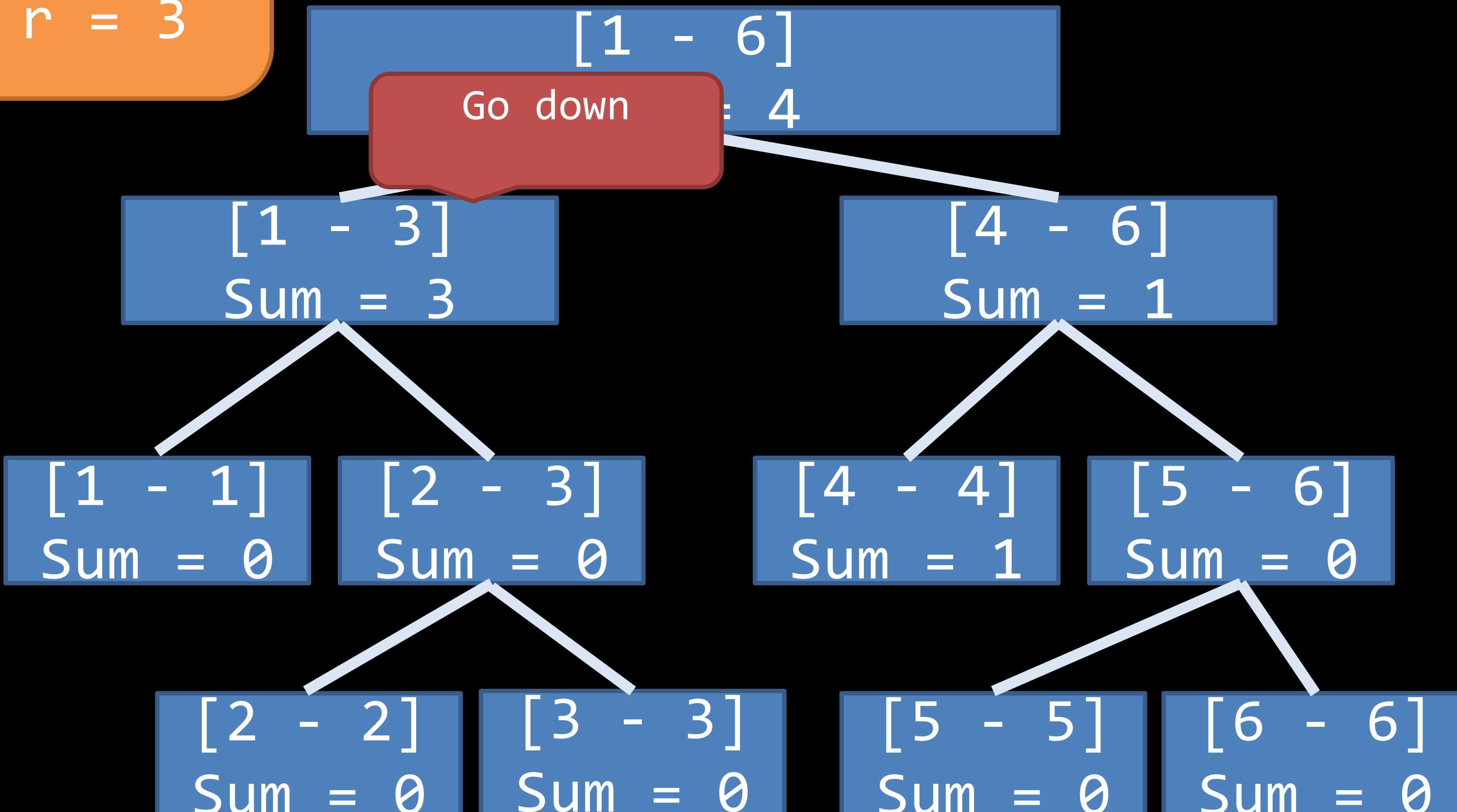
Query

Query
l = 2
r = 3



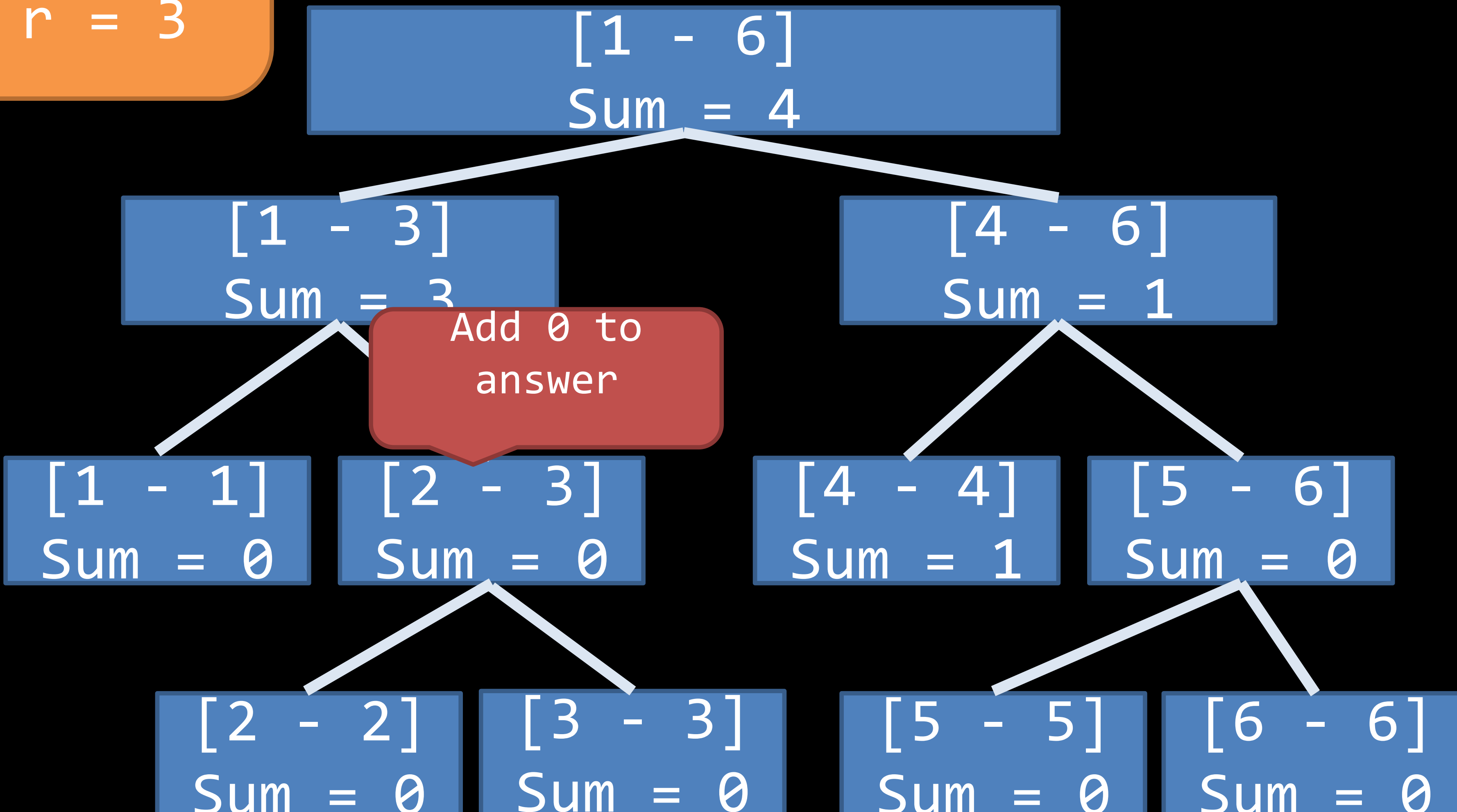
Query

Query
l = 2
r = 3



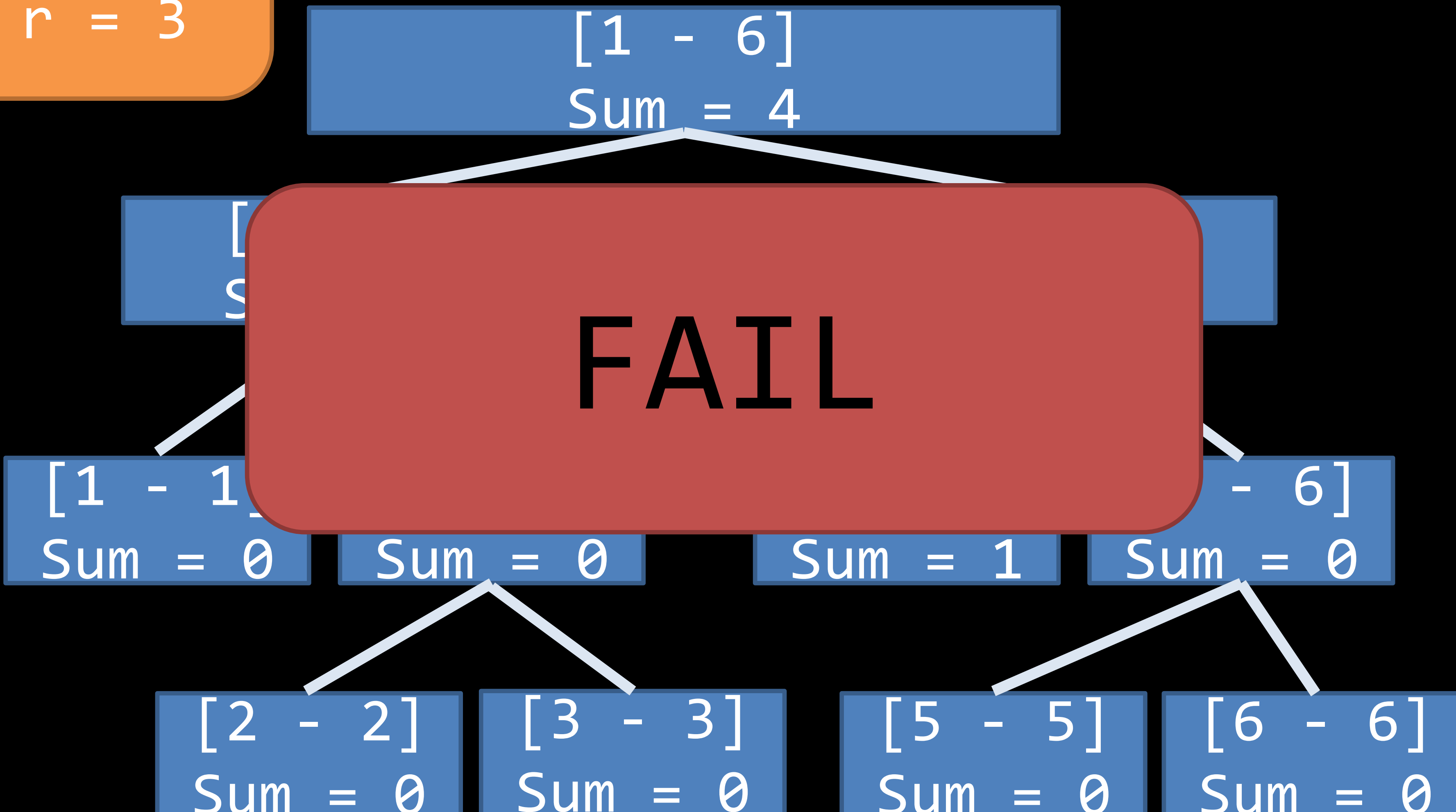
Query

Query
l = 2
r = 3



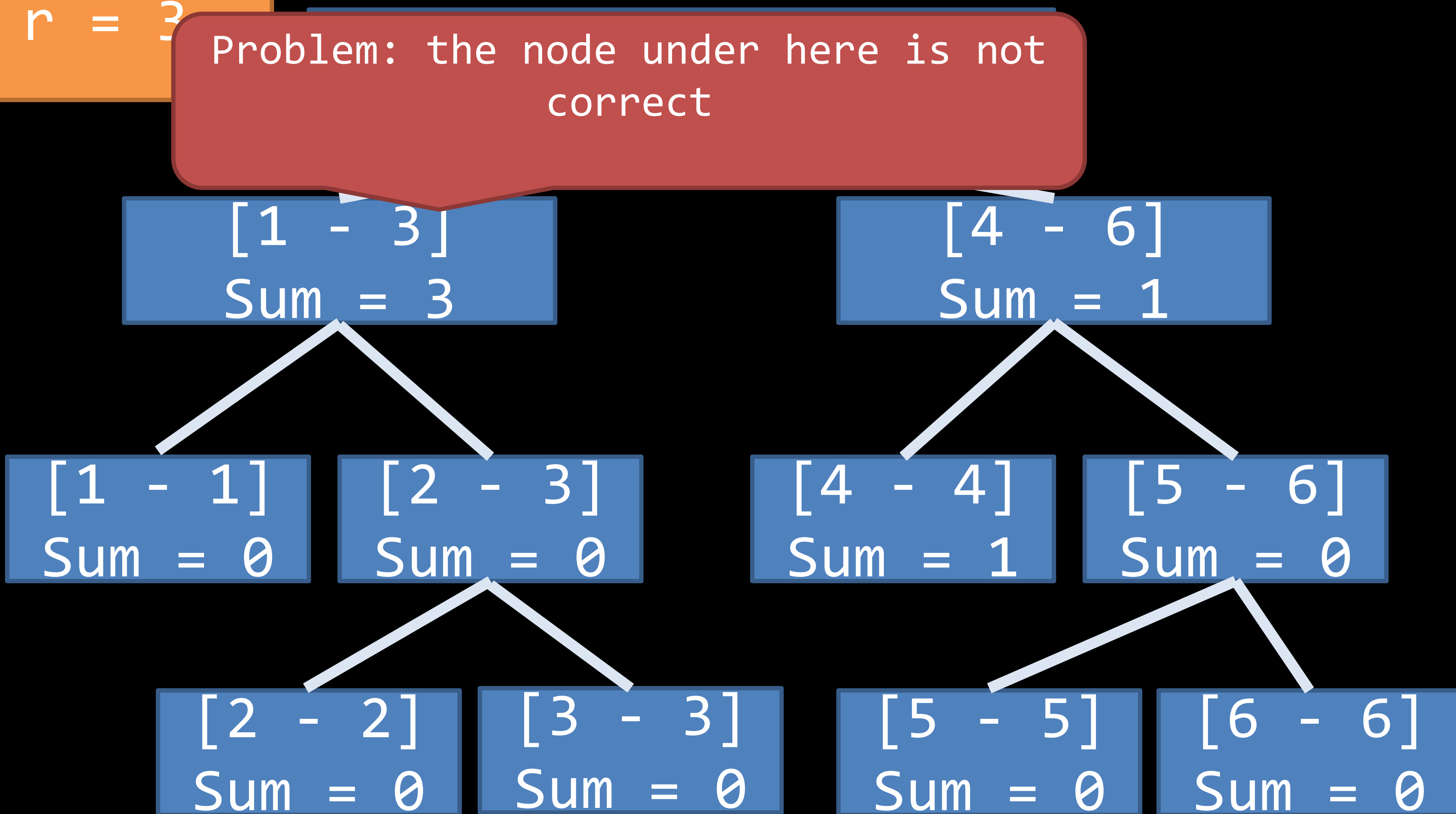
Query

Query
l = 2
r = 3



Query

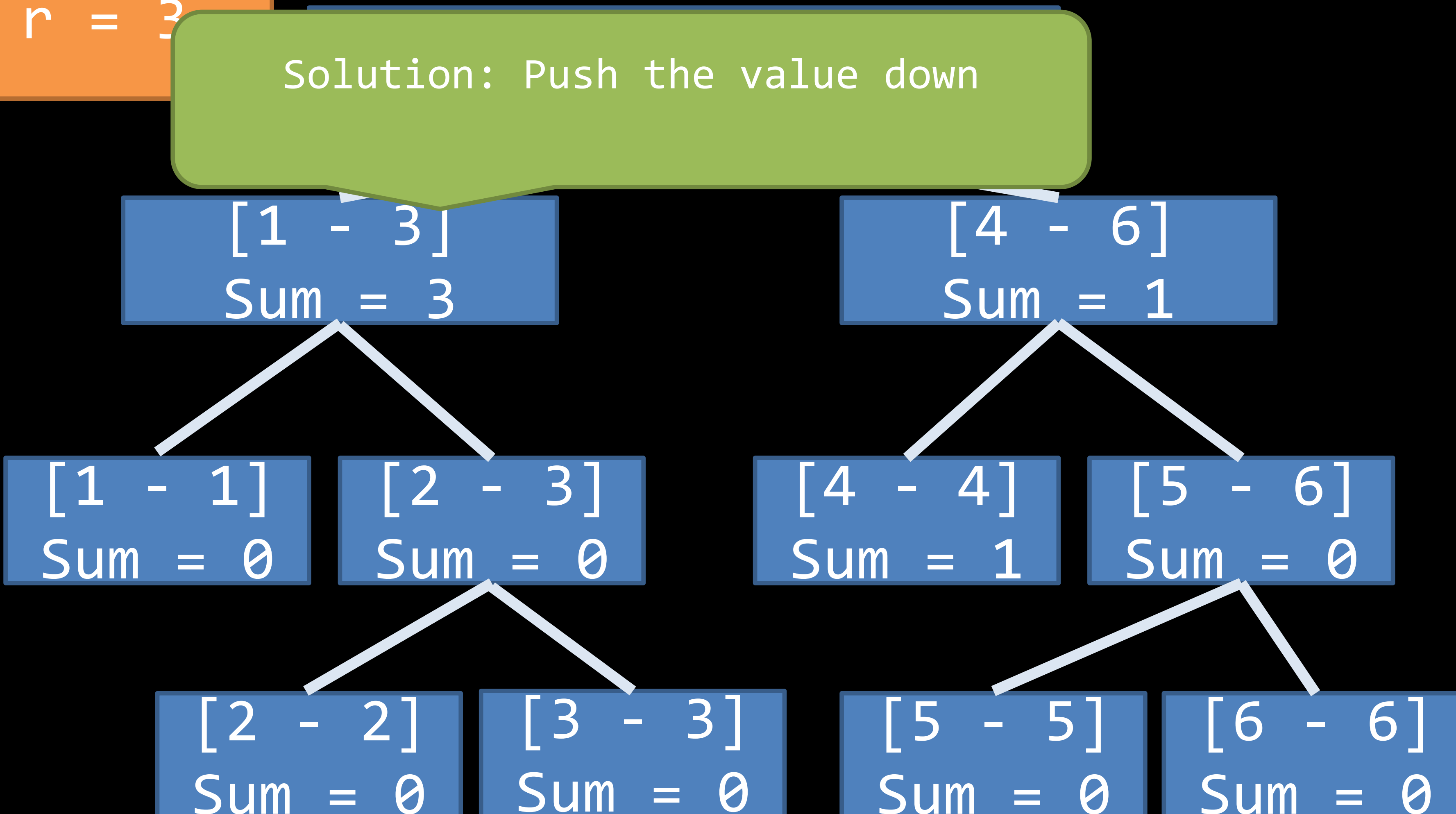
Query
l = 2
r = 3



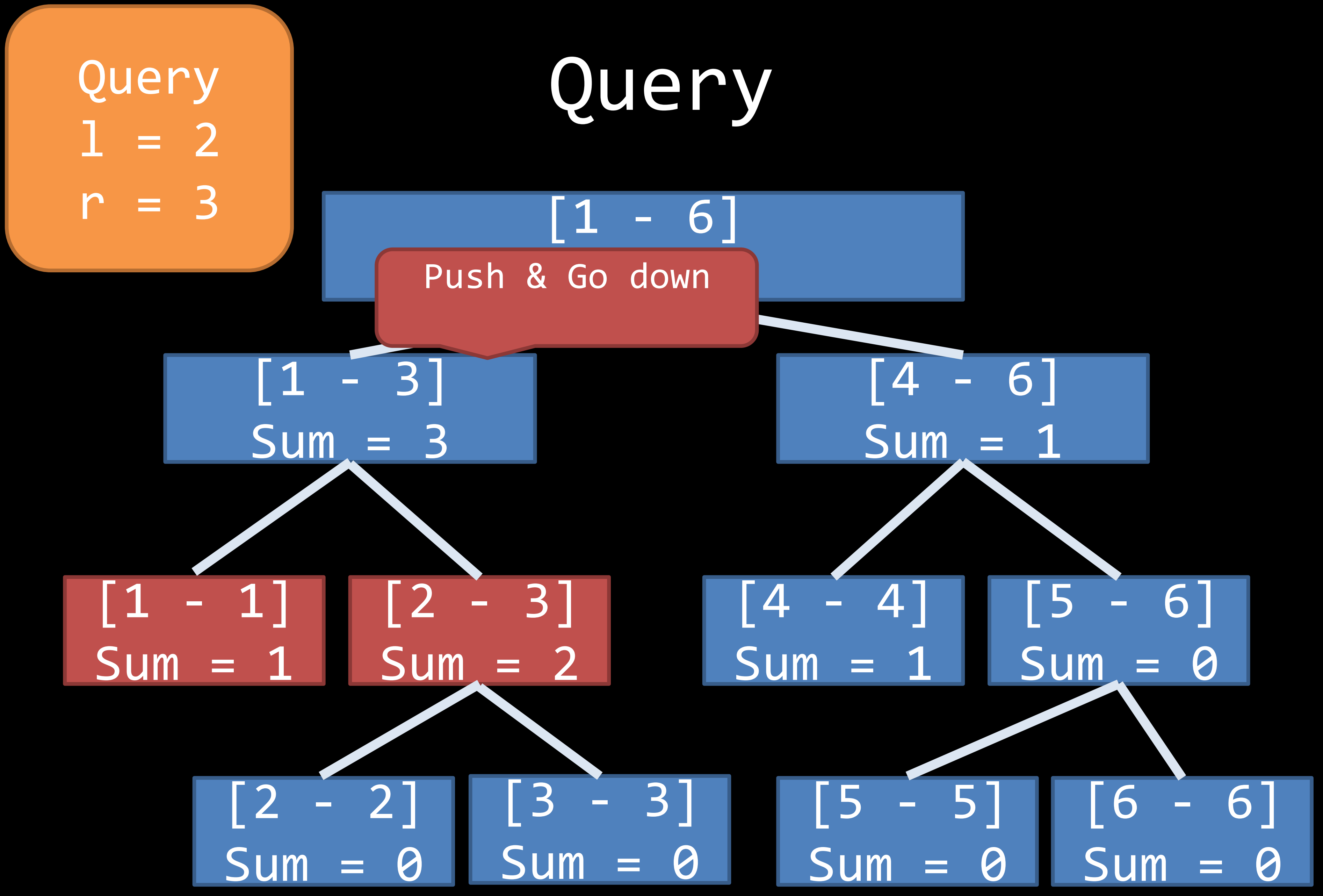
Query

Query
l = 2
r = 3

Solution: Push the value down

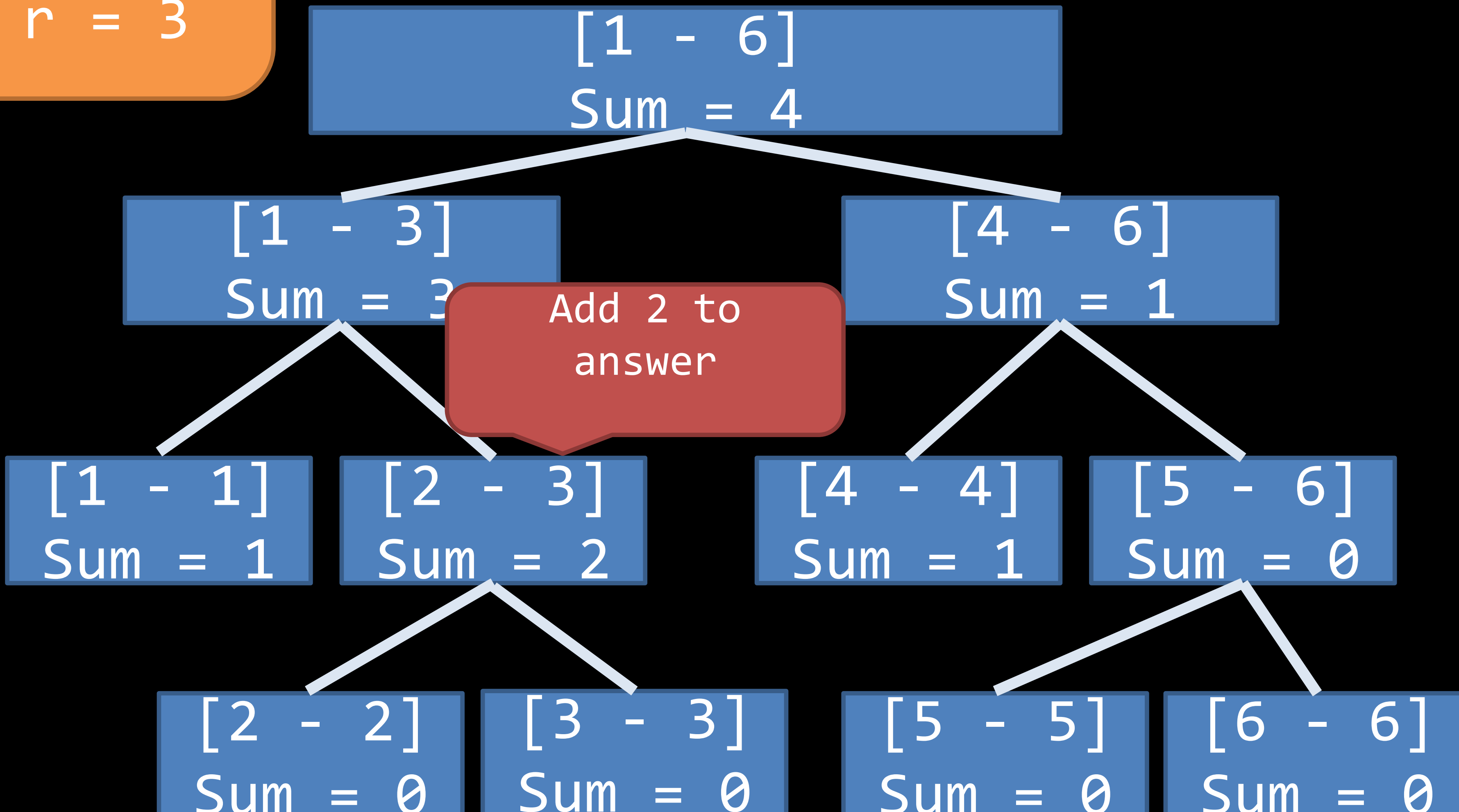


Query


Query
l = 2
r = 3
[1 - 6]
Push & Go down
[1 - 3]
Sum = 3
[4 - 6]
Sum = 1
[1 - 1]
Sum = 1
[2 - 3]
Sum = 2
[4 - 4]
Sum = 1
[5 - 6]
Sum = 0
[2 - 2]
Sum = 0
[3 - 3]
Sum = 0
[5 - 5]
Sum = 0
[6 - 6]
Sum = 0

Query
l = 2
r = 3

Query



Example 3

- Given A[1..n]
- There are 2 types of operations:
  - A[x] := v
  - Find the maximum continuous subsequence sum (i.e. find l, r such that A[l] + A[l+1] + … + A[r] is maximal)

Example 3

- What information a node need to store
- MCSS of the interval?
- Sum of the interval?

Example 3

- What information a node need to store
- Sum
- Prefix MCSS
- Suffix MCSS
- MCSS

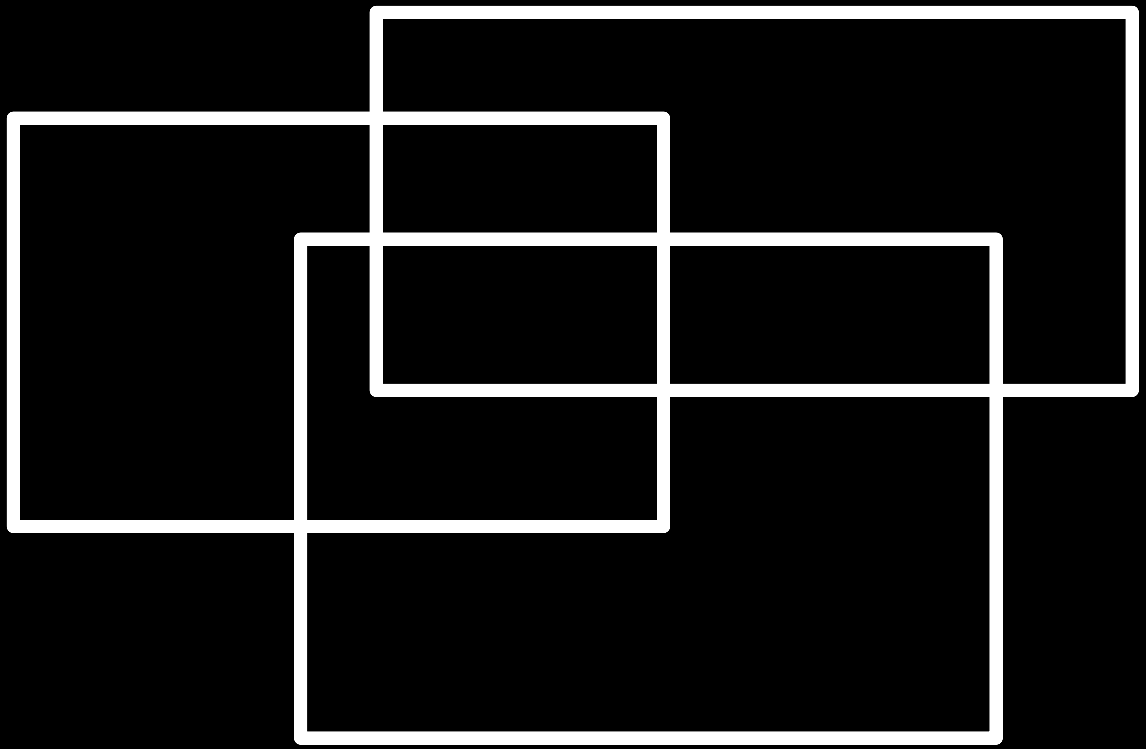
Example 3

- How to update?
- Left node: trivial
- Let l and r be left and right children
- Sum = Sum[l] + Sum[r]
- PrefixMCSS = max(PrefixMCSS[l], Sum[l] + PrefixMCSS[r])
- SuffixMCSS = max(SuffixMCSS[r], Sum[r] + SuffixMCSS[l])
- MCSS = max(MCSS[l], MCSS[r], SuffixMCSS[l] + PrefixMCSS[r])

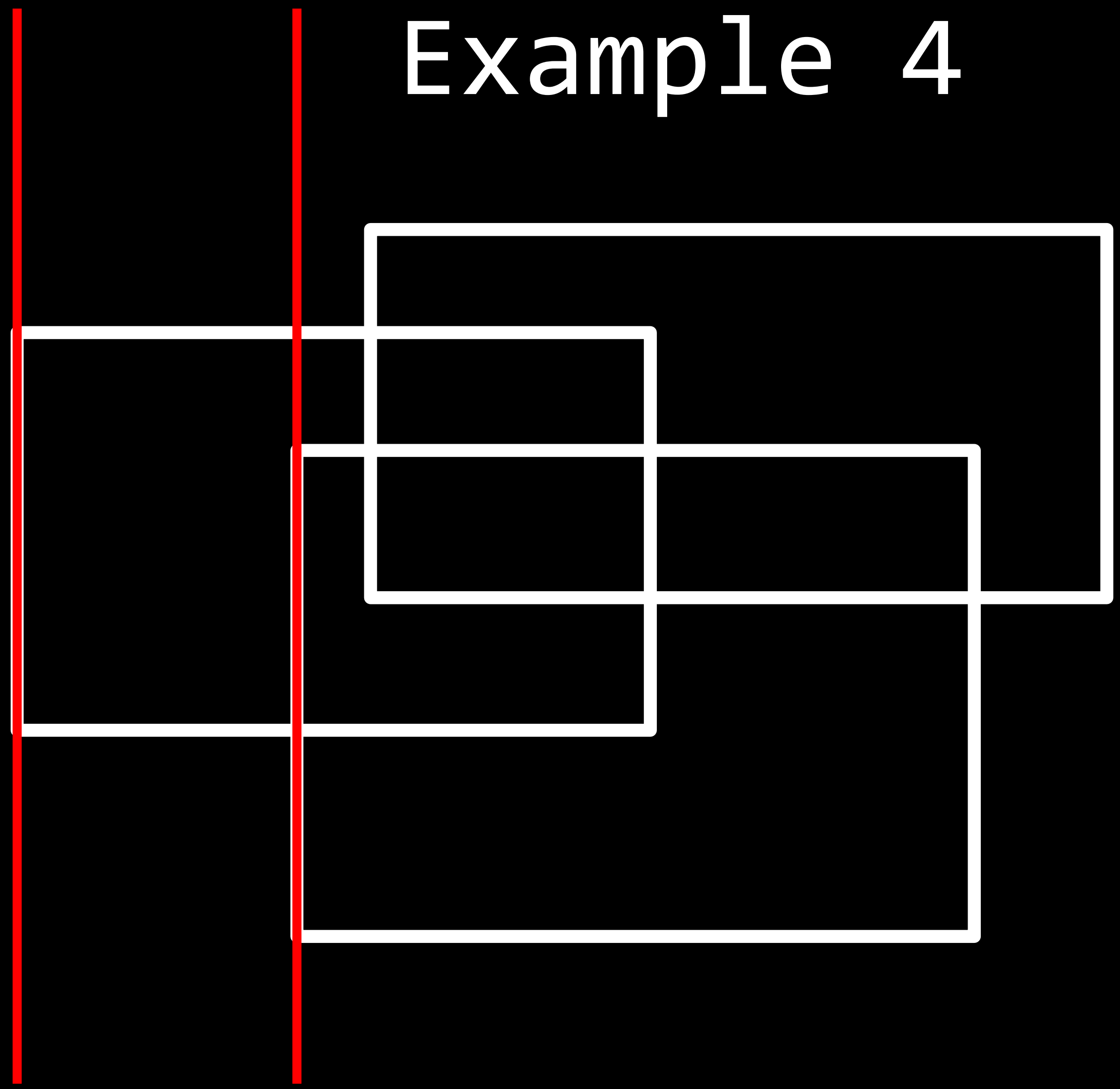
Example 4

- Given N rectangles in 2D plane
- Every vertex has integer coordinates
- Every edge parallel to X or Y Axis
- Find the union area

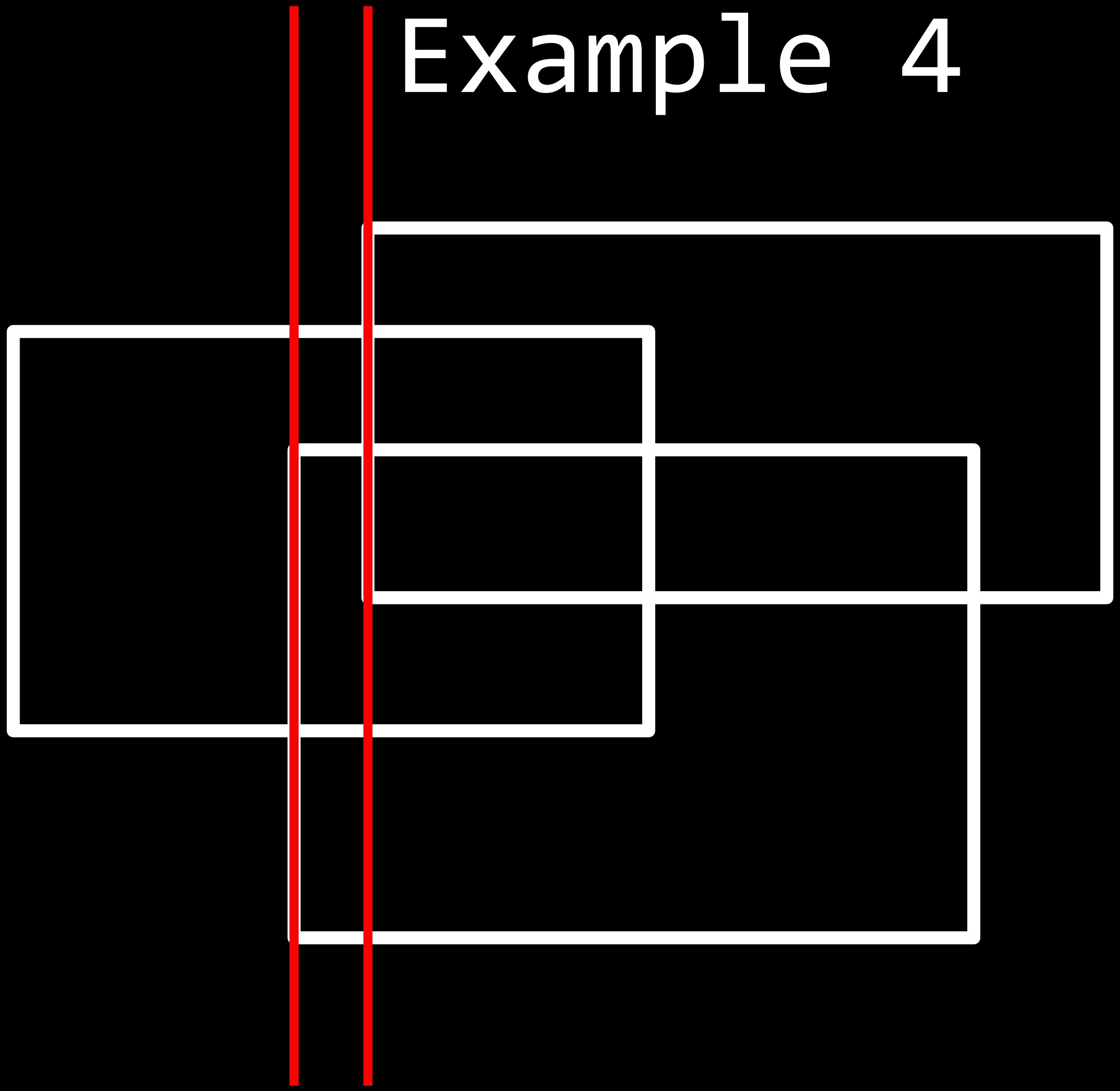
Example 4



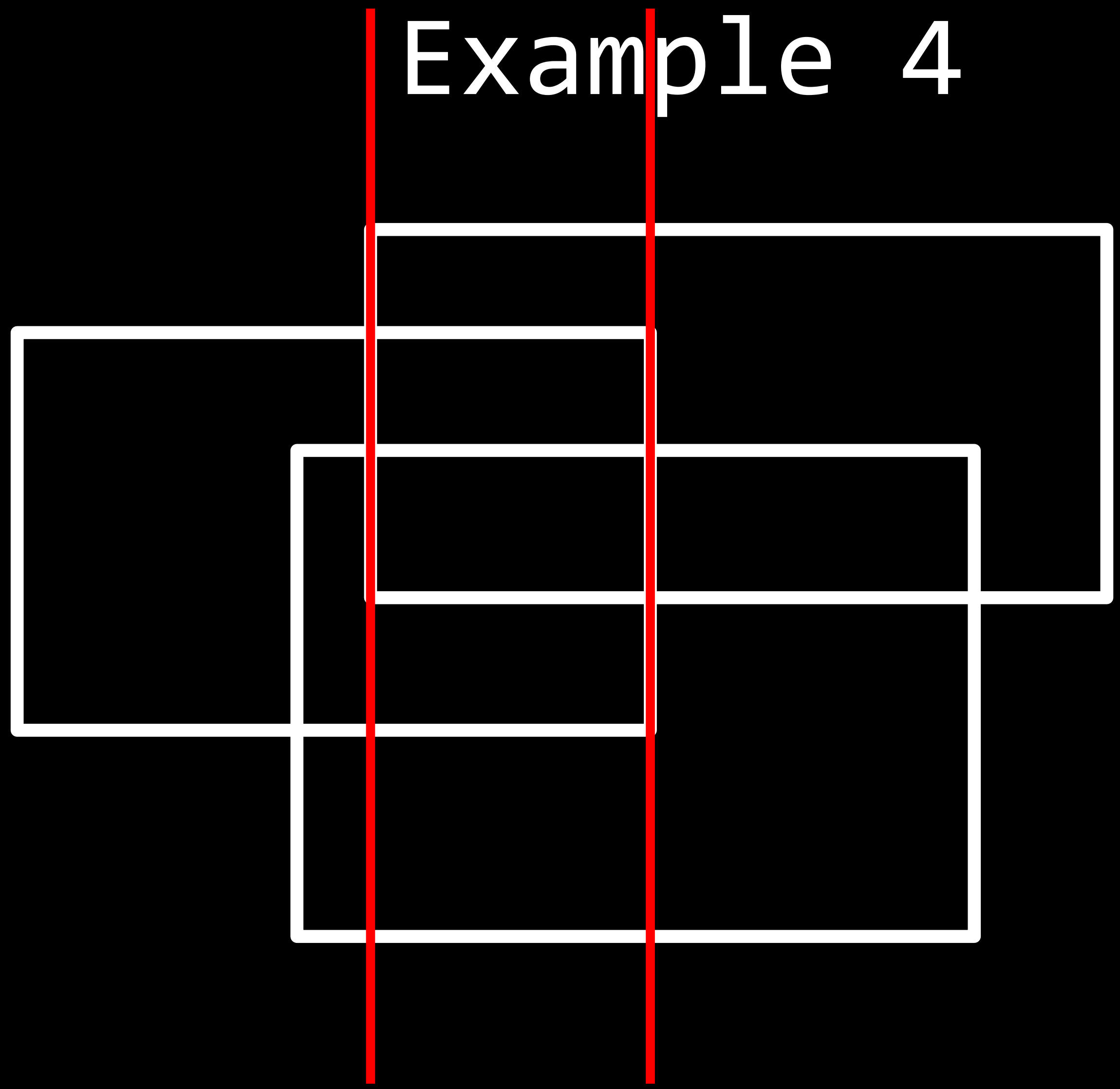
Example 4



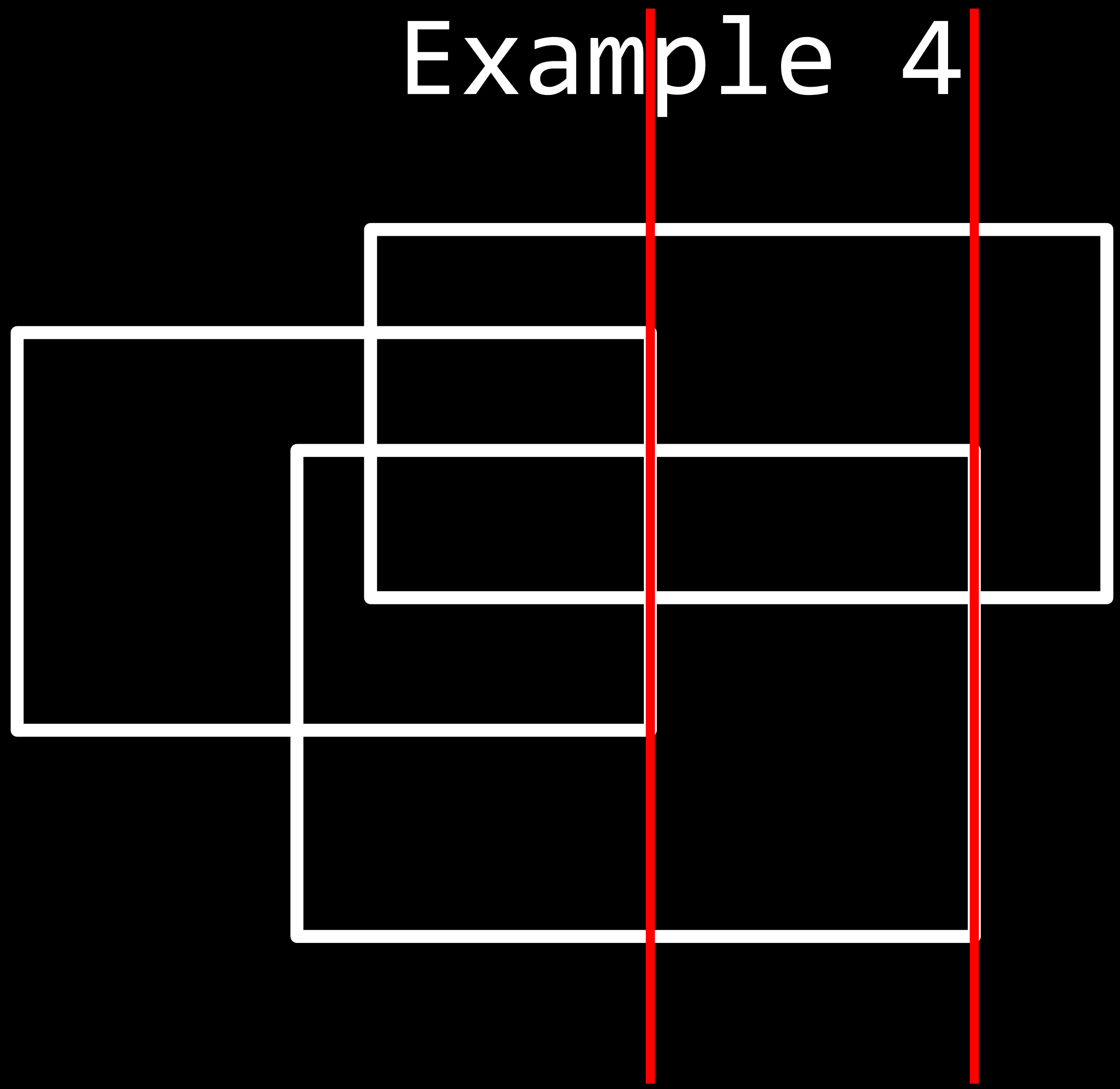
Example 4



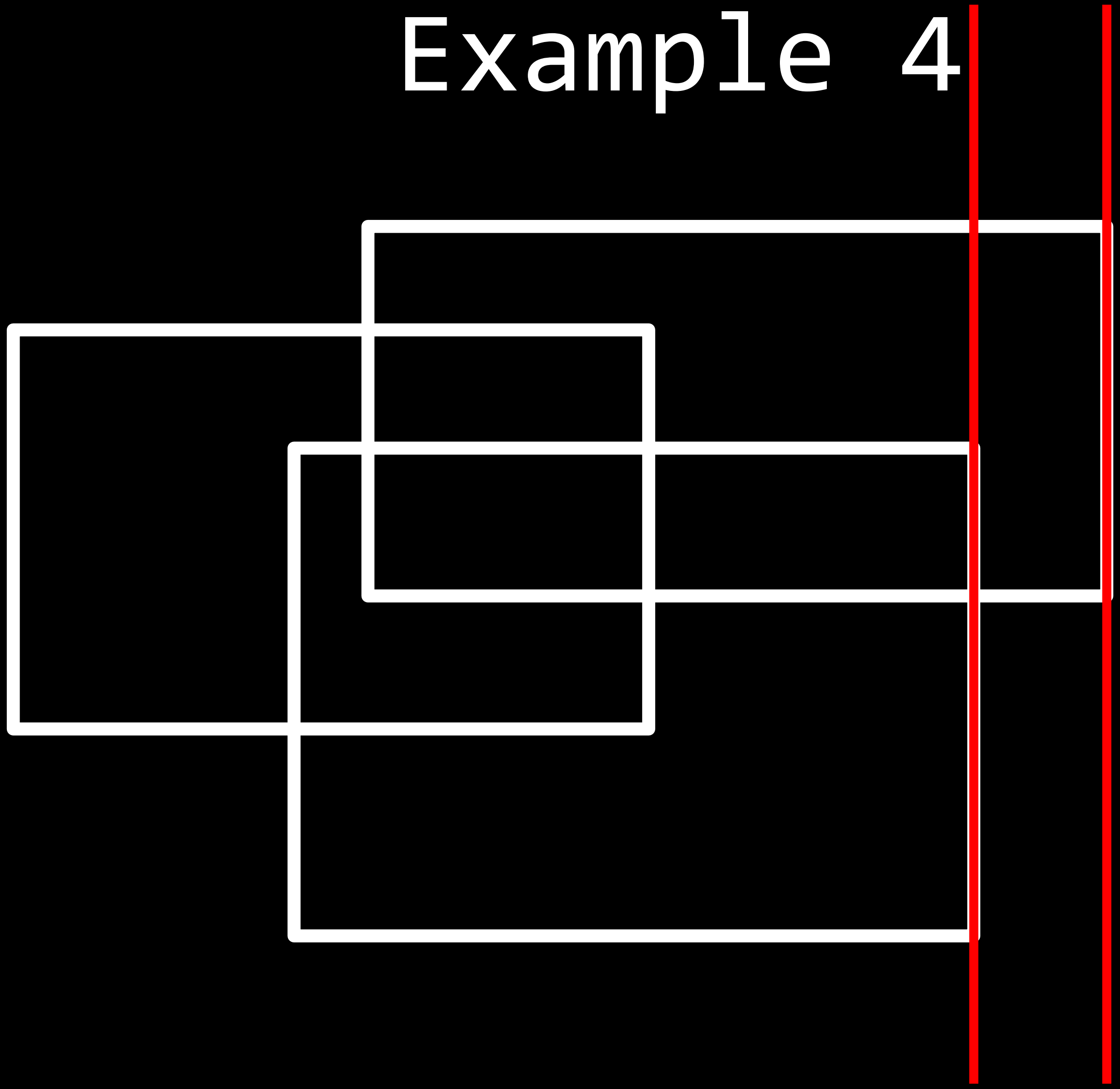
Example 4



Example 4



Example 4



Example 4

- Reduce the problem to supports the follow operations on 1..N
- Add 1 to A[l], A[l+1], …, A[r]
- Find the numbers of non-zero entries in A[1..N]

Problem

- In the previous examples, the ‘segment’ is always a small integer
- What if the segments are large or not integer?

Discretization

- Assume we have segments:
  - [4.3, 7.2]
  - [0.1, 10^{15}]
  - [7.2, 99]
- We can modify it to:
  - [2, 3]
  - [1, 5]
  - [3, 4]