Segment trees

Giulio Ermanno Pibiri
giulio.pibiri@di.unipi.it

18/10/2017

Problems we would like to solve efficiently

A	**3**	**1**	**-2**	**4**	**6**	**13**	**2**	**0**
	0	1	2	3	4	5	6	7

Problems we would like to solve efficiently

A	3	1	-2	4	6	13	2	0
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

Problems we would like to solve efficiently

A	**3**	**1**	**-2**	**4**	**6**	**13**	**2**	**0**
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

`sum`(3) = 6
`sum`(5) = 25

Problems we would like to solve efficiently

A	**3**	**1**	**-2**	**4**	**6**	**13**	**2**	**0**
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

`sum`(3) = 6

`sum`(5) = 25

`update`(2, 5)

Problems we would like to solve efficiently

A	**3**	**1**	**5**	**4**	**6**	**13**	**2**	**0**
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

`sum`(3) = 6
`sum`(5) = 25
`update`(2, 5)

Problems we would like to solve efficiently

A	**3**	**1**	**5**	**4**	**6**	**13**	**2**	**0**
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

`sum`(3) = 6
`sum`(5) = 25
`update`(2, 5)
`sum`(5) = 32

Problems we would like to solve efficiently

A	3	1	-2	4	6	13	2	0
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

`sum`(3) = 6
`sum`(5) = 25
`update`(2, 5)
`sum`(5) = 32

Problems we would like to solve efficiently

A	3	1	-2	4	6	13	2	0
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

`sum`(3) = 6
`sum`(5) = 25
`update`(2, 5)
`sum`(5) = 32

Range `MIN` (`MAX`) queries

Report the `MIN` (`MAX`) in A[i,j]

Problems we would like to solve efficiently

A	**3**	**1**	**-2**	**4**	**6**	**13**	**2**	**0**
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

`sum`(3) = 6

`sum`(5) = 25

`update`(2, 5)

`sum`(5) = 32

Range `MIN` (`MAX`) queries

Report the `MIN` (`MAX`) in A[i,j]

`min`(1,3) = -2

`max`(4,7) = 0

Problems we would like to solve efficiently

A	3	1	-2	4	6	13	2	0
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

`sum`(3) = 6
`sum`(5) = 25
`update`(2, 5)
`sum`(5) = 32

Range `MIN` (`MAX`) queries

Report the `MIN` (`MAX`) in A[i,j]

`min`(1,3) = -2
`max`(4,7) = 0

Range `SUM` queries

Report the sum of the elements in A[i,j]

Problems we would like to solve efficiently

A	**3**	**1**	**-2**	**4**	**6**	**13**	**2**	**0**
	0	1	2	3	4	5	6	7

(Static) Prefix sums

- `sum`(i) reports the sum of the first i+1 integers
- `update`(i, x) sets A[i] to x

`sum`(3) = 6
`sum`(5) = 25
`update`(2, 5)
`sum`(5) = 32

Range `MIN` (`MAX`) queries

Report the `MIN` (`MAX`) in A[i,j]

`min`(1,3) = -2
`max`(4,7) = 0

Range `SUM` queries

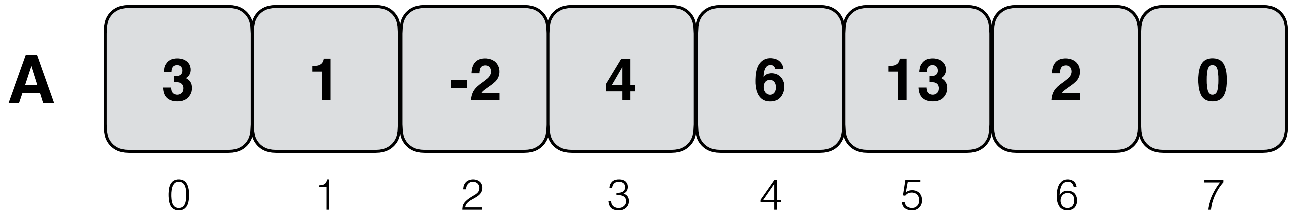
Report the sum of the elements in A[i,j]

`sum`(1,3) = 3
`sum`(4,7) = 21

Any solutions?

A	**3**	**1**	**-2**	**4**	**6**	**13**	**2**	**0**
	0	1	2	3	4	5	6	7

Any solutions?



1. Do nothing
2. Pre-calculate all queries

Any solutions?

A	3	1	-2	4	6	13	2	0
	0	1	2	3	4	5	6	7

1. Do nothing
2. Pre-calculate all queries

(Static) Prefix sums

1.
   `update`: O(1)
   `sum`: O(n)
   Space: no auxiliary space
2.
   `update`: O(n)
   `sum`: O(1)
   Space: no auxiliary space

3	4	2	6	12	25	27	27
0	1	2	3	4	5	6	7

Any solutions?

A							
3	**1**	**-2**	**4**	**6**	**13**	**2**	**0**
0	1	2	3	4	5	6	7

1. Do nothing
2. Pre-calculate all queries

(Static) Prefix sums

1.
   `update`: O(1)
   `sum`: O(n)
   Space: no auxiliary space
2.
   `update`: O(n)
   `sum`: O(1)
   Space: no auxiliary space

3	**4**	**2**	**6**	**12**	**25**	**27**	**27**
0	1	2	3	4	5	6	7

Range `MIN` (`MAX`) and `SUM` queries

1.
   Query time: O(n)
   Space: no auxiliary space
2.
   Query time: O(1)
   Space: $O(n^2)$
   Building time: $O(n^2)$

0	1	2	3	4	5	6	7	
3	**1**	**-2**	**-2**	**-2**	**-2**	**-2**	**-2**	0
	1	**-2**	**-2**	**-2**	**-2**	**-2**	**-2**	1
		-2	**-2**	**-2**	**-2**	**-2**	**-2**	2
			4	**4**	**4**	**2**	**0**	3
				6	**6**	**2**	**0**	4
					13	**2**	**0**	5
						2	**0**	6
							0	7

An efficient solution

Remember
An efficient solution is the one that gives
guaranteed good running times for
all operations.

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

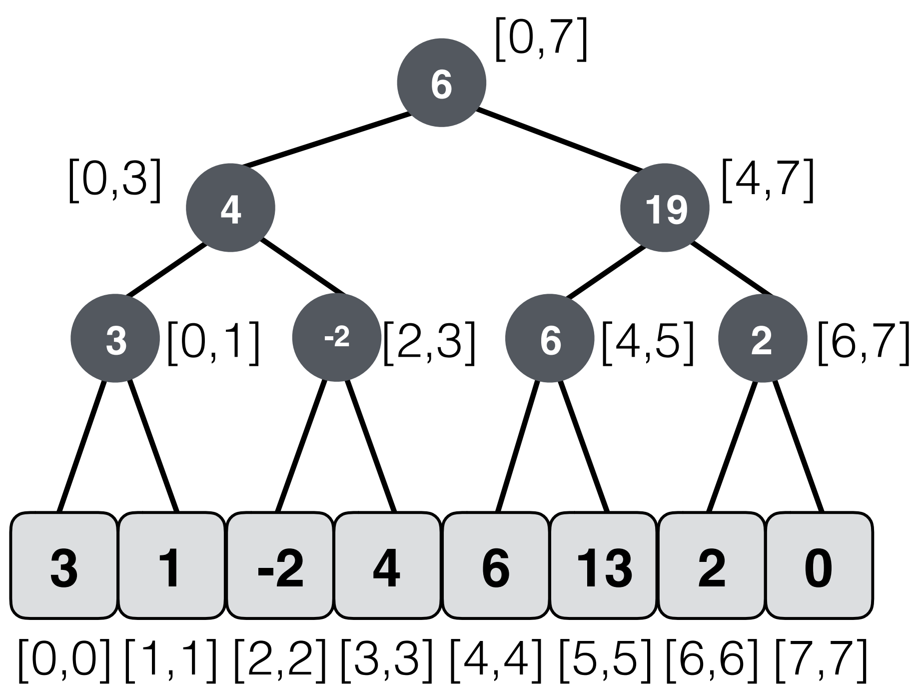
Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums

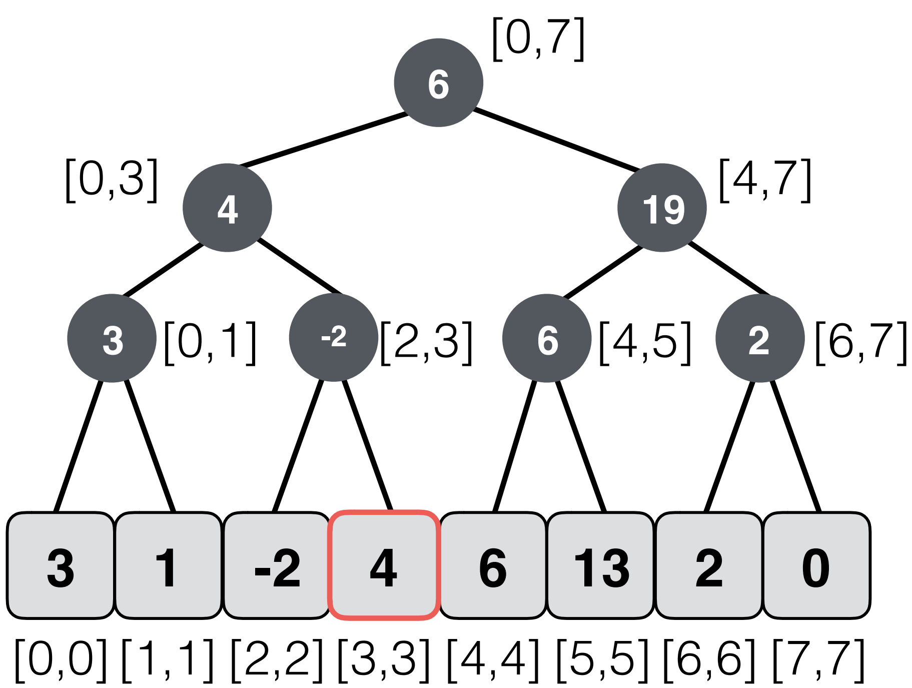


An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



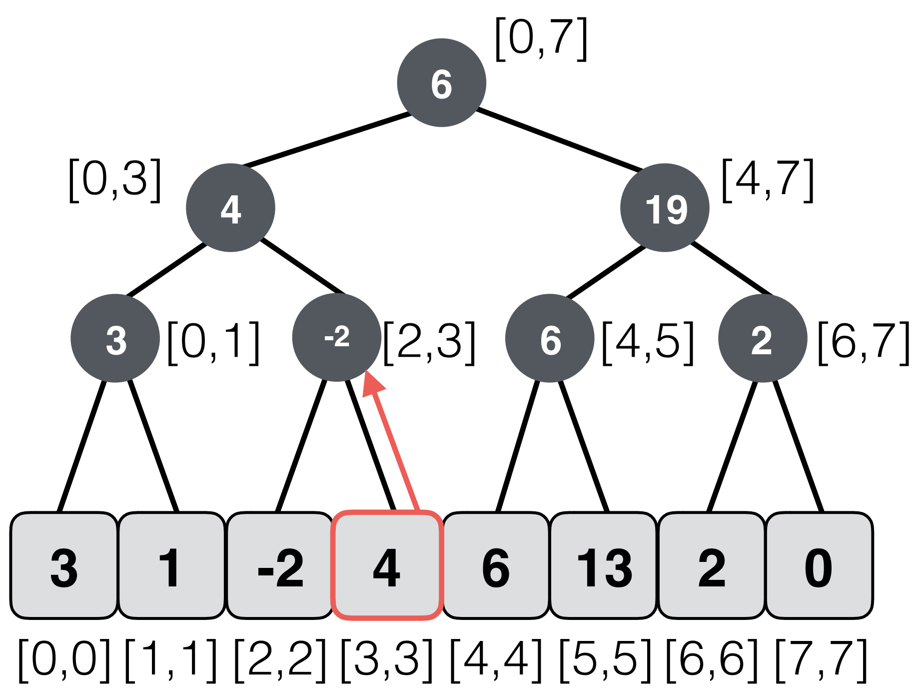
`sum`(3) = (4) +

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



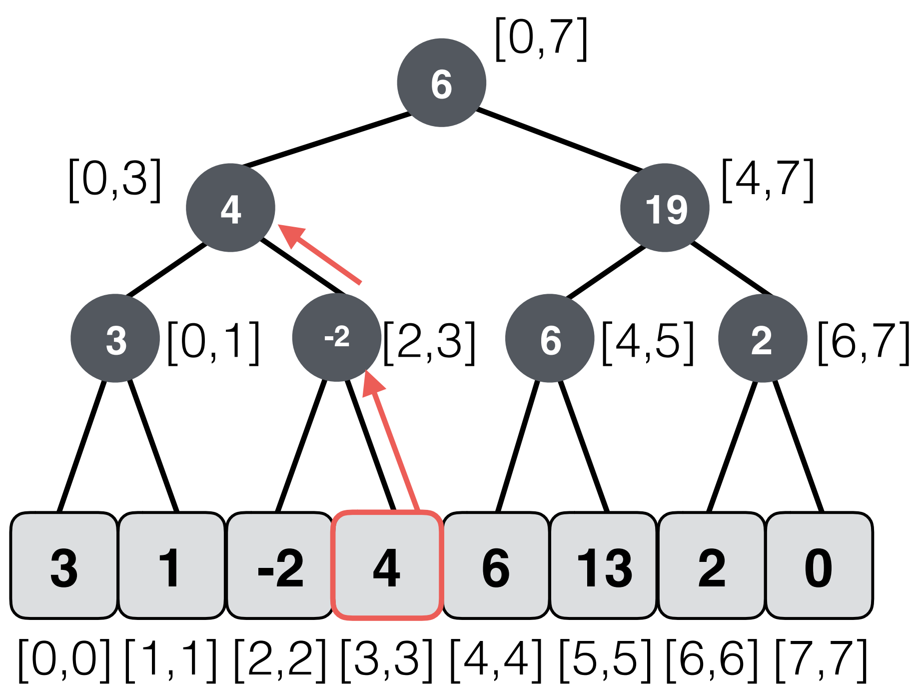
`sum`(3) = (4) + (-2) +

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



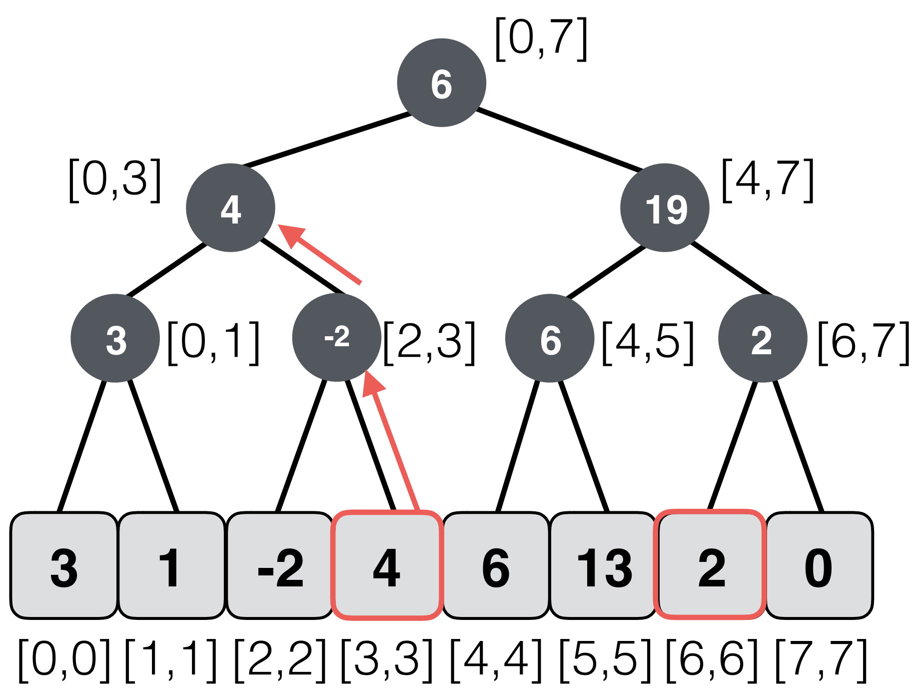
`sum`(3) = (4) + (-2) + (4) = 6

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



`sum`(3) = (4) + (-2) + (4) = 6

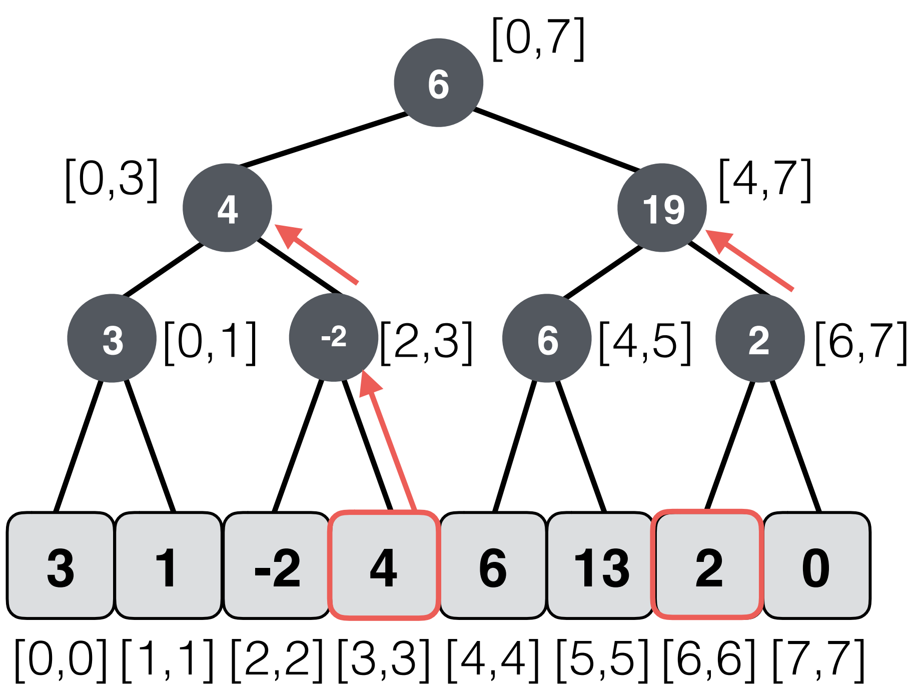
`sum`(6) = (2) +

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



`sum`(3) = (4) + (-2) + (4) = 6

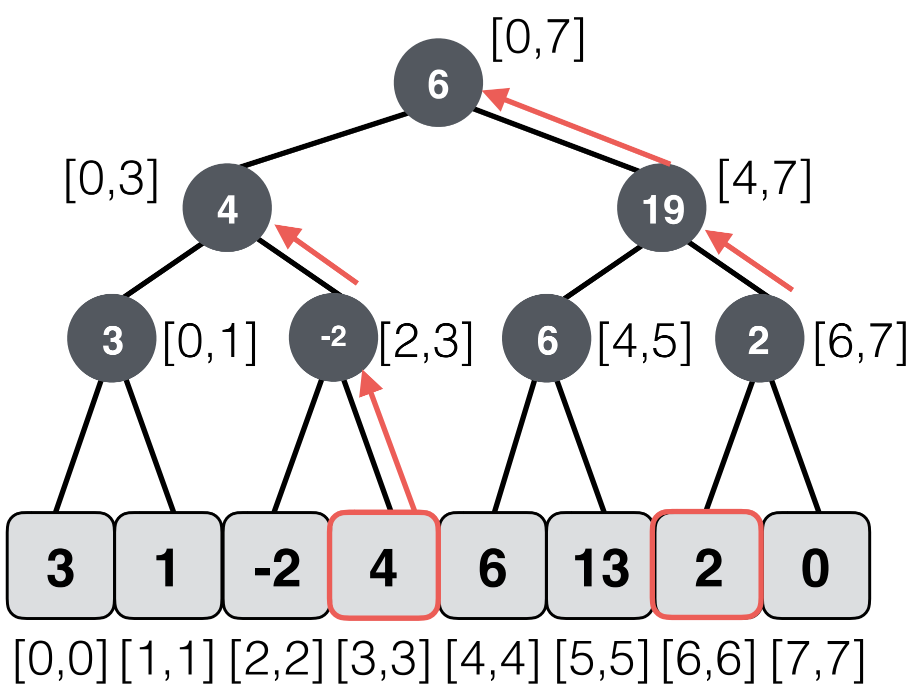
`sum`(6) = (2) + (19)+

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



`sum`(3) = (4) + (-2) + (4) = 6

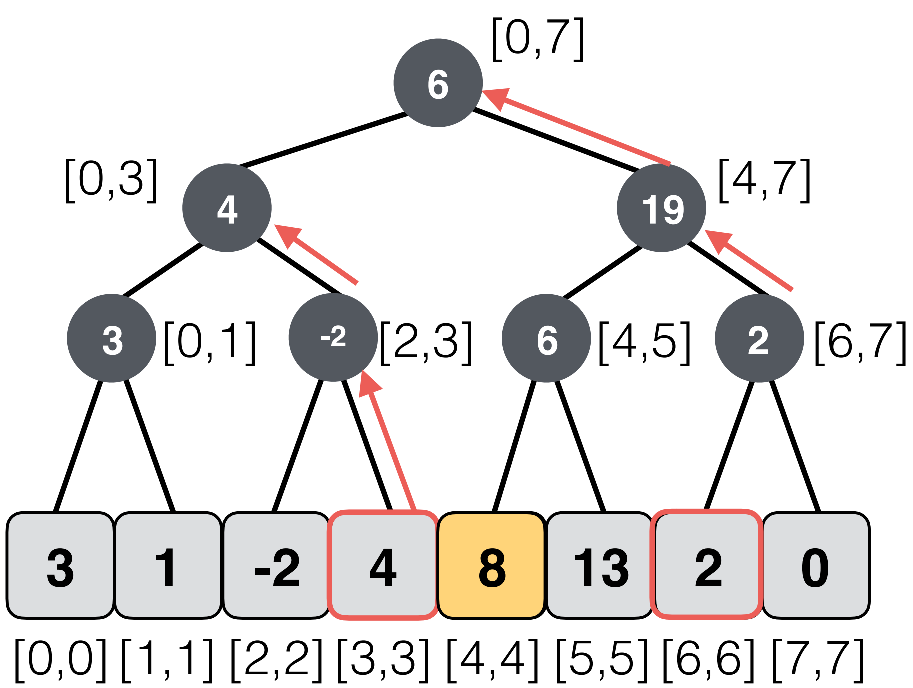
`sum`(6) = (2) + (19)+ (6) = 27

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



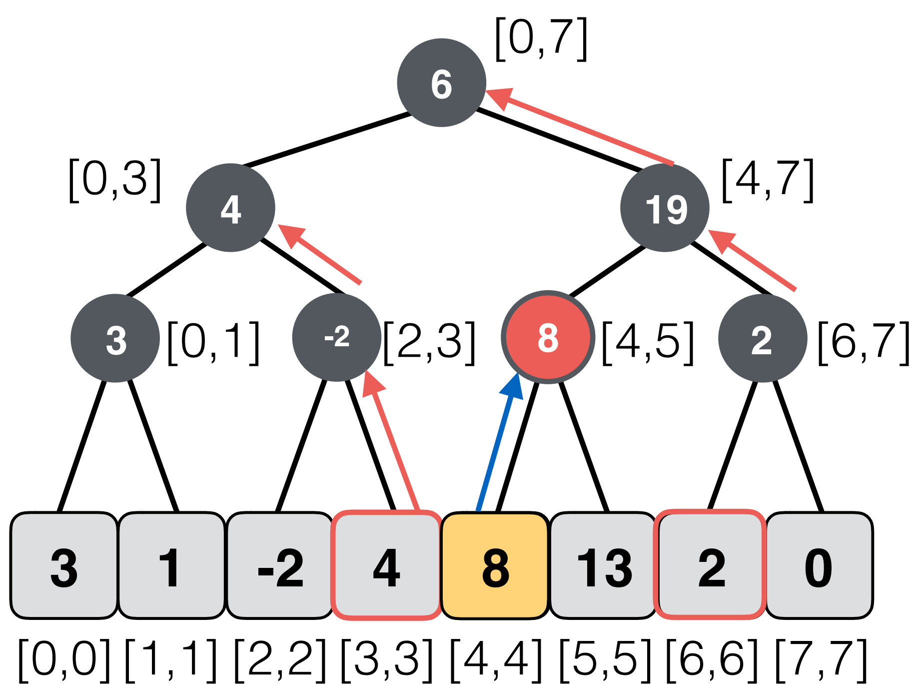
`sum`(3) = (4) + (-2) + (4) = 6
`sum`(6) = (2) + (19)+ (6) = 27
`update`(4, 8)

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



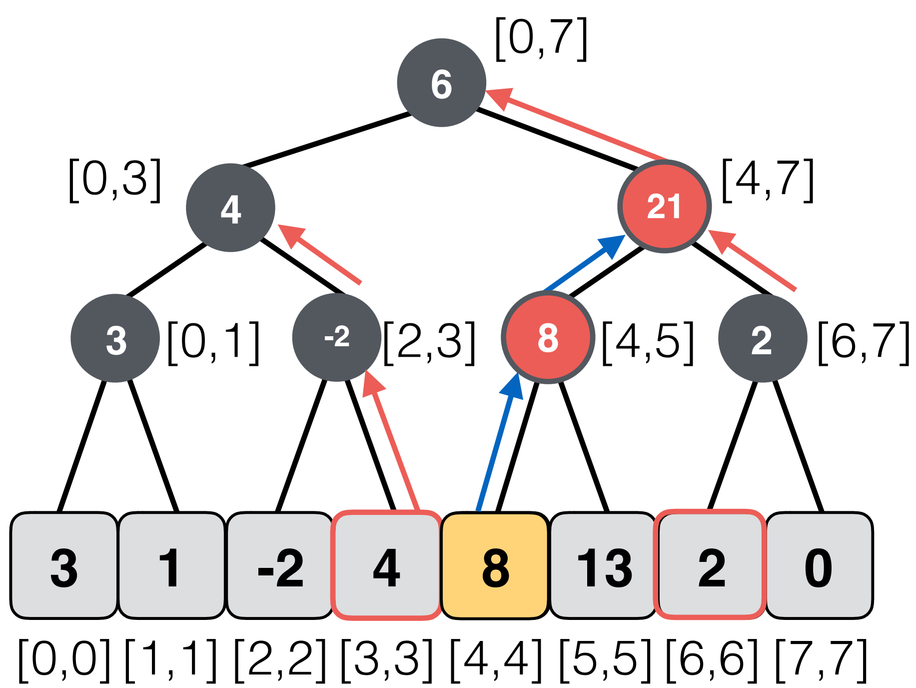
`sum`(3) = (4) + (-2) + (4) = 6
`sum`(6) = (2) + (19)+ (6) = 27
`update`(4, 8)

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



`sum`(3) = (4) + (-2) + (4) = 6

`sum`(6) = (2) + (19)+ (6) = 27

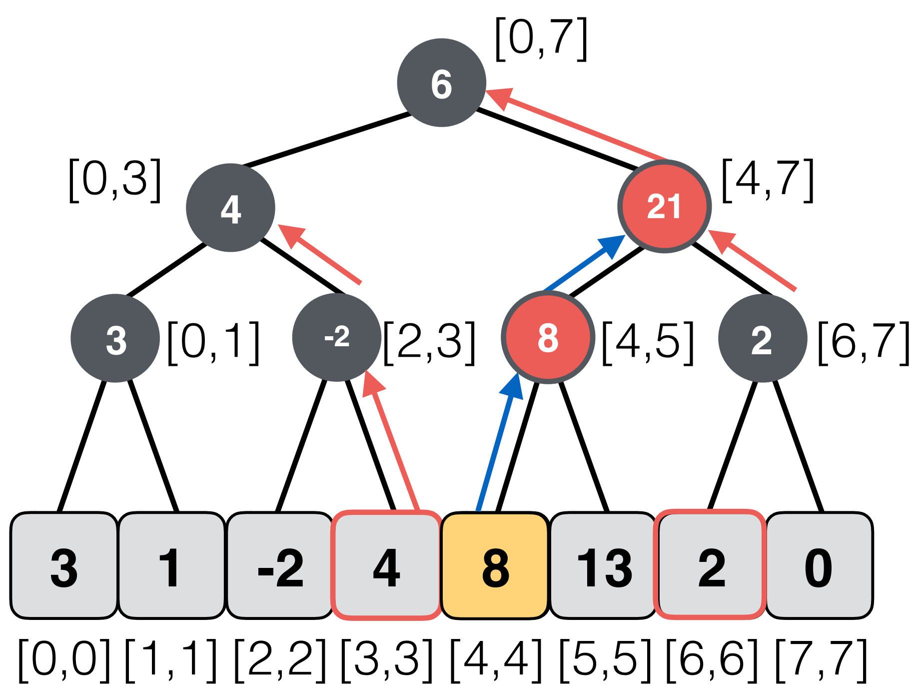
`update`(4, 8)

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums



`sum`(3) = (4) + (-2) + (4) = 6

`sum`(6) = (2) + (19)+ (6) = 27

`update`(4, 8)

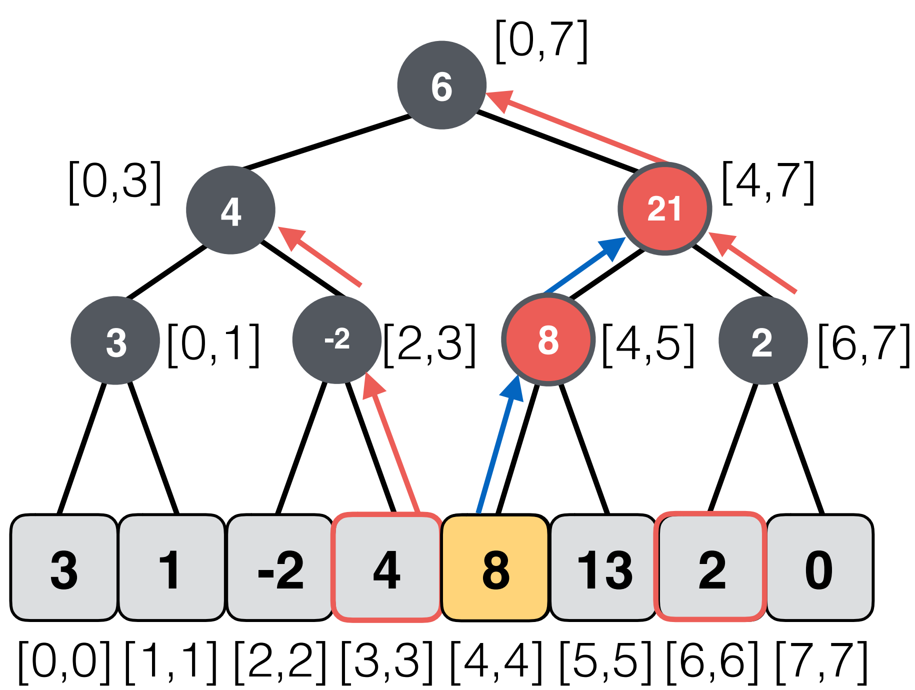
`sum` and `update` in O(log n)

An efficient solution

Remember
An efficient solution is the one that gives *guaranteed good running times* for **all** operations.

Idea
Impose a complete (static) binary tree over the array: a **segment tree**.

(Static) Prefix sums

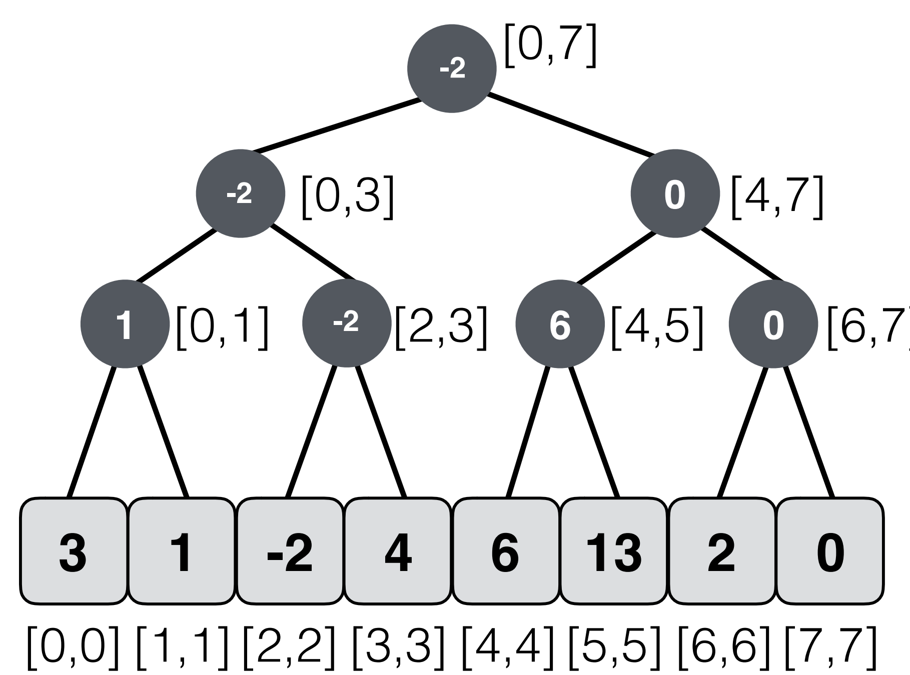


`sum`(3) = (4) + (-2) + (4) = 6
`sum`(6) = (2) + (19)+ (6) = 27
`update`(4, 8)

`sum` and `update` in O(log n)

What we consider next, stay tuned!

Range `MIN` queries

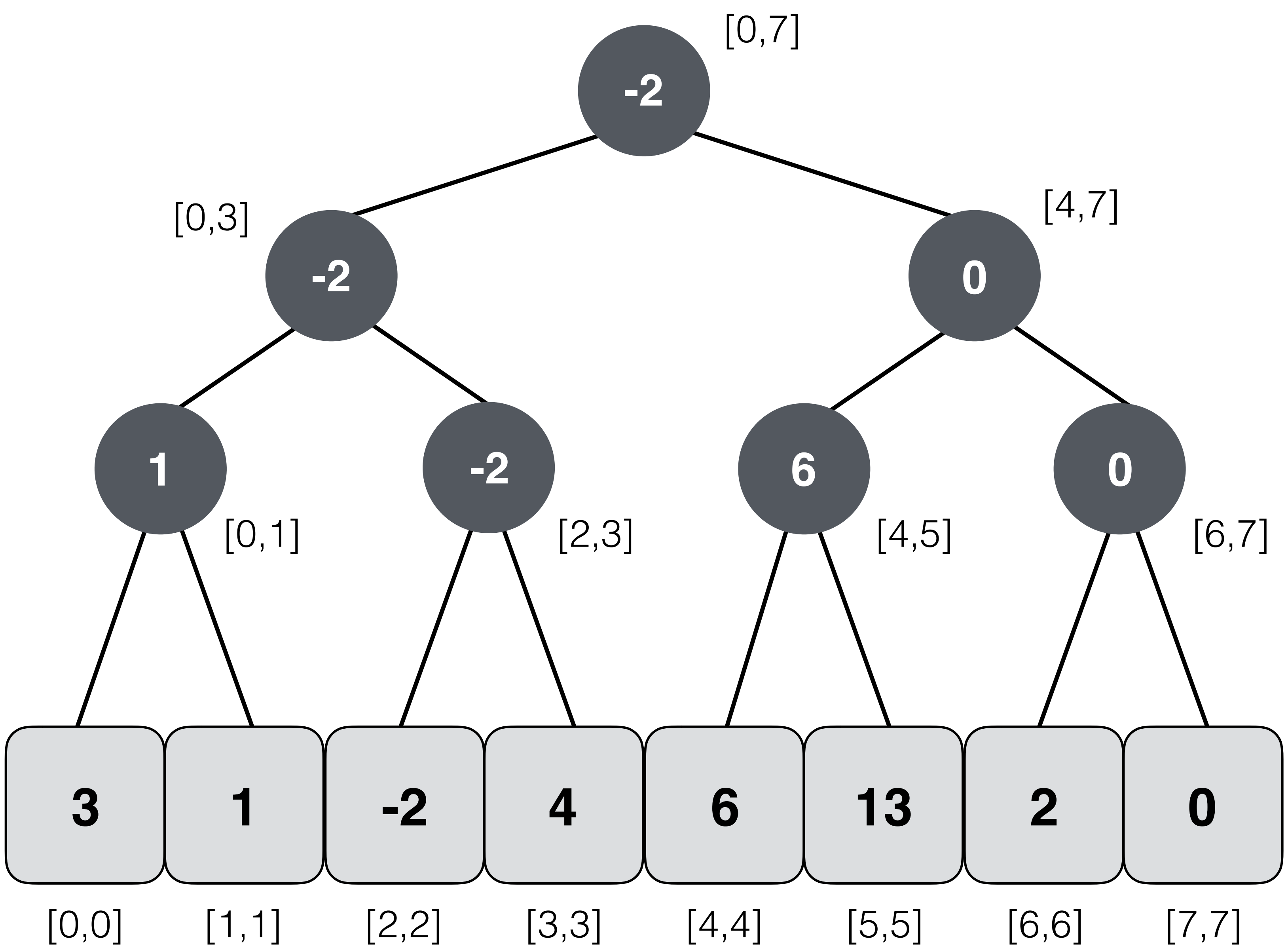


Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value

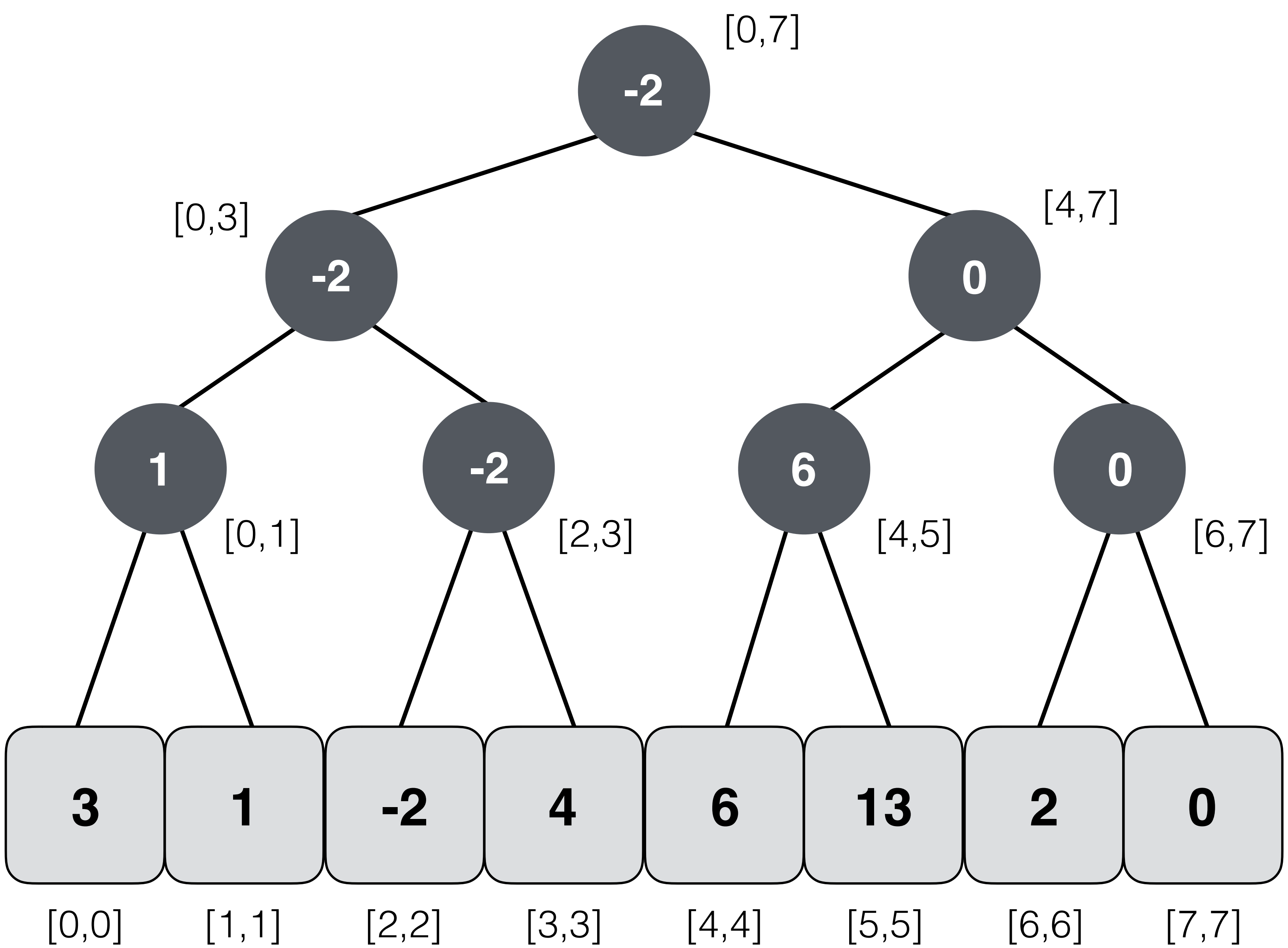


Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



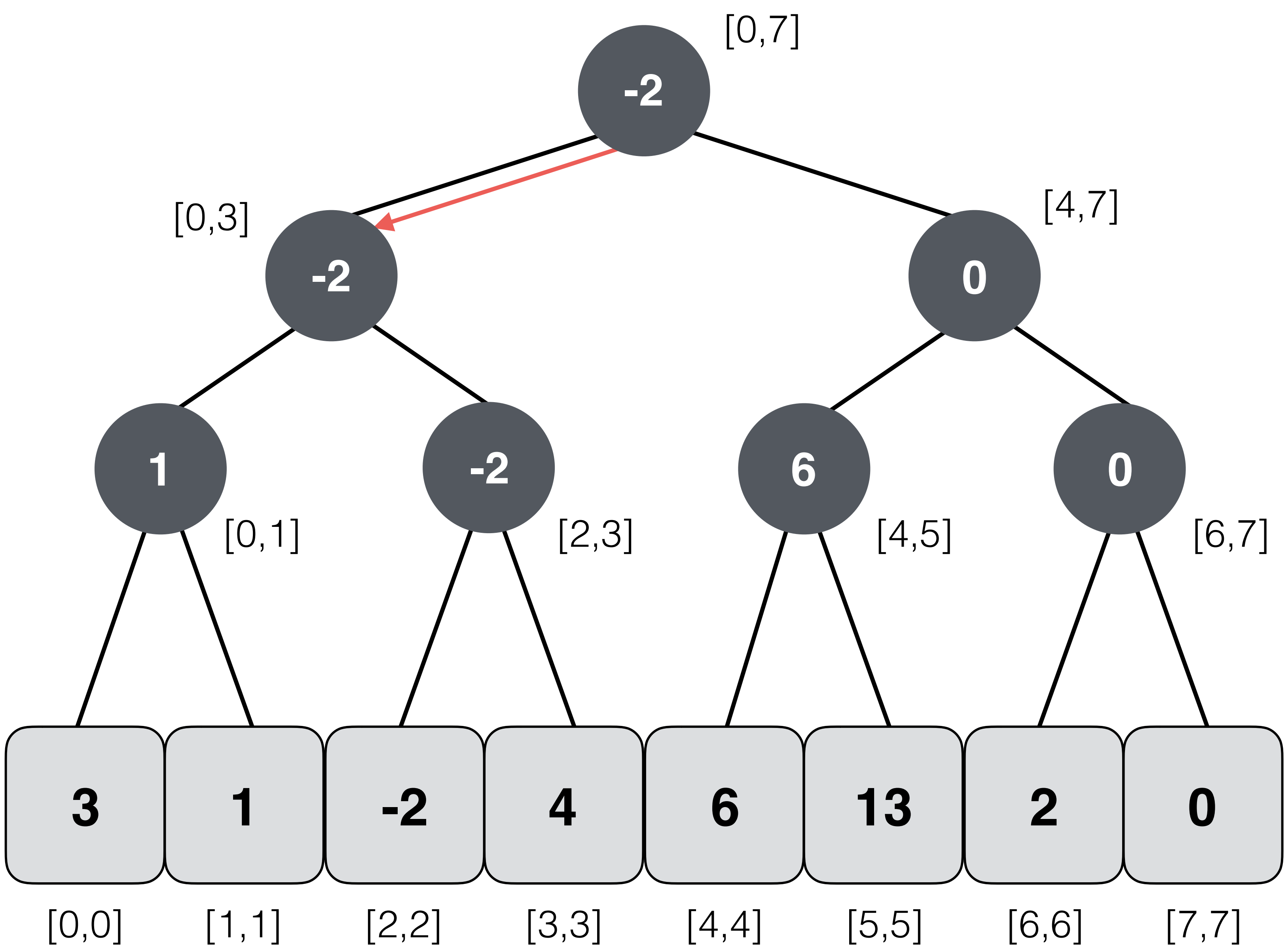
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



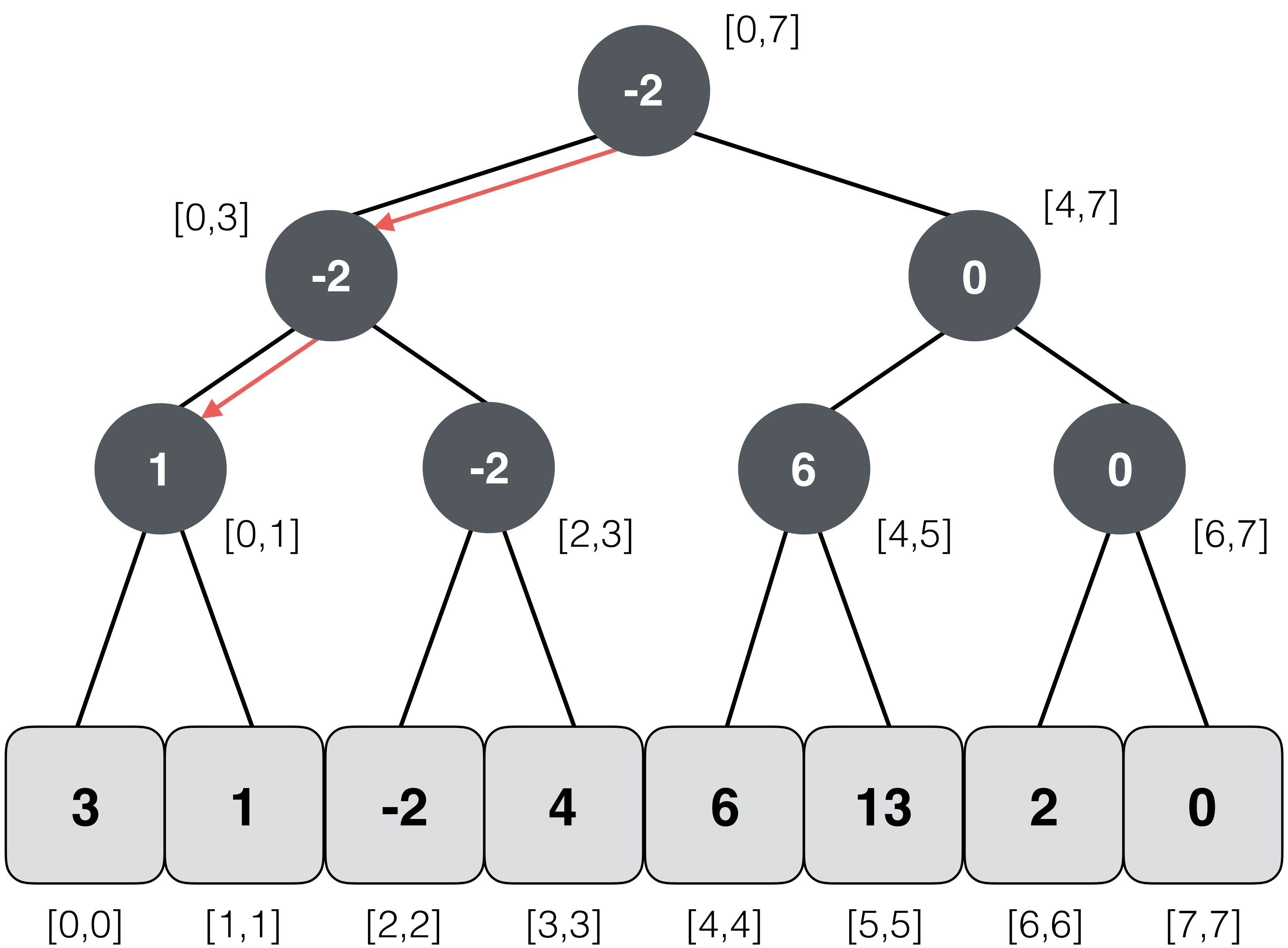
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



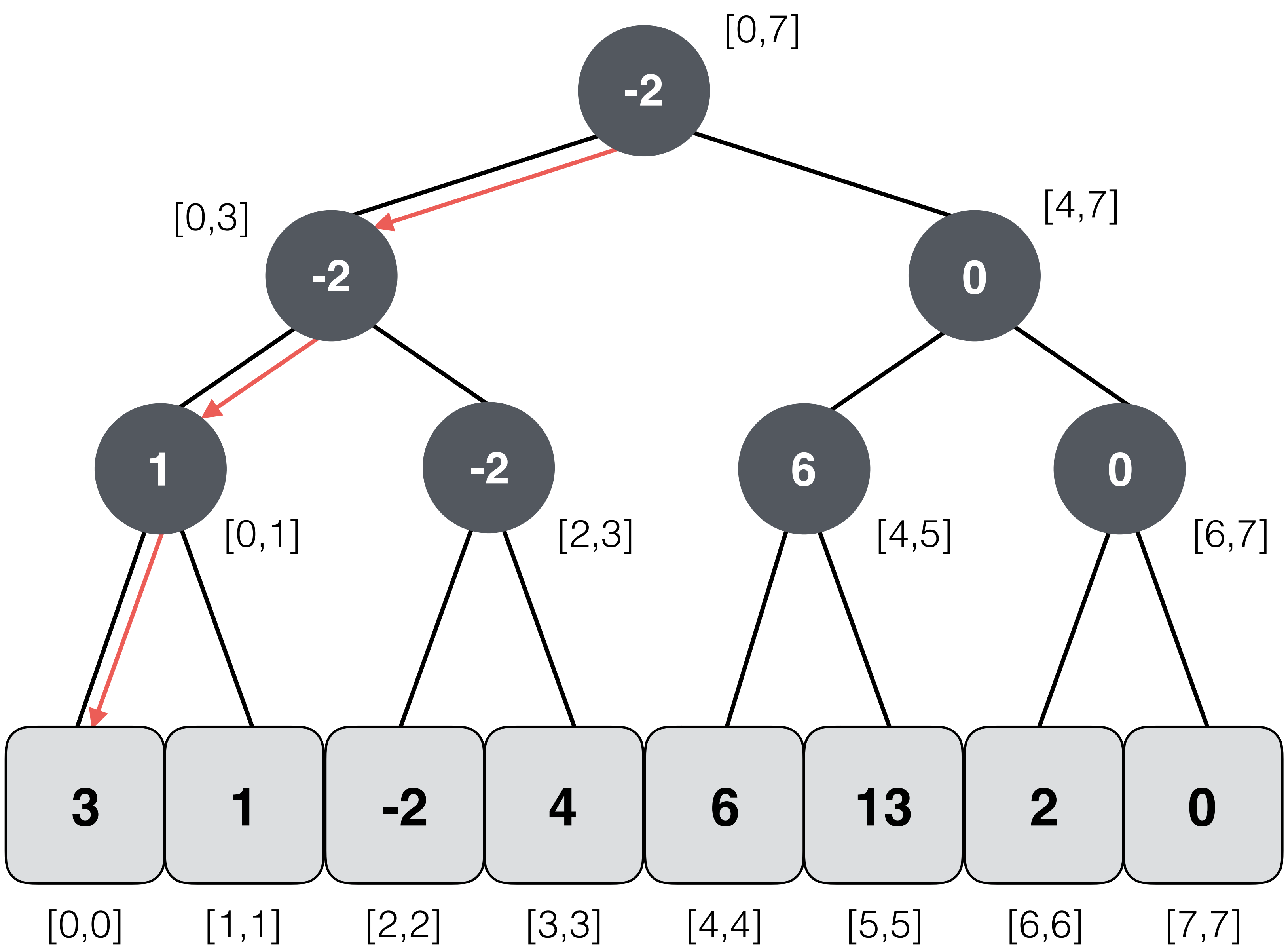
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



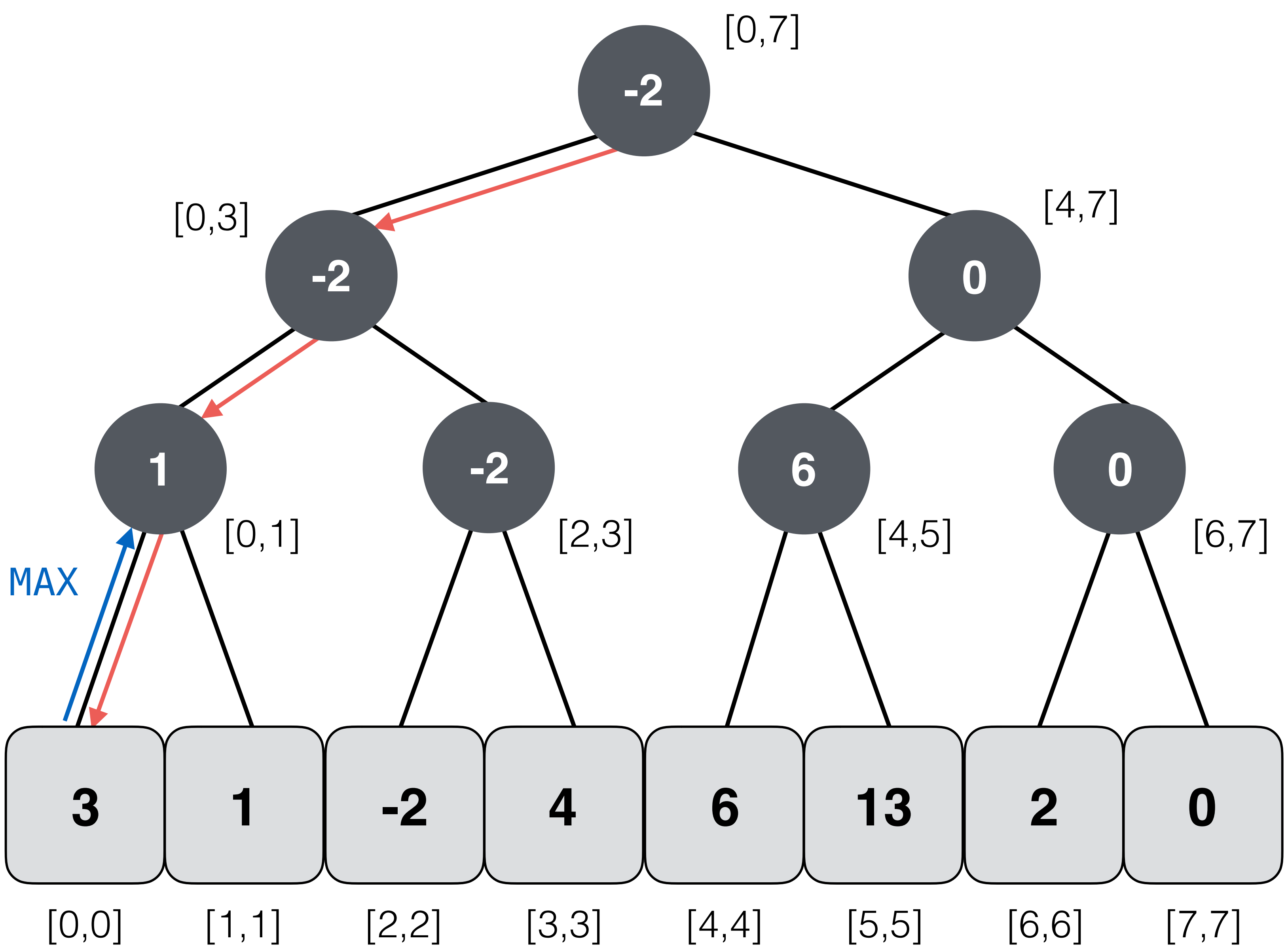
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



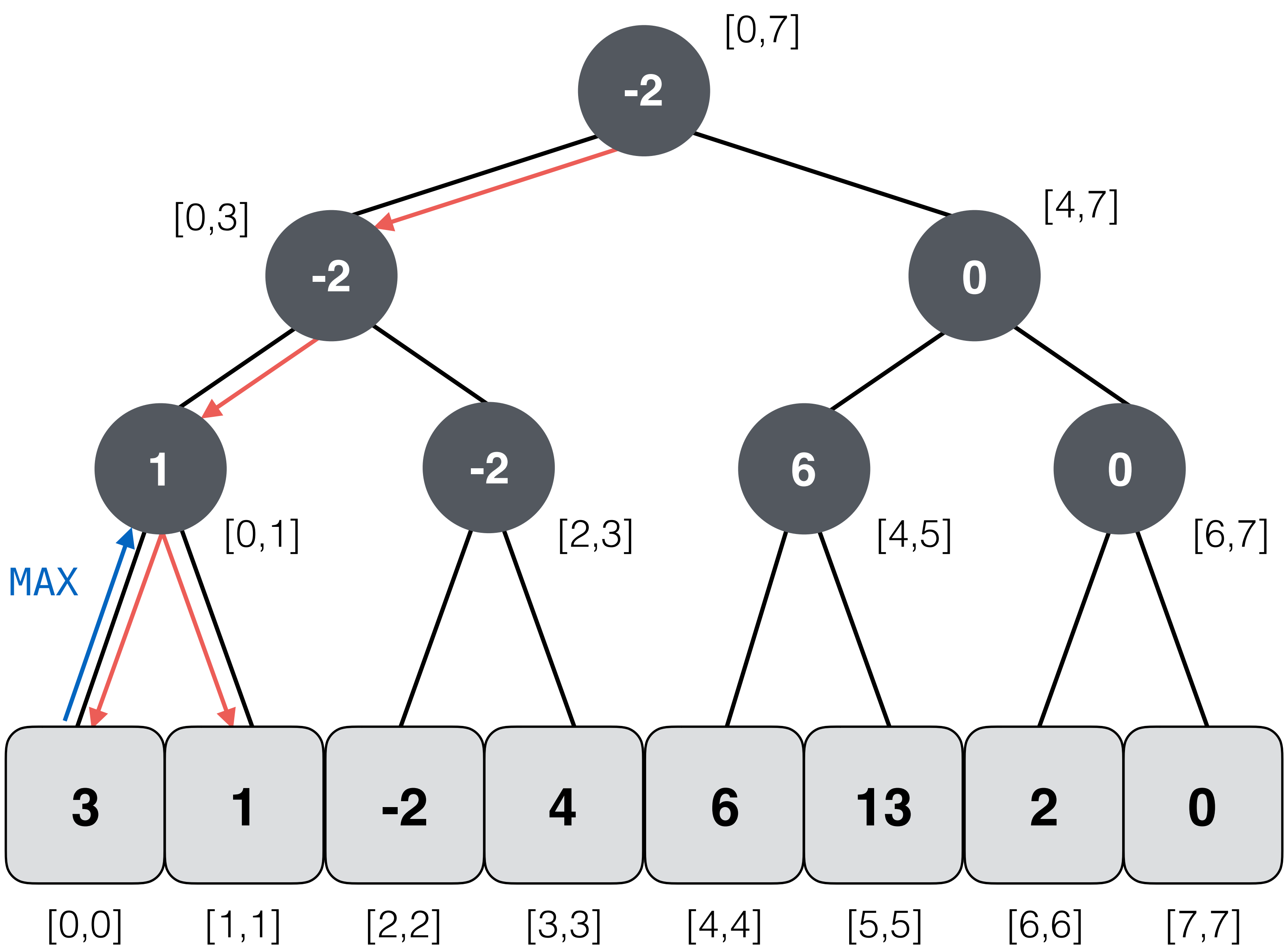
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



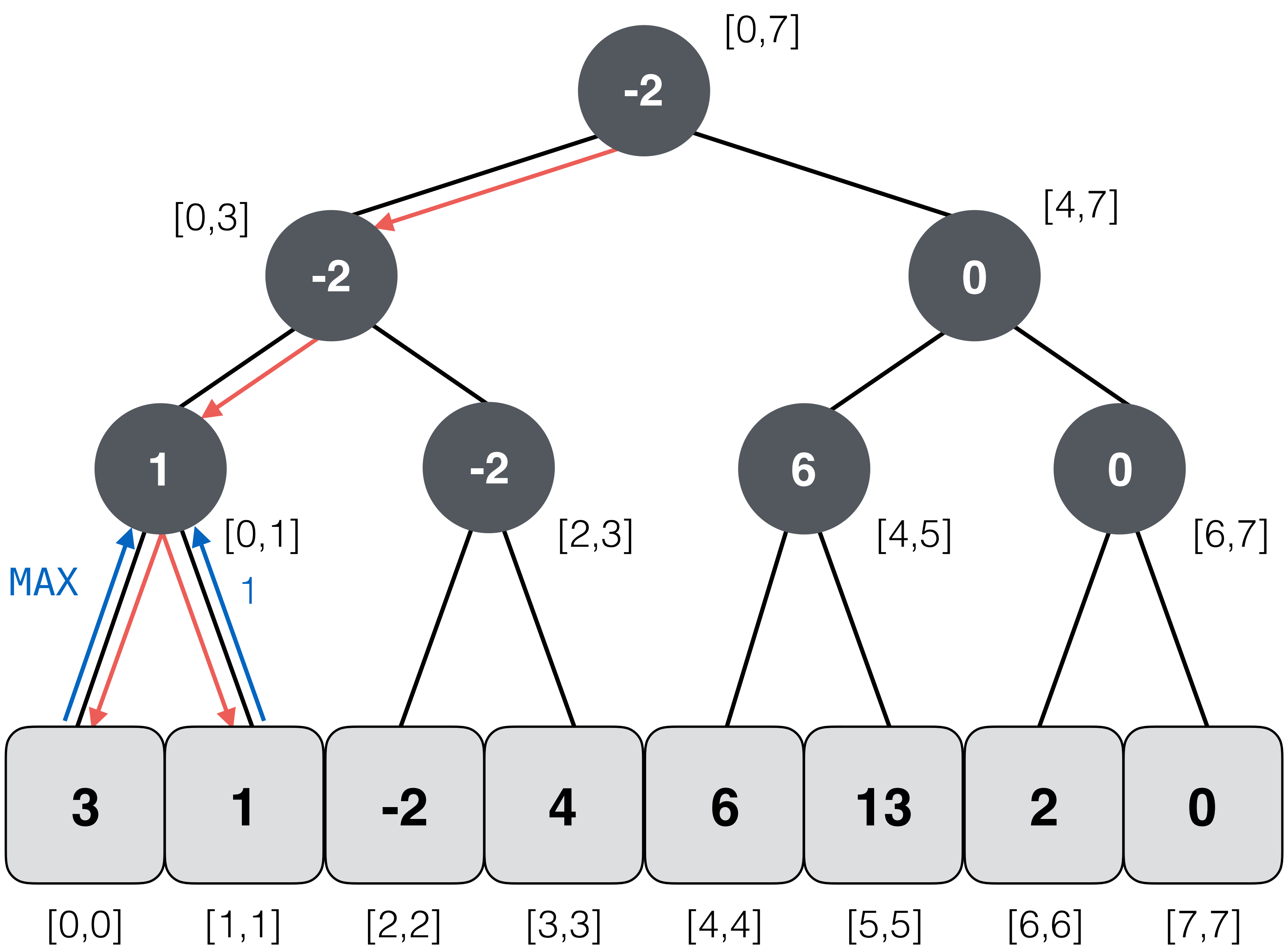
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



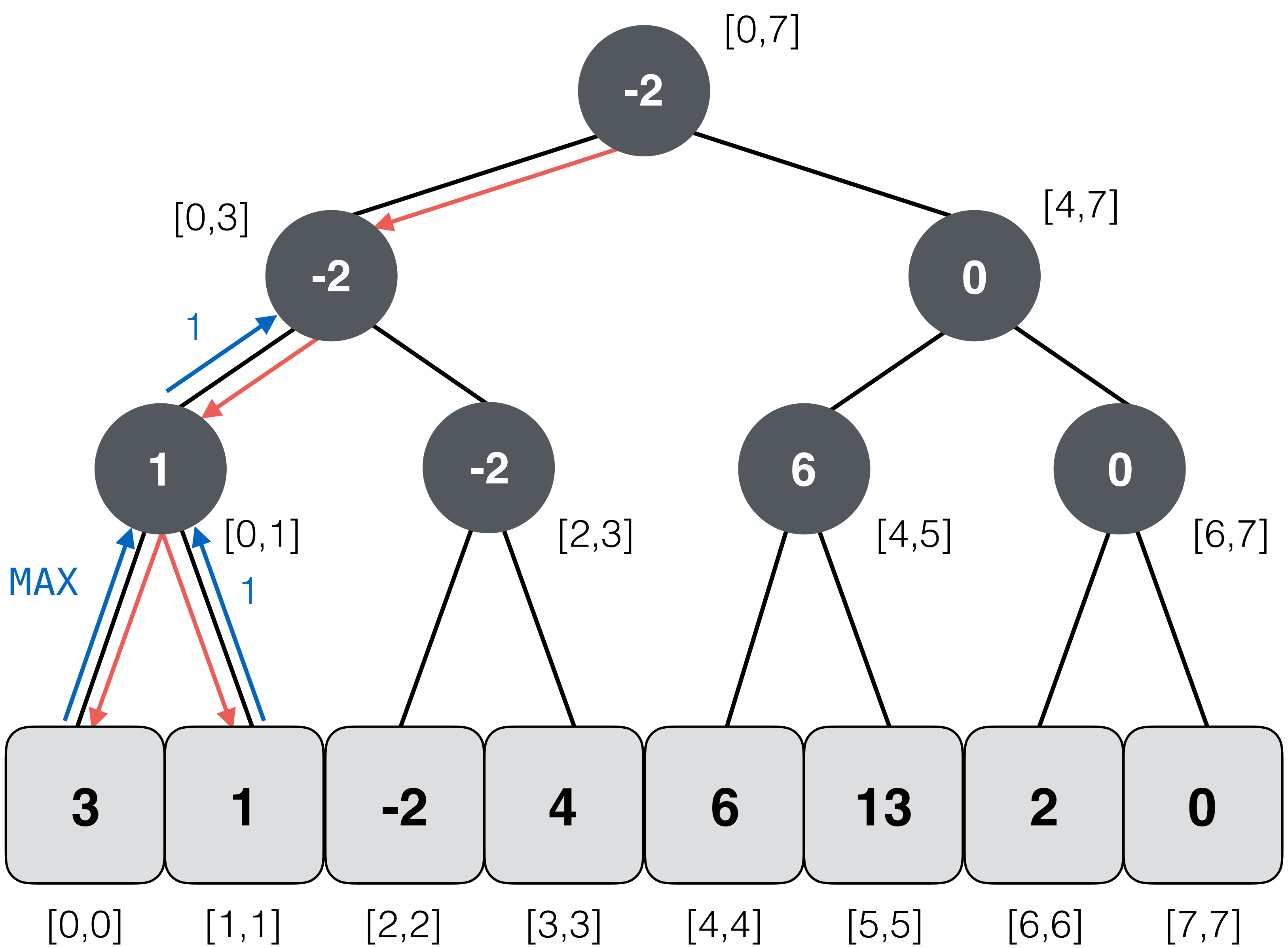
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



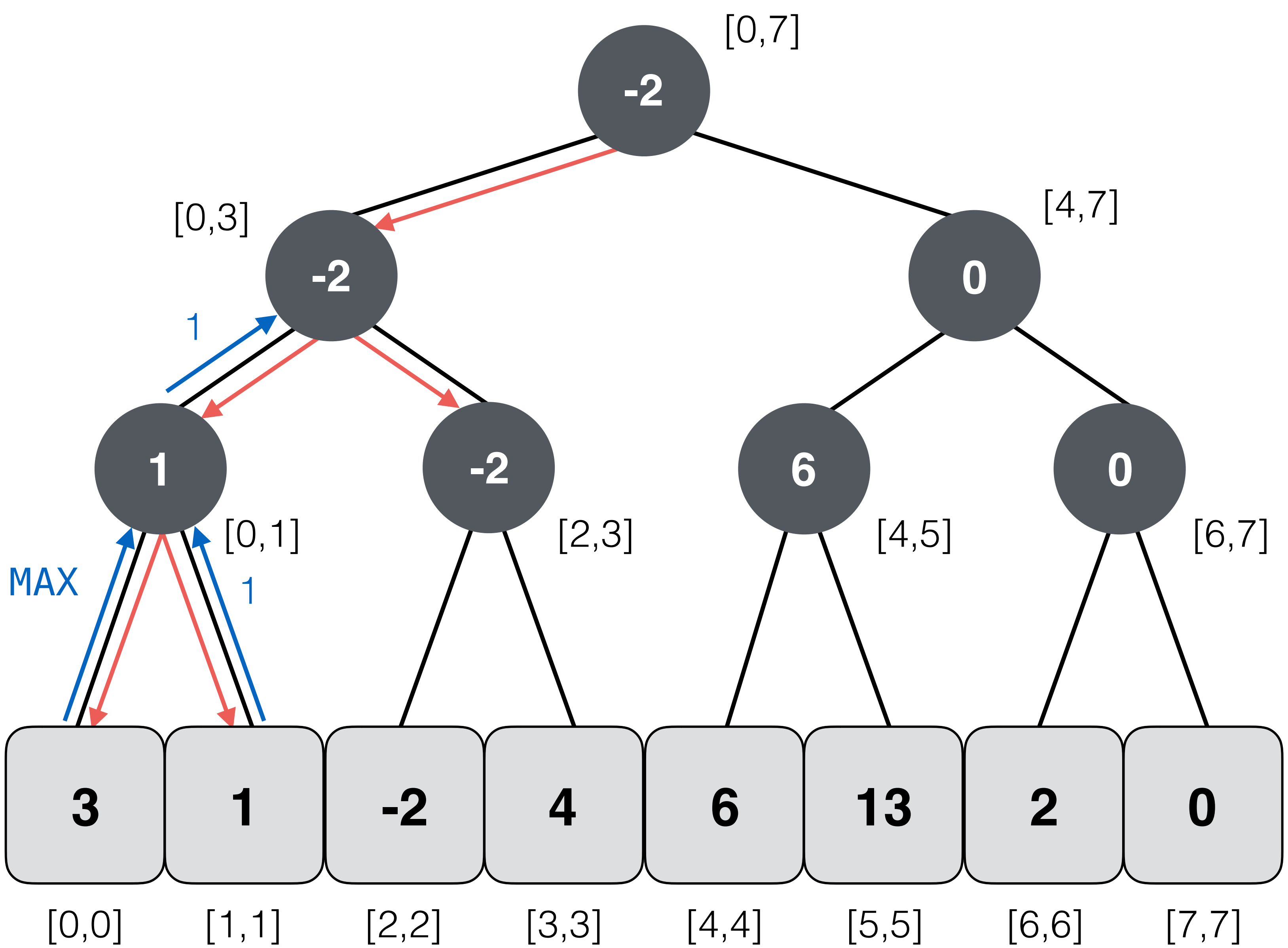
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



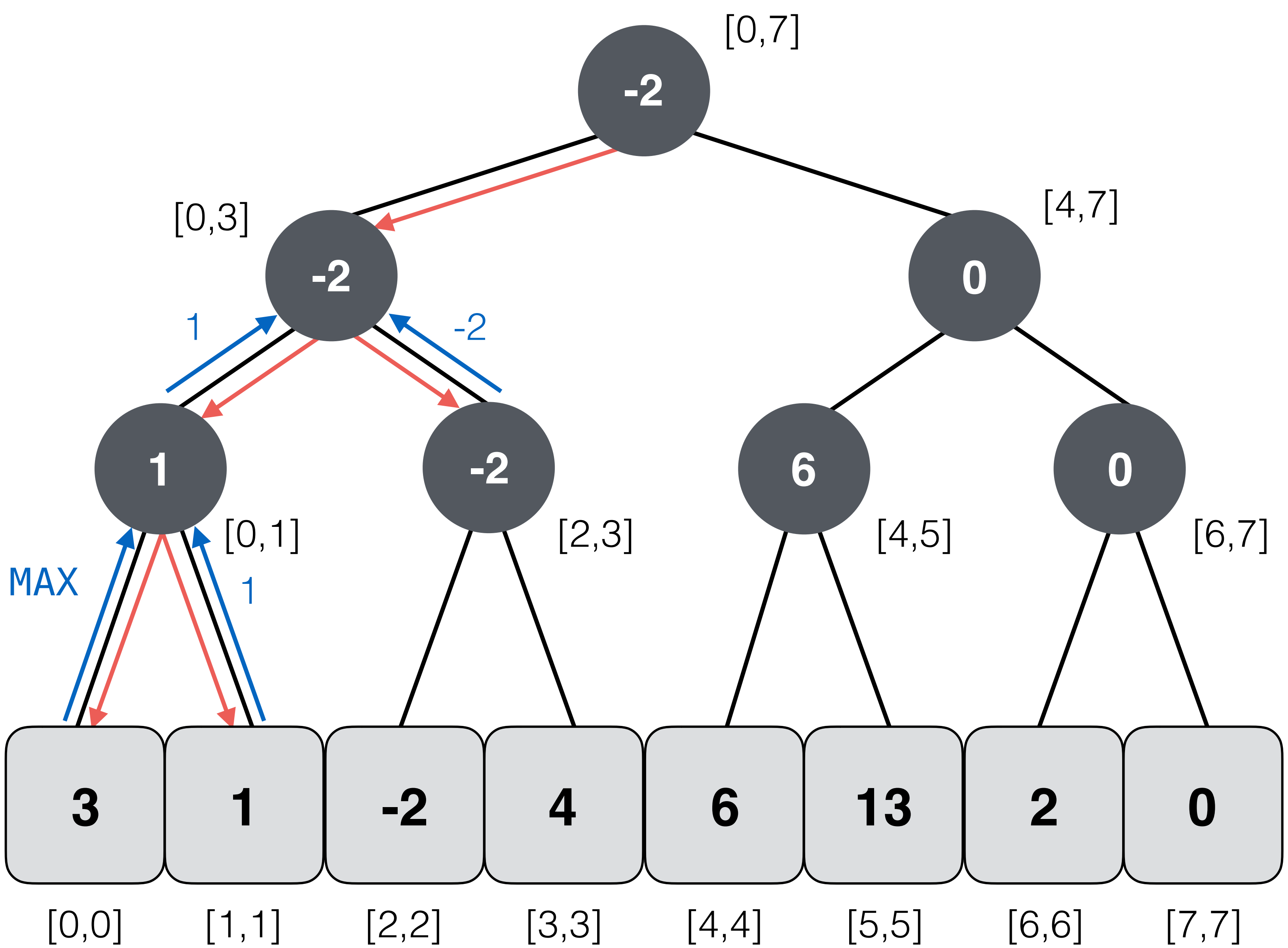
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



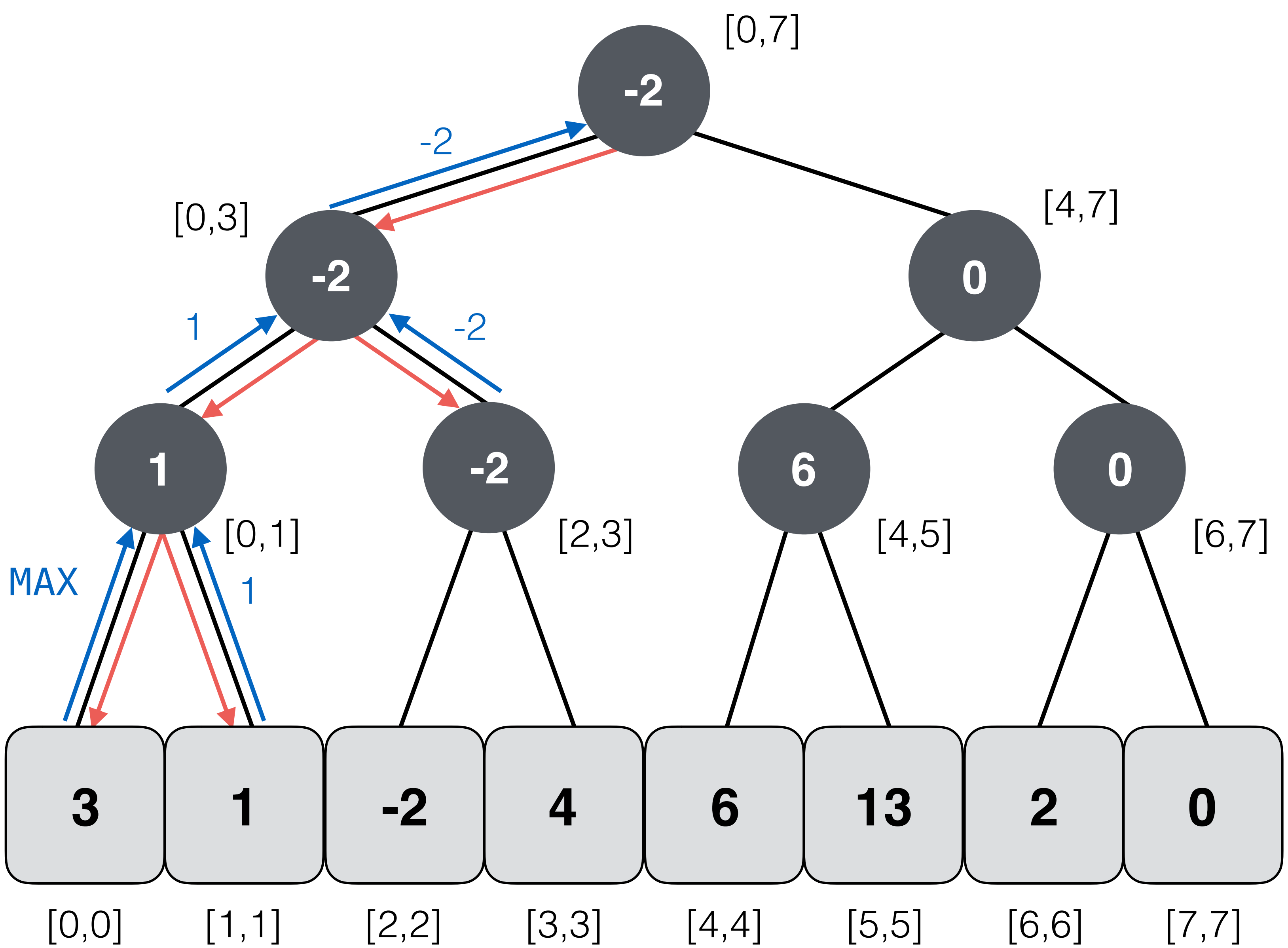
`min`(1,3)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



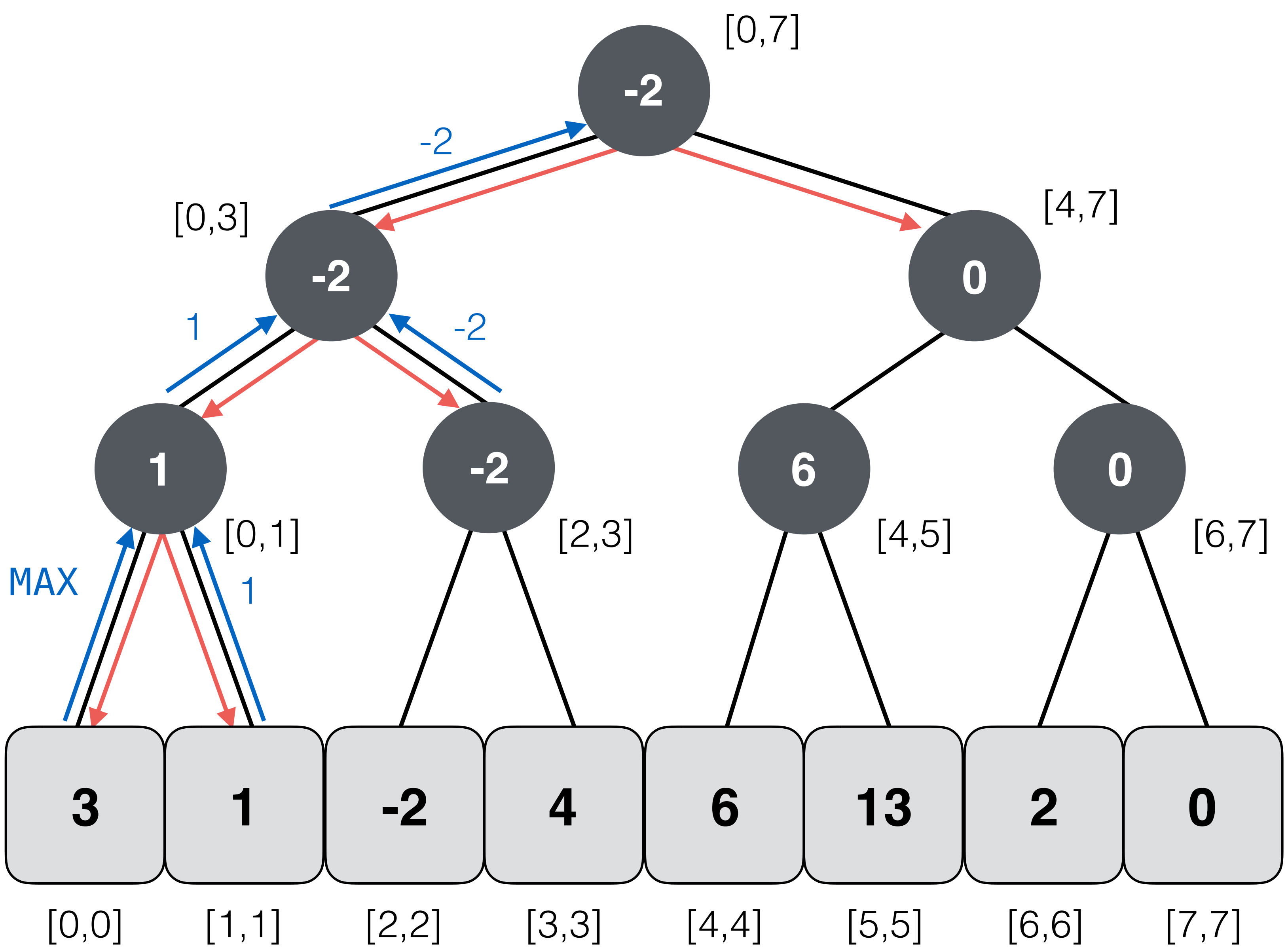
`min`(1,3)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return MAX value



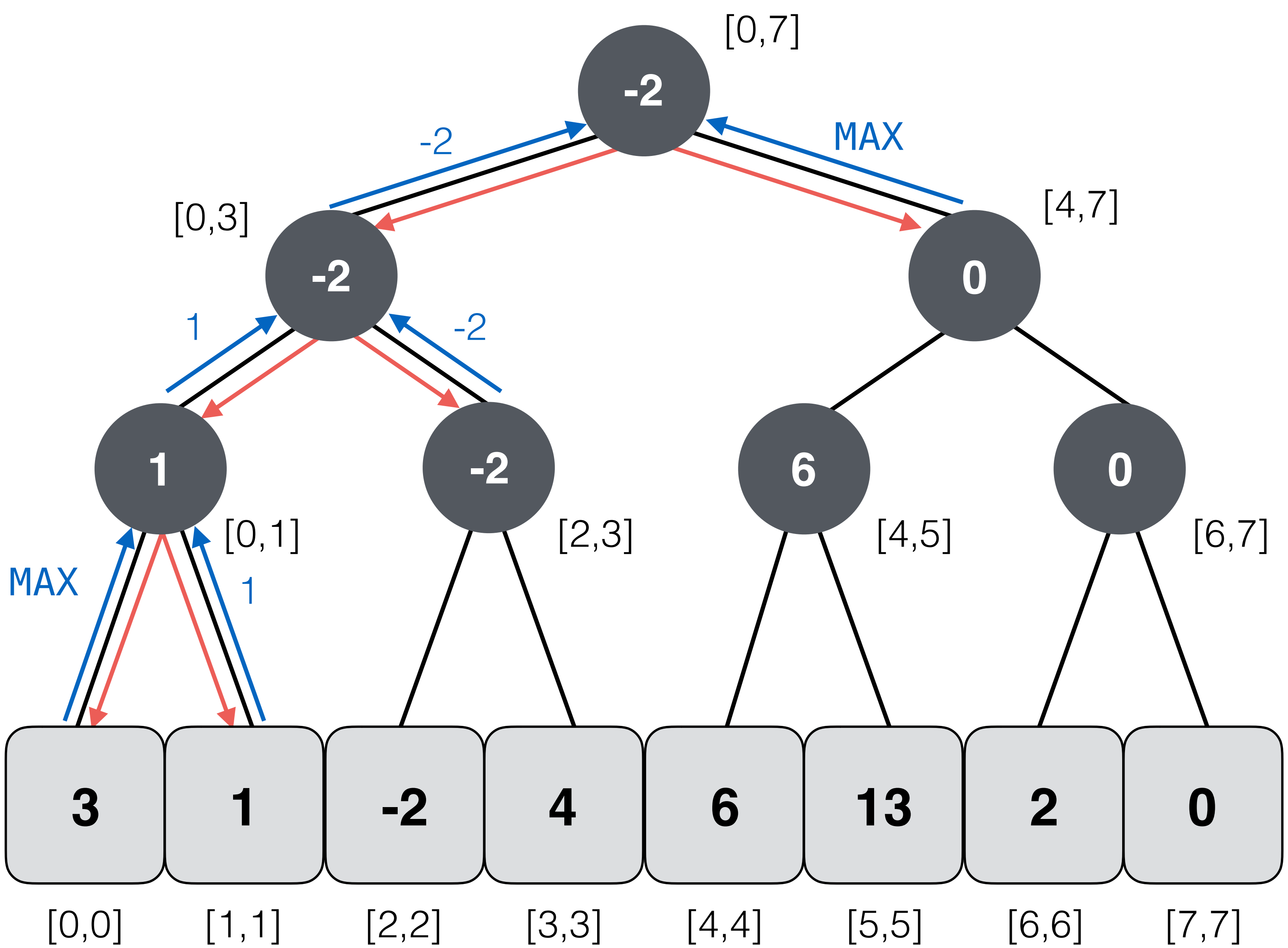
min(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



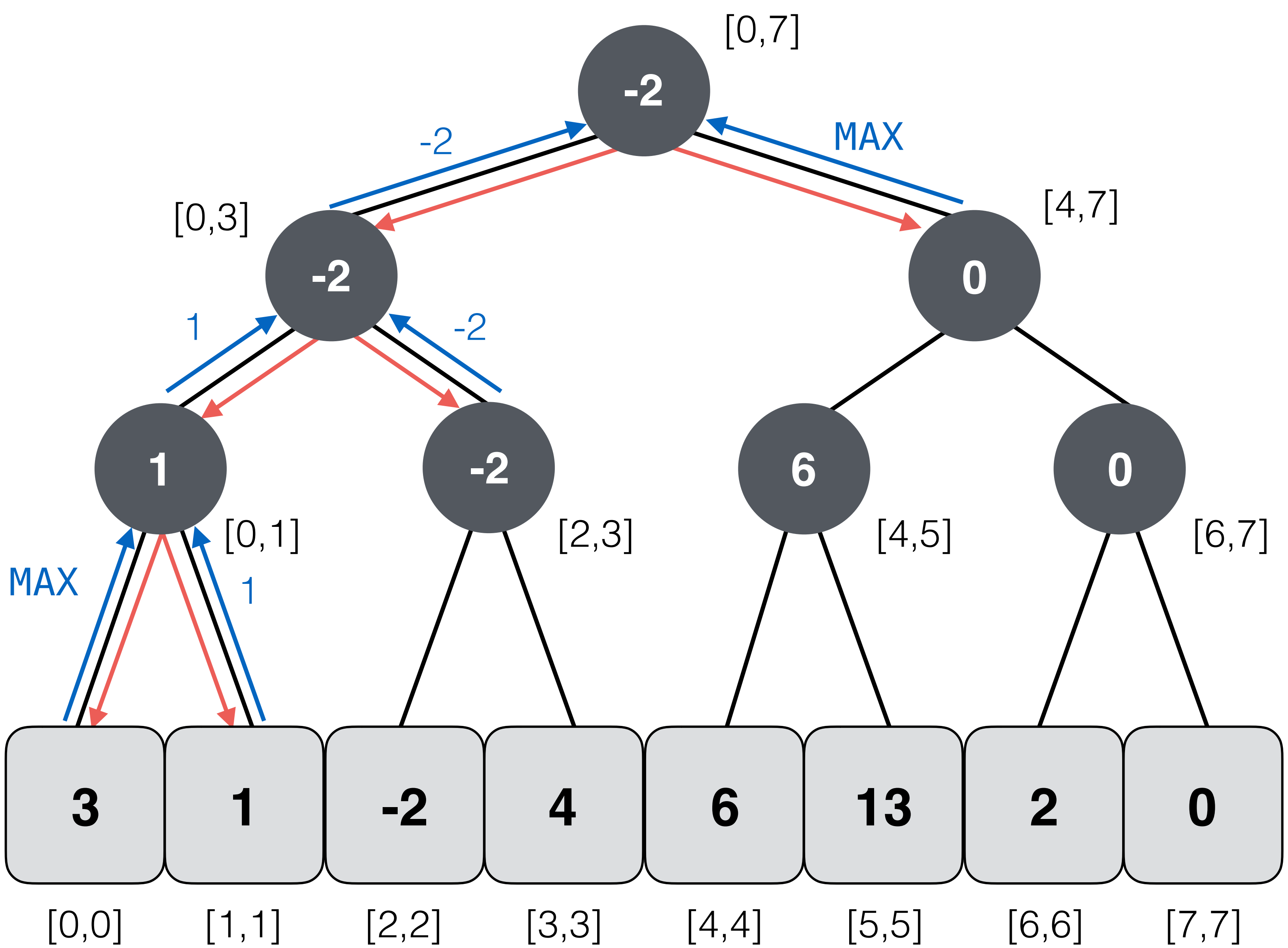
`min`(1,3)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value

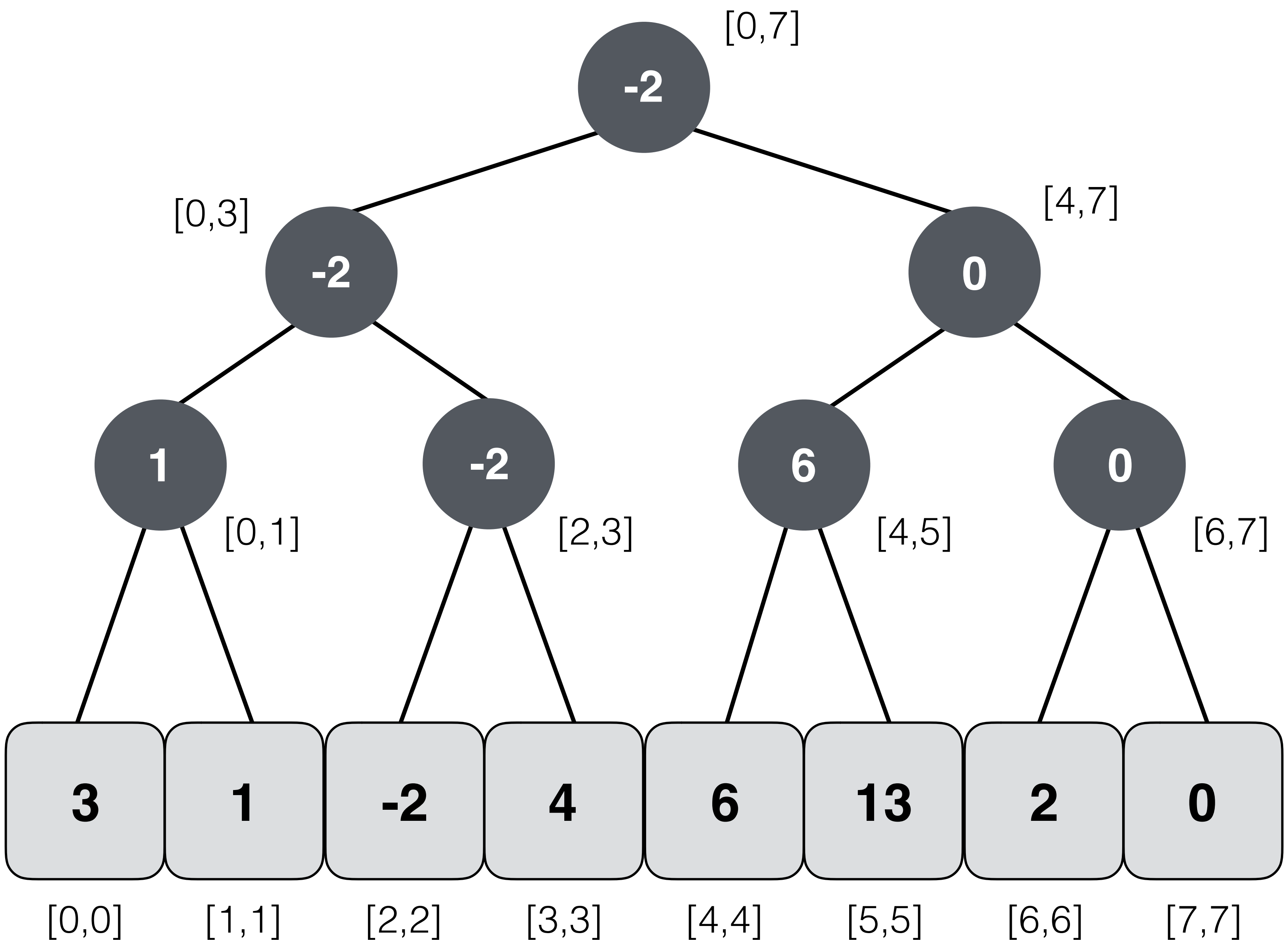
`min`(1,3) = -2

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



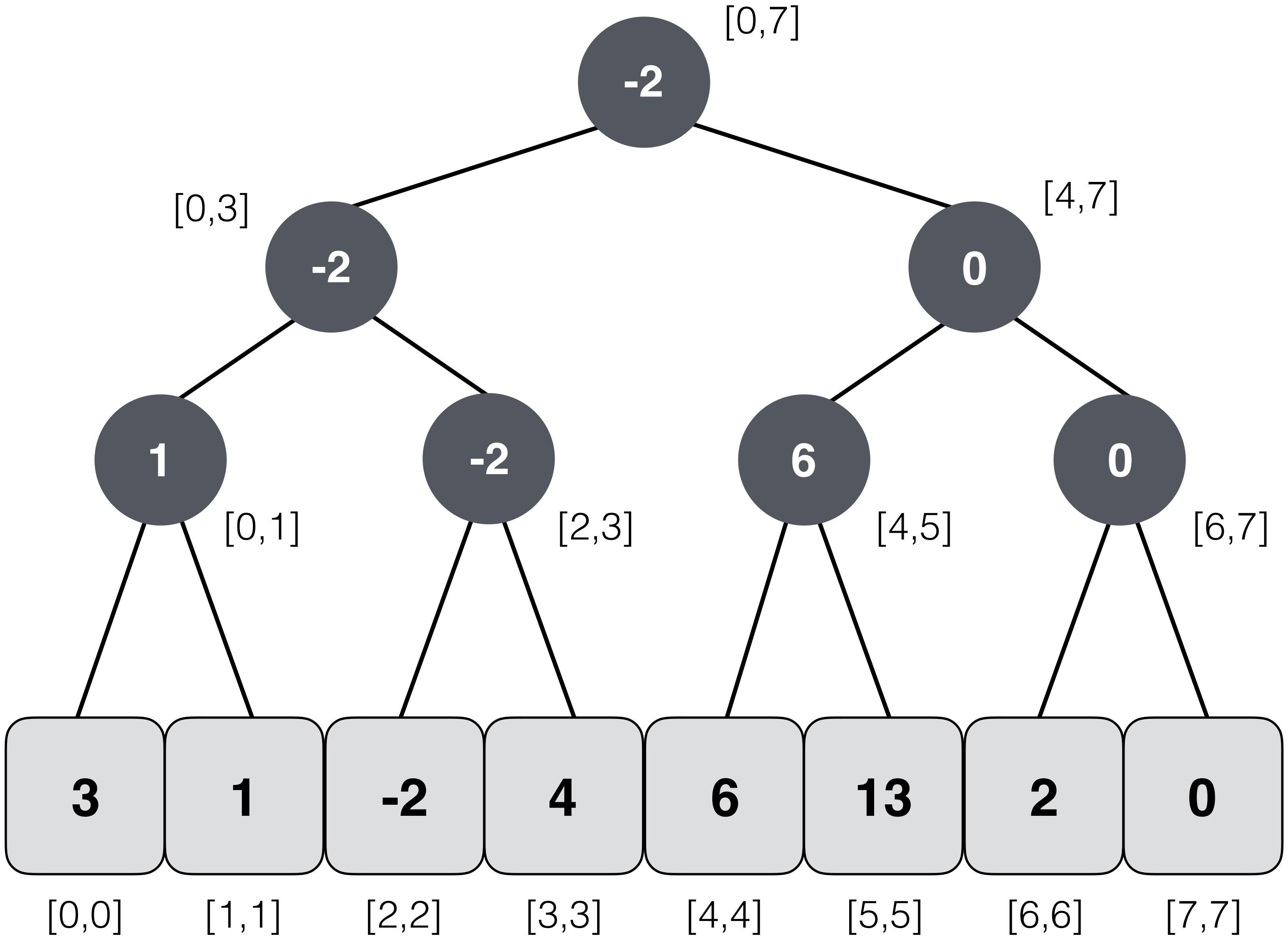
`min`(1,3) = -2

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

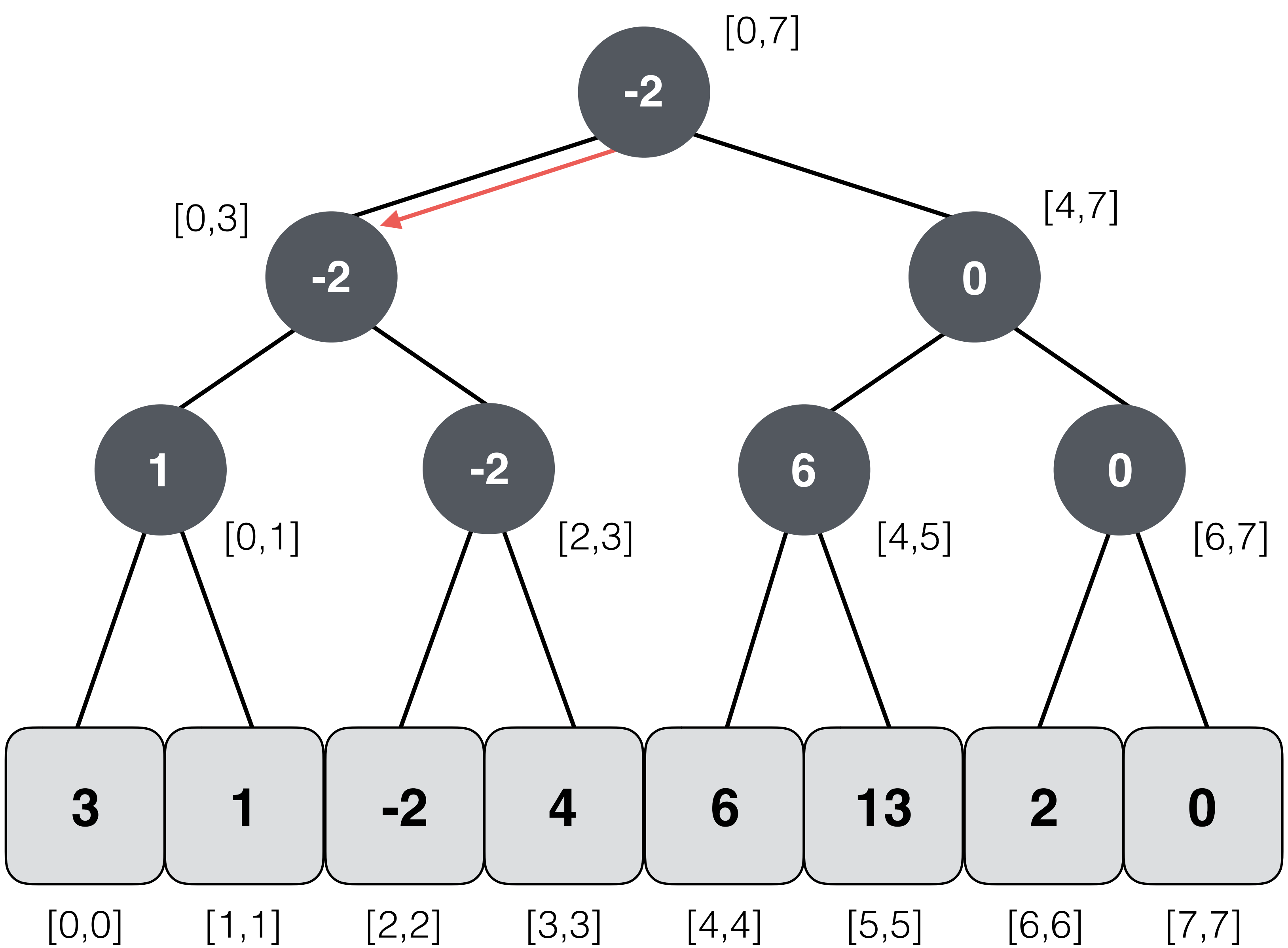
`min`(3,6)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

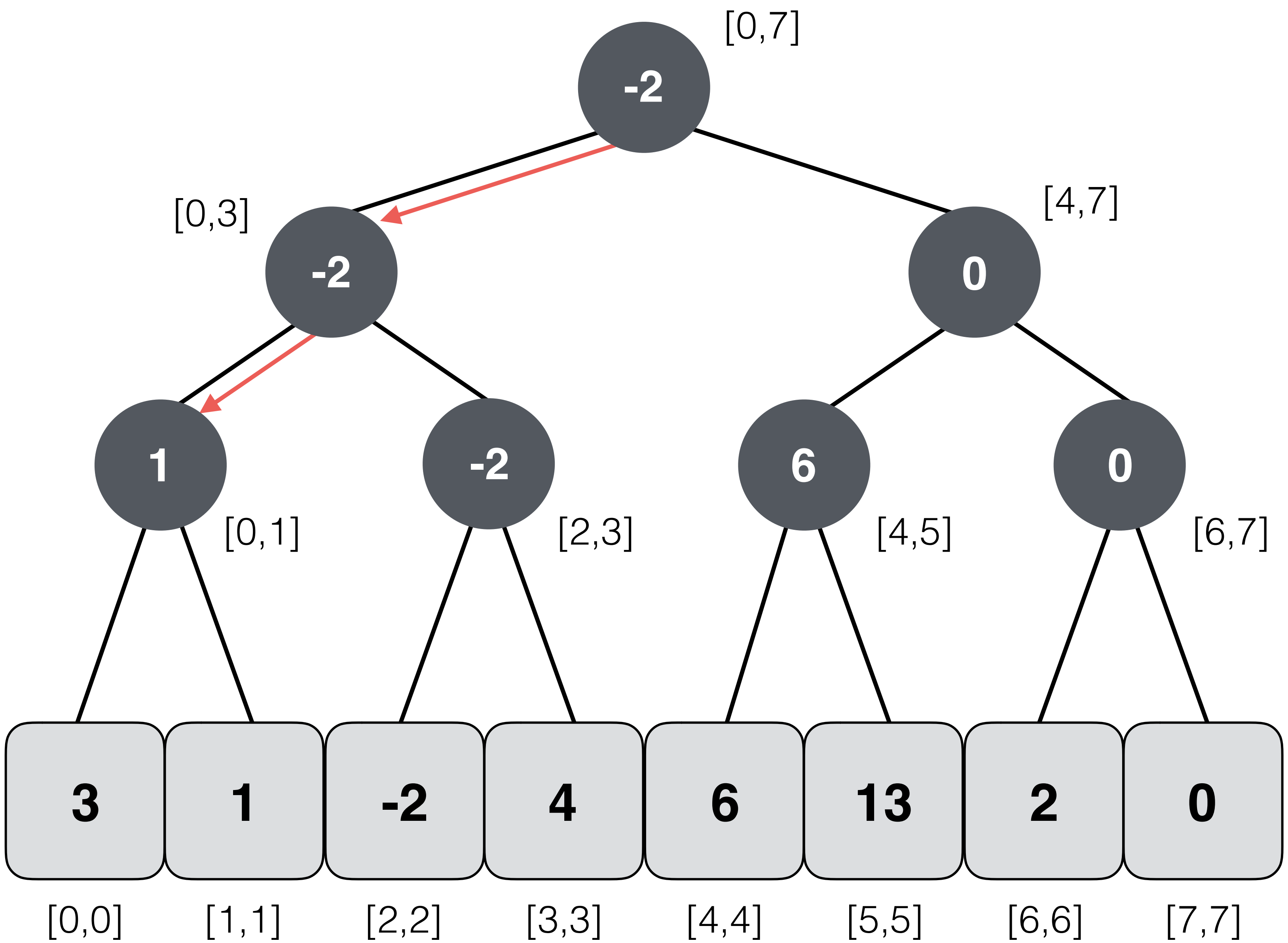
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

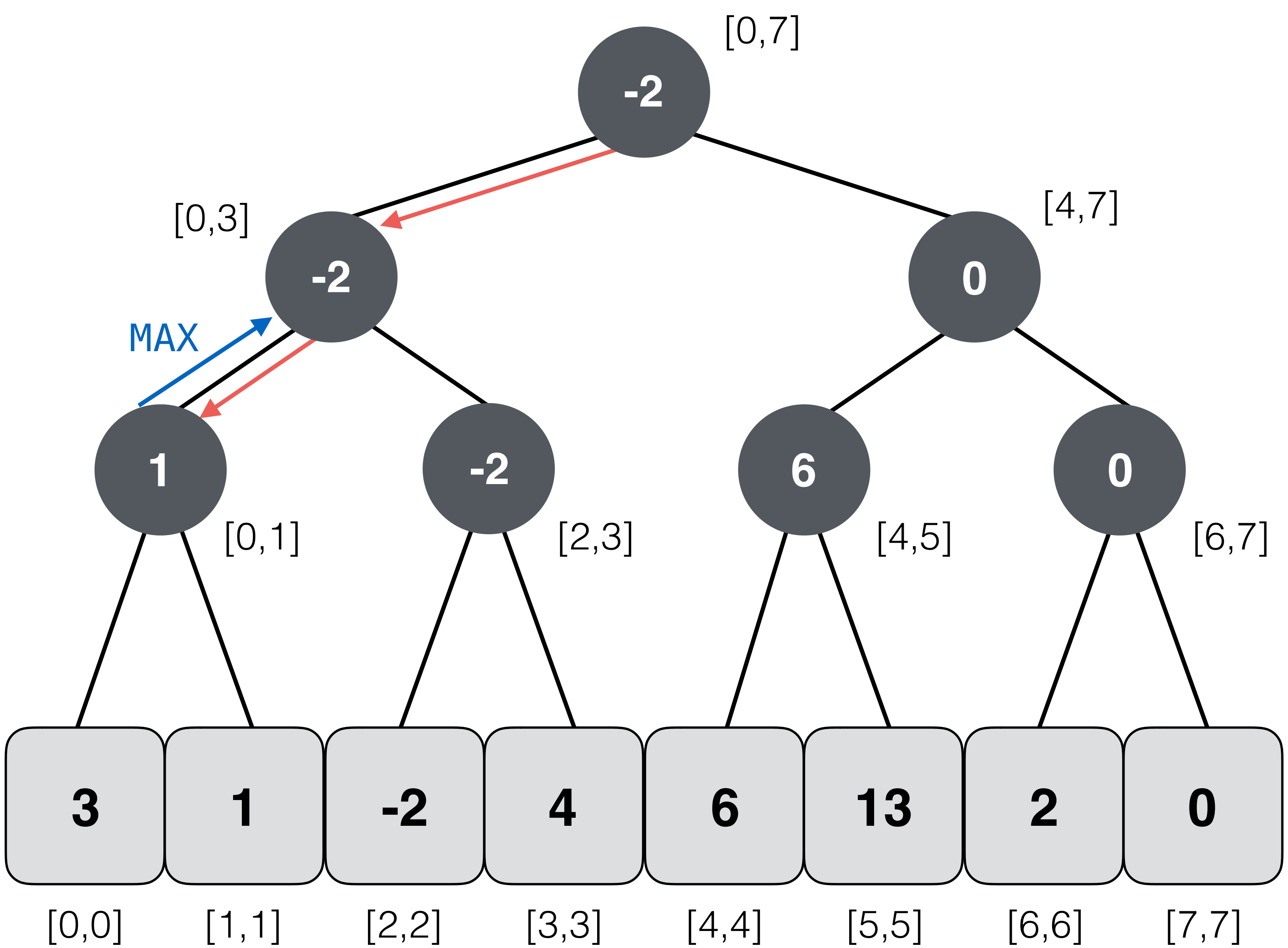
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

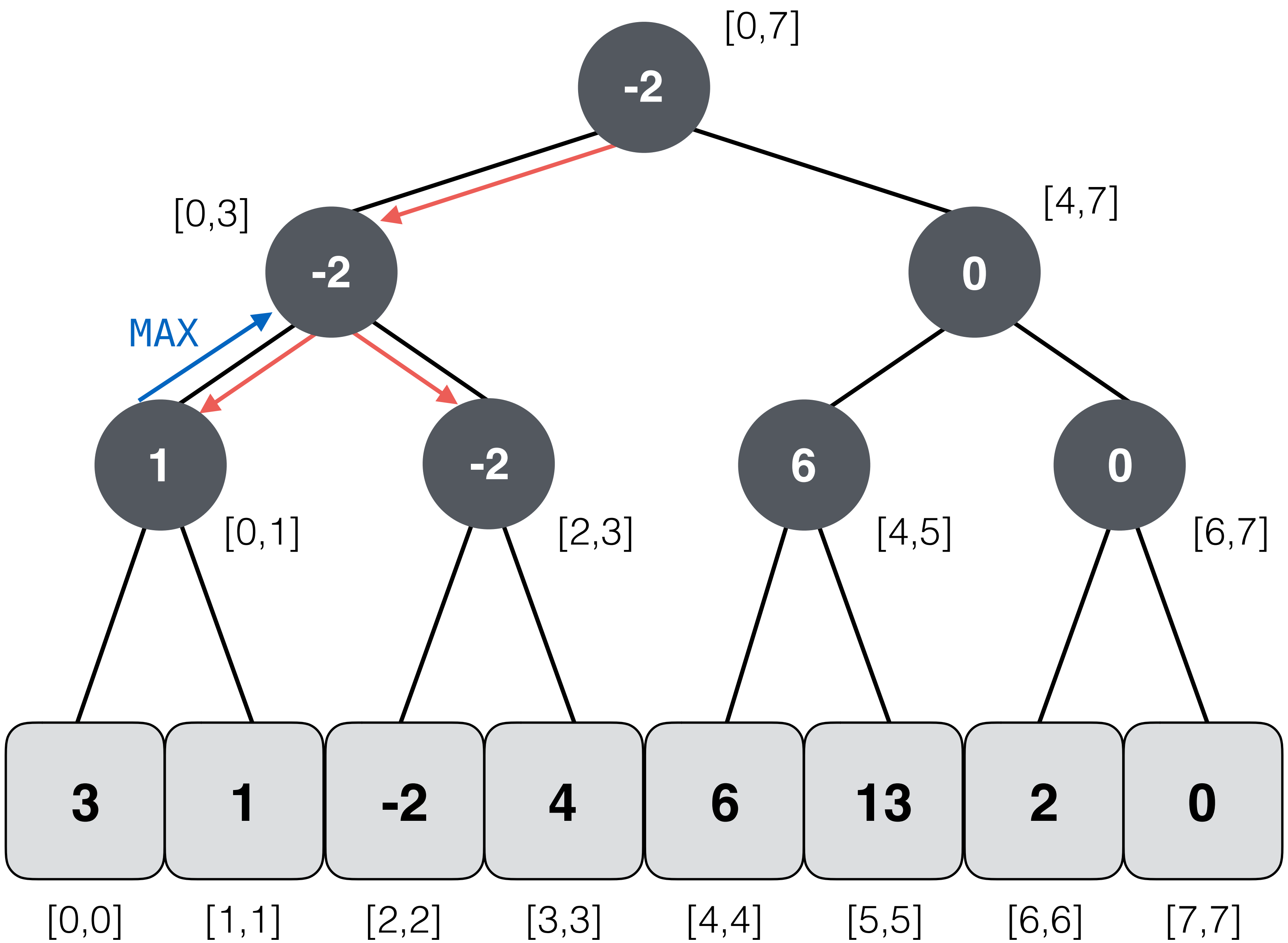
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

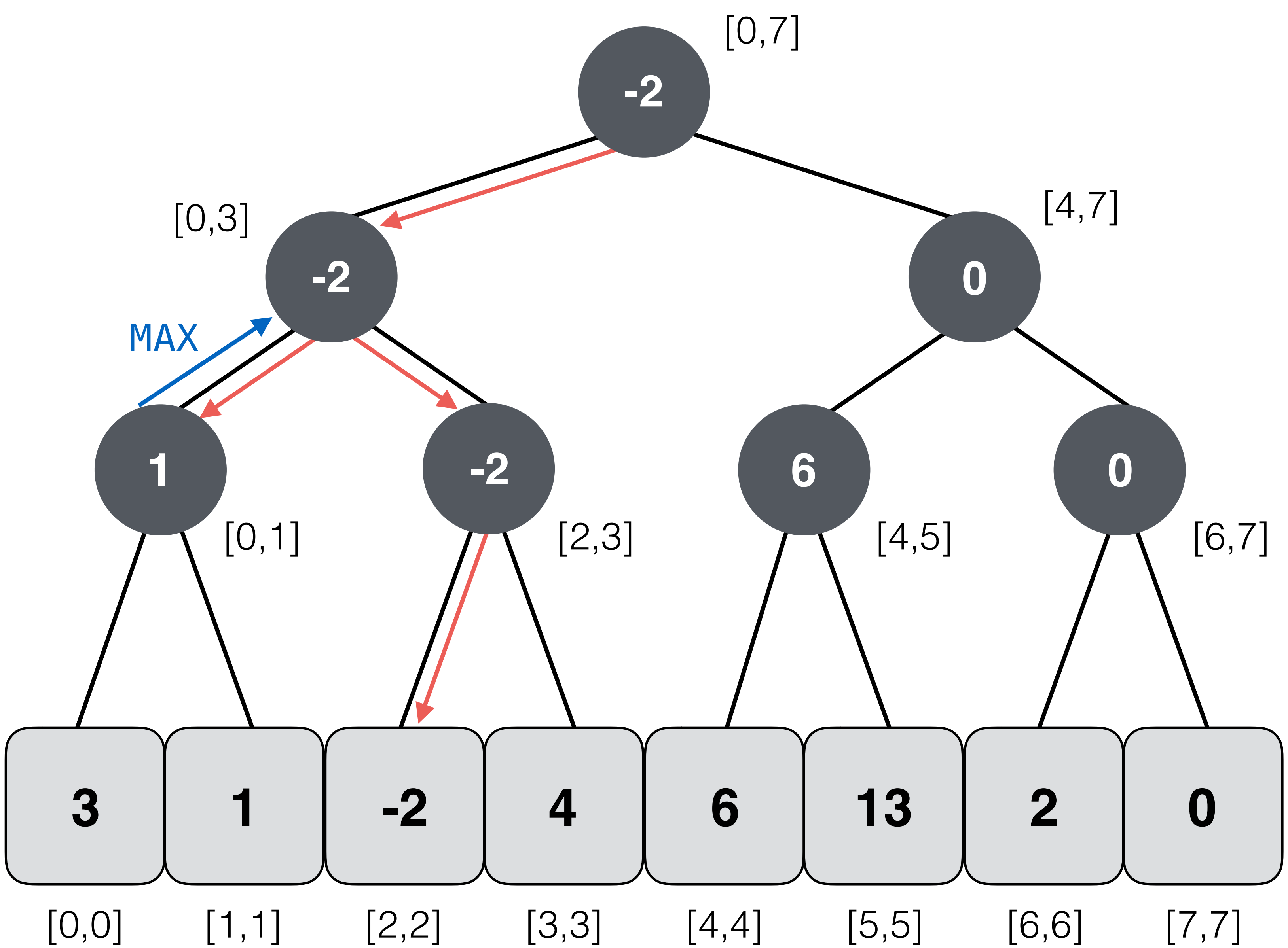
`min`(3,6)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

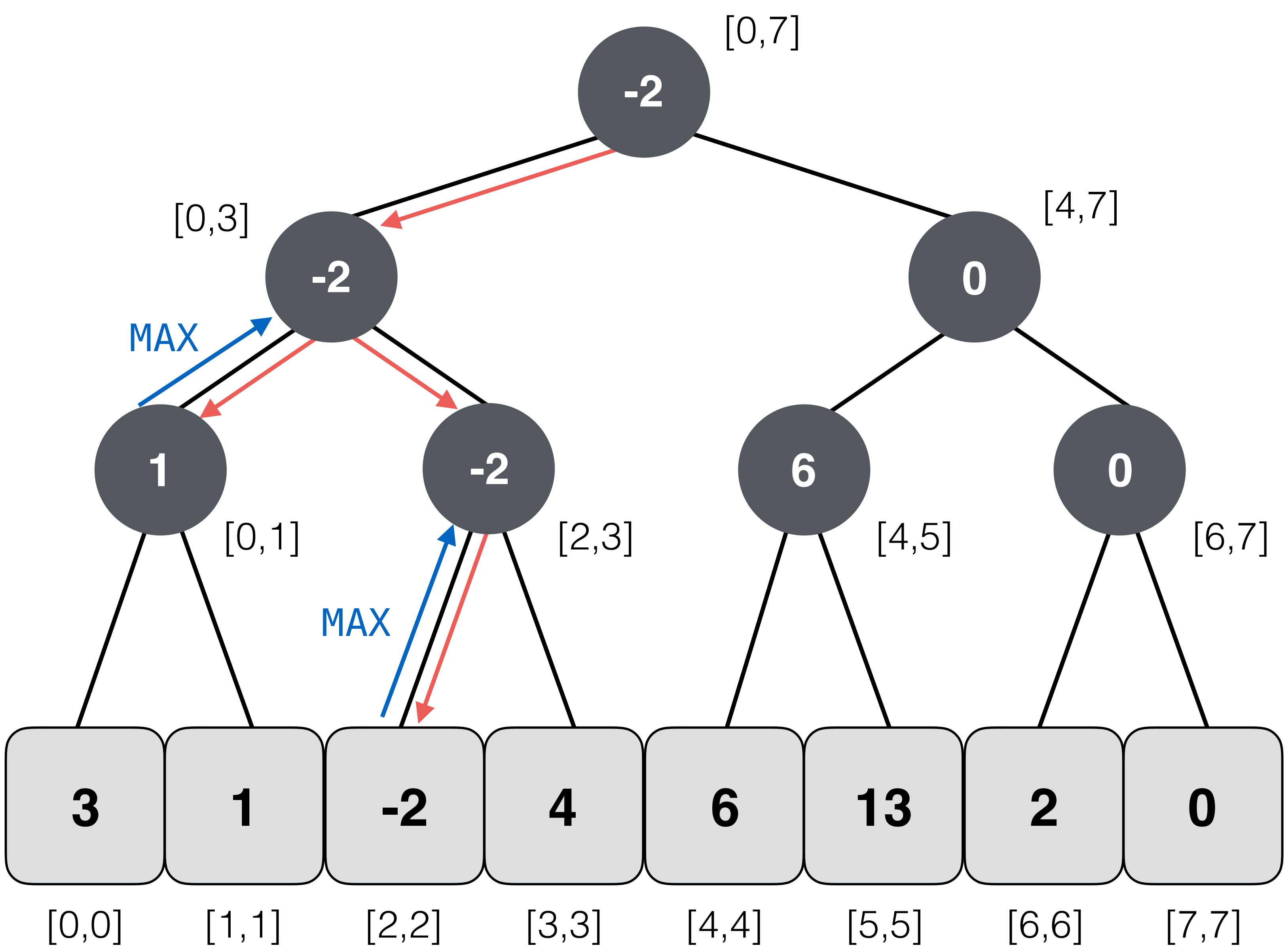
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

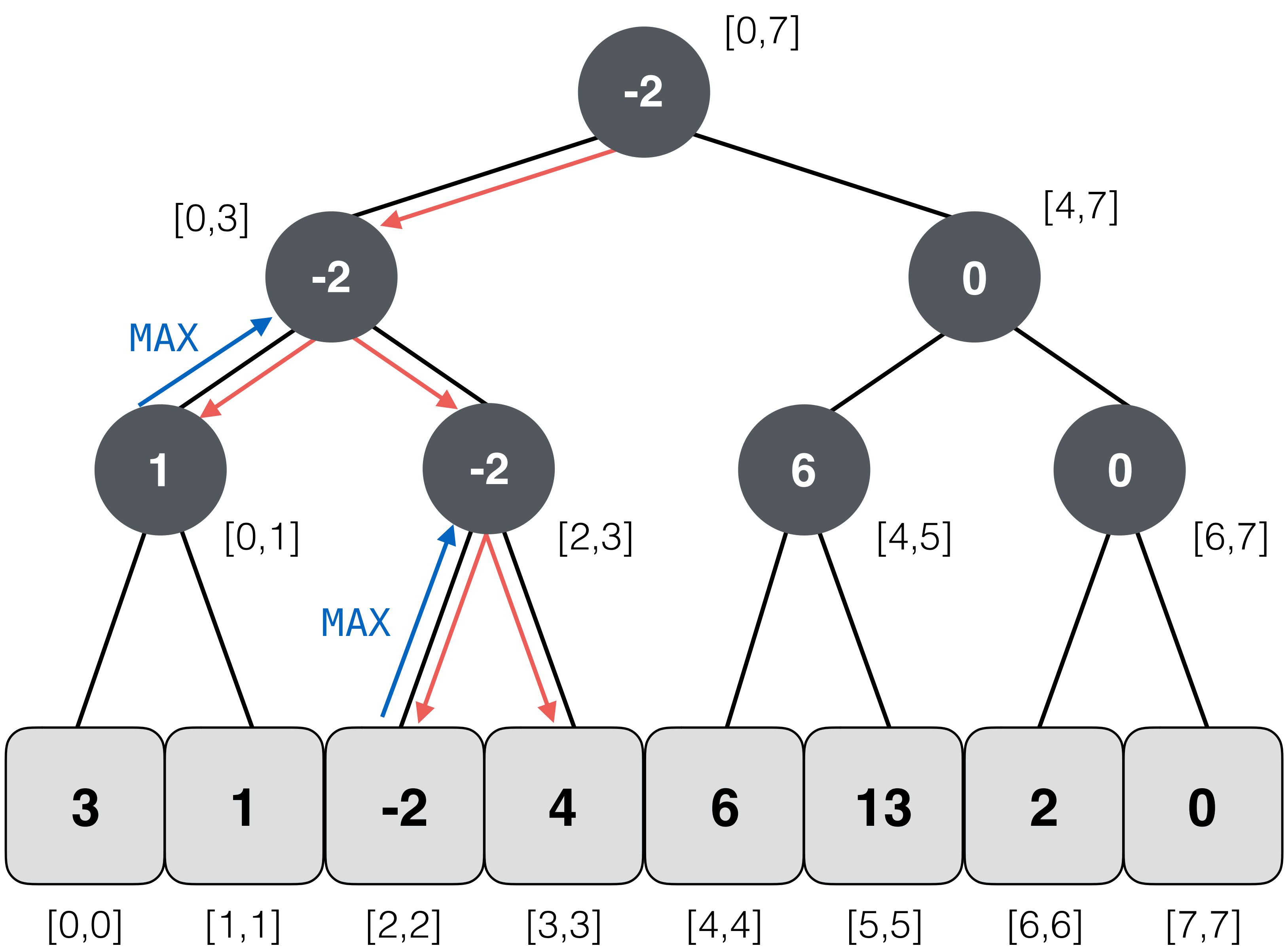
`min`(3,6)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

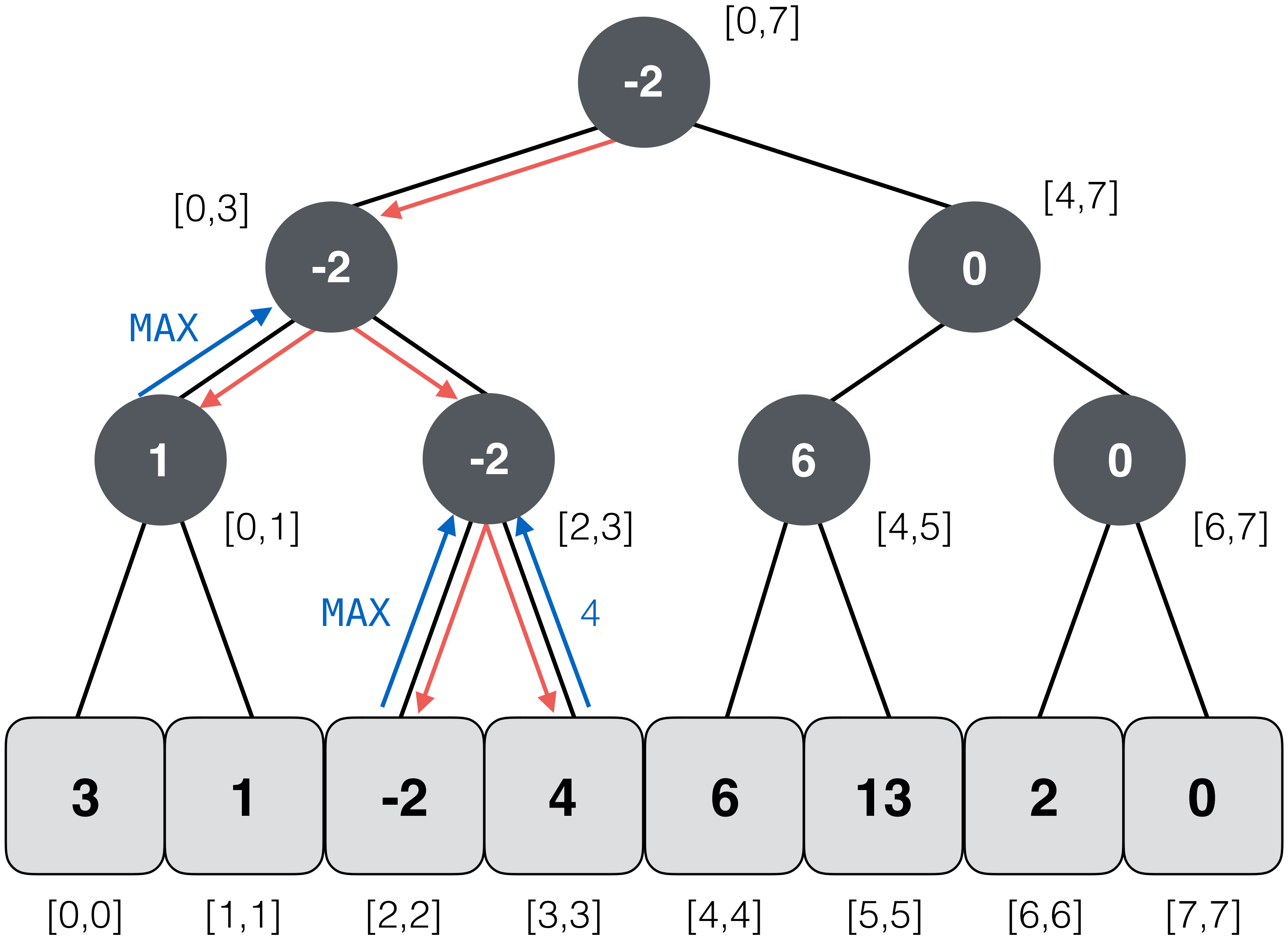
`min`(3,6)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

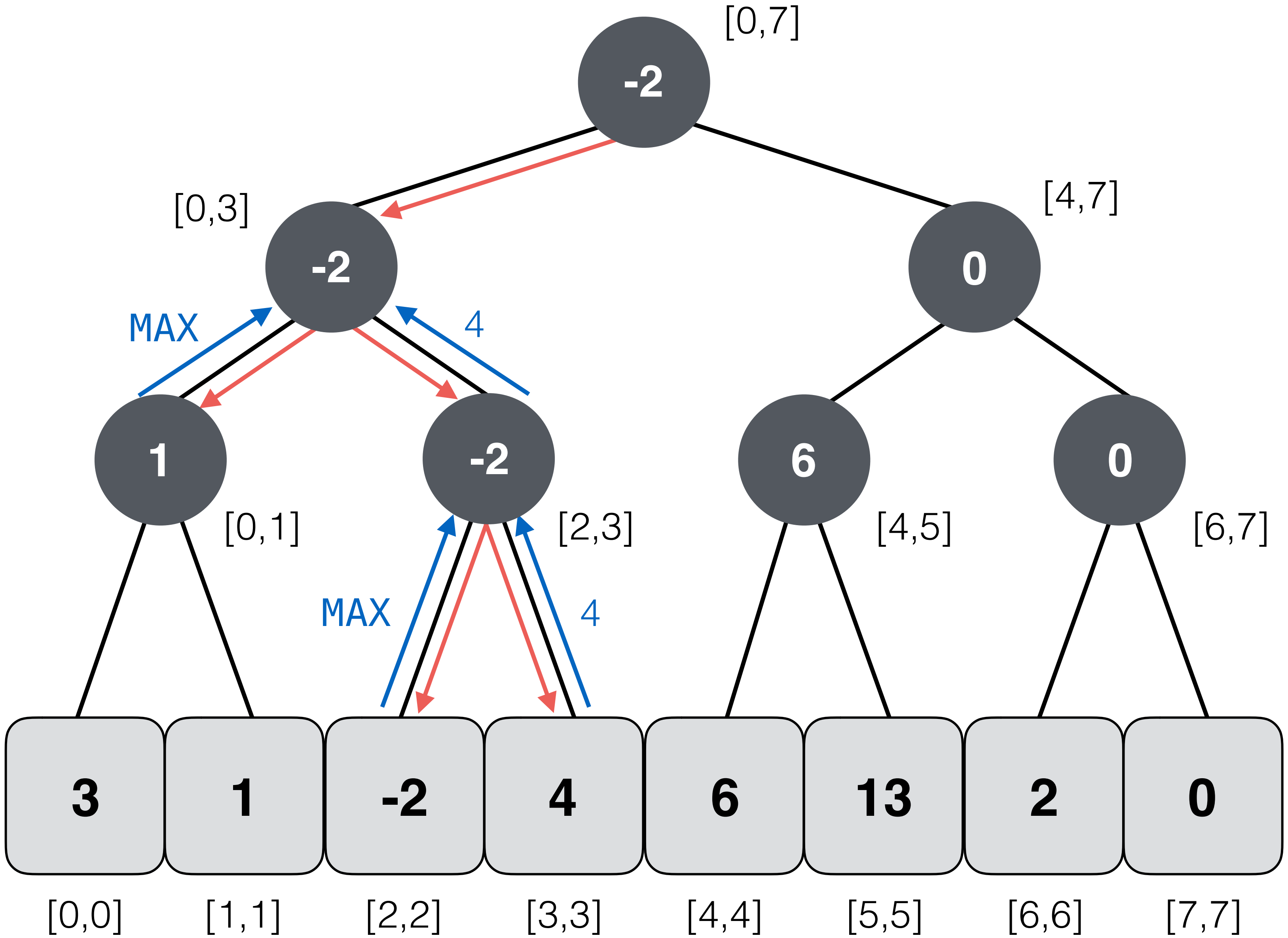
`min`(3,6)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

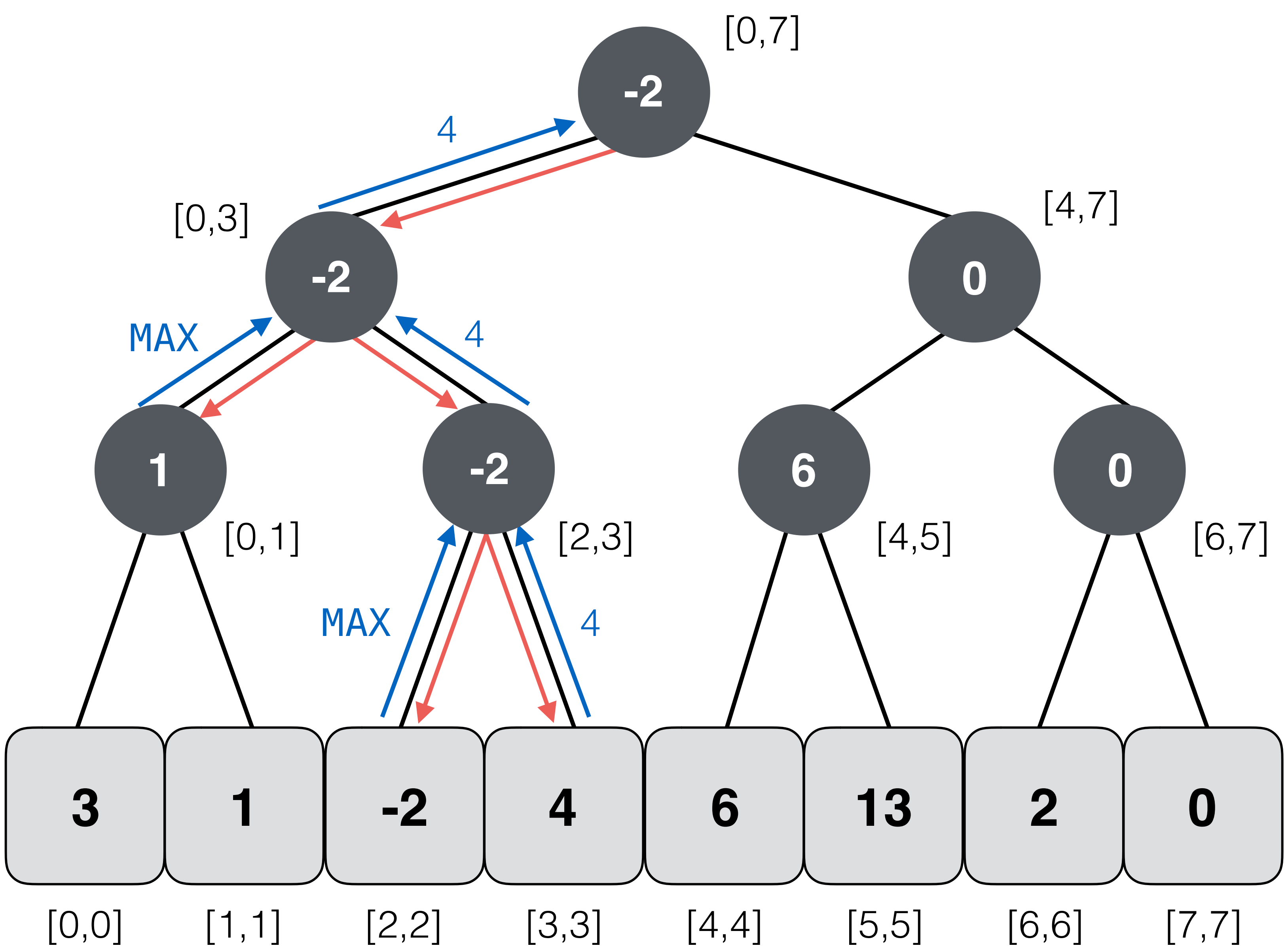
`min`(3,6)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

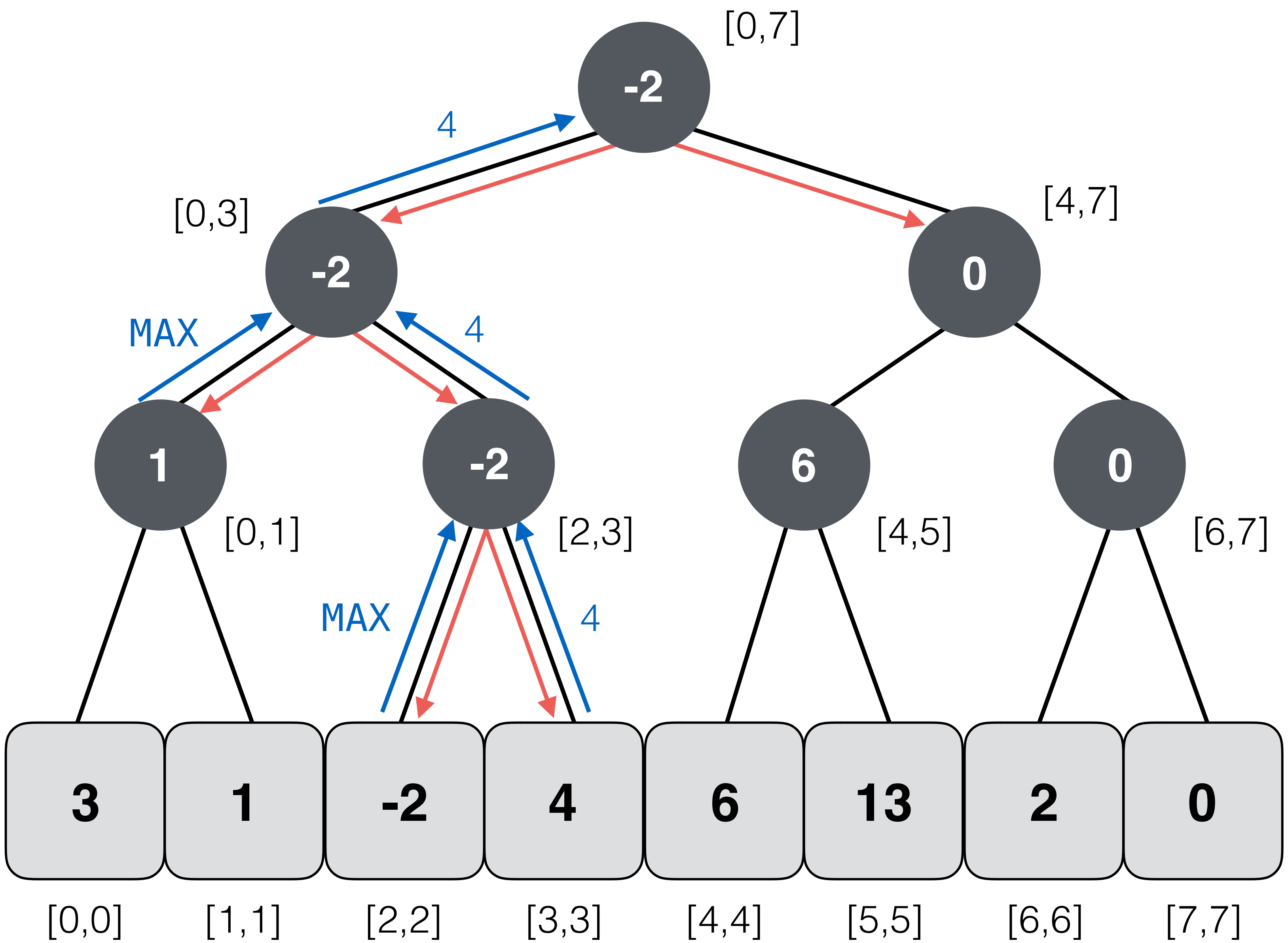
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

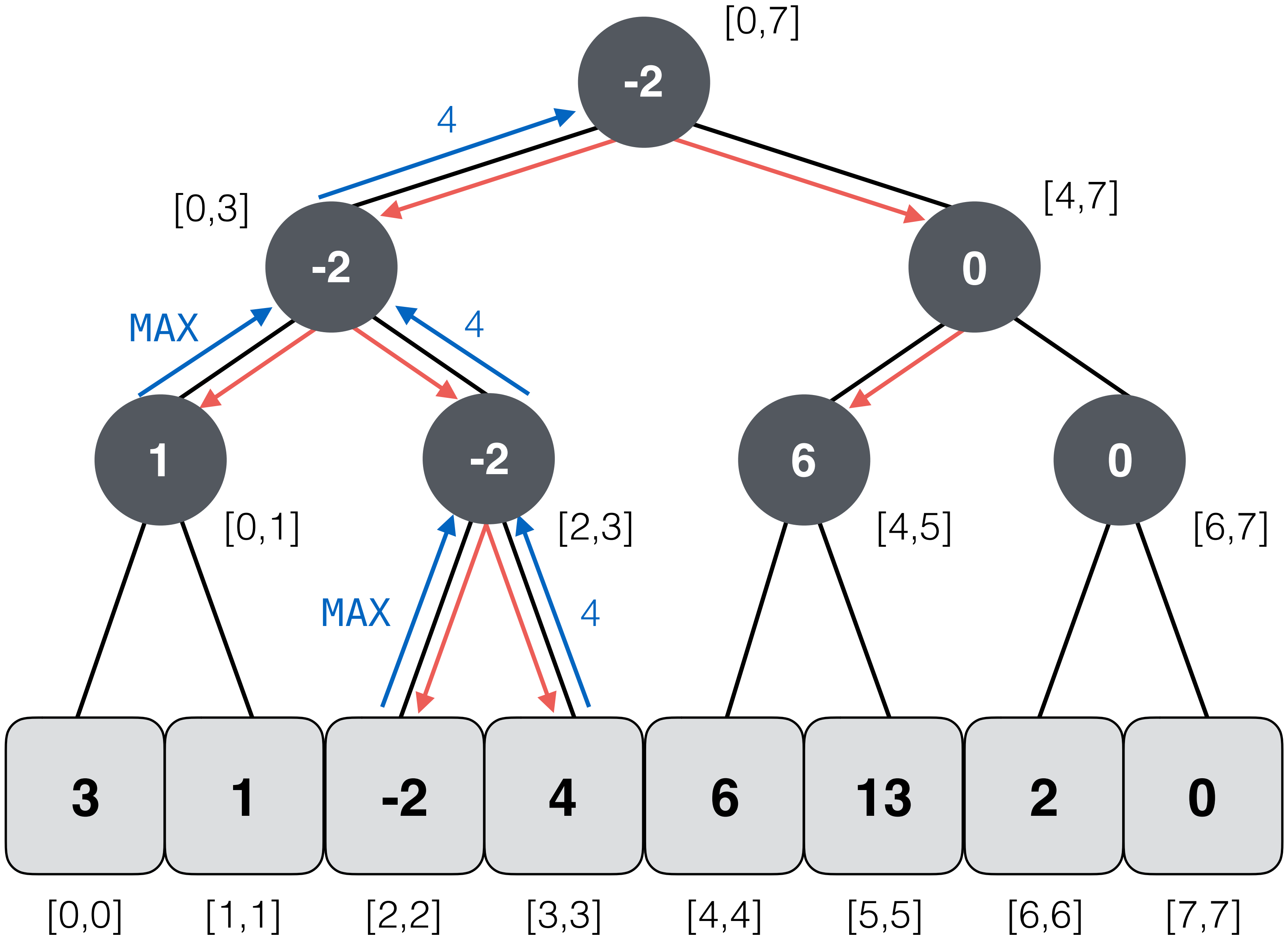
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

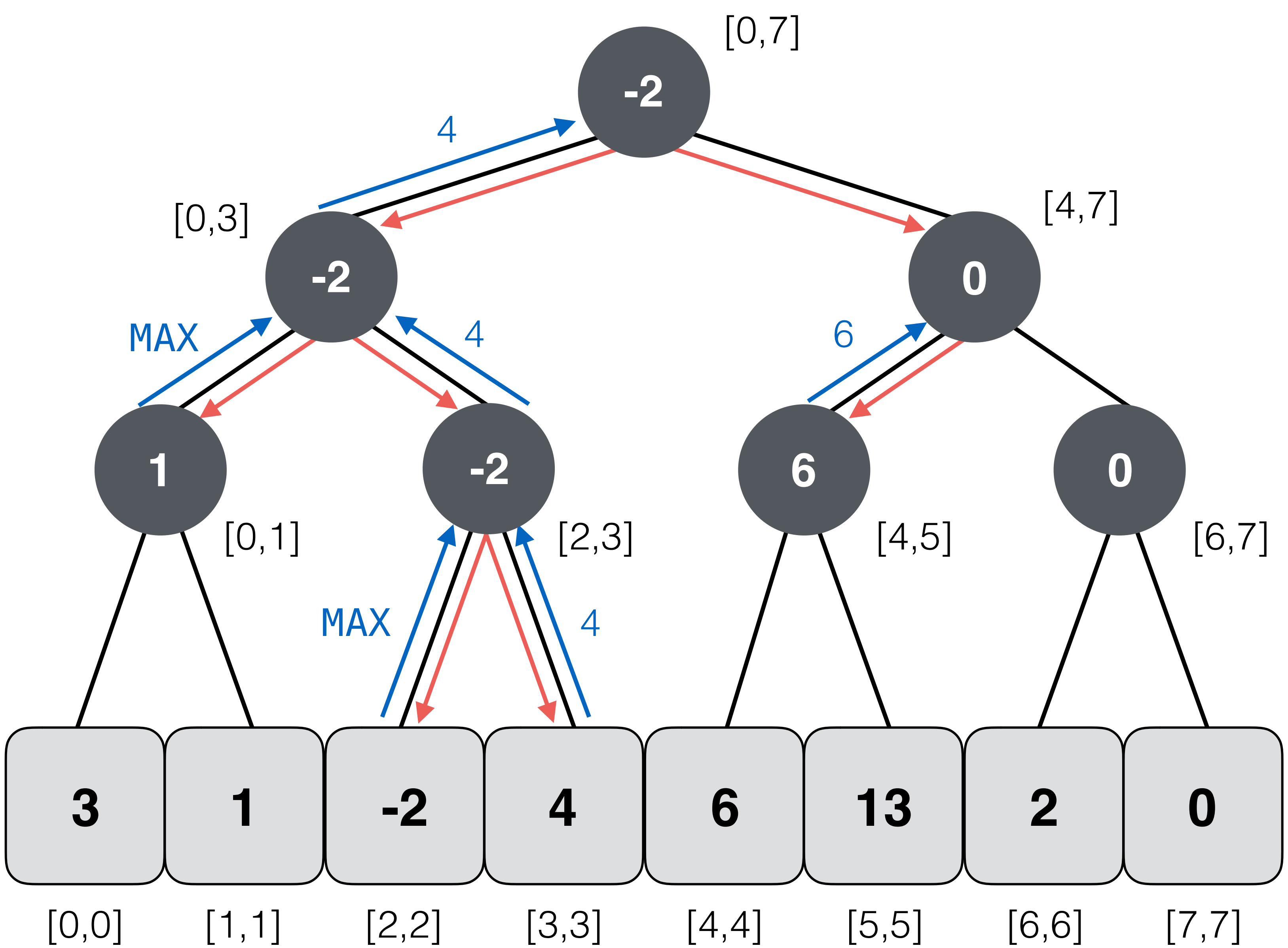
`min`(3,6)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

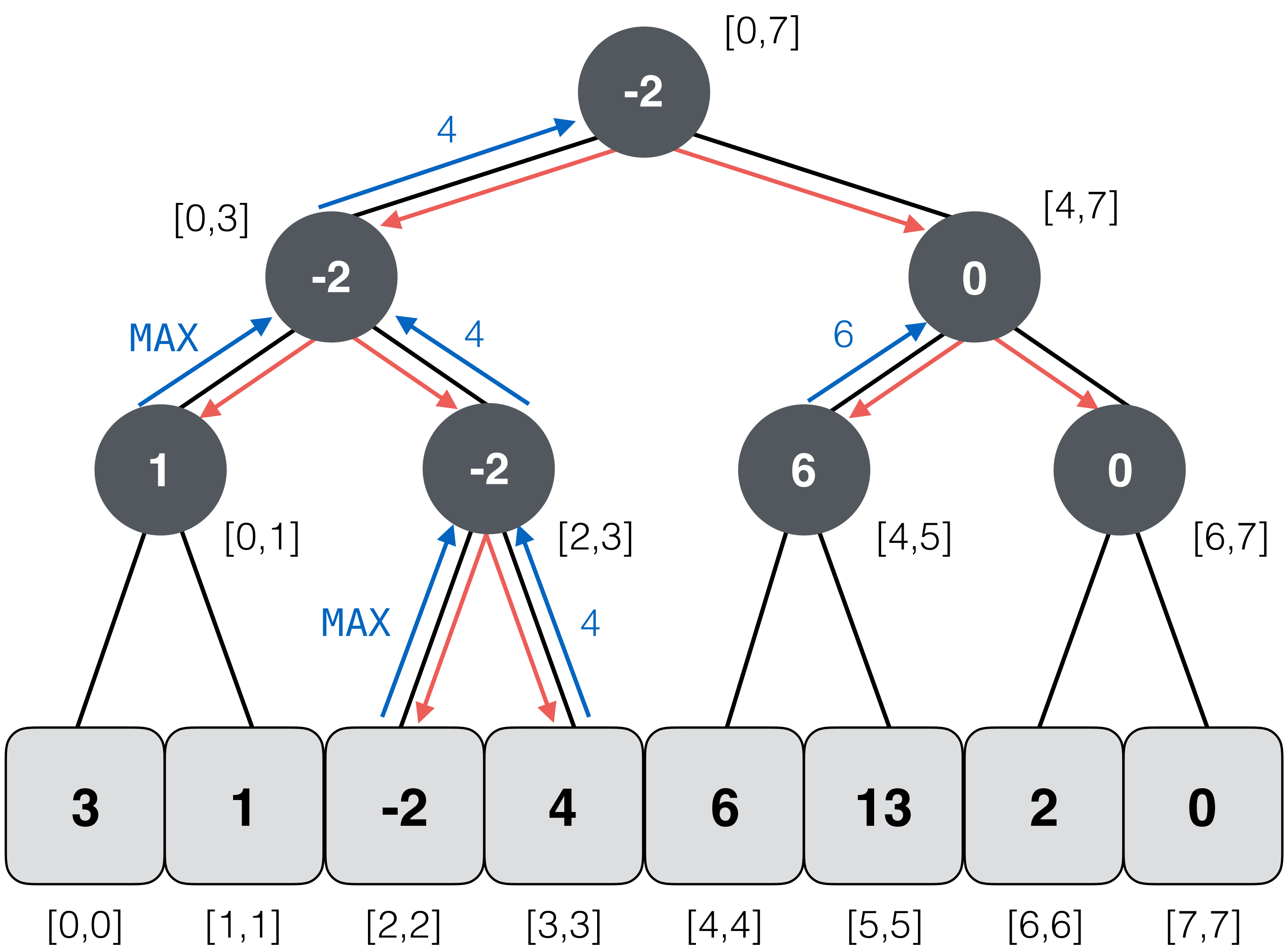
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

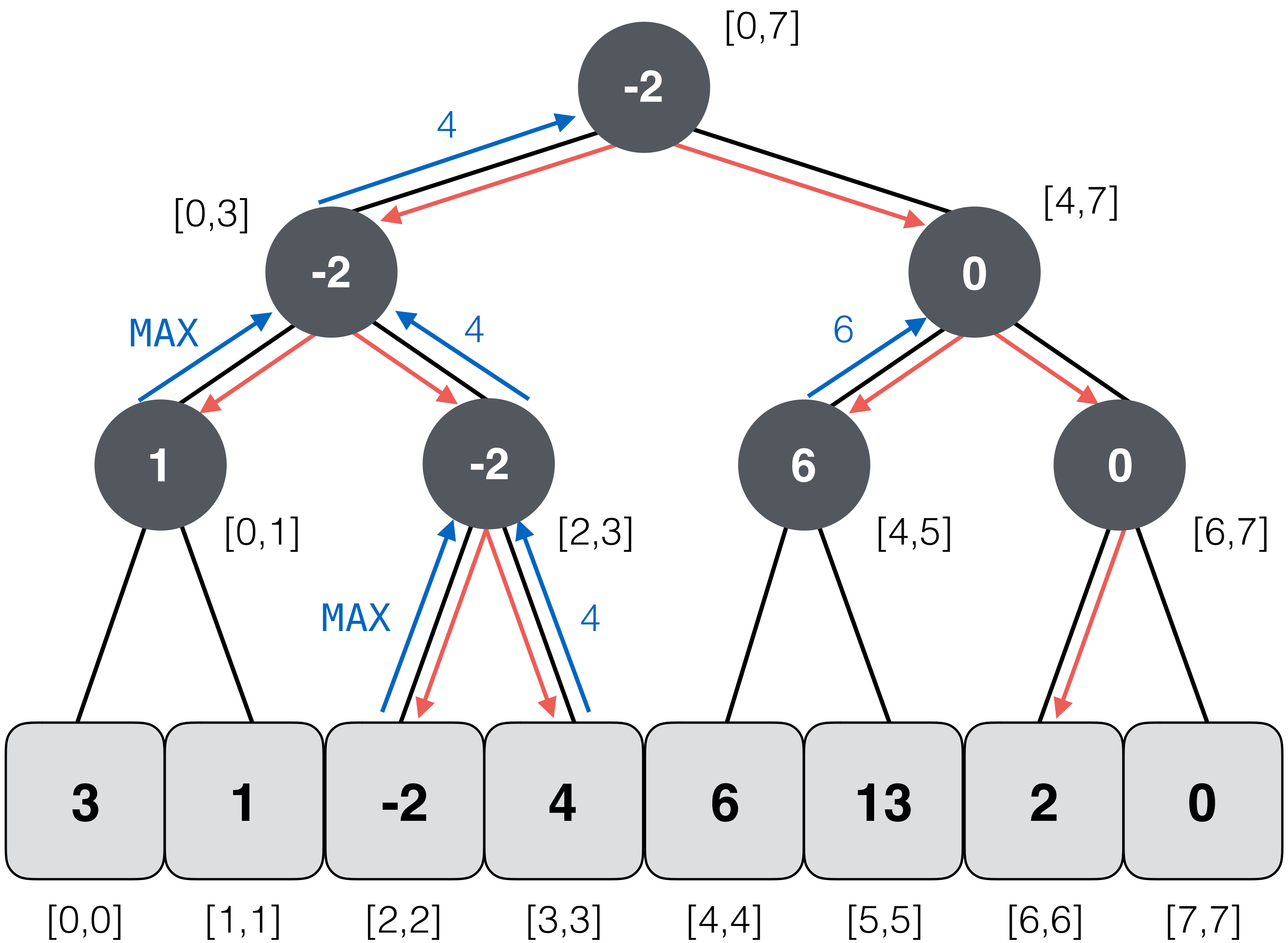
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

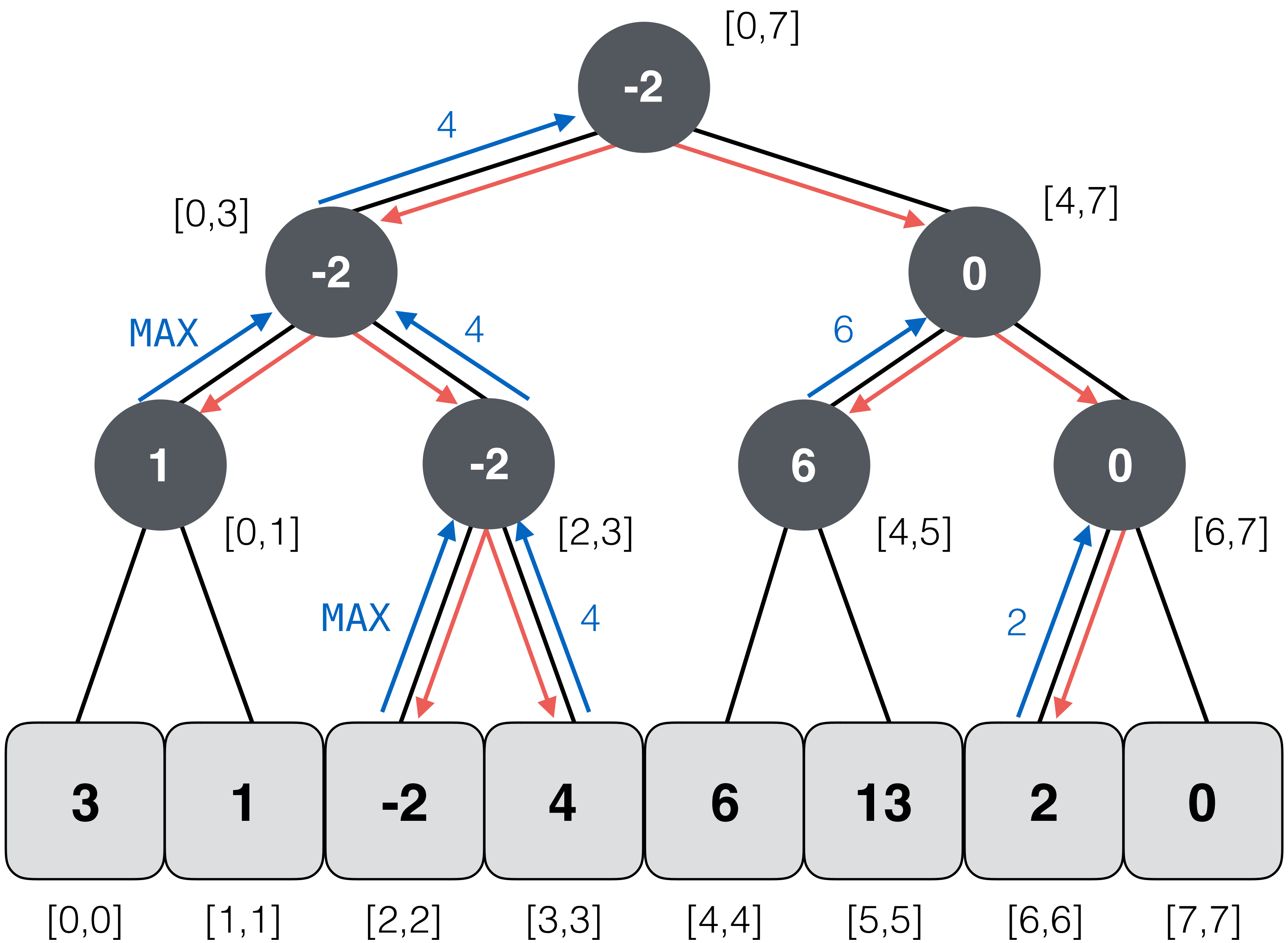
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

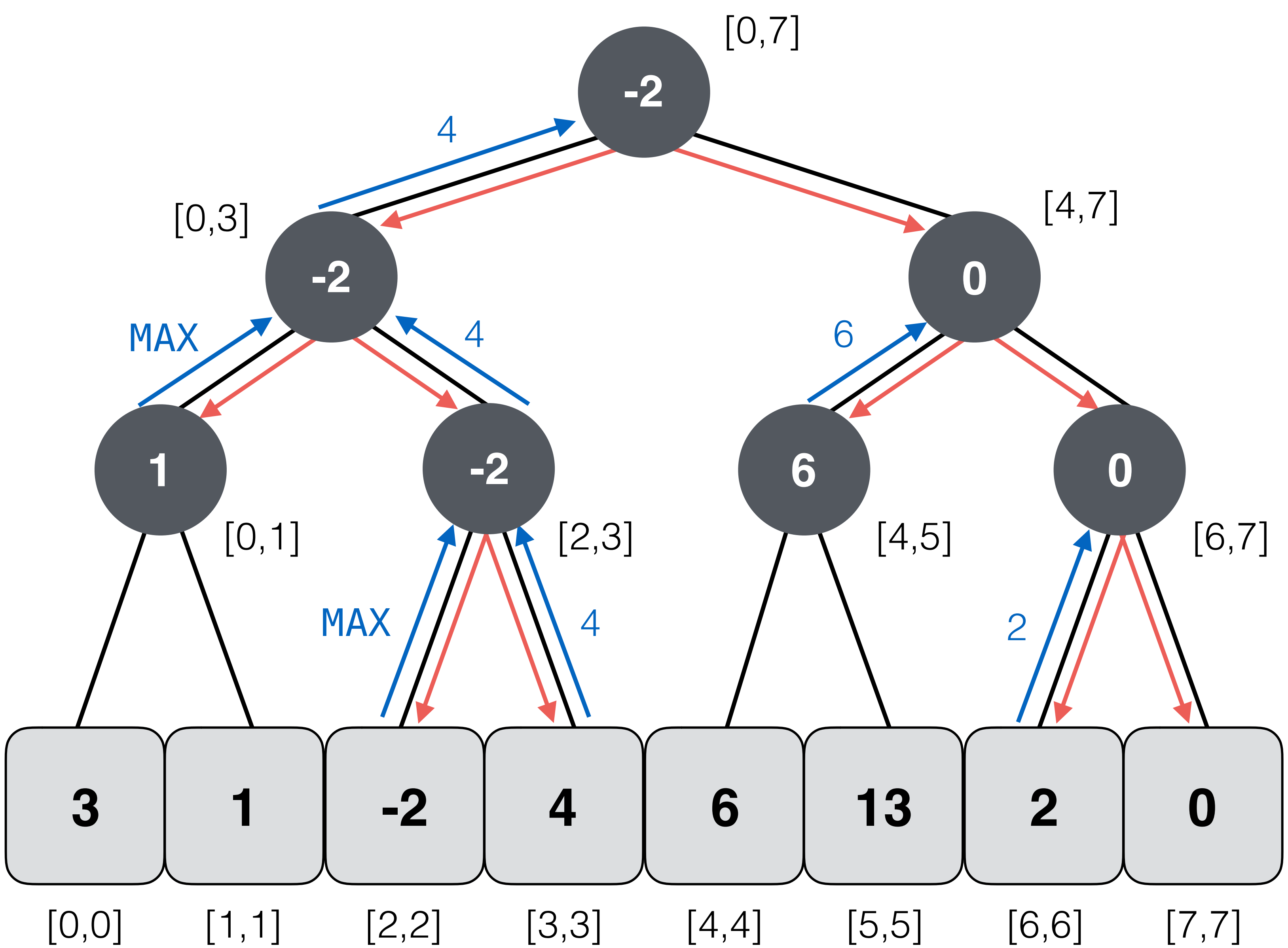
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

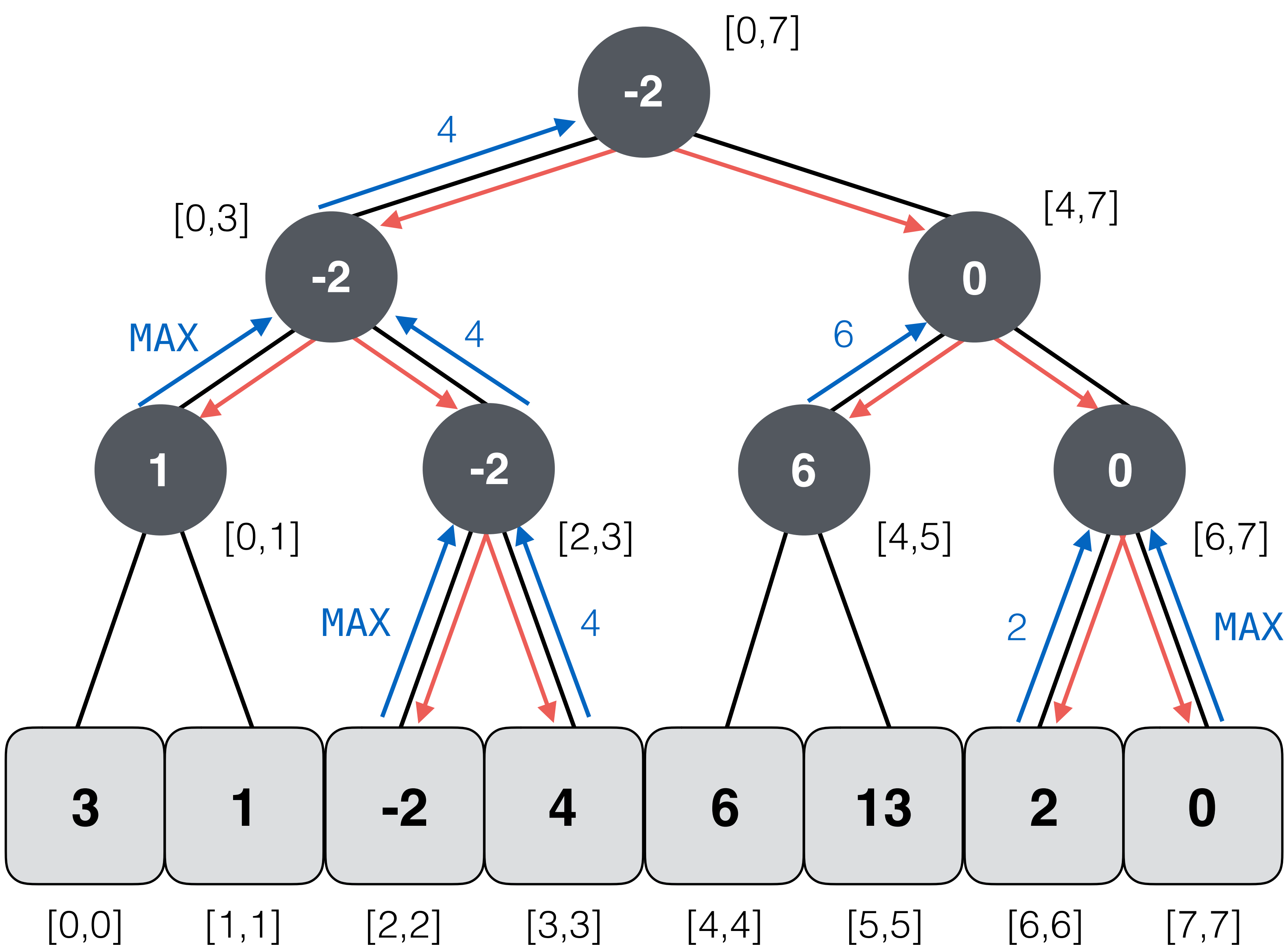
`min`(3,6)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

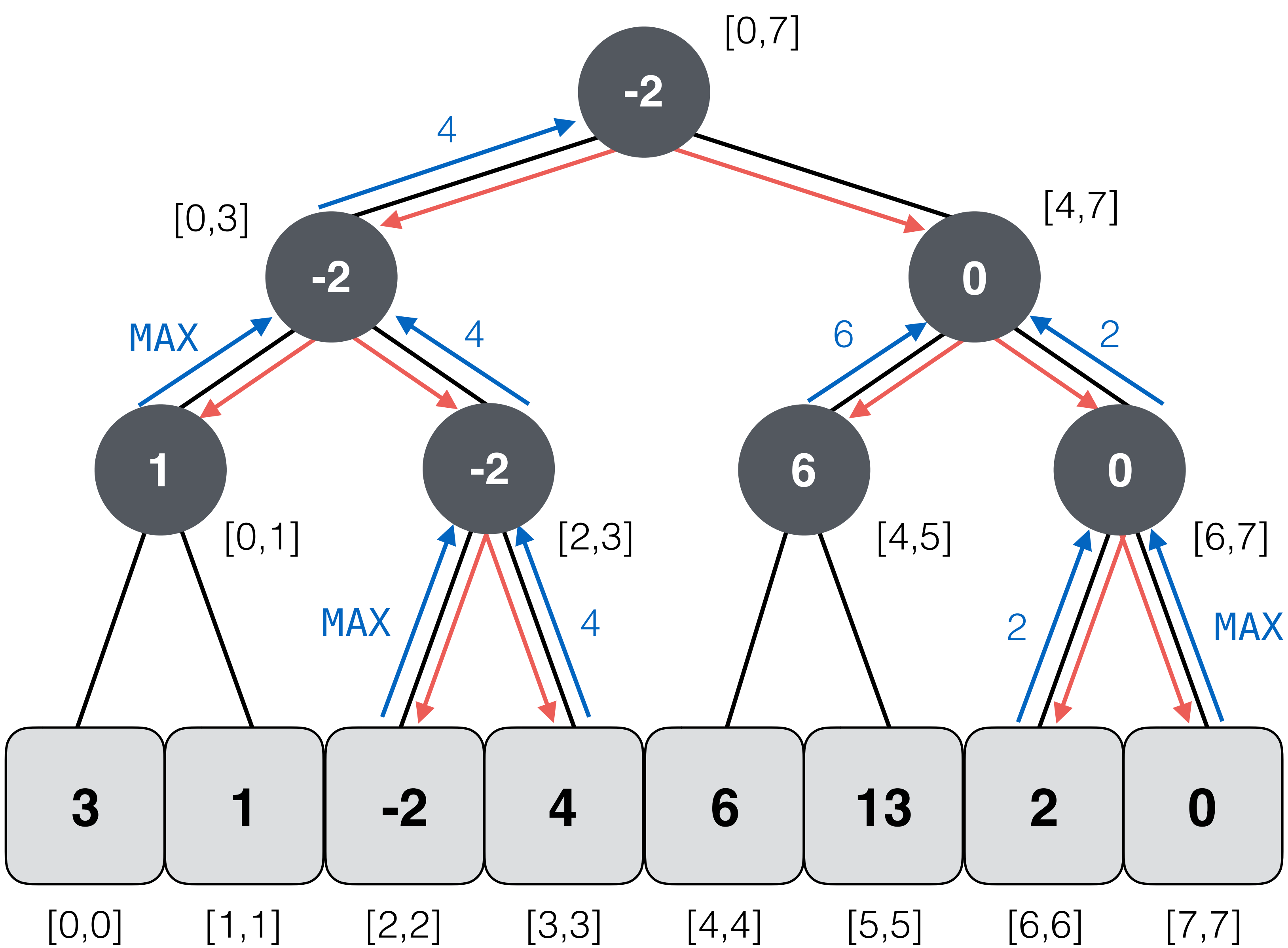
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

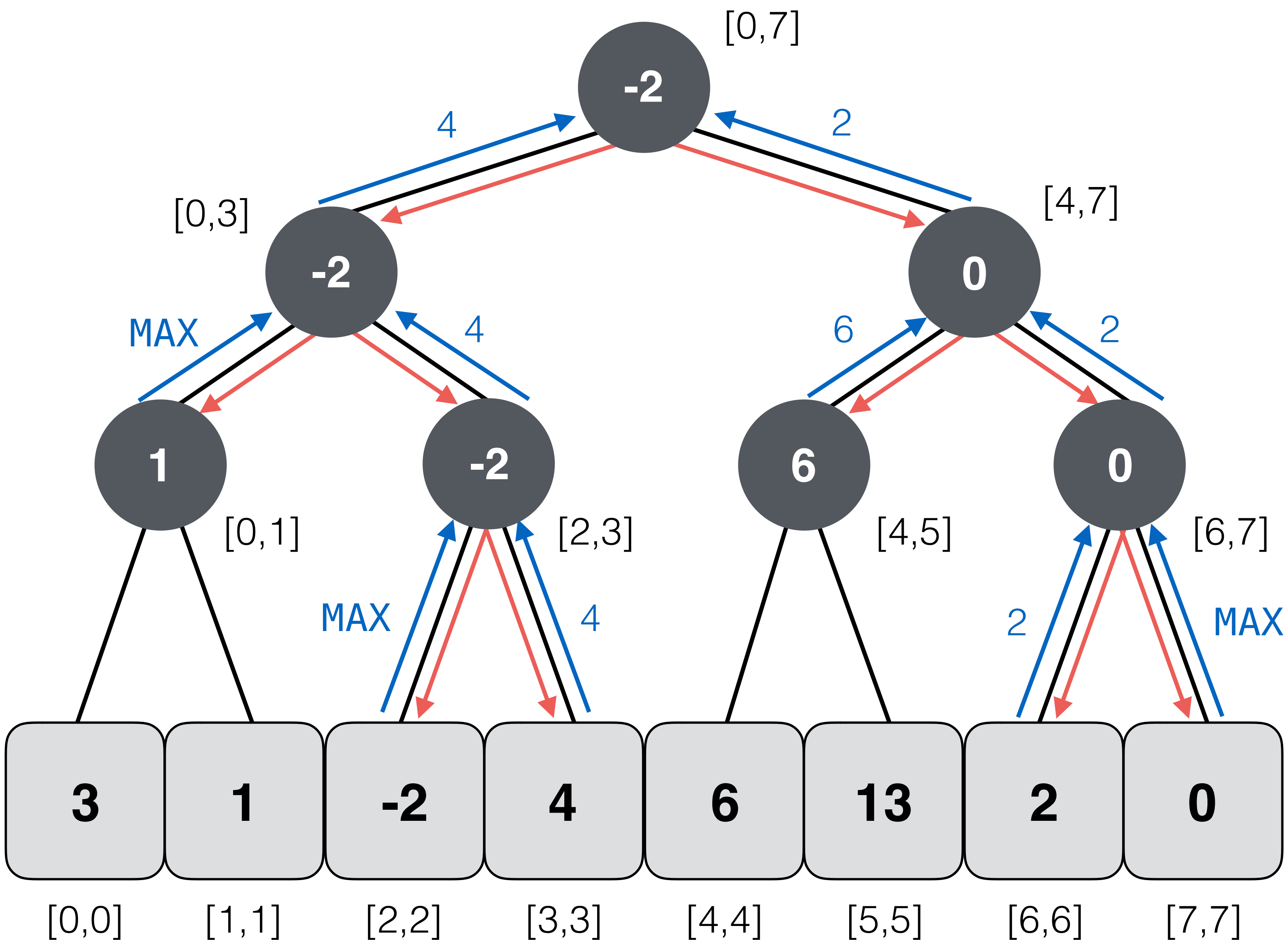
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

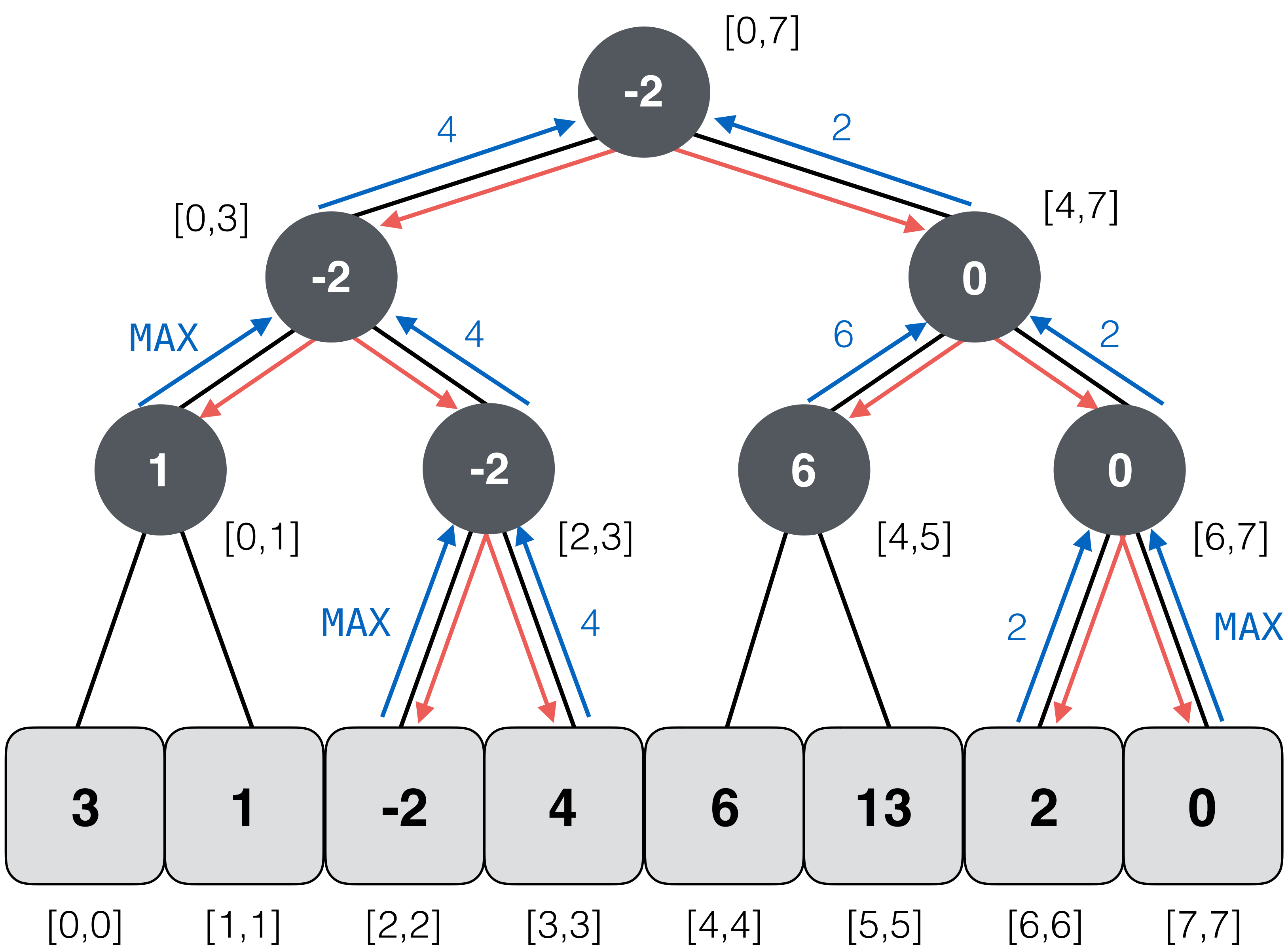
`min`(3,6)

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

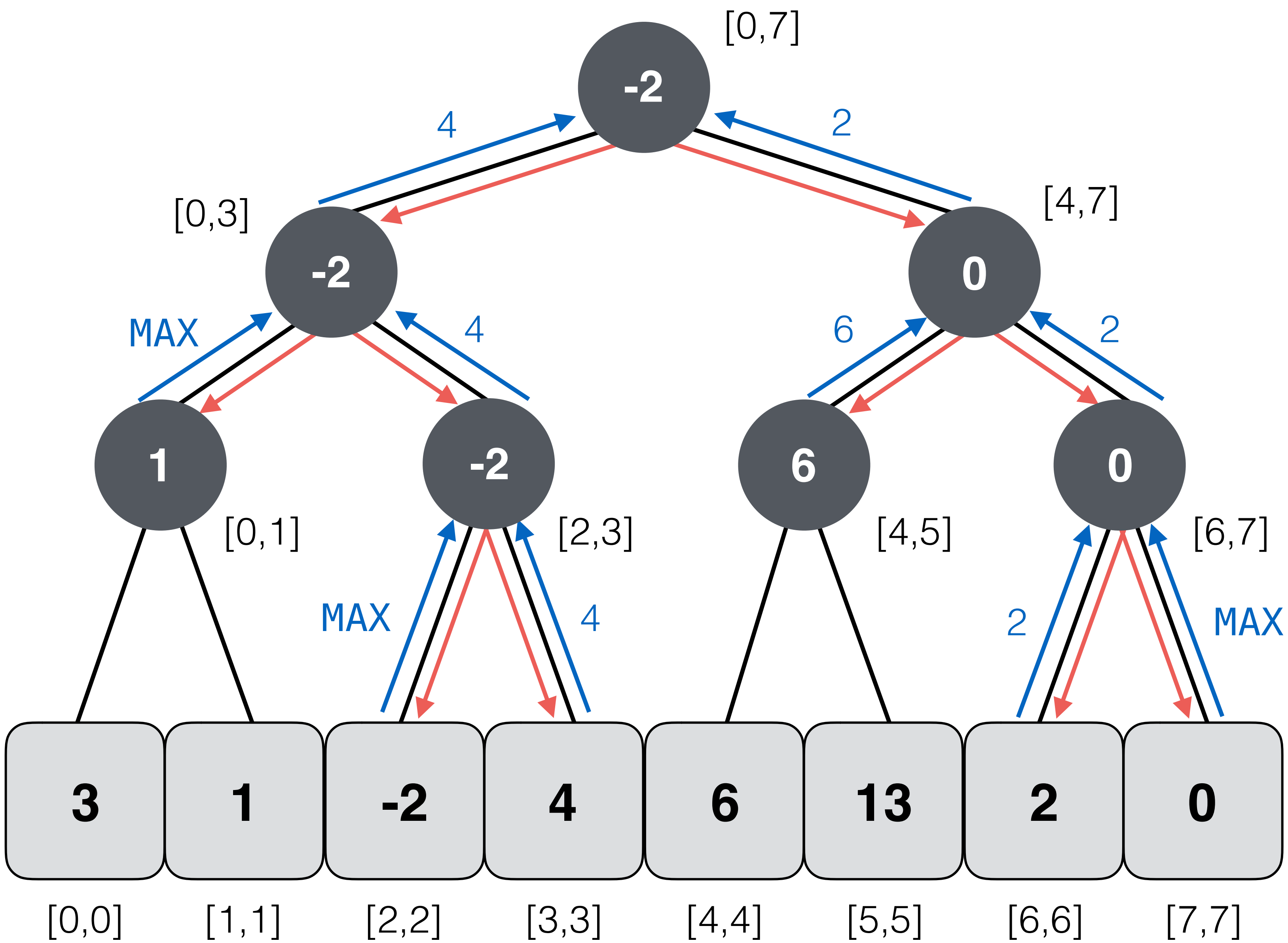
`min`(3,6) = 2

Range `MIN` Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

`min`(3,6) = 2

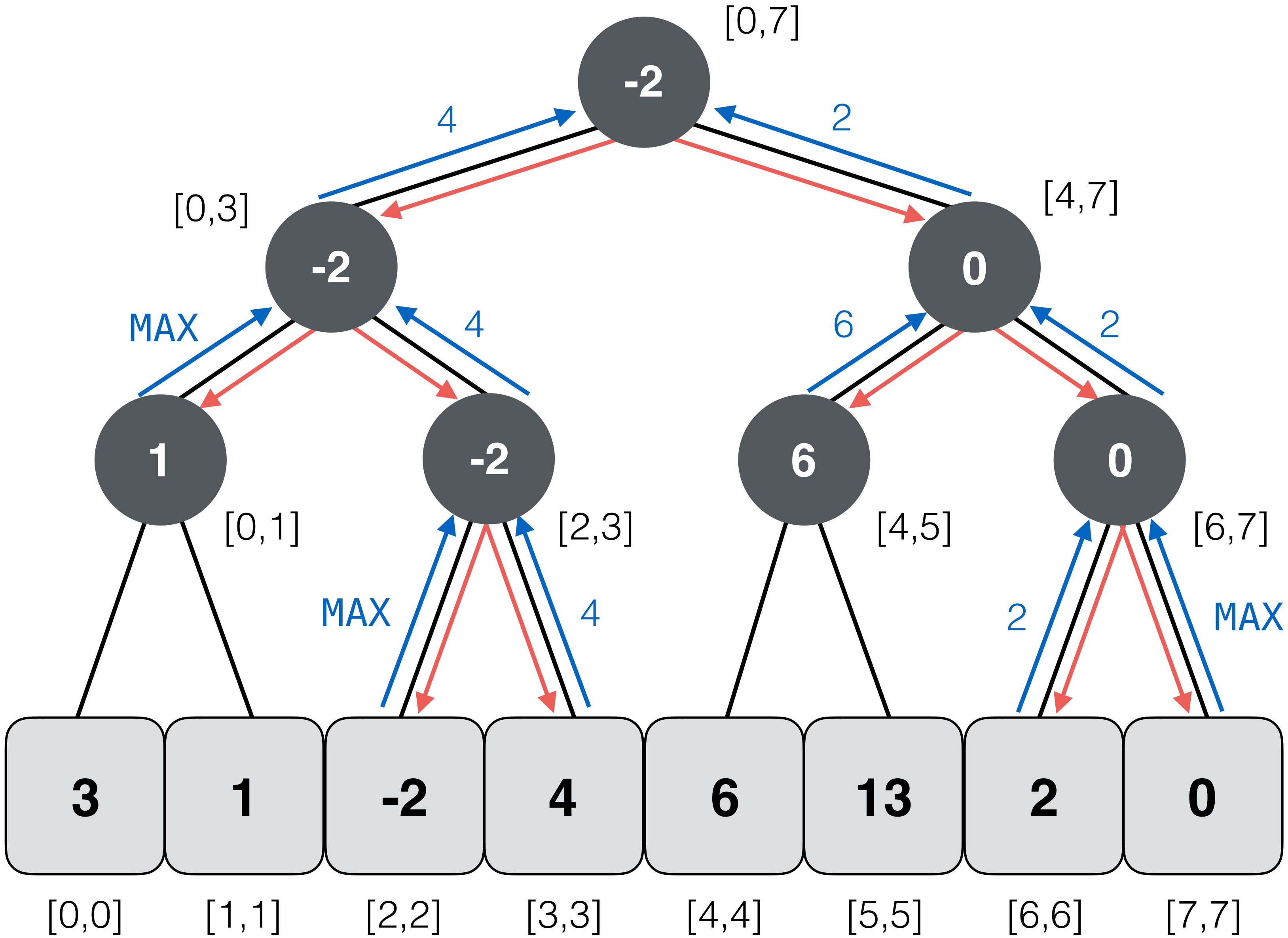
Query time: O(log n)
Space: O(n)
Building time: O(n)

Range MIN Queries with Segment Trees

Consider a segment tree with n leaves (2n - 1 nodes in total).

Given an interval [i,j], search for it in the tree.

- **partial overlap**: search in both subtrees
- **total overlap**: stop and return value at node
- **no overlap**: stop and return `MAX` value



`min`(1,3) = -2

`min`(3,6) = 2

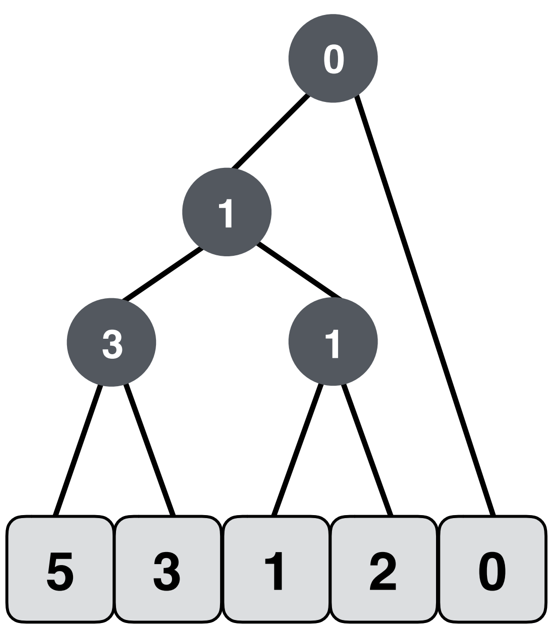
Query time: O(log n)
Space: O(n)
Building time: O(n)

How do we represent trees?

```
node* root = nullptr;
std::deque<node*> q;
int n = 0;
std::cin >> n;

for (int i = 0; i < n; ++i) {
    int x = 0;
    std::cin >> x;
    node* n = new node(x);
    q.push_back(n);
}

node* last = nullptr;
if (n % 2) {
    last = q.back();
    q.pop_back();
}

auto min_parent = [&](node* left, node* right) {
    int min = std::min<int>(left->key, right->key);
    node* parent = new node(min, left, right);
    q.push_back(parent);
};

while (q.size() != 1) {
    min_parent(q[0], q[1]);
    q.pop_front();
    q.pop_front();
}

if (last != nullptr) {
    min_parent(q.front(), last);
    q.pop_front();
}

root = q.front();
```

Pointers

```
struct node {
    node(int k,
         node* l = nullptr,
         node* r = nullptr)
       : key(k), left(l), right(r)
    {}

    int key;
    node* left;
    node* right;
};
```

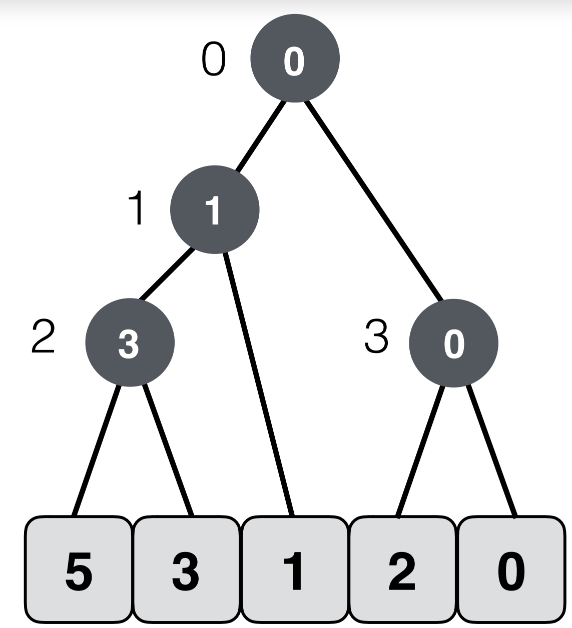


How do we represent trees?

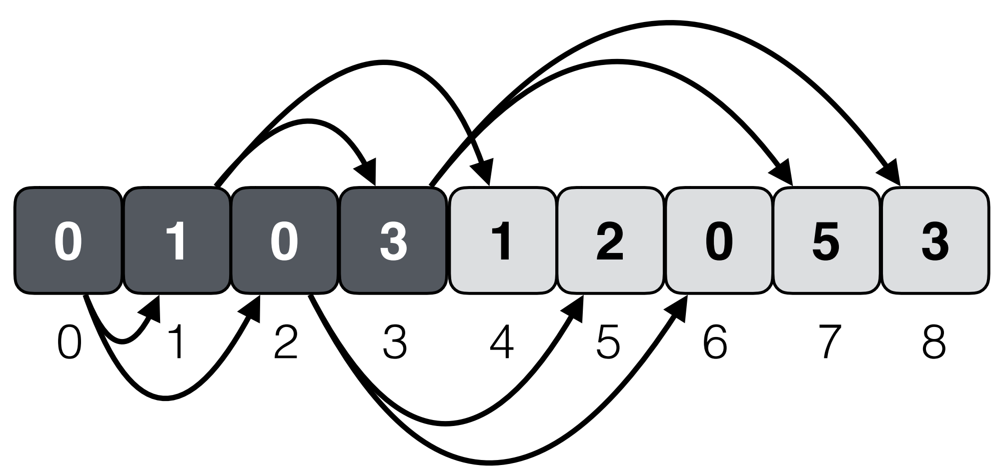
Arrays

```
std::vector<int> tree;

int n = 0;
std::cin >> n;
int tree_size = 2 * n - 1;
tree.resize(tree_size);

int h = ceil(log2(n));
 // left-most node id following level order
int left_most_node = (int(1) << (h - 1)) - 1;
int offset = 2 * left_most_node + 1;

// set leaves circularly

// 1. go forward
int i = 0;
for (int j = offset; j != tree_size; ++i, ++j) {
    int x = 0;
    std::cin >> x;
    tree[j] = x;
}

// 2. fall back
for (int j = 0; i != n; ++i, ++j) {
    int x = 0;
    std::cin >> x;
    tree[n - 1 + j] = x;
}

// set internal nodes
for (int i = tree_size - 1; i != 0; i -= 2) {
    int min = std::min<int>(tree[i], tree[i - 1]);
    tree[PARENT(i)] = min;
}
```

```
#define LEFT(i)     2 * i + 1
#define RIGHT(i)    2 * i + 2
#define PARENT(i) (i - 1) / 2
```



Pointers are implicit!



How do we represent trees?

Remember

Be skeptic: *measure* first and then conclude.

How do we represent trees?

Remember
Be skeptic: *measure* first and then conclude.

Pointers vs. **Arrays**

How do we represent trees?

Remember

Be skeptic: *measure* first and then conclude.

Pointers VS. **Arrays**

Experiment over 5 million nodes

Visit the tree and compute the sum of all nodes.

```
→  segment_trees git:(master) ✗ g++ -std=c++11 -O3 trees_with_pointers.cpp -o trees_with_pointers
→  segment_trees git:(master) ✗ ./trees_with_pointers < input7
building took: 19.3895 [sec]
sum is: 3676408720
sum took: 0.118645 [sec]
→  segment_trees git:(master) ✗ g++ -std=c++11 -O3 trees_with_arrays.cpp -o trees_with_arrays
→  segment_trees git:(master) ✗ ./trees_with_arrays < input7
building took: 18.0724 [sec]
sum is: 3676408920
sum took: 0.0118351 [sec]
→  segment_trees git:(master) ✗
```

How do we represent trees?

Remember

Be skeptic: *measure* first and then conclude.

Pointers VS. **Arrays**

Experiment over 5 million nodes

Visit the tree and compute the sum of all nodes.

```
→  segment_trees git:(master) ✗ g++ -std=c++11 -O3 trees_with_pointers.cpp -o trees_with_pointers
→  segment_trees git:(master) ✗ ./trees_with_pointers < input7
building took: 19.3895 [sec]
sum is: 3676408720
sum took: 0.118645 [sec]
→  segment_trees git:(master) ✗ g++ -std=c++11 -O3 trees_with_arrays.cpp -o trees_with_arrays
→  segment_trees git:(master) ✗ ./trees_with_arrays < input7
building took: 18.0724 [sec]
sum is: 3676408920
sum took: 0.0118351 [sec]
→  segment_trees git:(master) ✗
```

10X

How do we represent trees?

Remember
Be skeptic: *measure* first and then conclude.

Pointers VS. **Arrays**

Experiment over 5 million nodes

Visit the tree and compute the sum of all nodes.

```
→  segment_trees git:(master) ✗ g++ -std=c++11 -O3 trees_with_pointers.cpp -o trees_with_pointers
→  segment_trees git:(master) ✗ ./trees_with_pointers < input7
building took: 19.3895 [sec]
sum is: 3676408720
sum took: 0.118645 [sec]
→  segment_trees git:(master) ✗ g++ -std=c++11 -O3 trees_with_arrays.cpp -o trees_with_arrays
→  segment_trees git:(master) ✗ ./trees_with_arrays < input7
building took: 18.0724 [sec]
sum is: 3676408920
sum took: 0.0118351 [sec]
→  segment_trees git:(master) ✗
```

10X

OK, we are going to adopt the array-based representation!

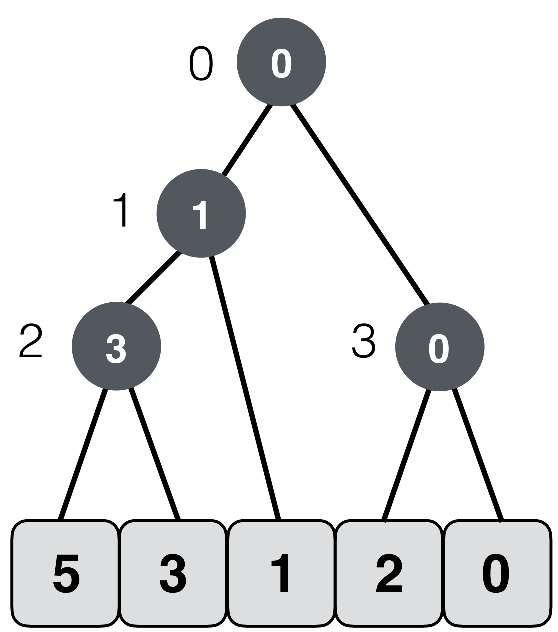
Building Segment Trees recursively

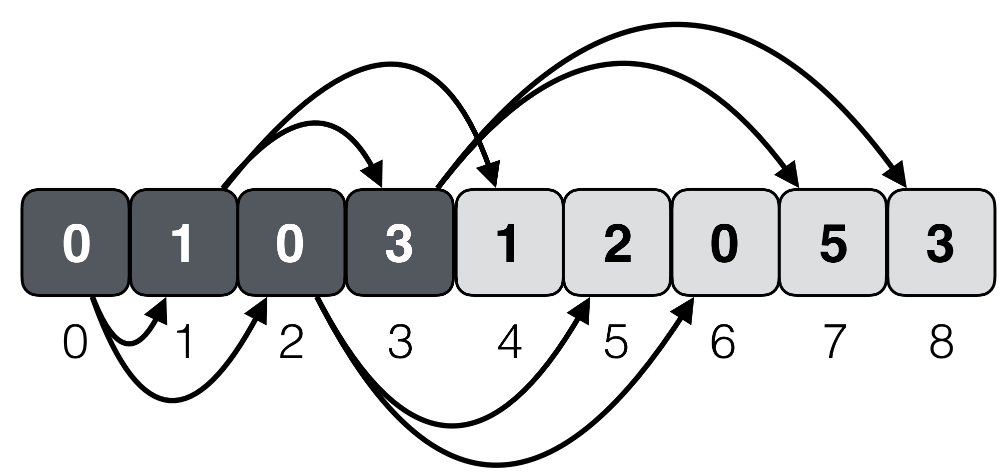
```
size_t n = leaves.size();
// round up to the next power of 2
size_t m = size_t(1) << static_cast<size_t>(ceil(log2(n)));
m_tree.resize(2 * m - 1, m_traits.invalid);

build(leaves, 0, m - 1, 0);
```

```
struct type_traits {
    IntType invalid;
    BinaryFunct funct;
};
```

```
void build(std::vector<IntType> const& leaves, size_t lo, size_t hi, size_t pos) {
    if (lo == hi) {
        m_tree[pos] = leaves[lo];
        return;
    }
    size_t mid = (lo + hi) / 2;
    build(leaves, lo, mid, LEFT(pos));
    build(leaves, mid + 1, hi, RIGHT(pos));
    m_tree[pos] = m_traits.funct(m_tree[LEFT(pos)], m_tree[RIGHT(pos)]);
}
```

```
#define LEFT(i)      2 * i + 1
#define RIGHT(i)     2 * i + 2
#define PARENT(i) (i - 1) / 2
```

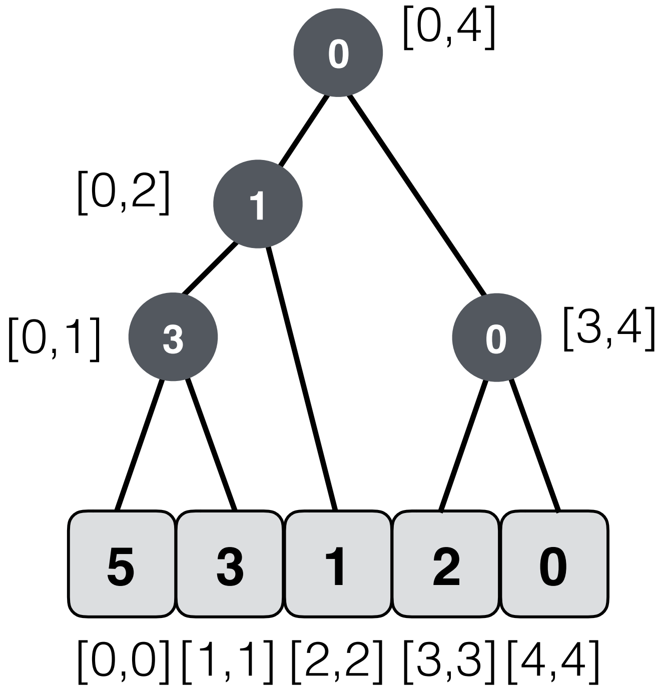




Range (MIN) Queries with Segment Trees

```
IntType rmq(range const& query, range node_segment, size_t pos) {

    if (query.lo <= node_segment.lo
    and query.hi >= node_segment.hi) { // total overlap
        return m_tree[pos];
    }
    if (query.lo > node_segment.hi
    or  query.hi < node_segment.lo) { // no overlap
        return m_traits.invalid;
    }

    // partial overlap
    size_t mid = (node_segment.lo + node_segment.hi) / 2;
    return m_traits.funct(
        rmq(query, {node_segment.lo, mid}, LEFT(pos)),
        rmq(query, {mid + 1, node_segment.hi}, RIGHT(pos))
    );
}
```

```
struct range {
    range(size_t l, size_t h)
        : lo(l), hi(h)
    {}

    size_t lo, hi;
};
```

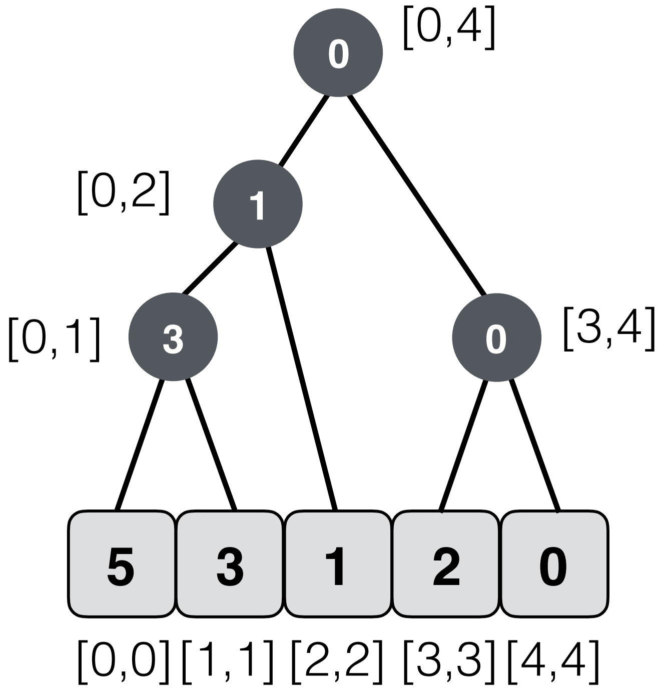


Range (MIN) Queries with Segment Trees

```
IntType rmq(range const& query, range node_segment, size_t pos) {

    if (query.lo <= node_segment.lo
    and query.hi >= node_segment.hi) { // total overlap
        return m_tree[pos];
    }
    if (query.lo > node_segment.hi
    or  query.hi < node_segment.lo) { // no overlap
        return m_traits.invalid;
    }

    // partial overlap
    size_t mid = (node_segment.lo + node_segment.hi) / 2;
    return m_traits.funct(
        rmq(query, {node_segment.lo, mid}, LEFT(pos)),
        rmq(query, {mid + 1, node_segment.hi}, RIGHT(pos))
    );
}
```

```
struct range {
    range(size_t l, size_t h)
        : lo(l), hi(h)
    {}

    size_t lo, hi;
};
```

min(1,3) = ?

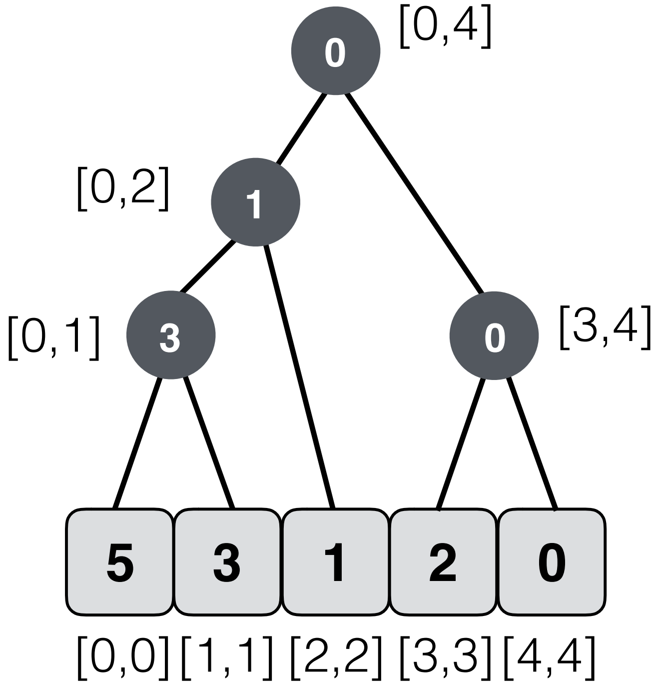


Updating Segment Trees

Let's add two new operations (updates):

- `update`(i, x) which increments the i-th leaf by x;
- `update_range`(i, j ,x) which increments all leaves from i to j by x.

```
void update(size_t i, IntType delta, range node_segment, size_t pos) {

    if (i > node_segment.hi
    or  i < node_segment.lo) {
        return;
    }

    if (node_segment.lo == node_segment.hi) { // leaf
        m_tree[pos] += delta;
        return;
    }

    size_t mid = (node_segment.lo + node_segment.hi) / 2;
    update(i, delta, {node_segment.lo, mid}, LEFT(pos));
    update(i, delta, {mid + 1, node_segment.hi}, RIGHT(pos));
    m_tree[pos] = m_traits.funct(m_tree[LEFT(pos)], m_tree[RIGHT(pos)]);
}
```



Updating Segment Trees

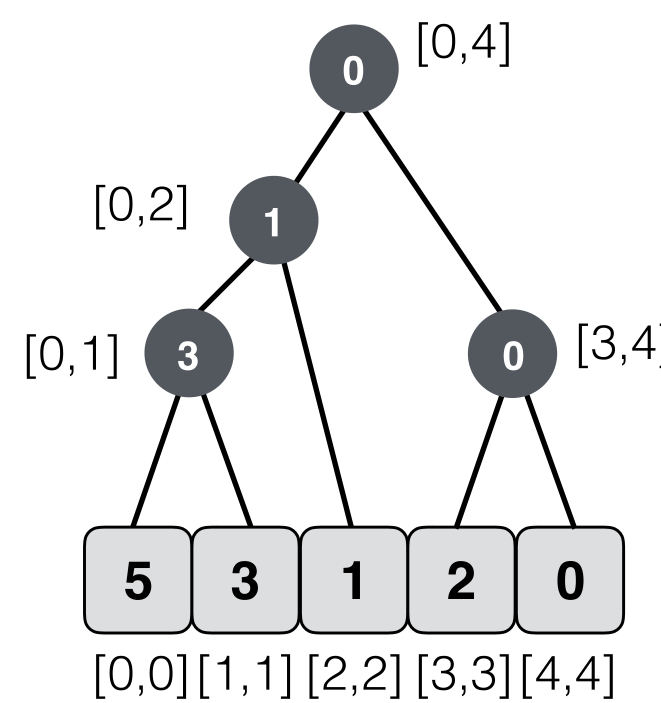
Let's add two new operations (updates):

- `update`(i, x) which increments the i-th leaf by x;
- `update_range`(i, j ,x) which increments all leaves from i to j by x.

```
         range const& query
void update(size_t i, IntType delta, range node_segment, size_t pos) {

    if (i > node_segment.hi
    or  i < node_segment.lo) {
        return;
    }
```

```
if (query.lo > node_segment.hi
or  query.hi < node_segment.lo) {
    return;
}
```

```
    if (node_segment.lo == node_segment.hi) { // leaf
        m_tree[pos] += delta;
        return;
    }

    size_t mid = (node_segment.lo + node_segment.hi) / 2;
    update(i, delta, {node_segment.lo, mid}, LEFT(pos));
    update(i, delta, {mid + 1, node_segment.hi}, RIGHT(pos));
    m_tree[pos] = m_traits.funct(m_tree[LEFT(pos)], m_tree[RIGHT(pos)]);
}
```



Benchmarking Segment Trees

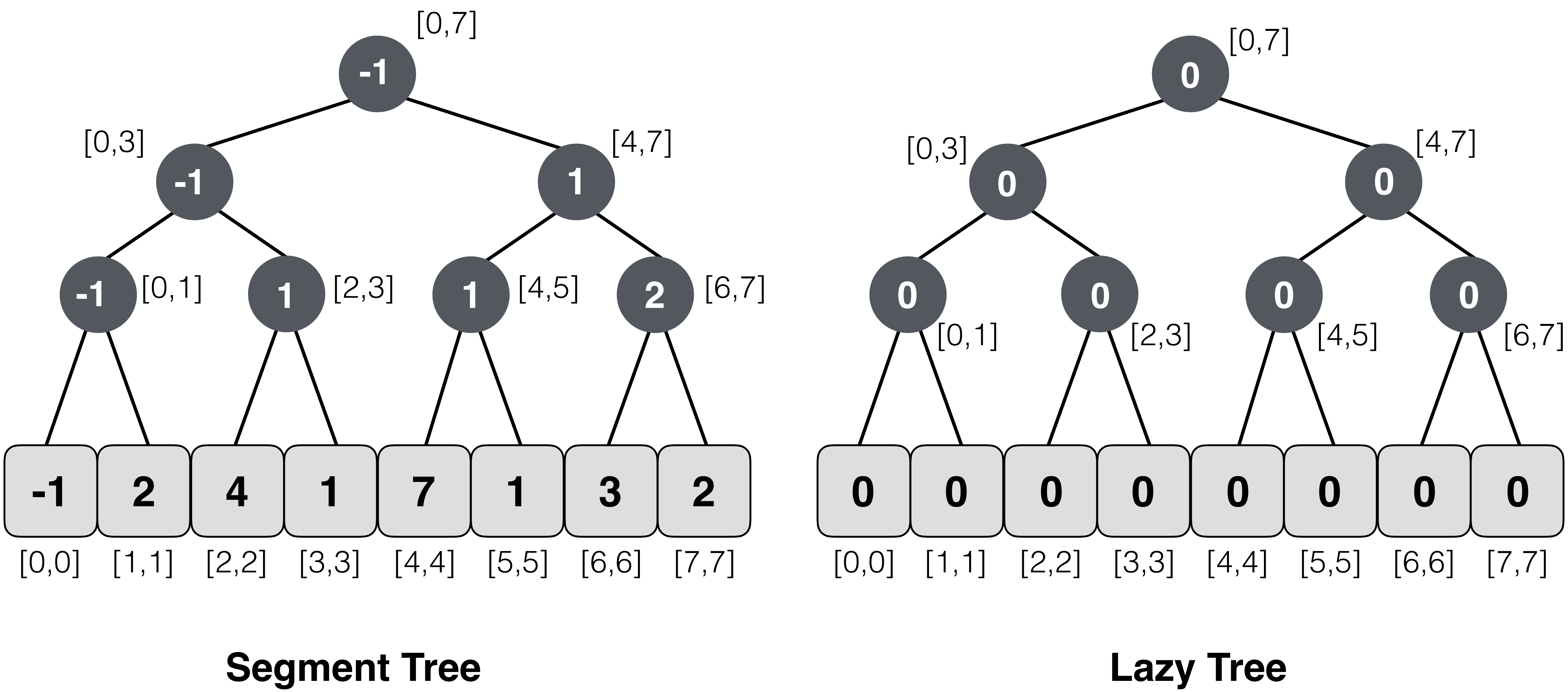
```
→  segment_trees git:(master) ✗ python gen_data.py 5000000 100000 10000 > input13
→  segment_trees git:(master) ✗ ./rmq_segment_tree < input13
parsing the input took: 18.5193 [sec]
building tree with 5000000 leaves
building took: 0.314939 [sec]
executing 100000 range queries
average query time: 1.74382 [musec]
executing 10000 updates
average update time: 0.561733 [musec]
executing 10000 range updates
average range update time: 2.55461 [musec]
→  segment_trees git:(master) ✗
```

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

Lazy Propagation in Segment Trees

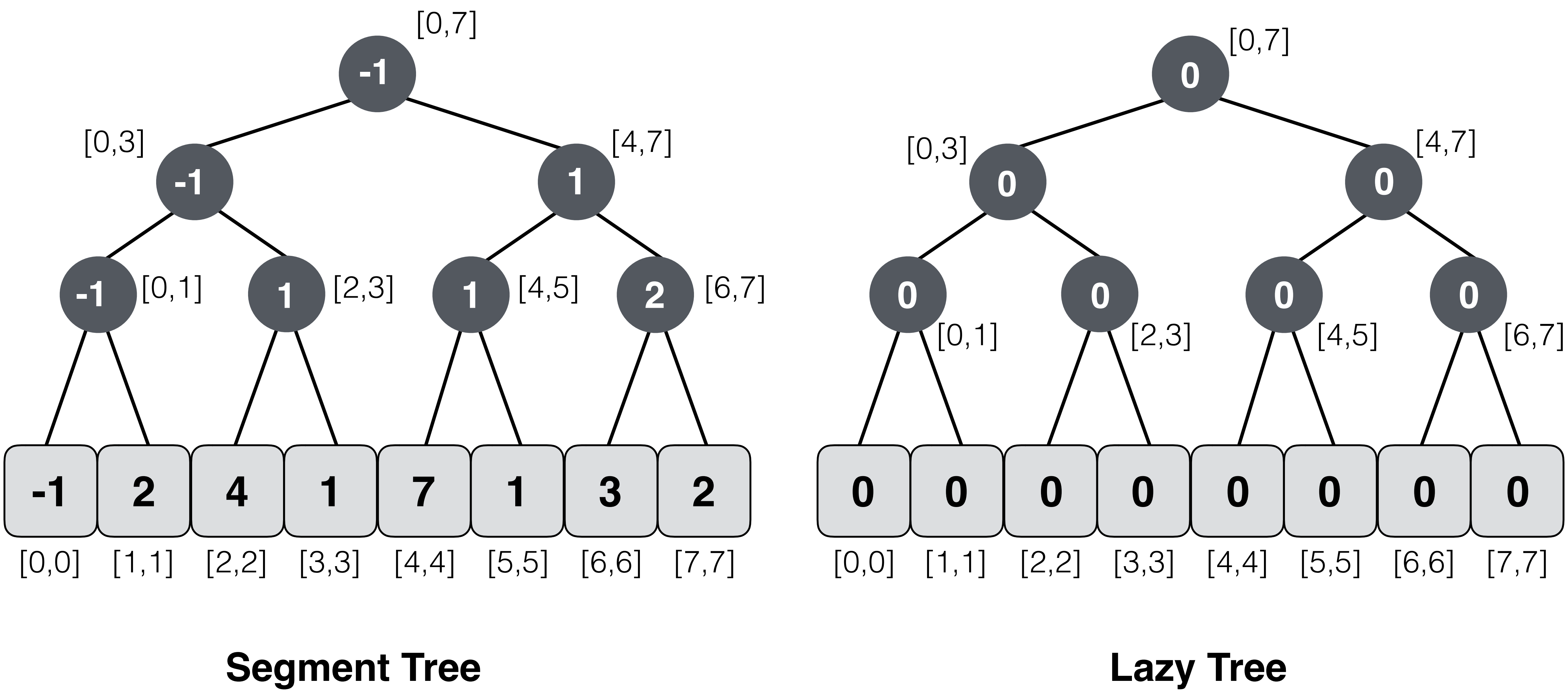
Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.



Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

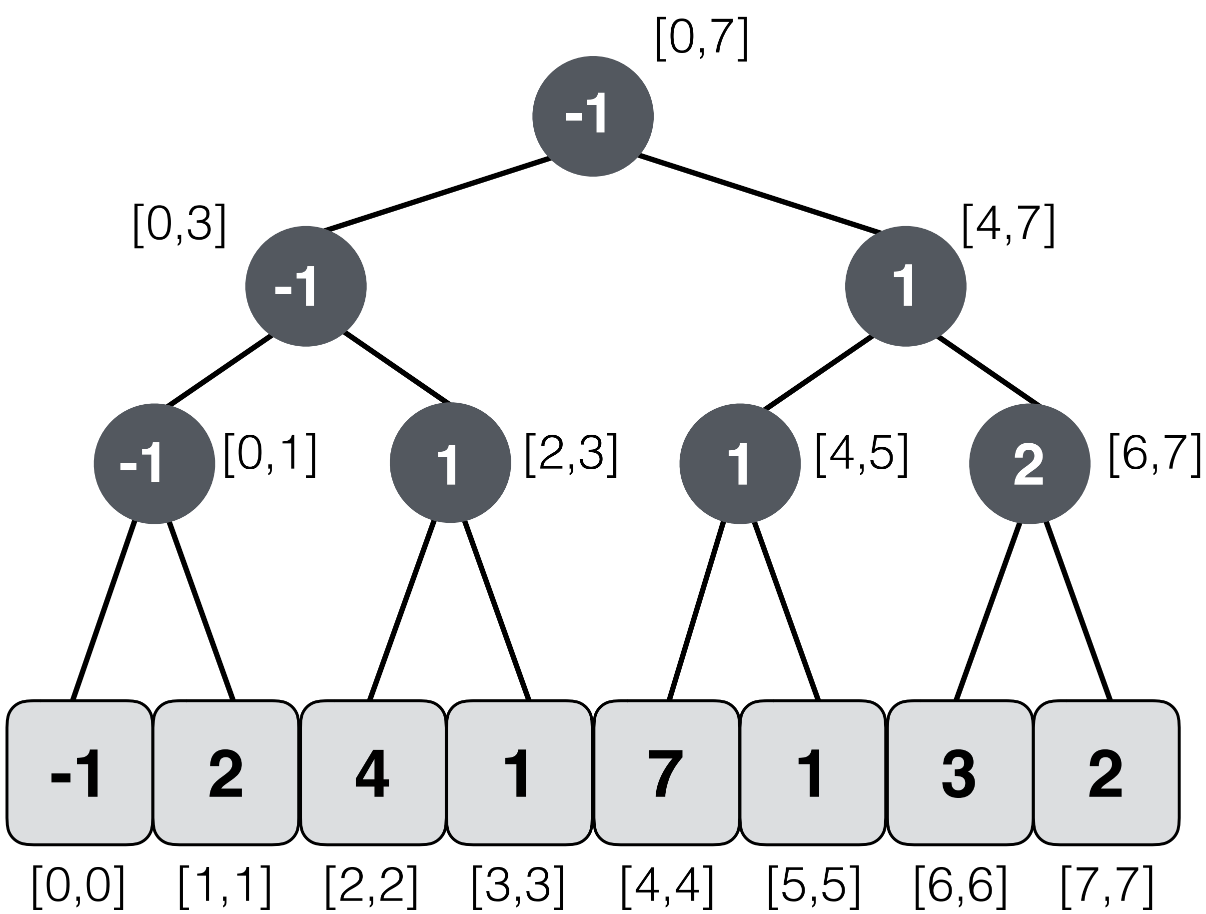
`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



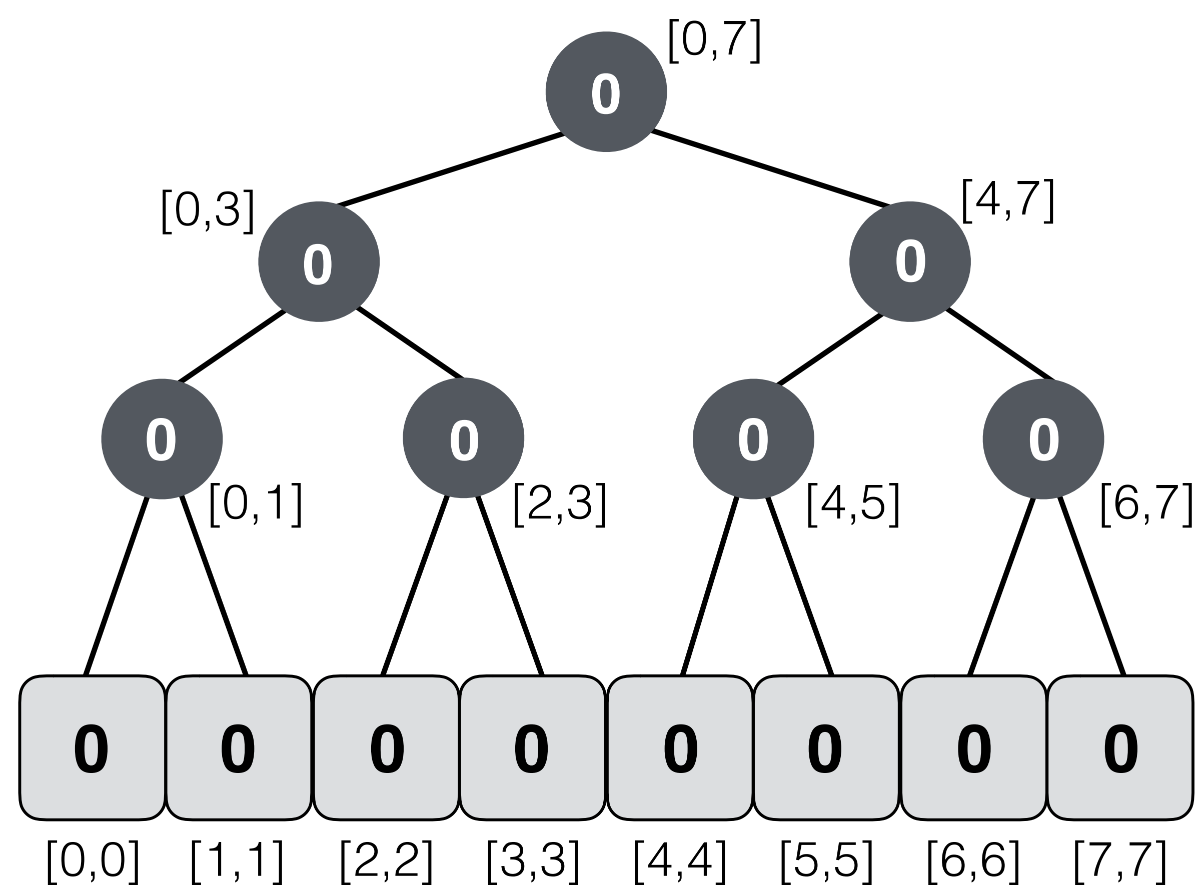
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

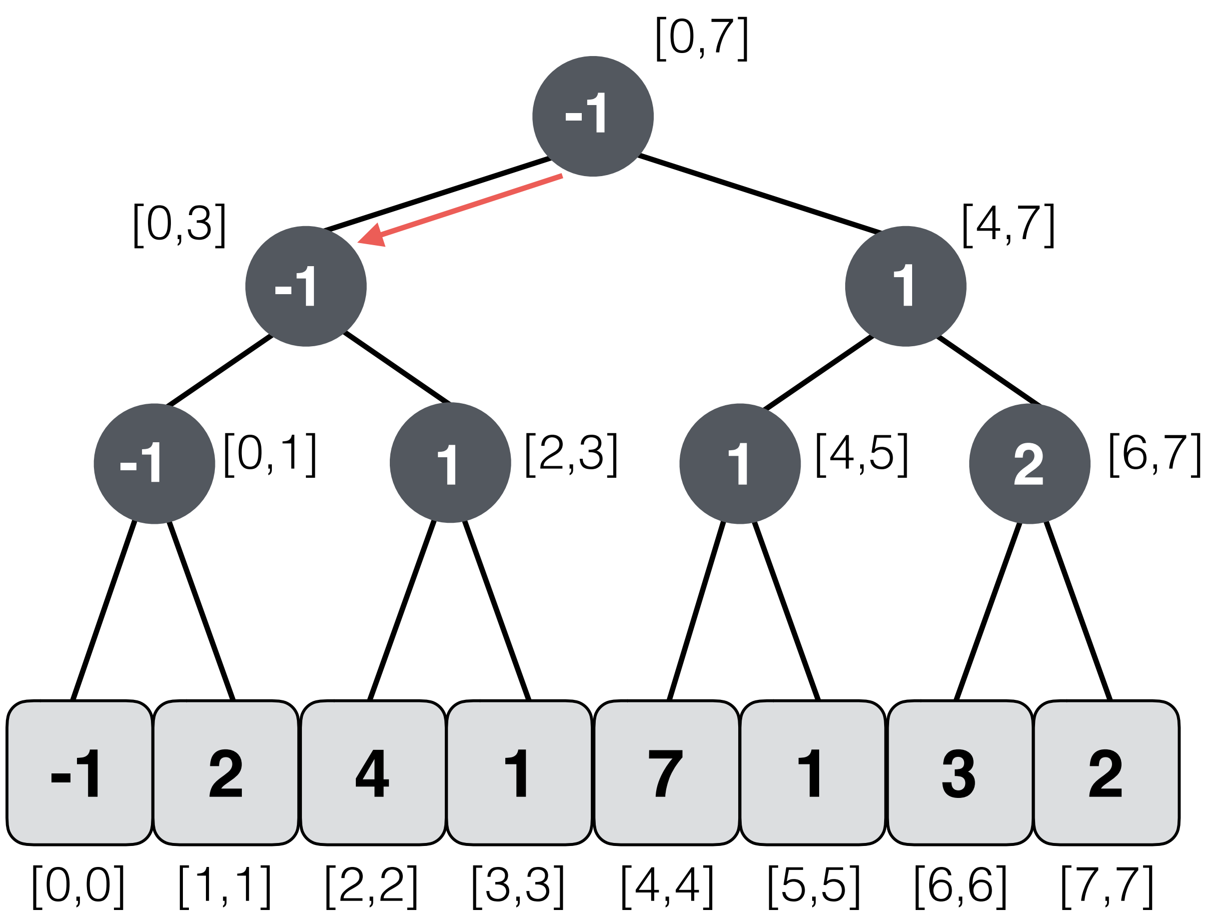


Lazy Tree

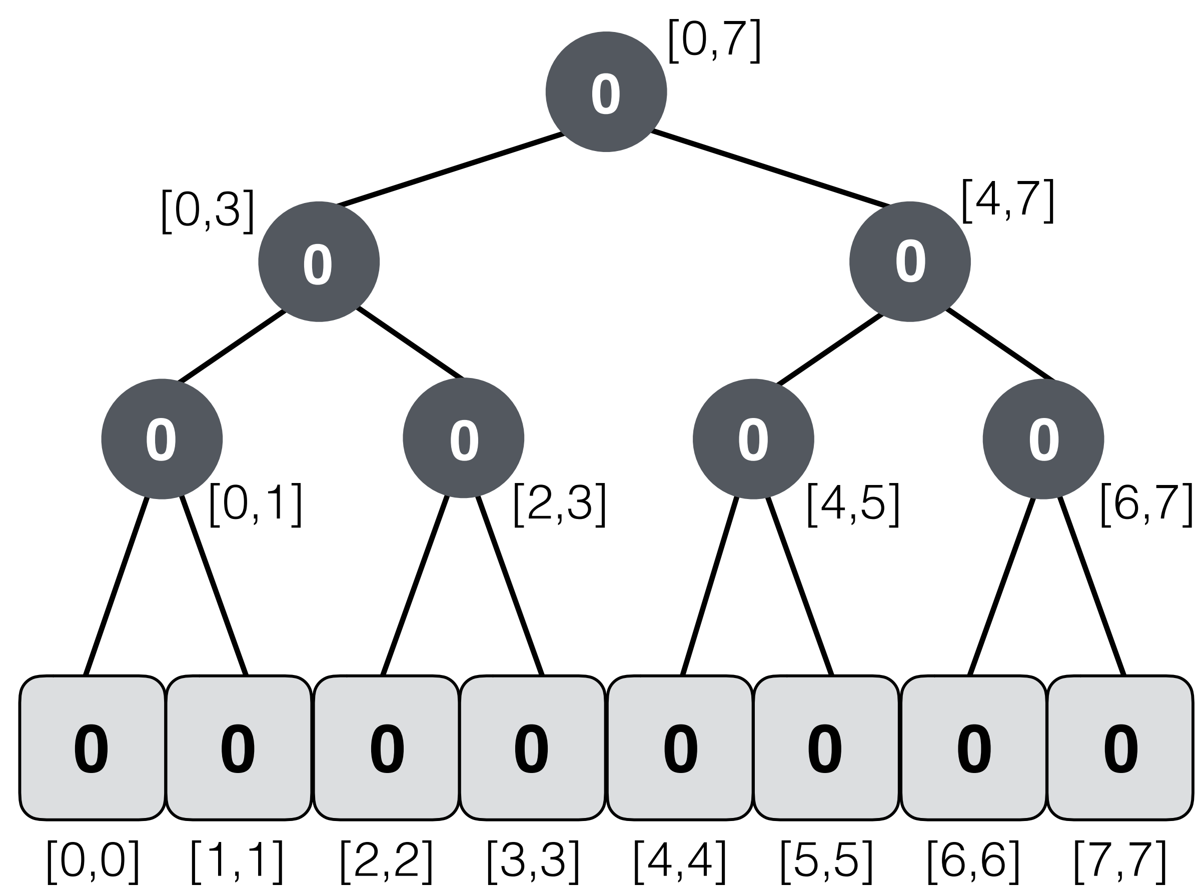
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

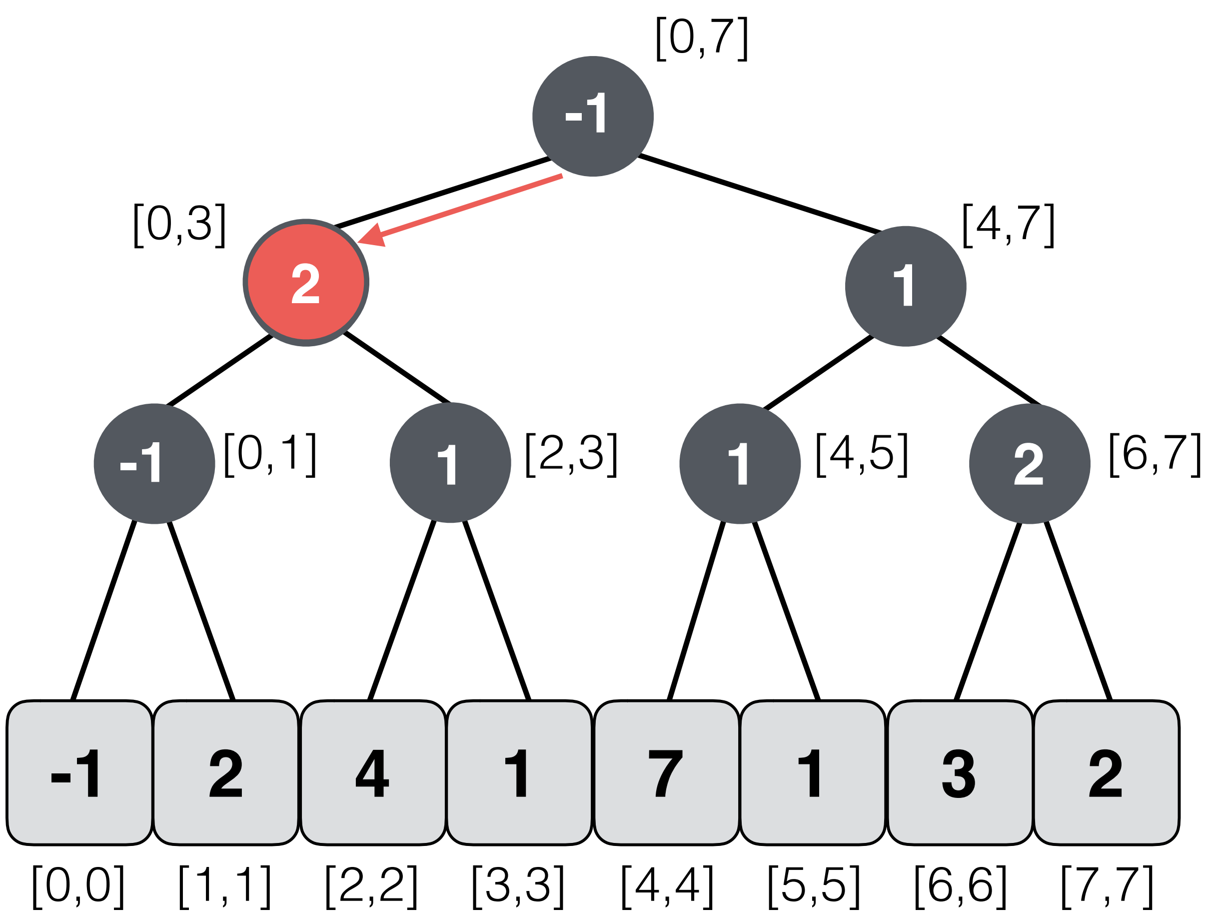


Lazy Tree

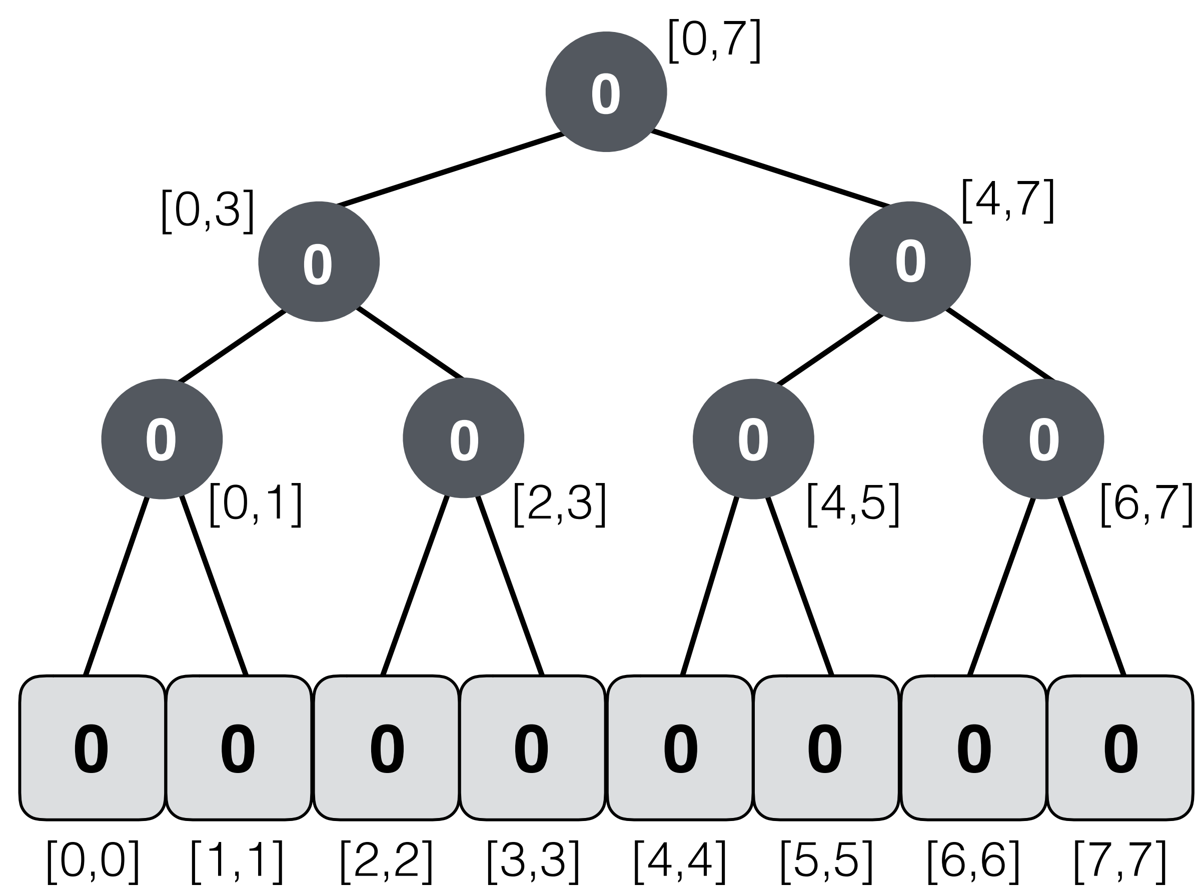
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

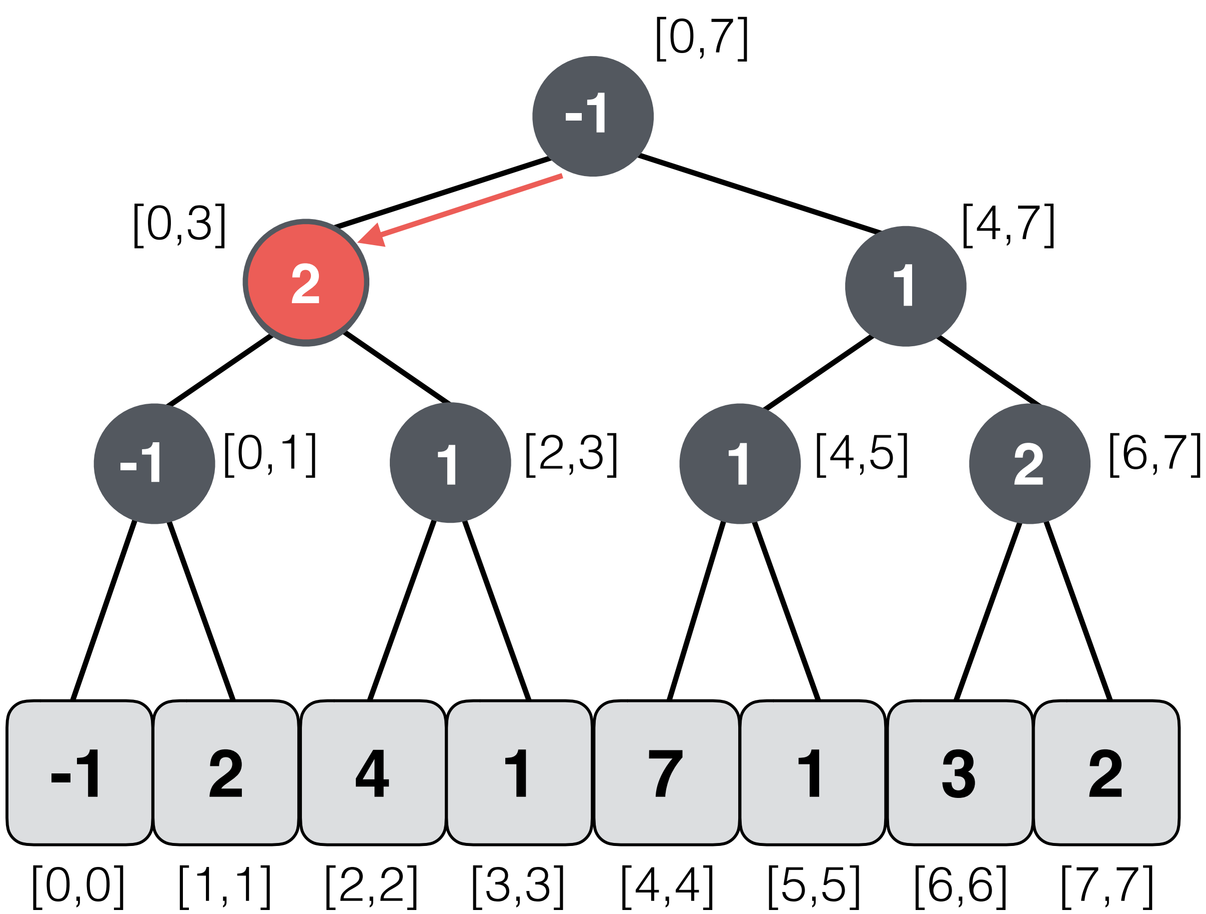


Lazy Tree

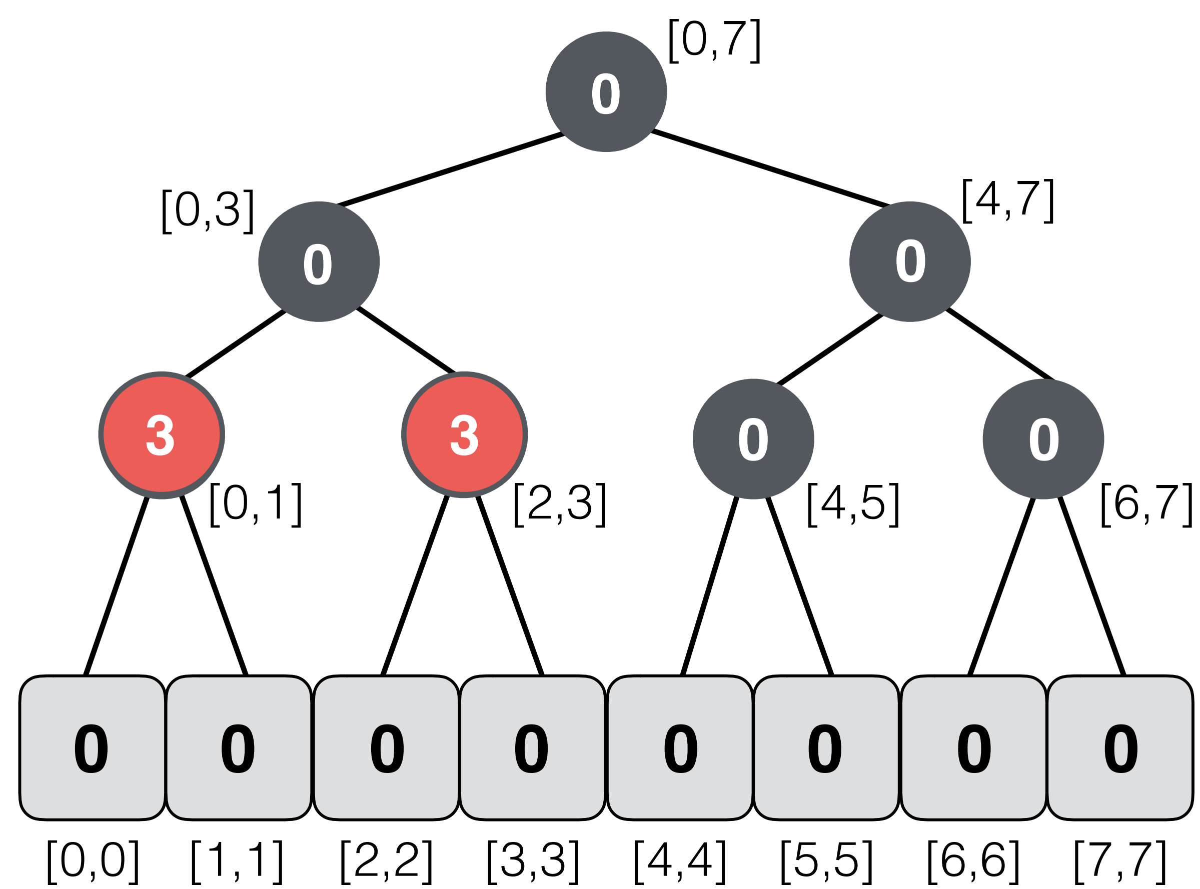
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

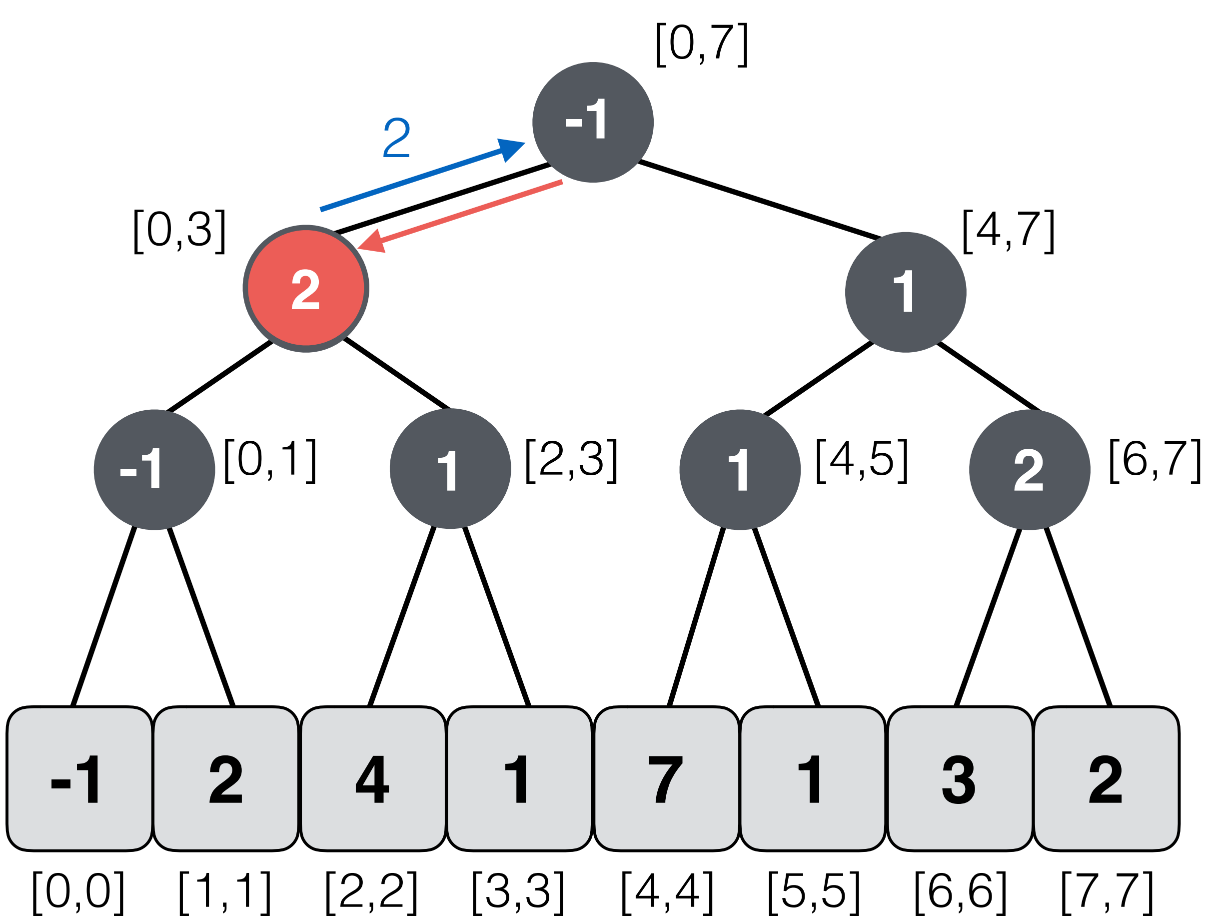


Lazy Tree

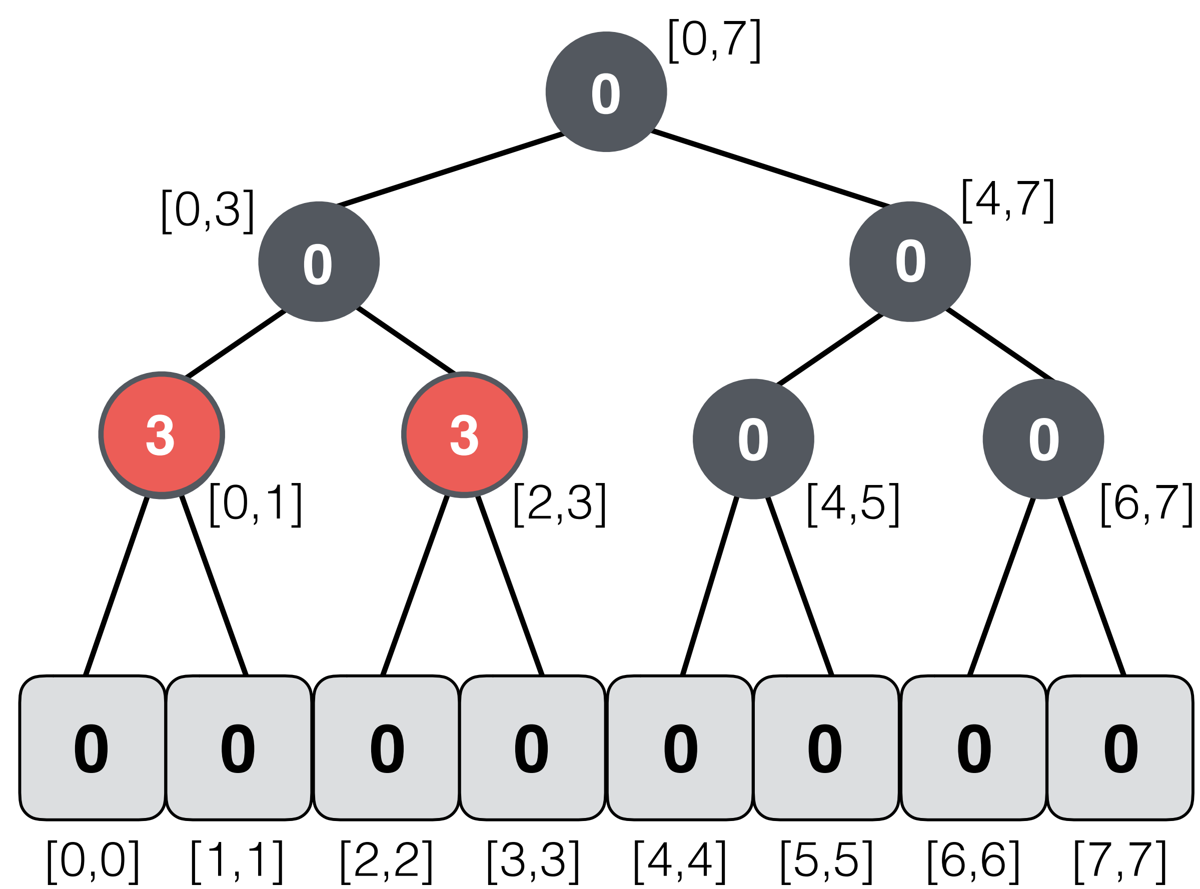
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

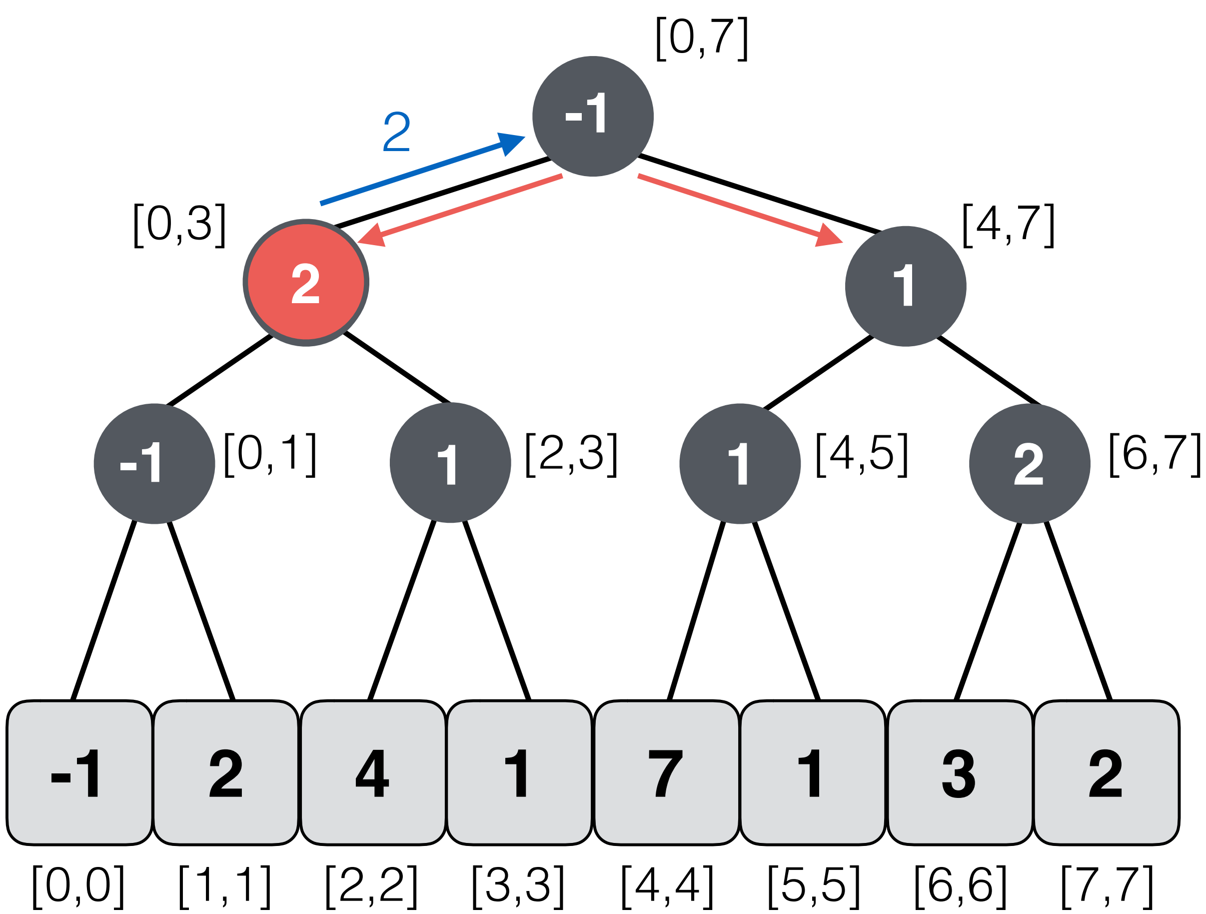


Lazy Tree

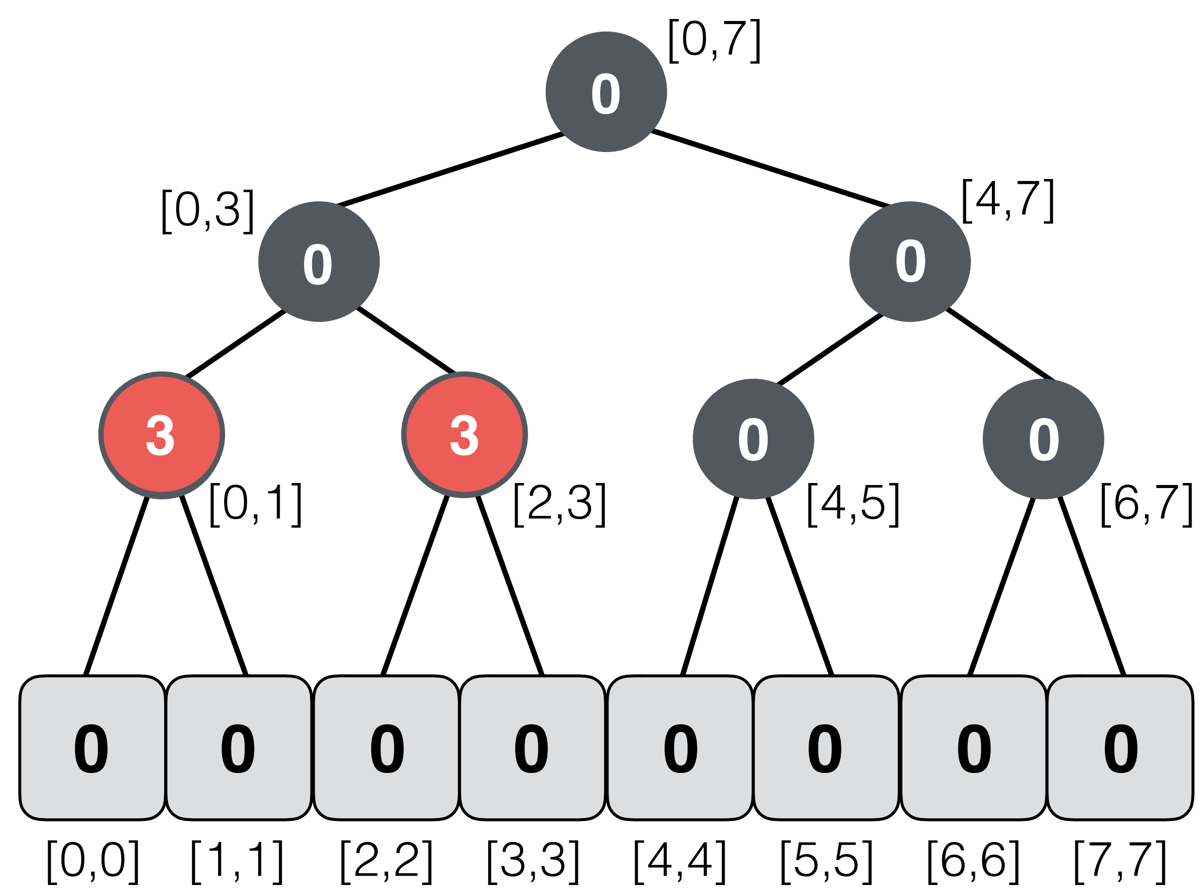
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

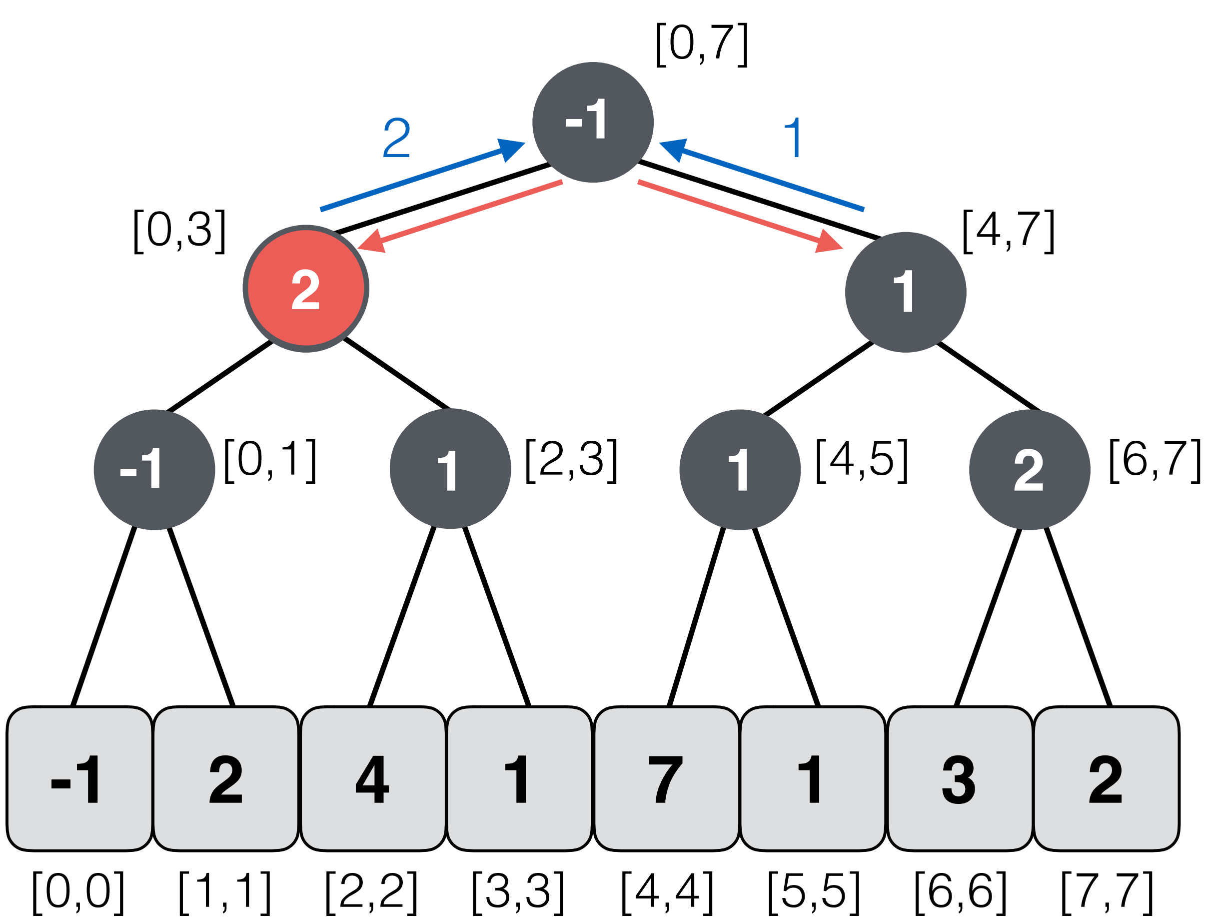


Lazy Tree

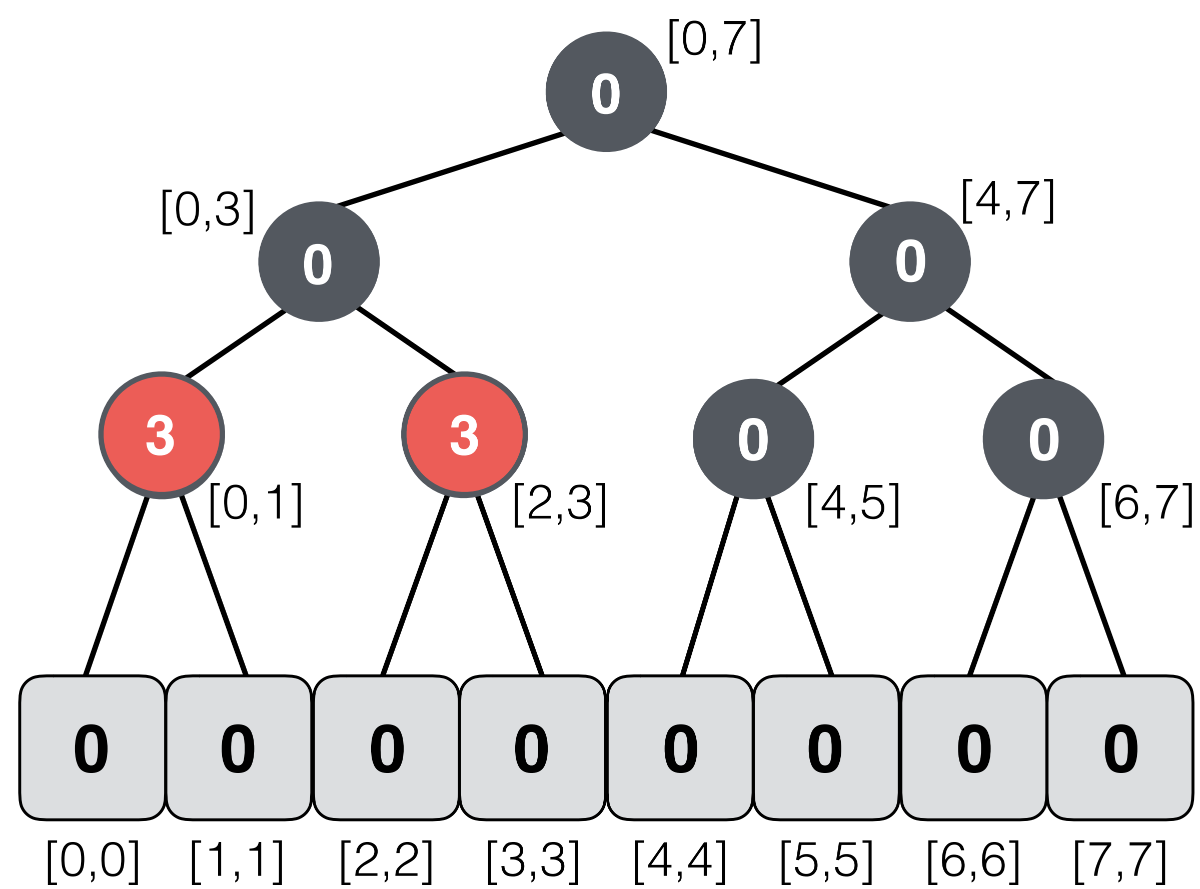
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



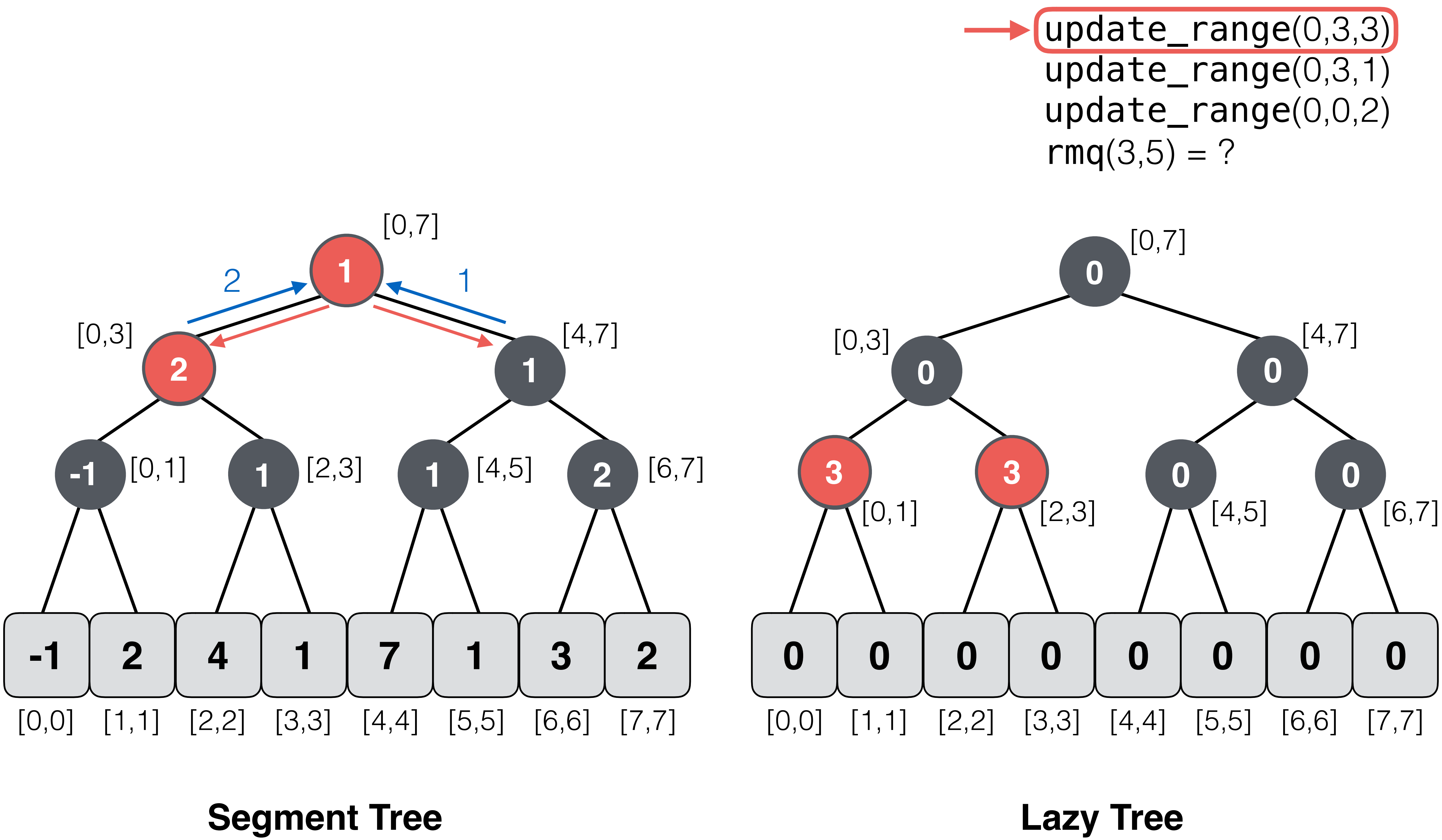
Segment Tree



Lazy Tree

Lazy Propagation in Segment Trees

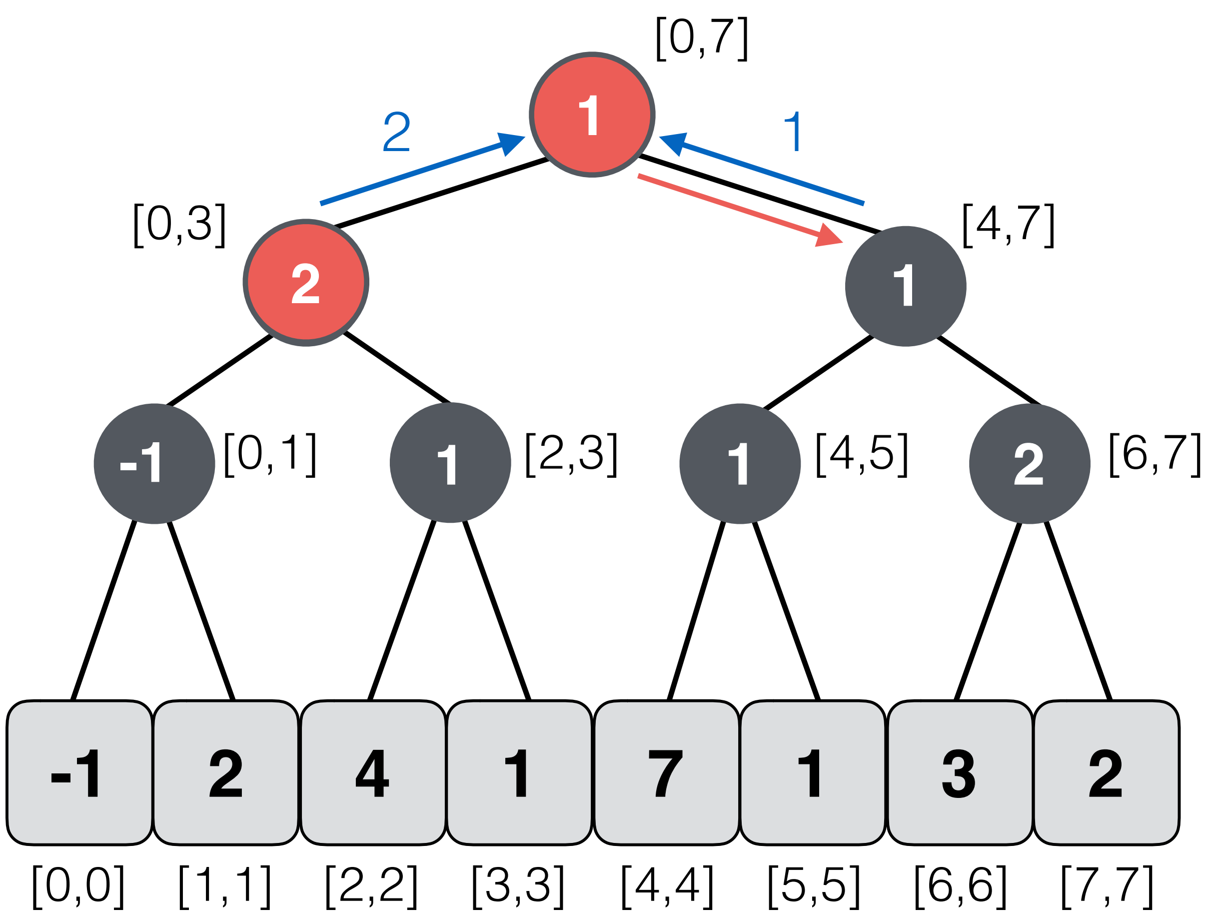
Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.



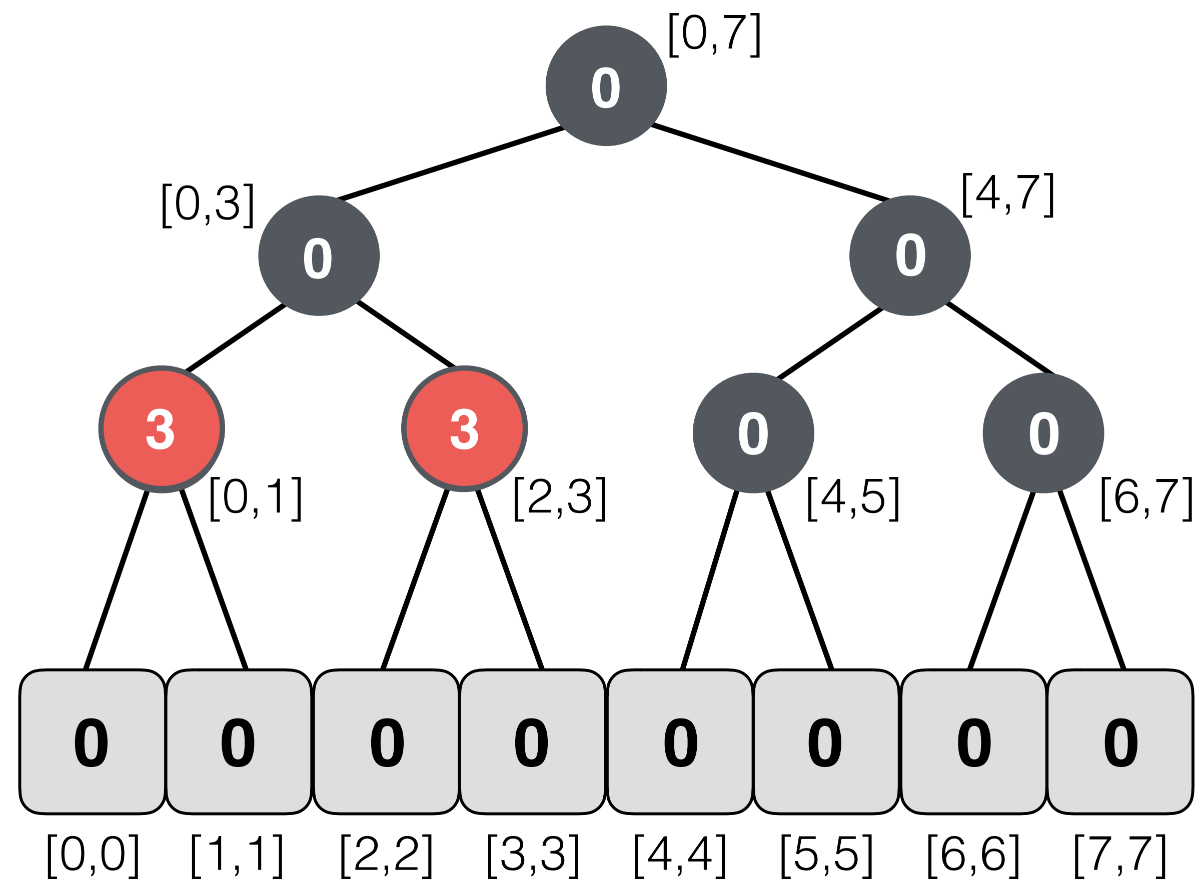
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?



Segment Tree

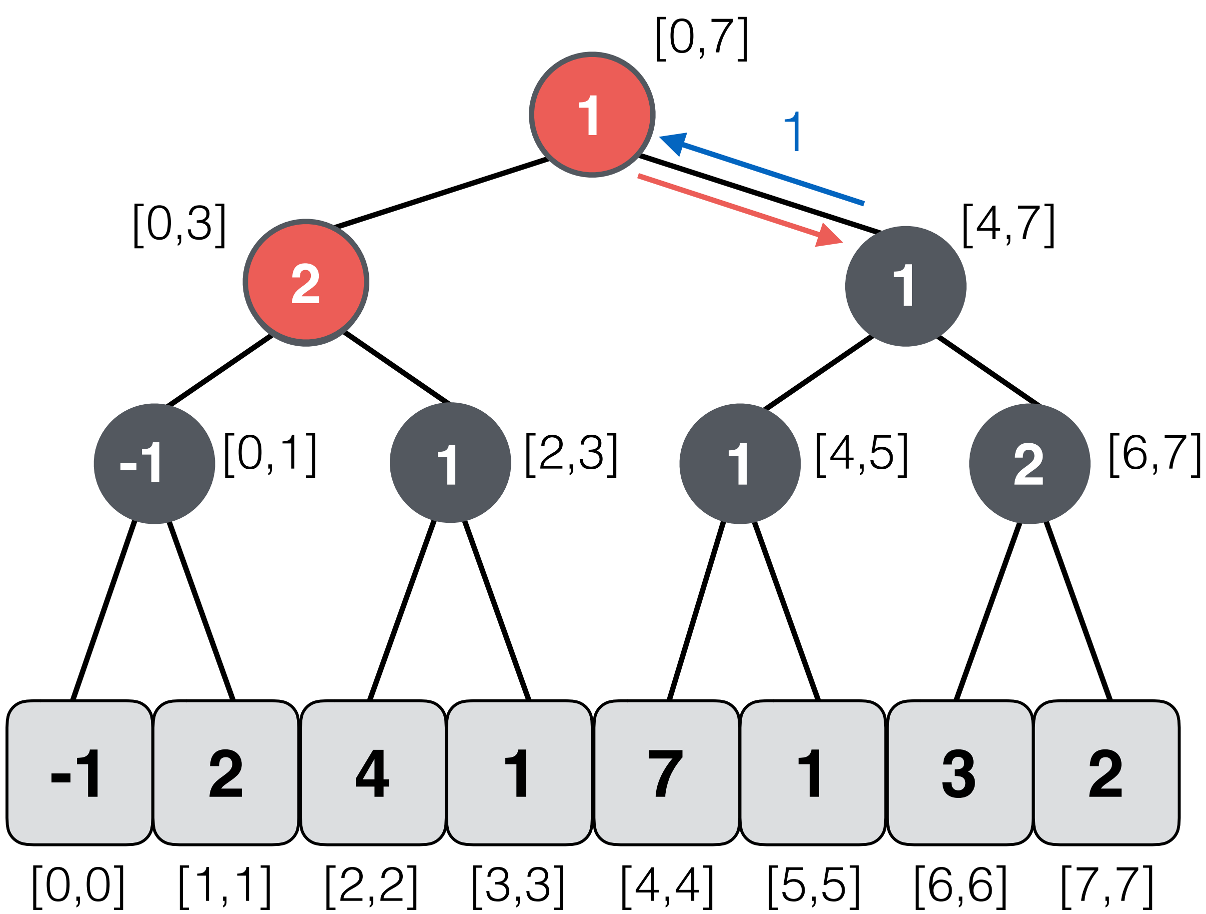


Lazy Tree

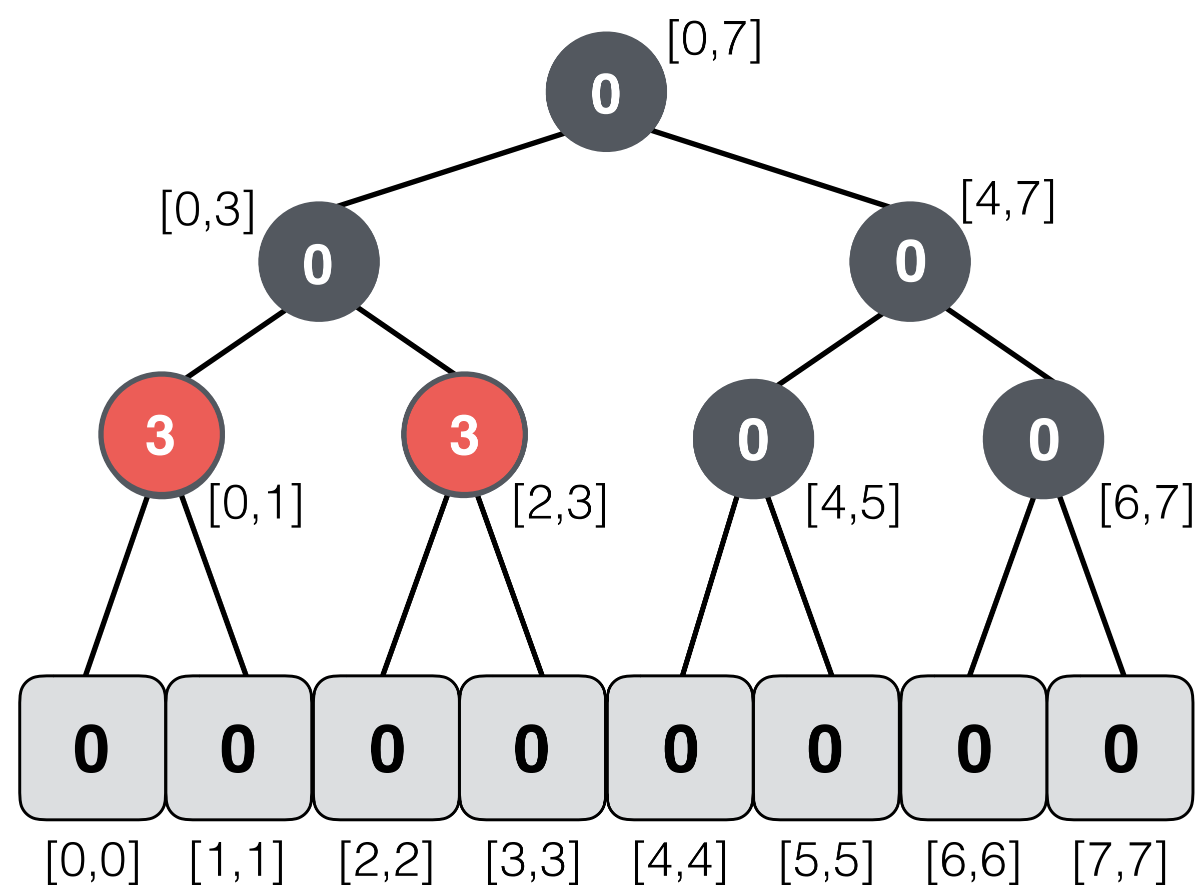
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

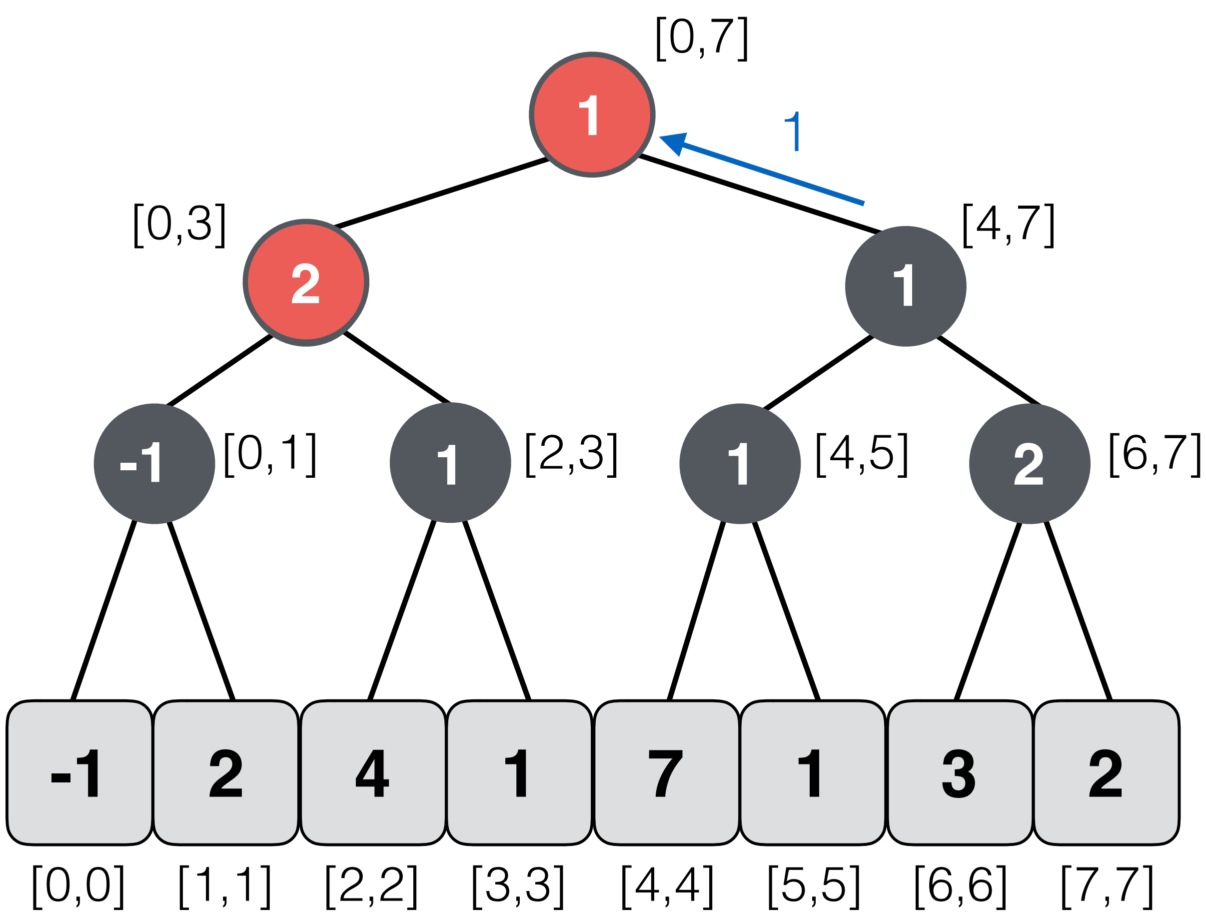


Lazy Tree

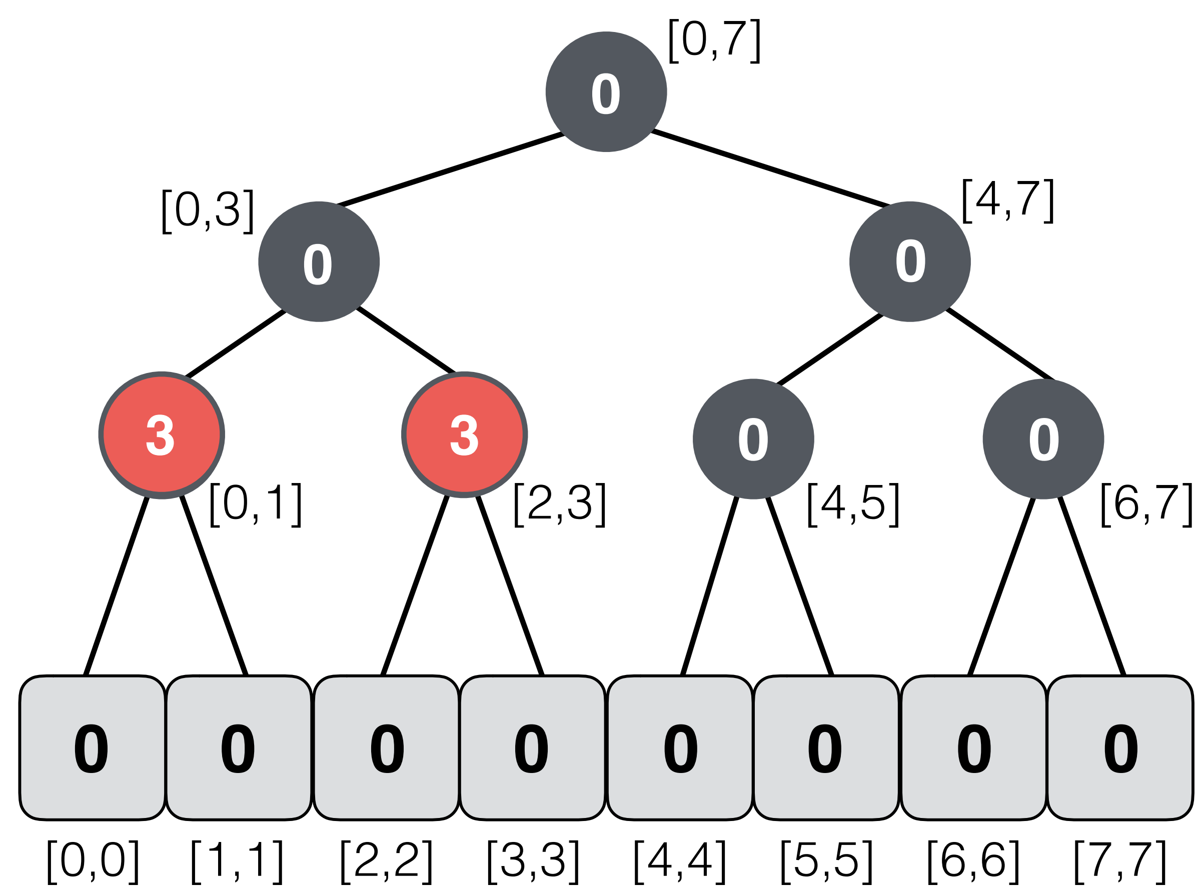
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

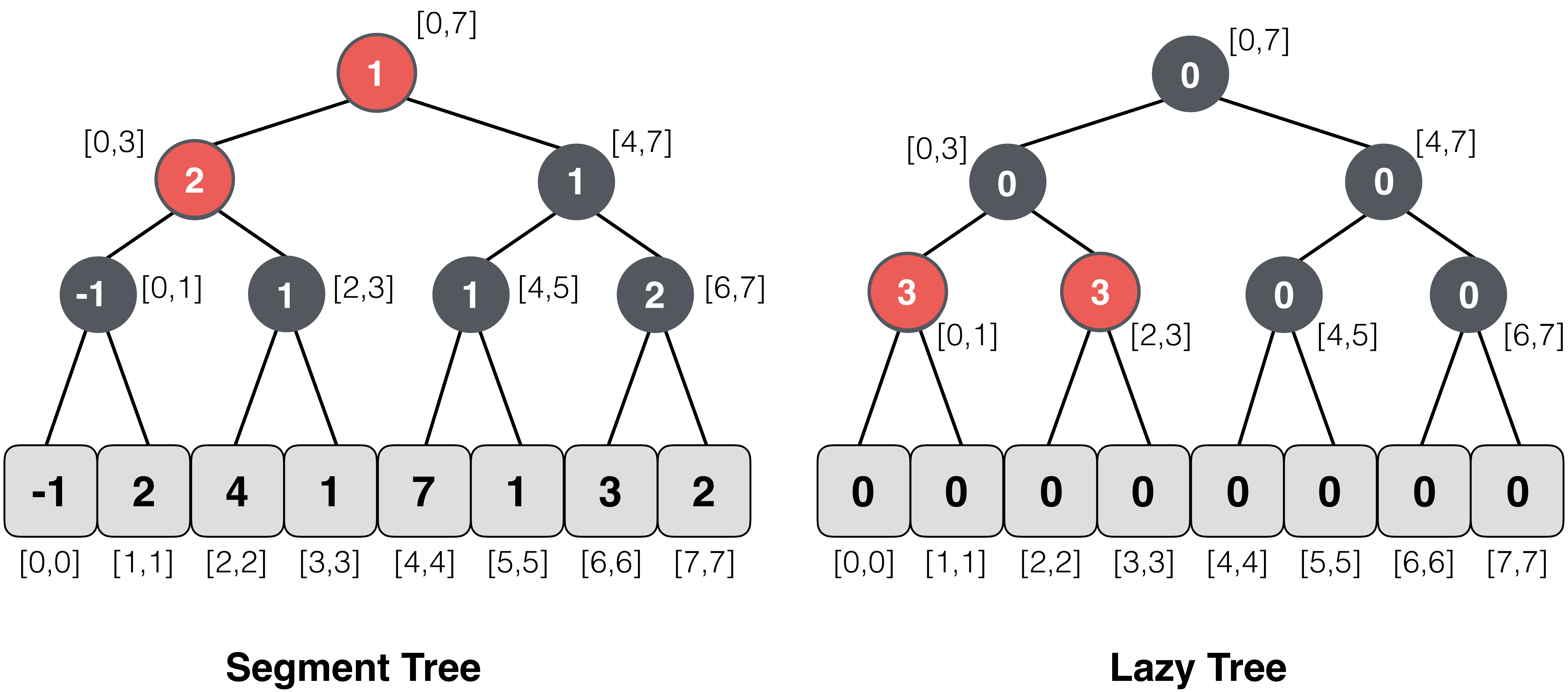


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

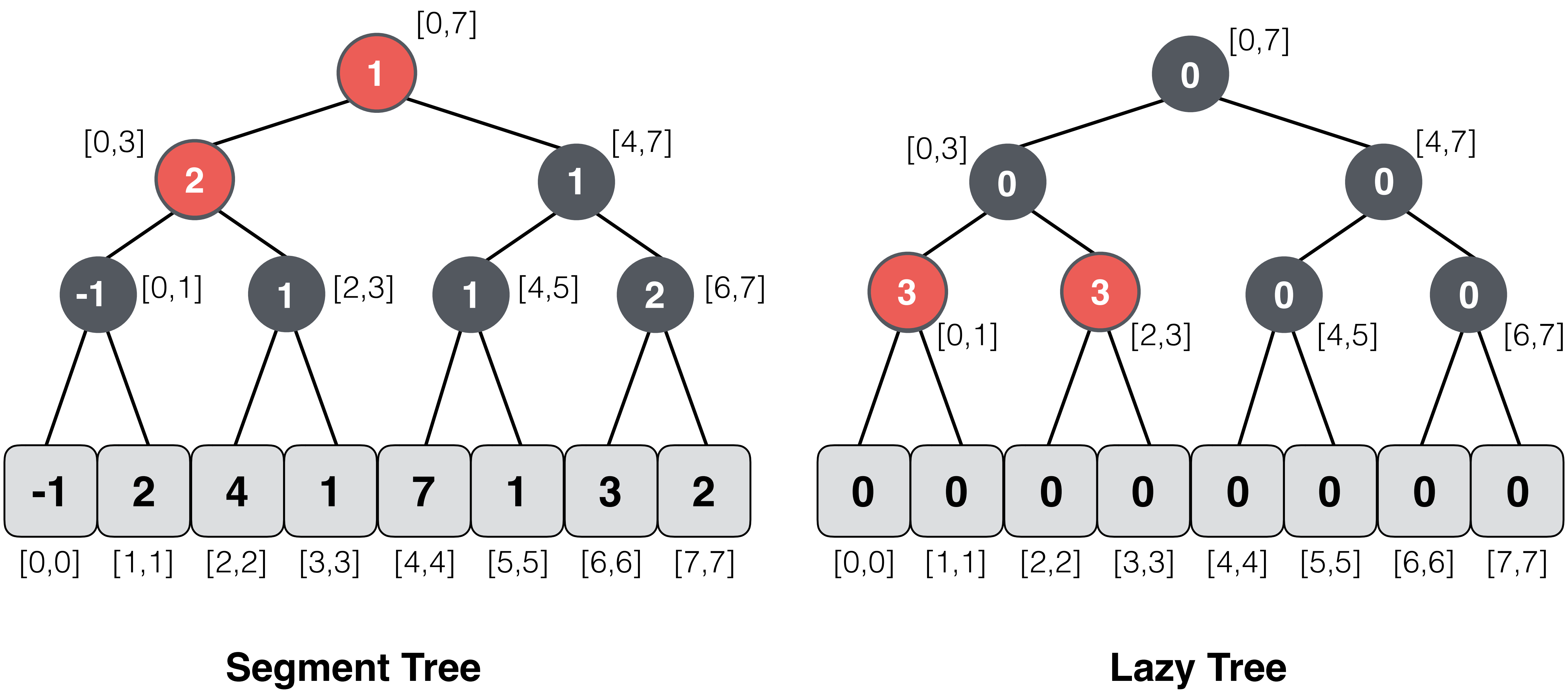
`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

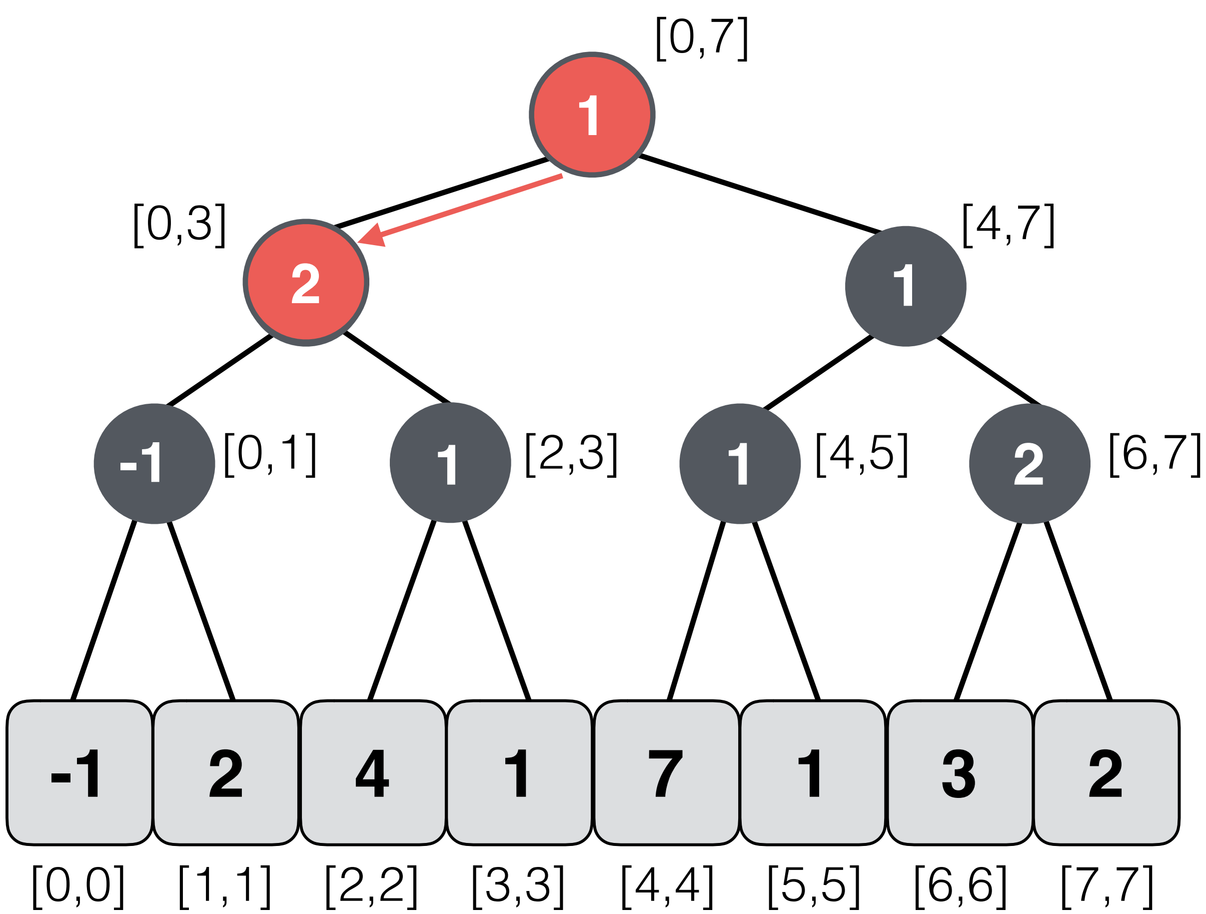
```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



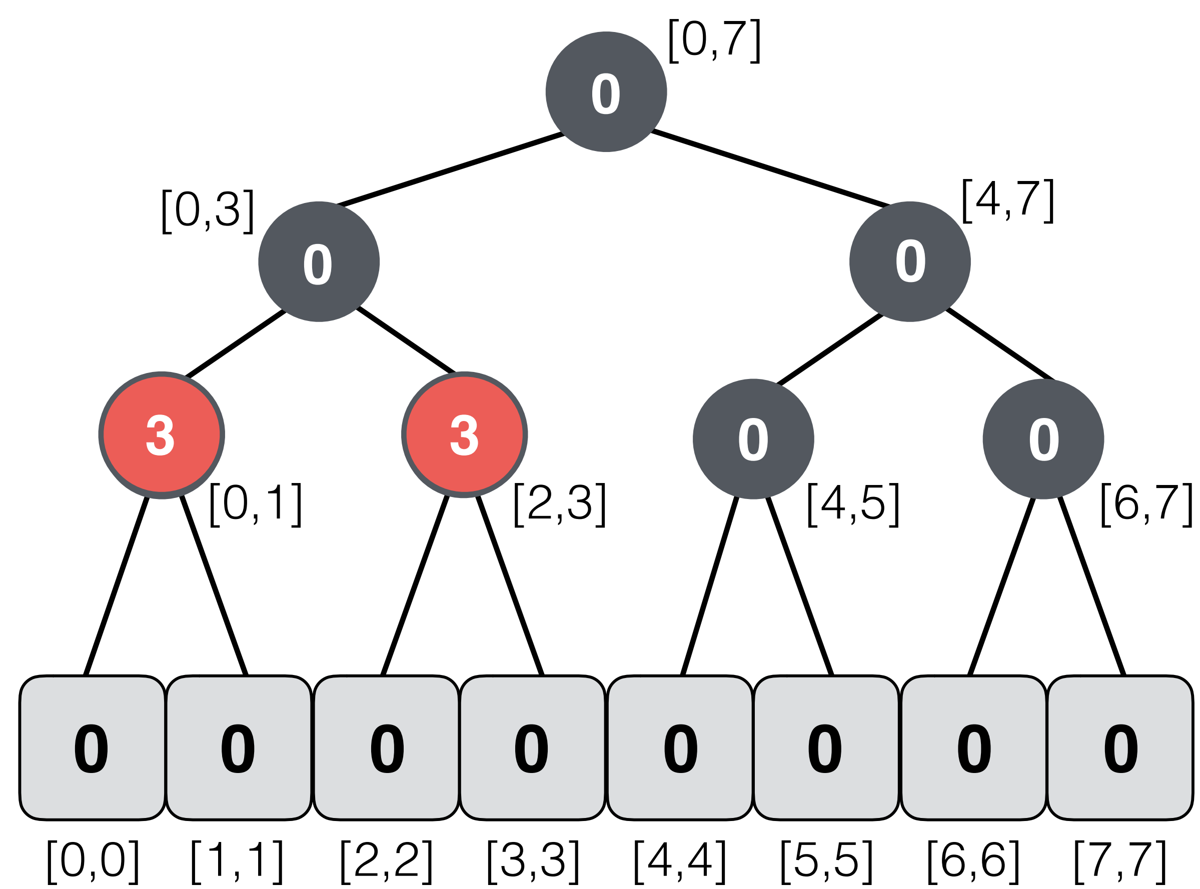
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

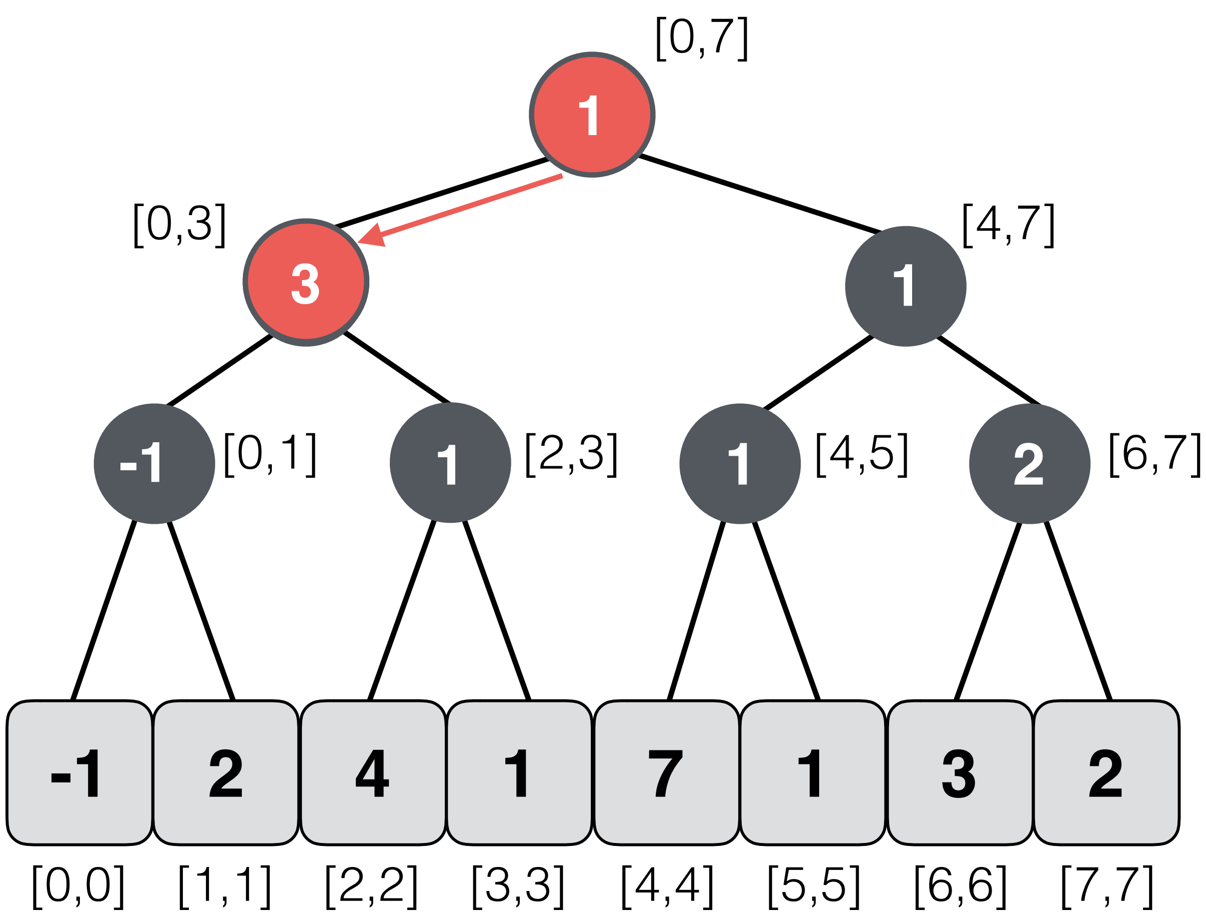


Lazy Tree

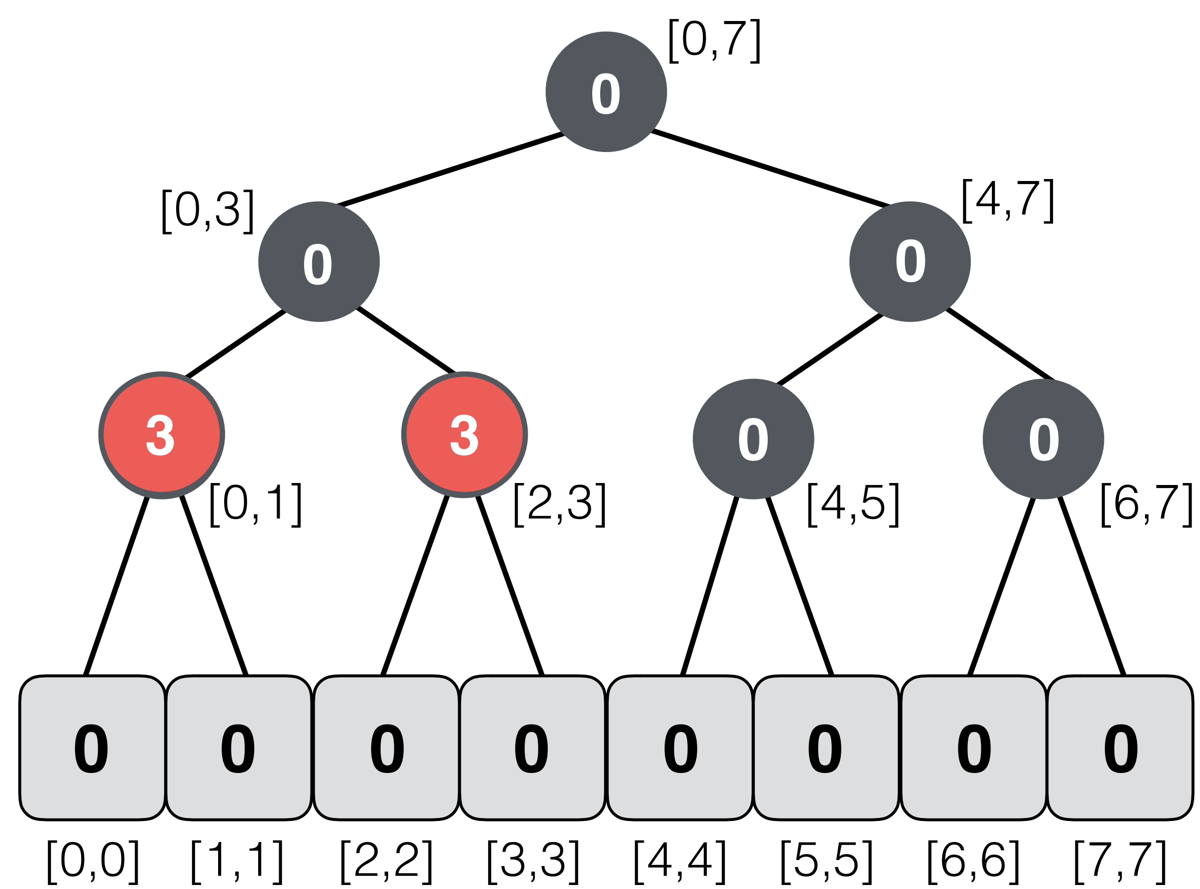
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

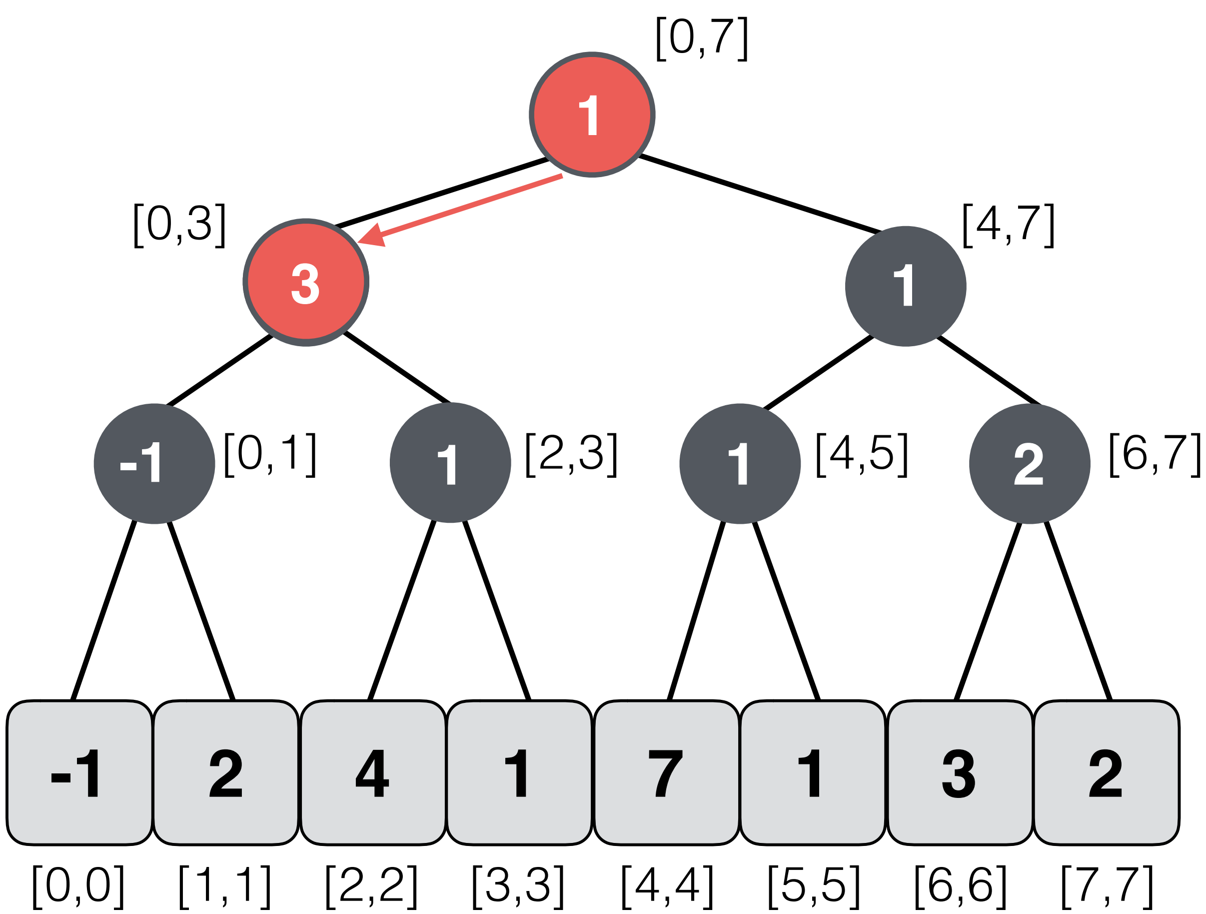


Lazy Tree

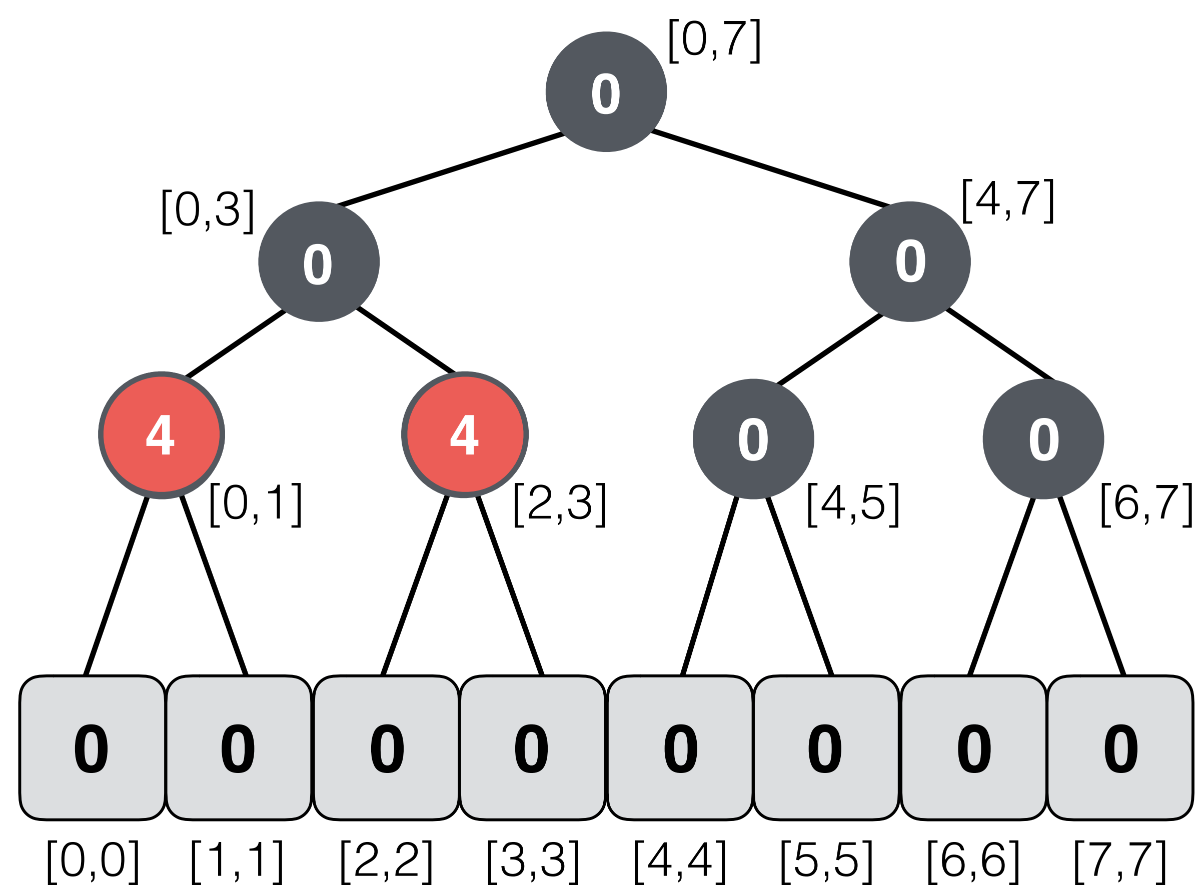
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

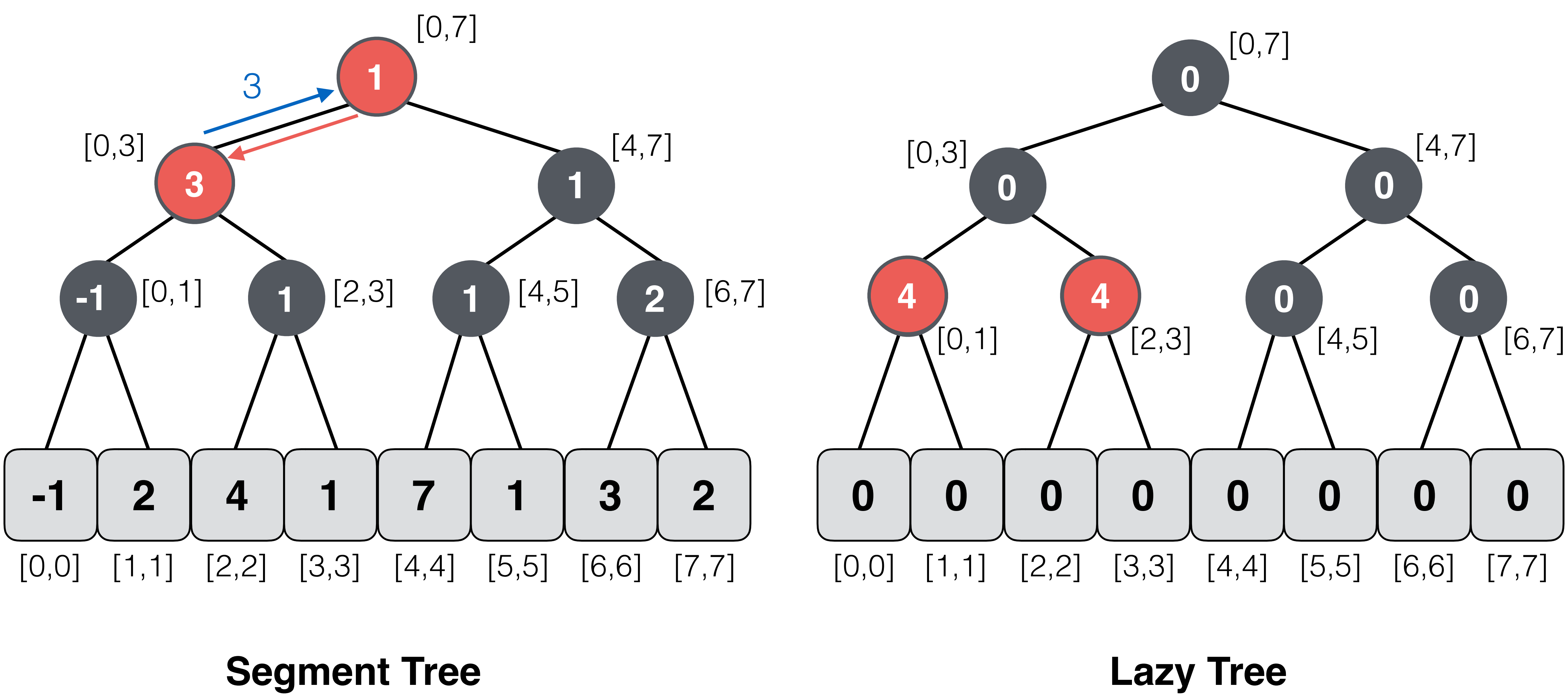


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

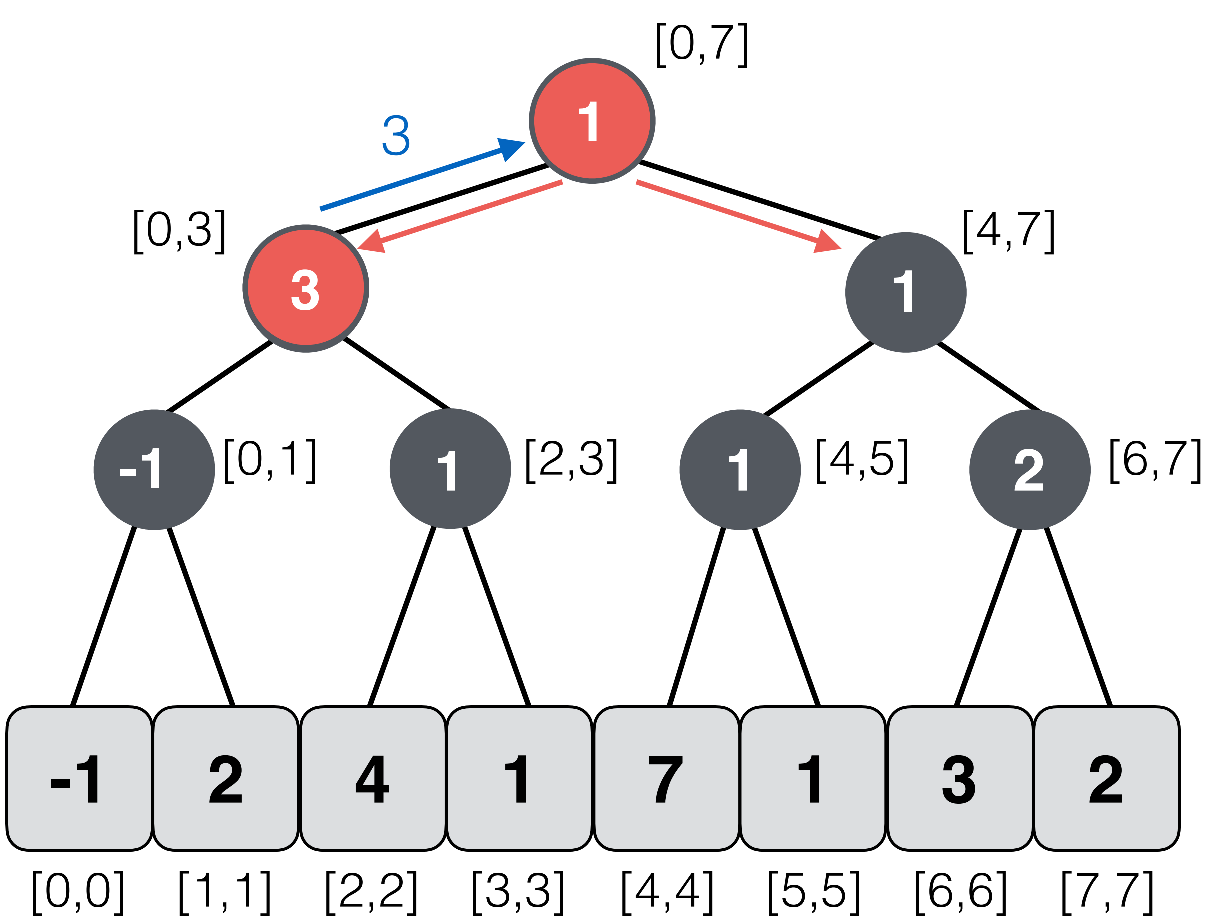
`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



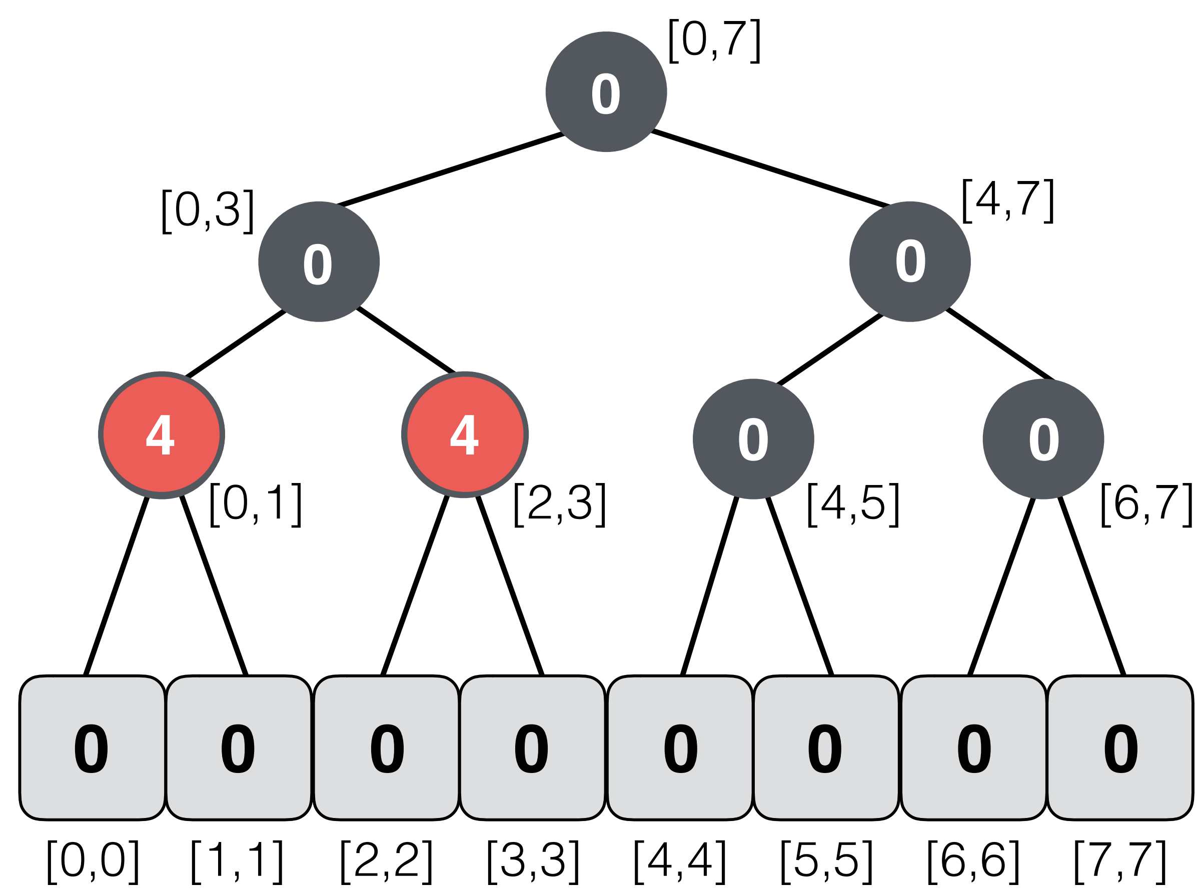
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
$\boxed{\texttt{update\_range}(0,3,1)}$
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

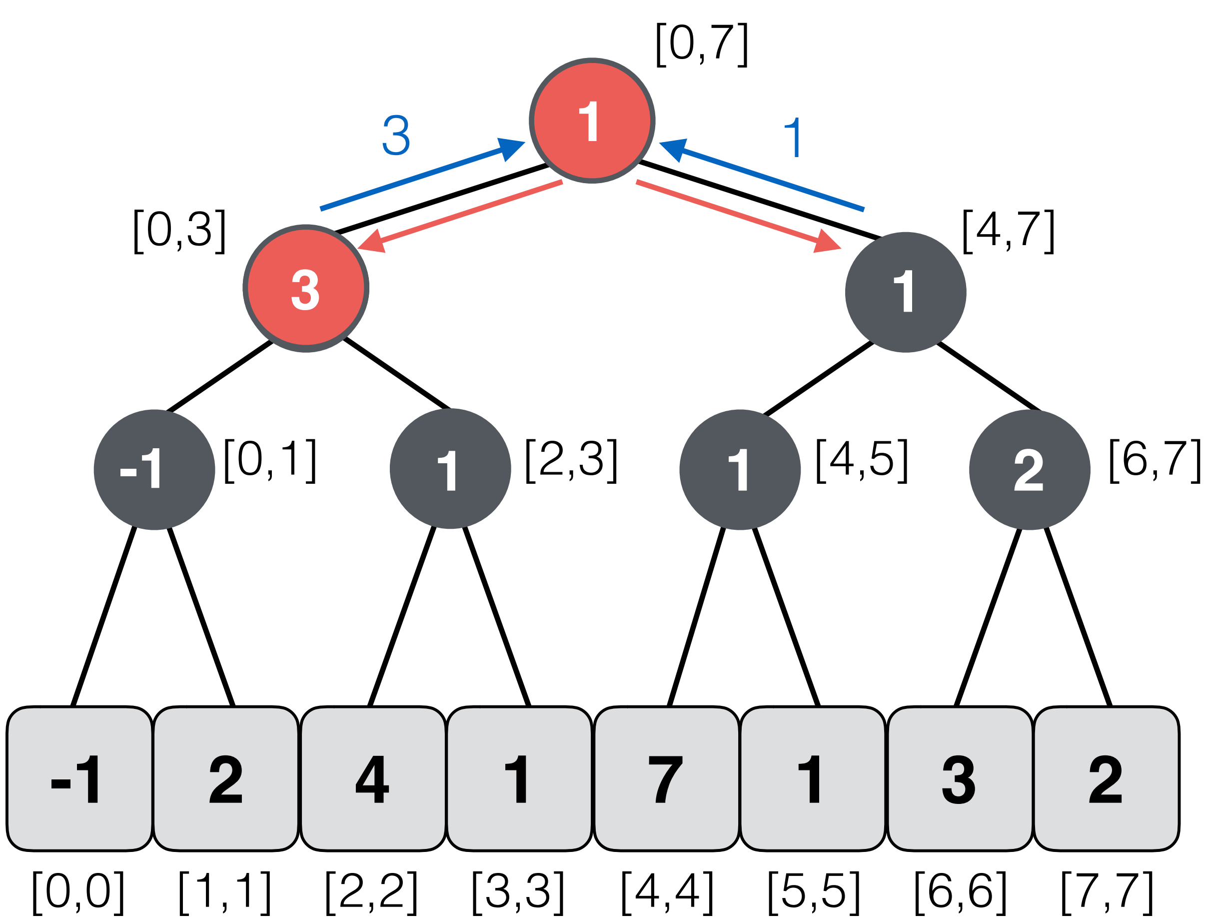


Lazy Tree

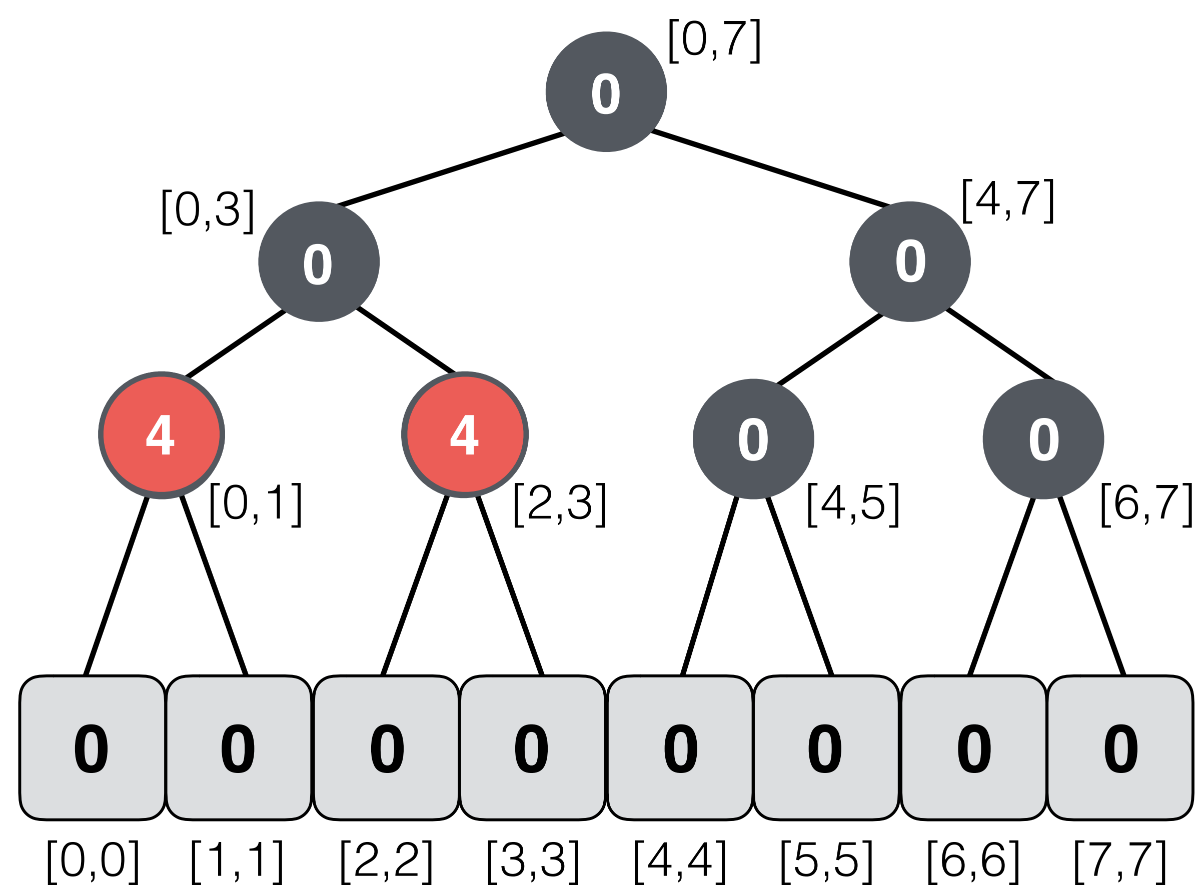
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

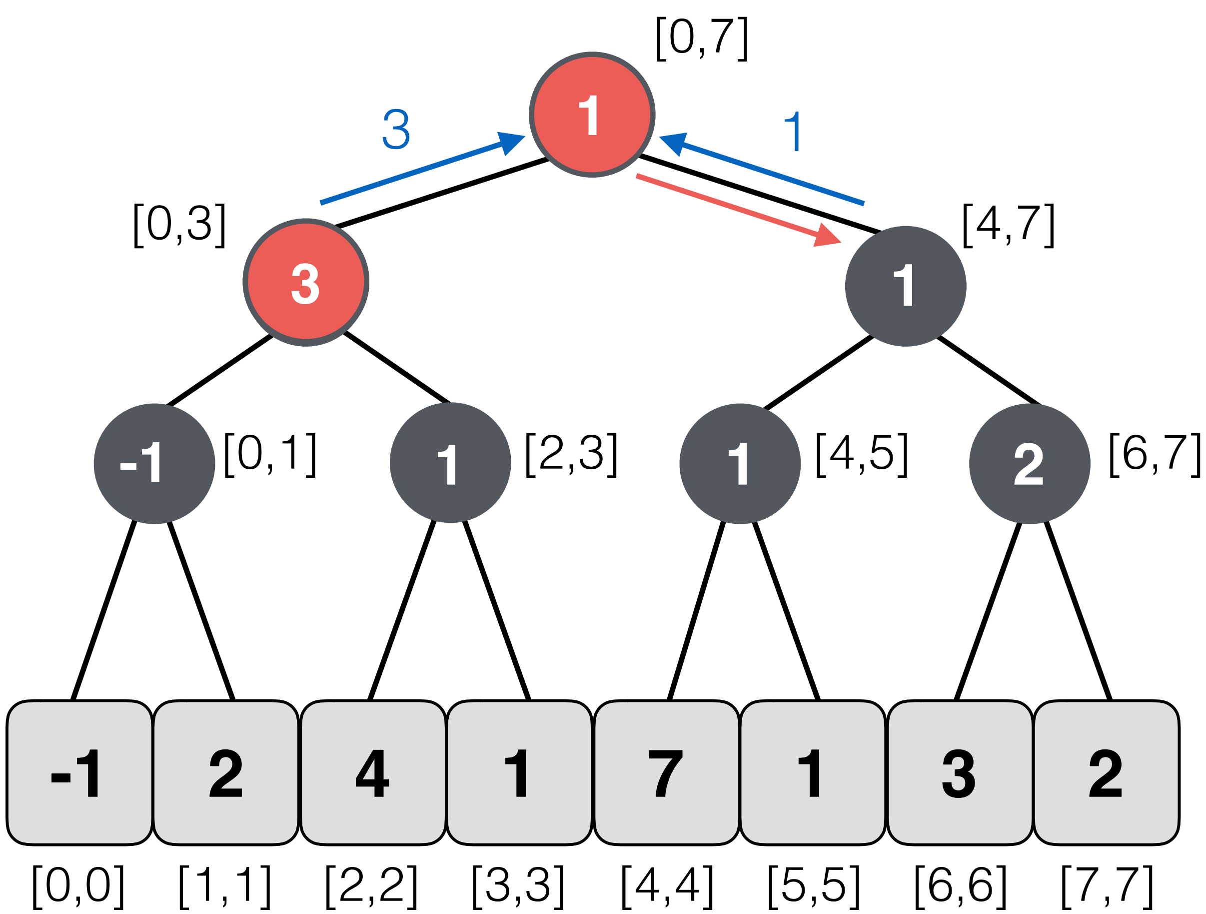


Lazy Tree

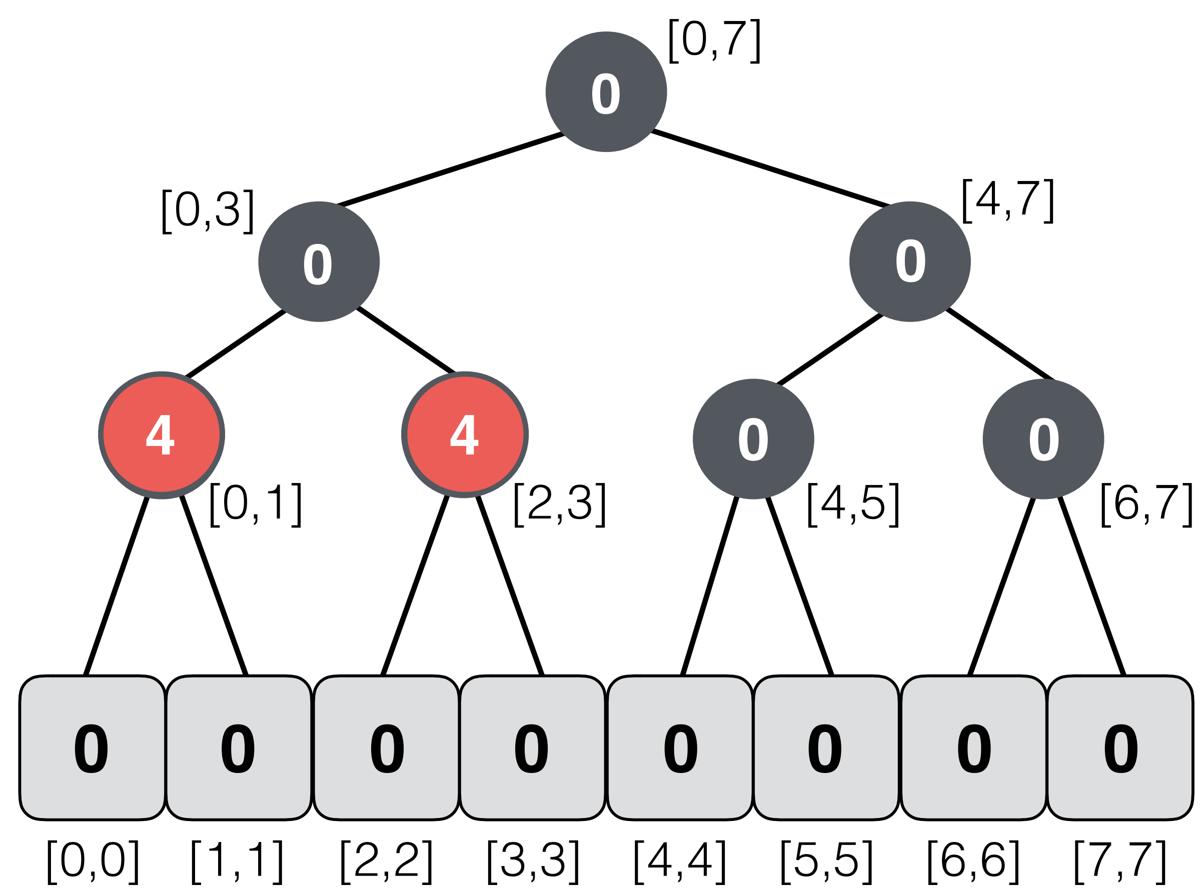
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

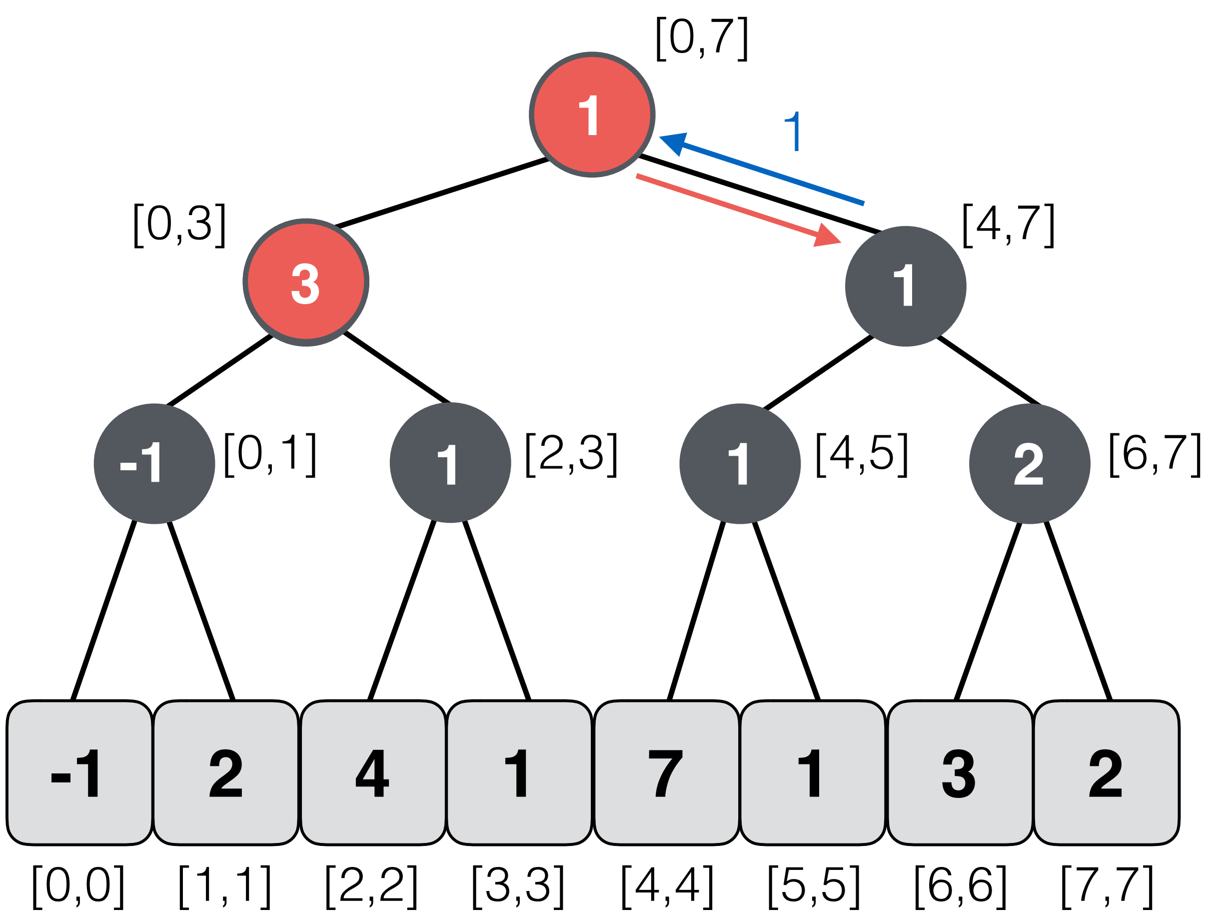


Lazy Tree

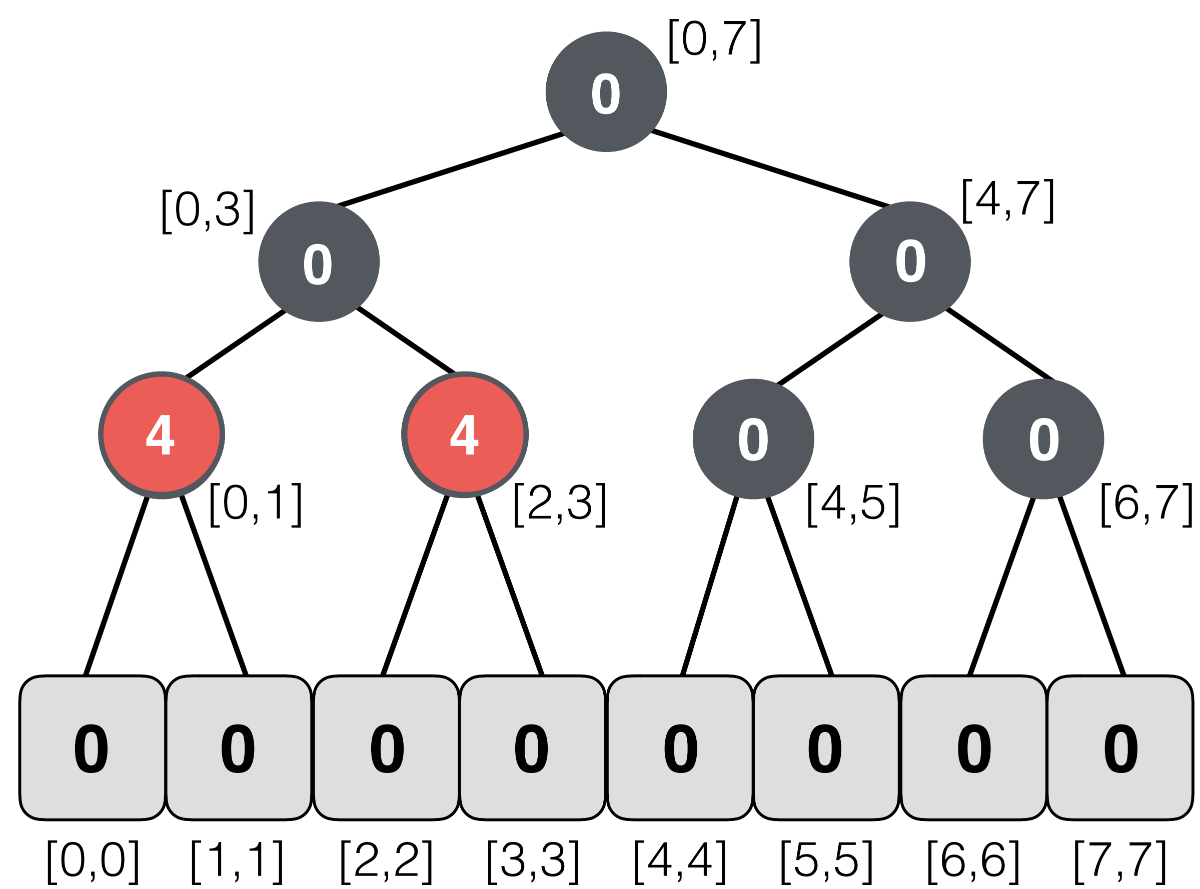
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

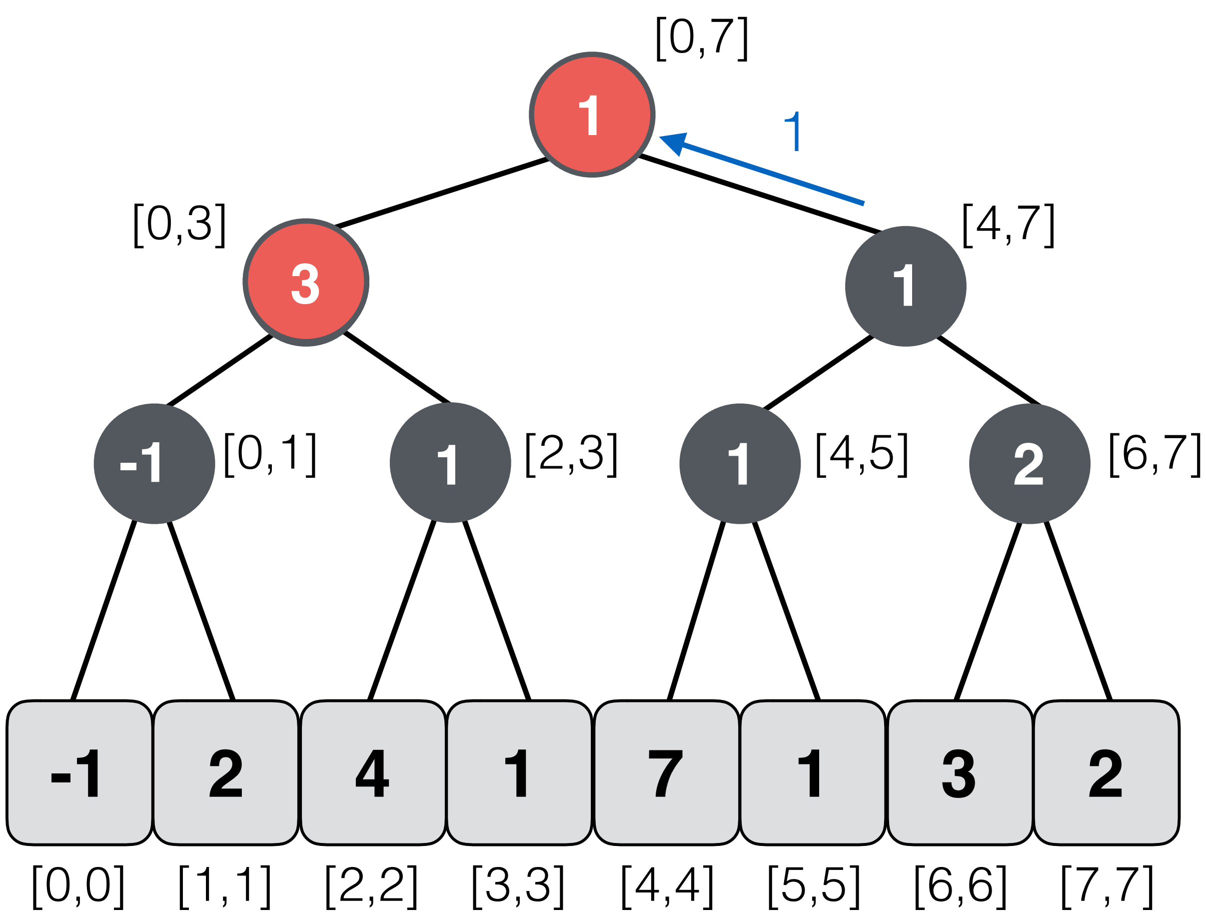


Lazy Tree

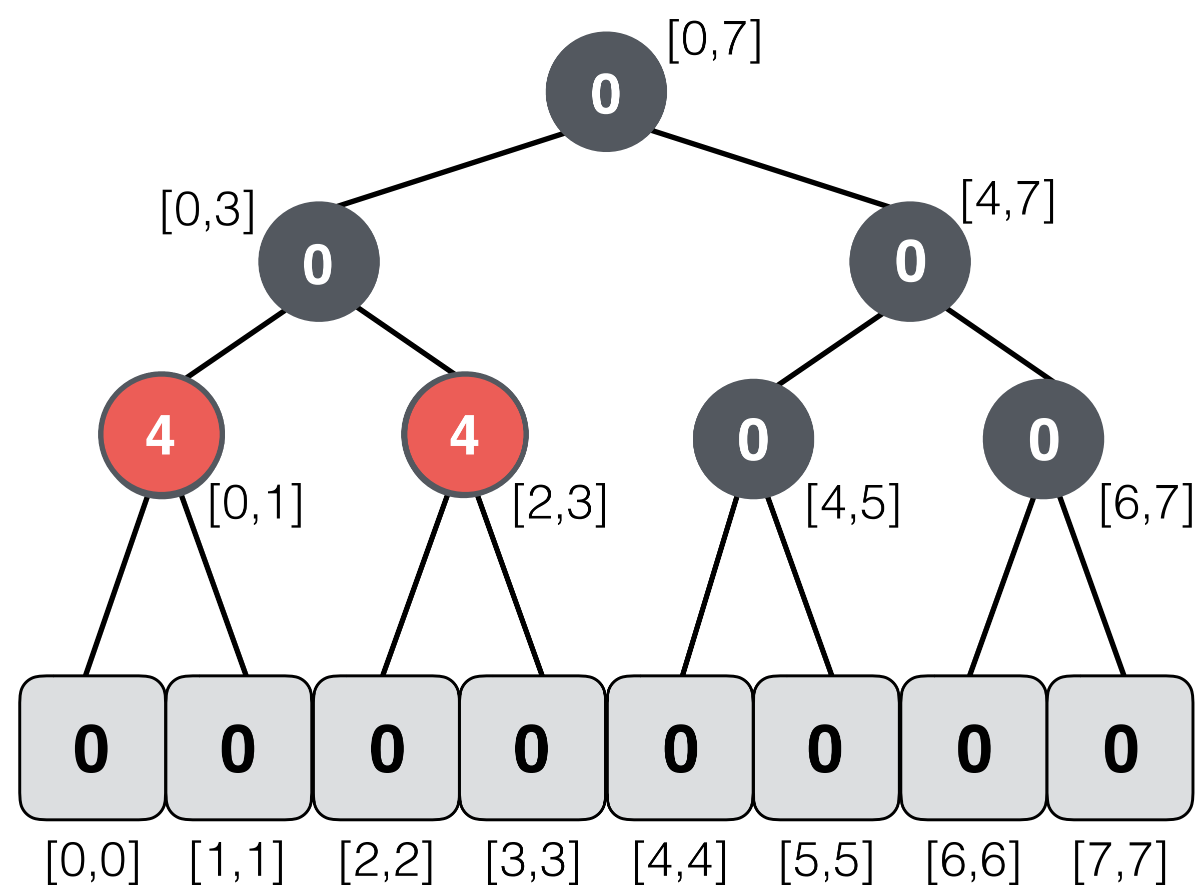
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

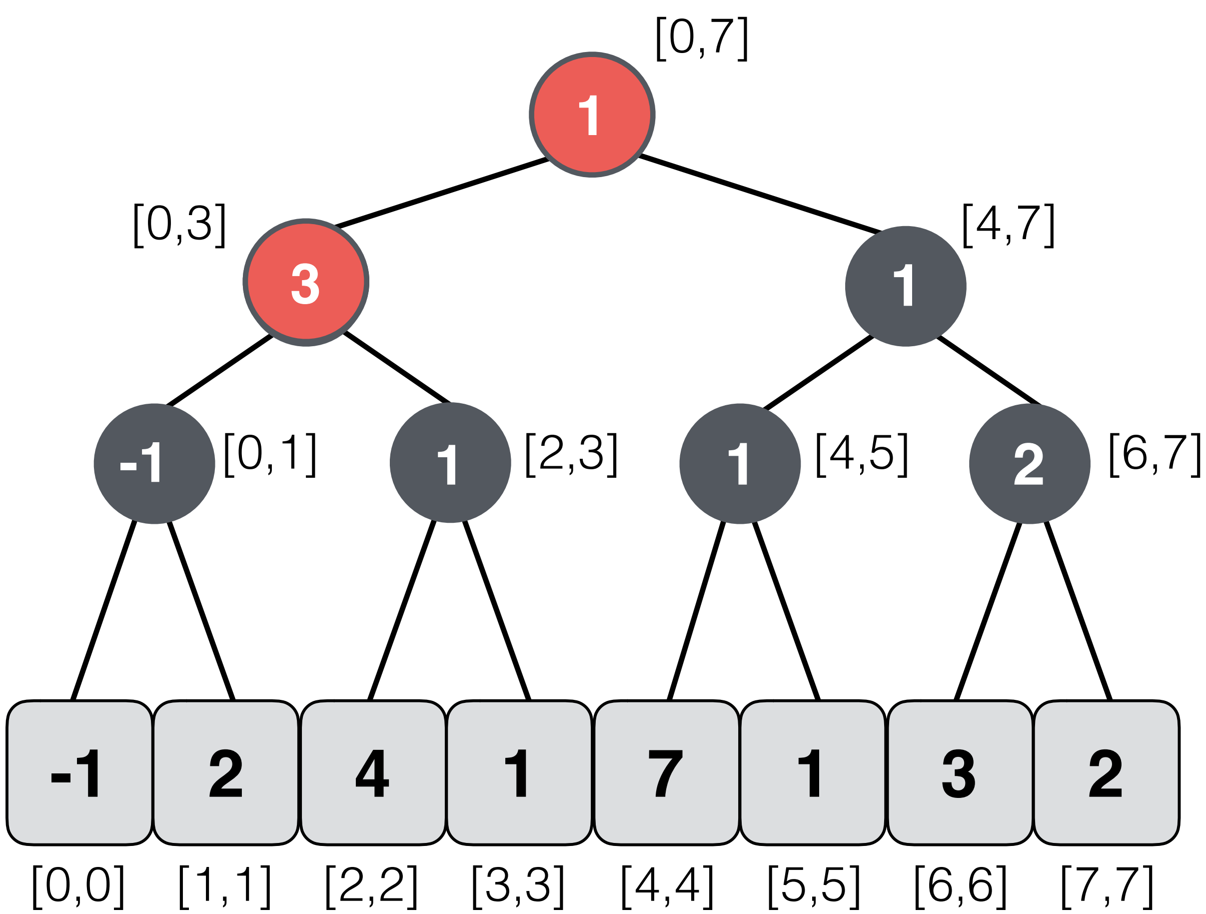


Lazy Tree

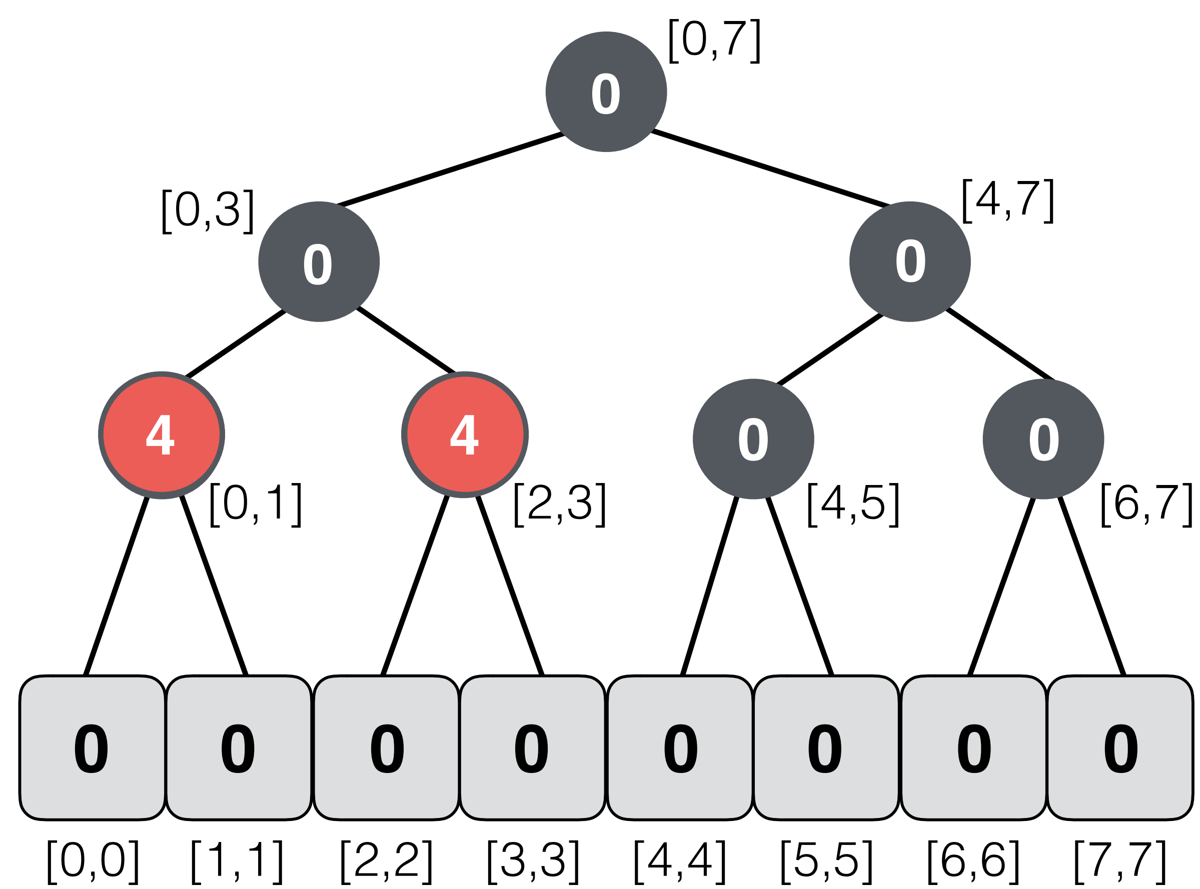
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

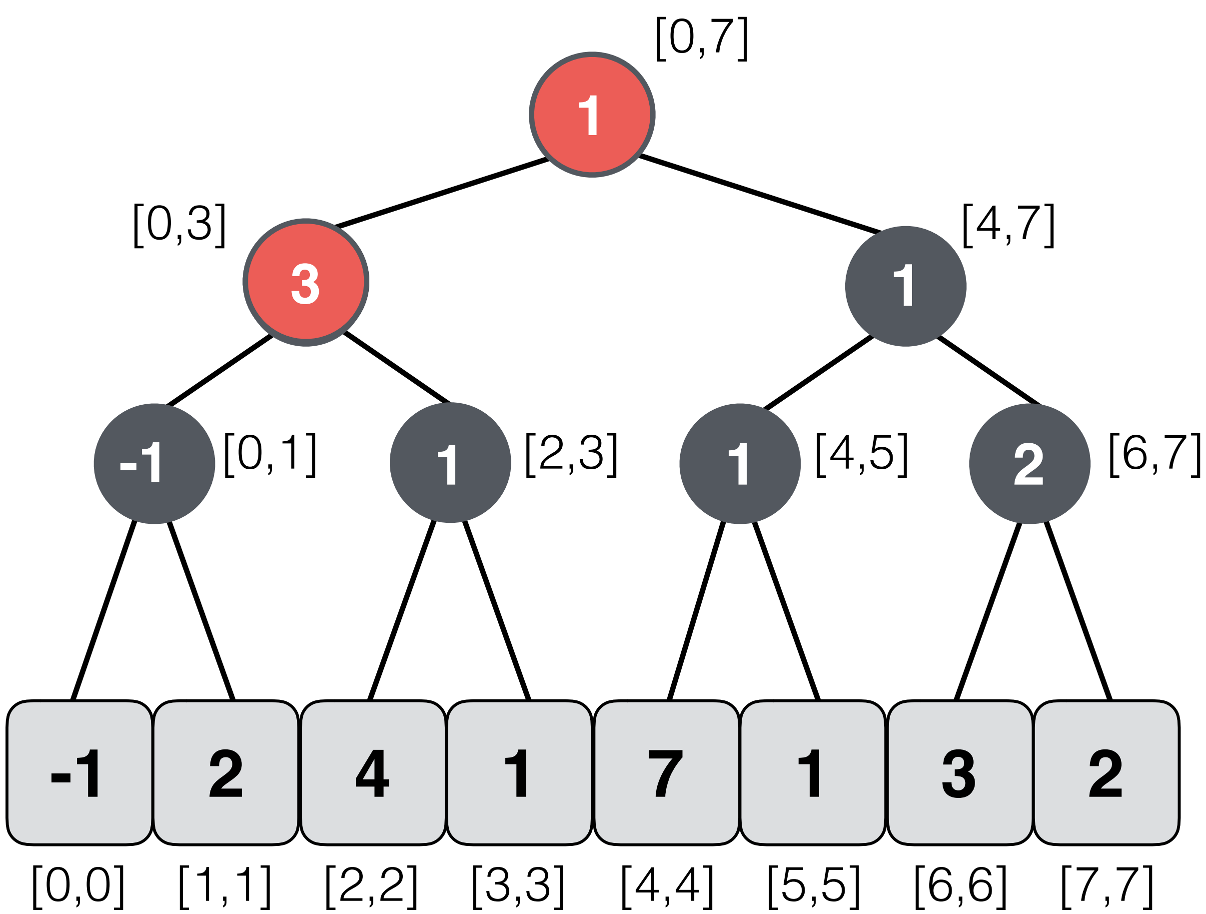


Lazy Tree

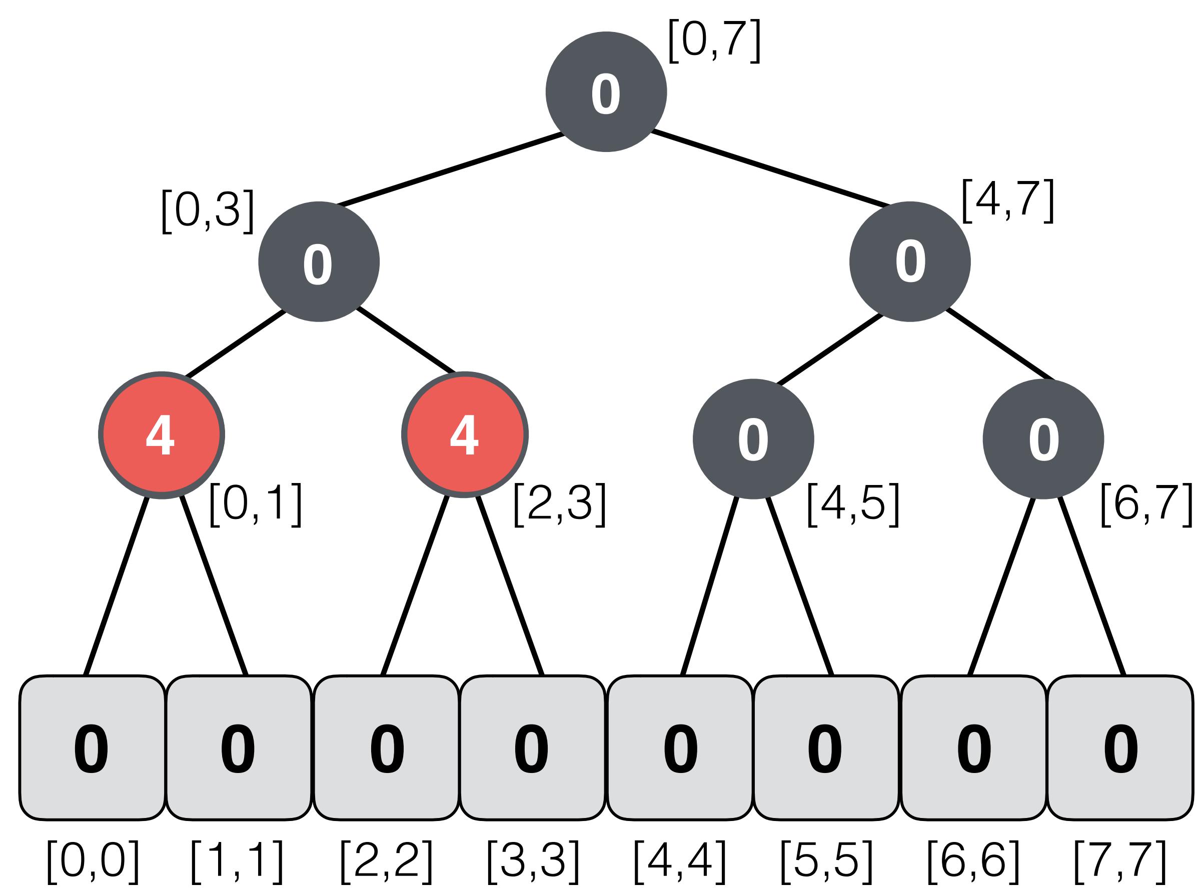
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

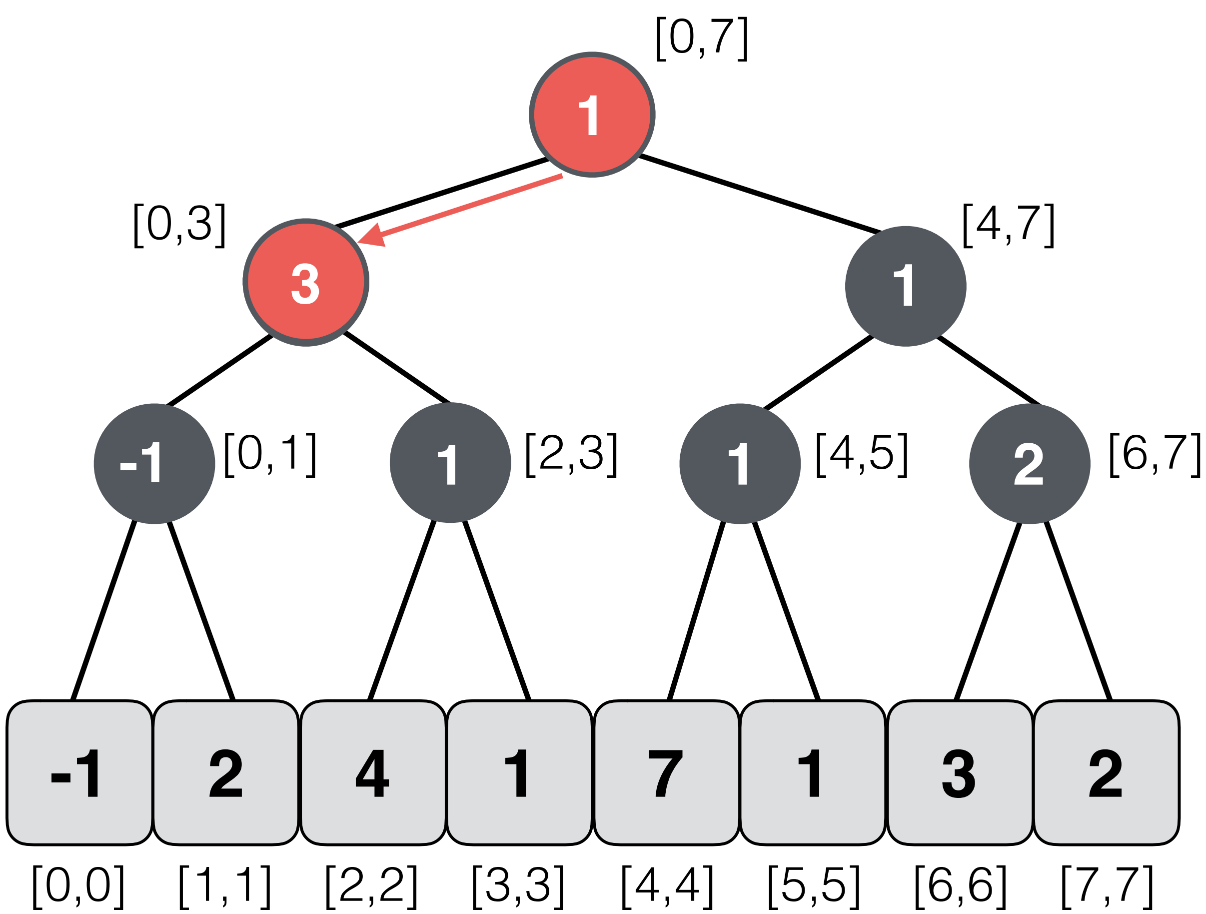


Lazy Tree

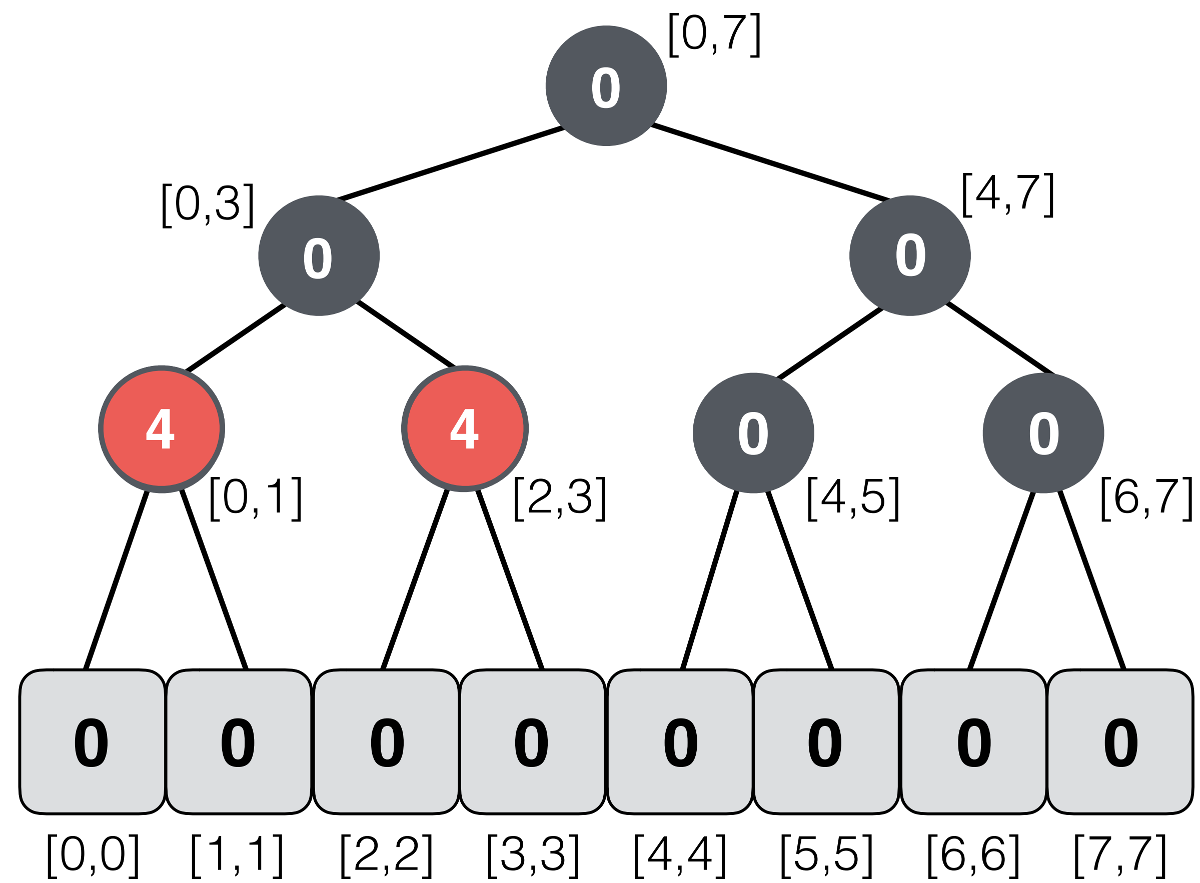
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

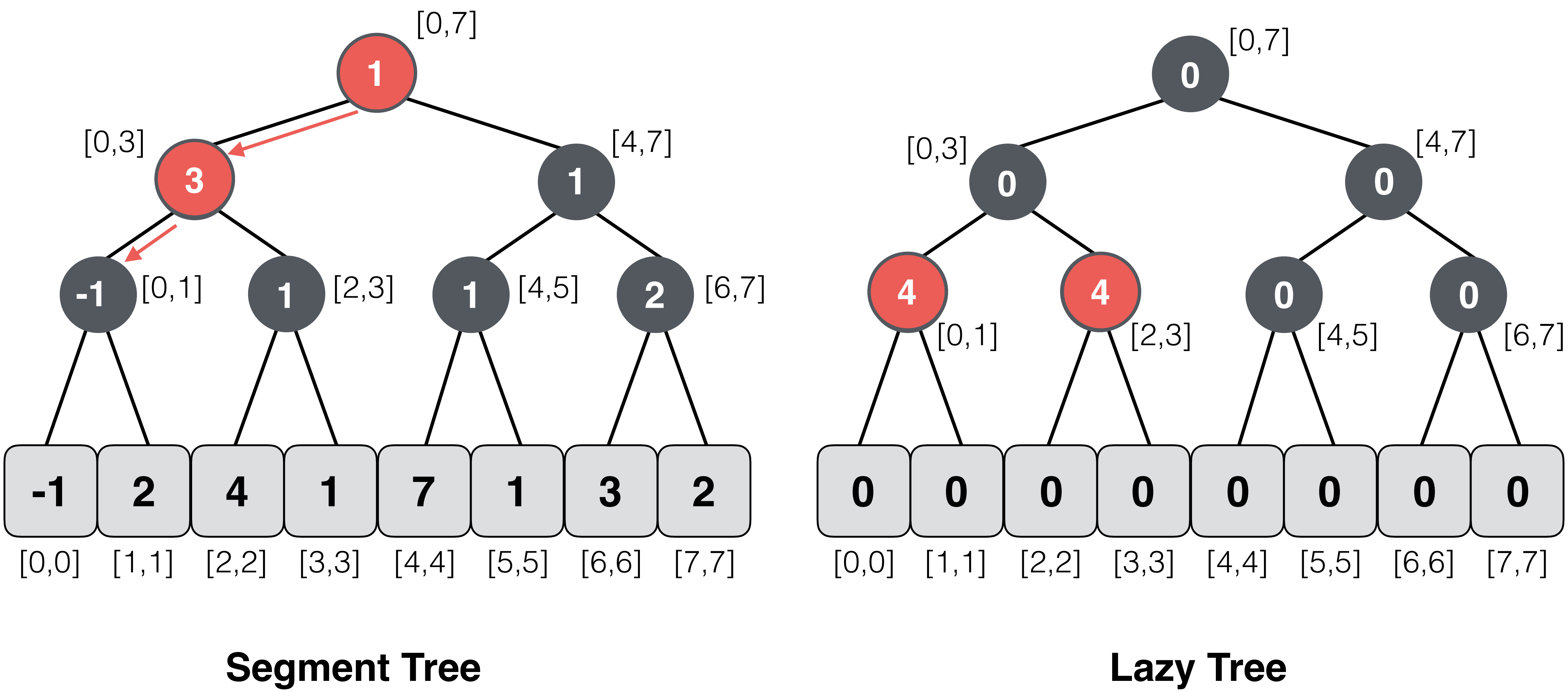


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

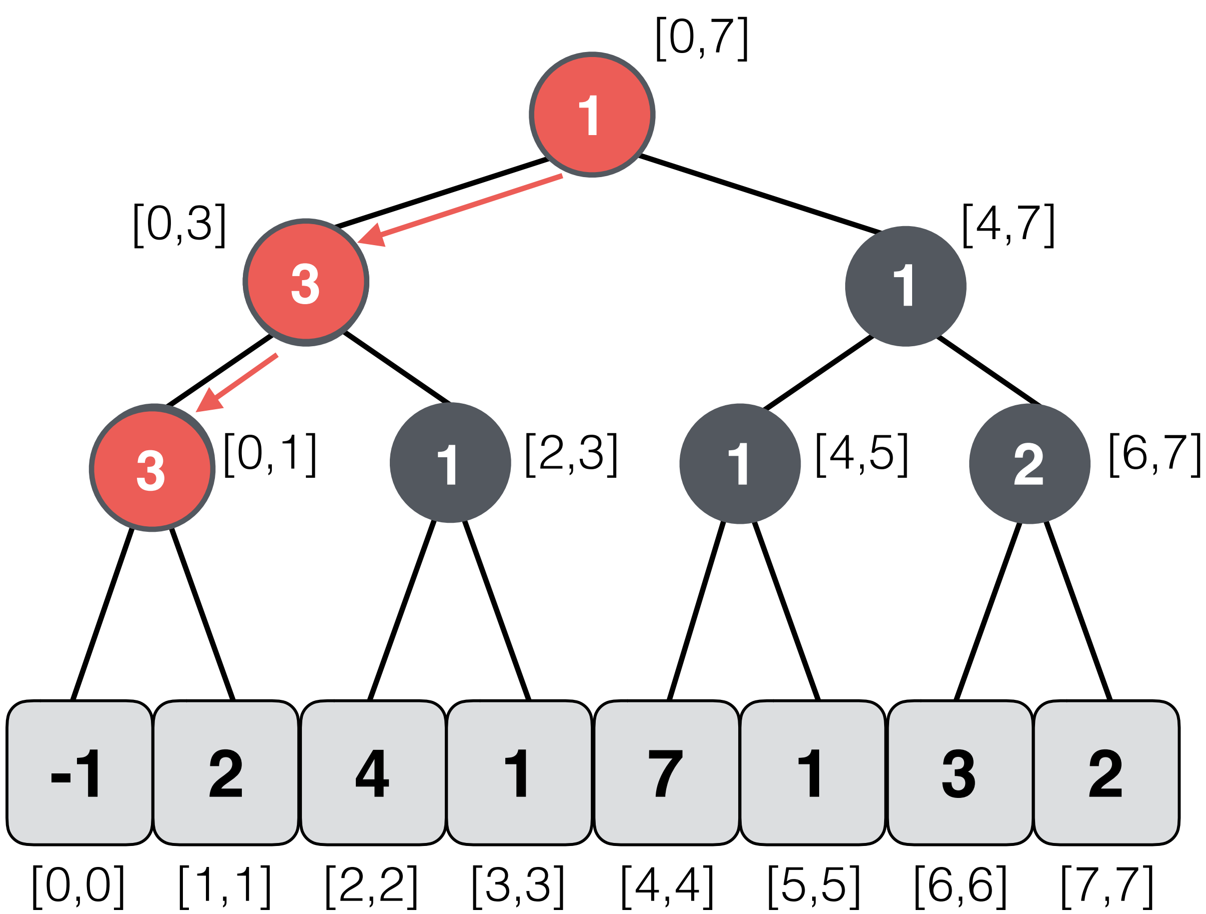
`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



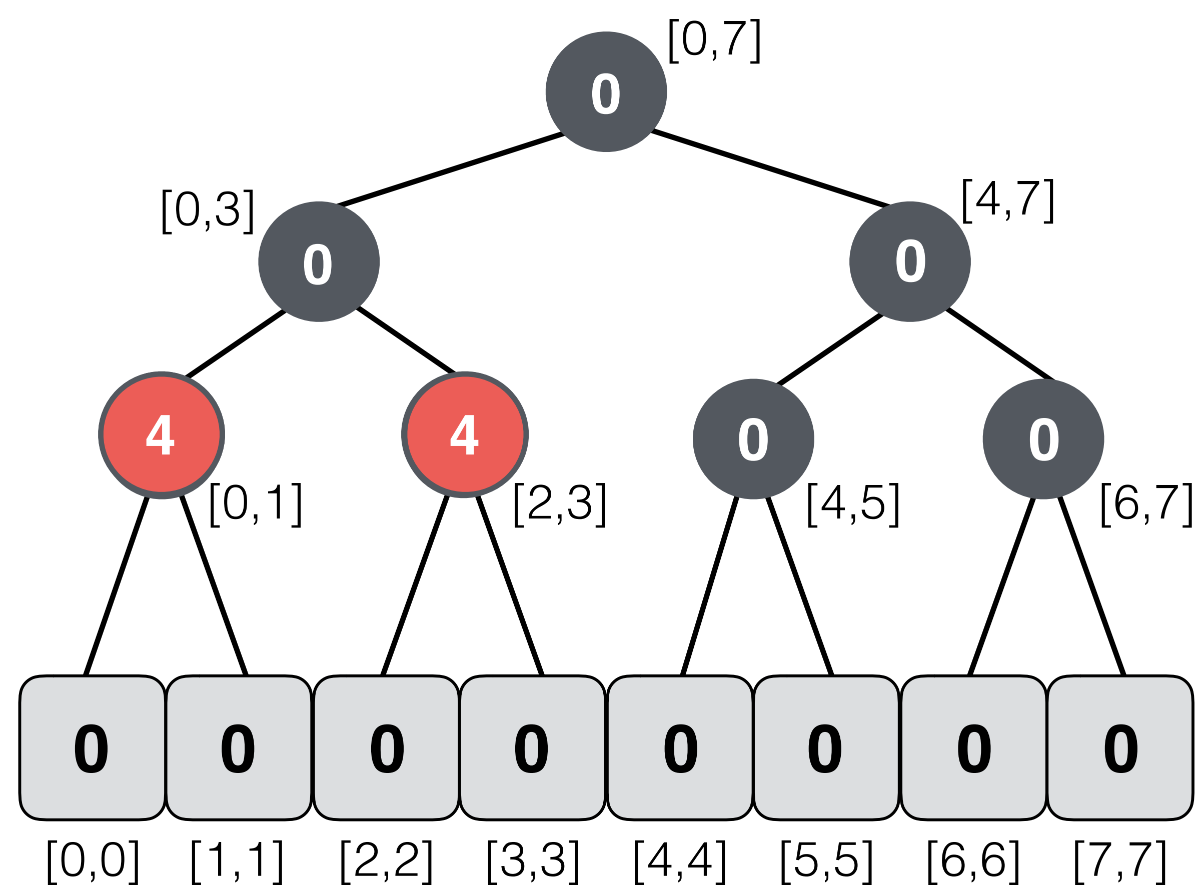
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

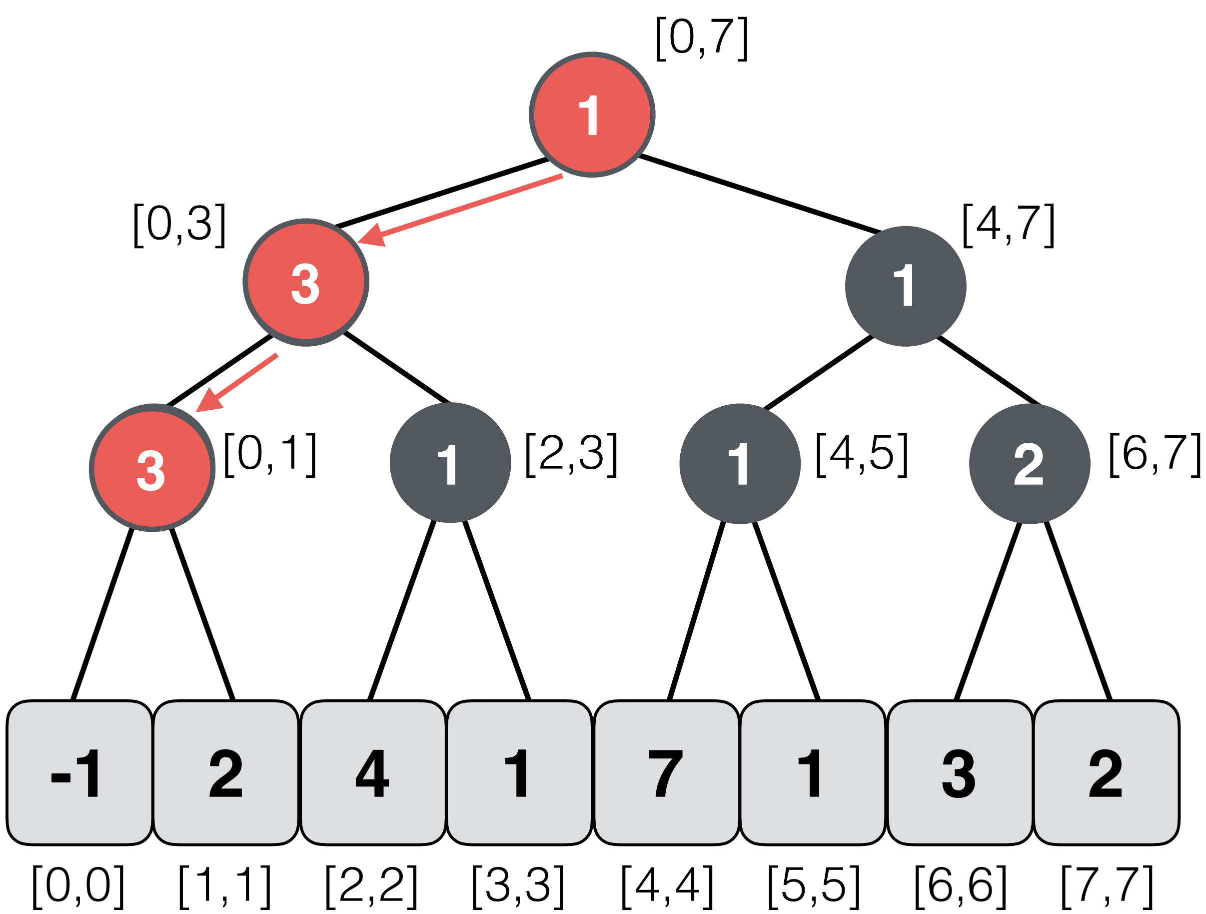


Lazy Tree

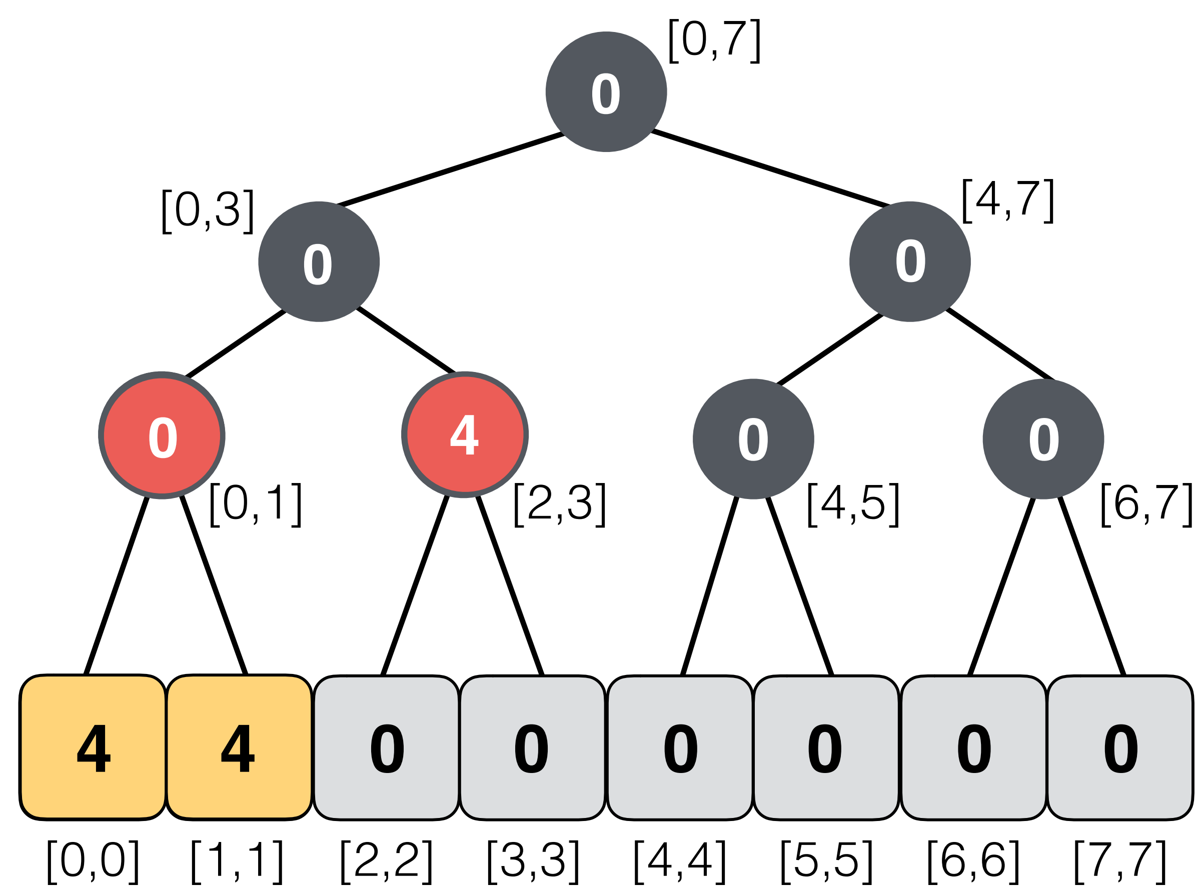
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

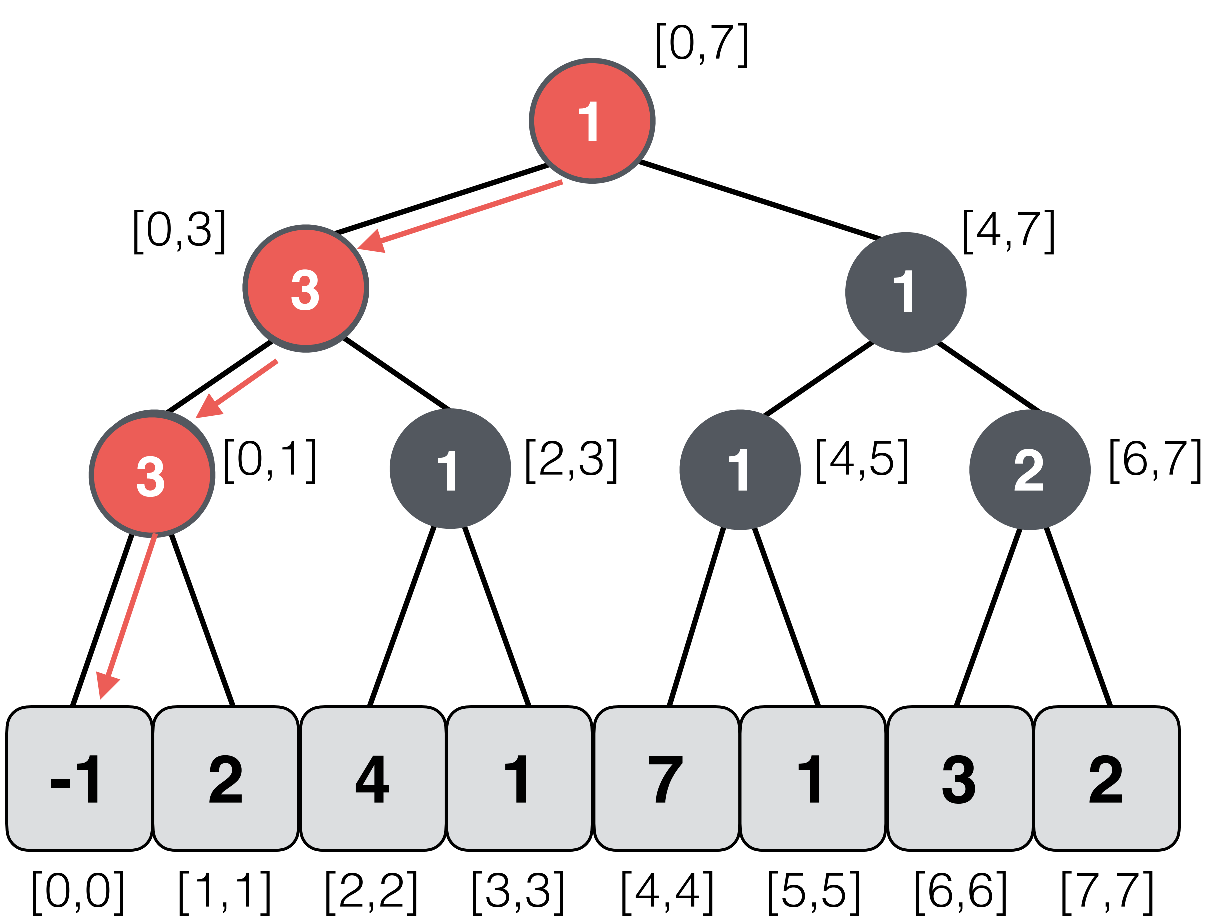


Lazy Tree

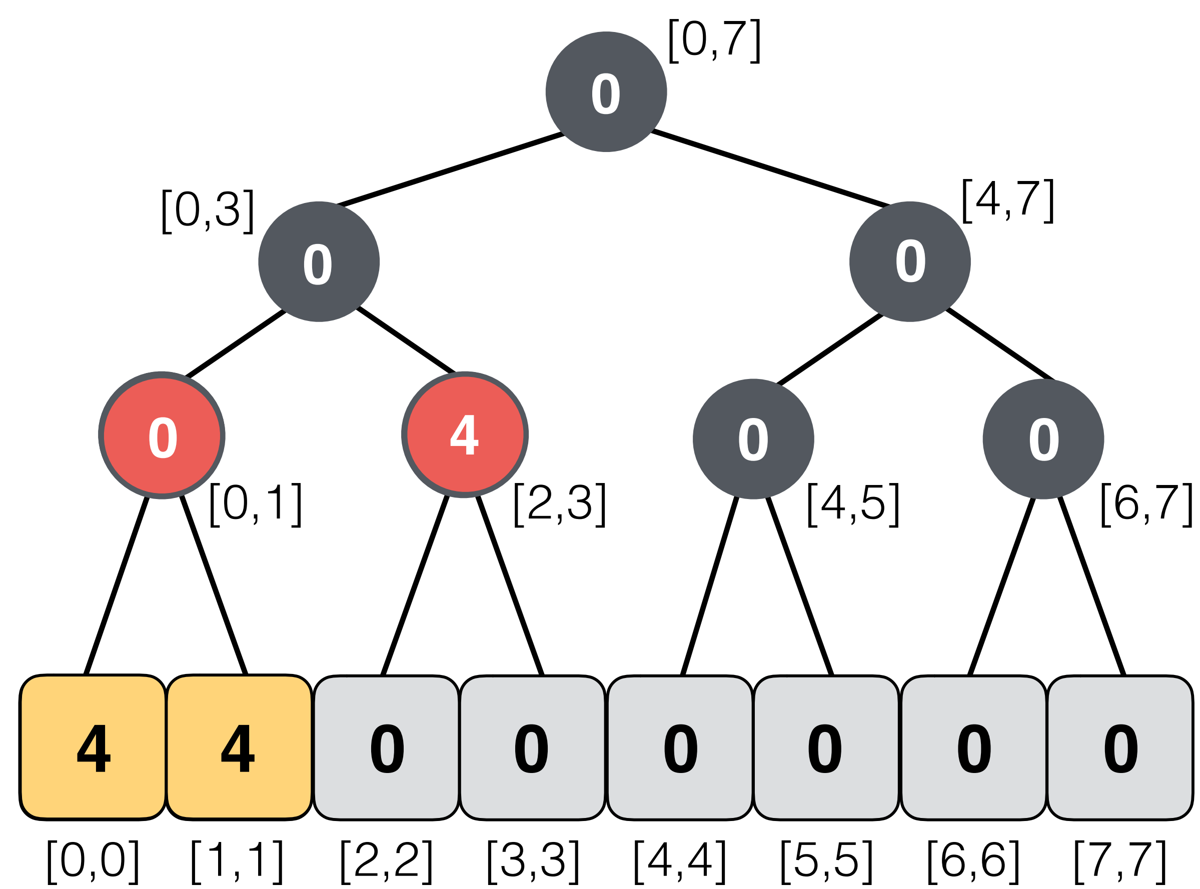
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

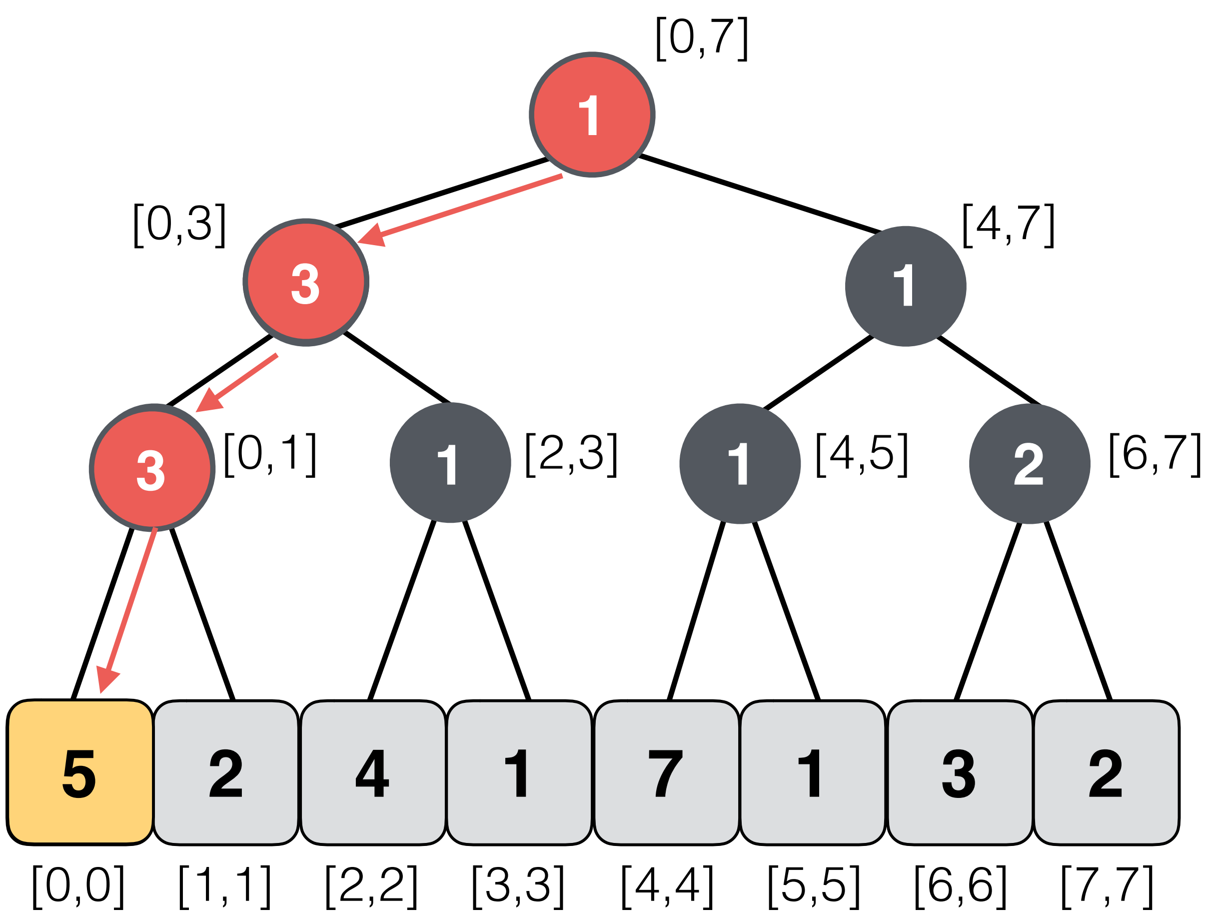


Lazy Tree

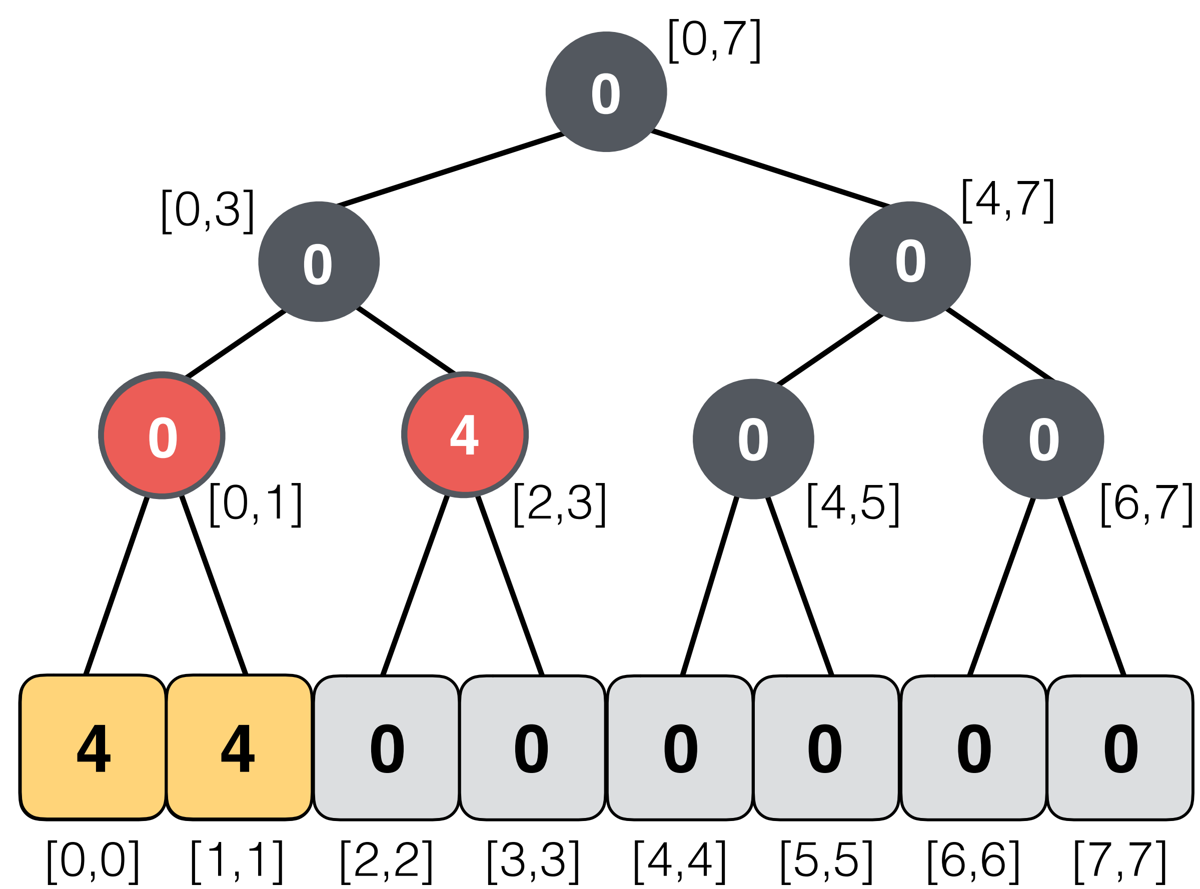
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

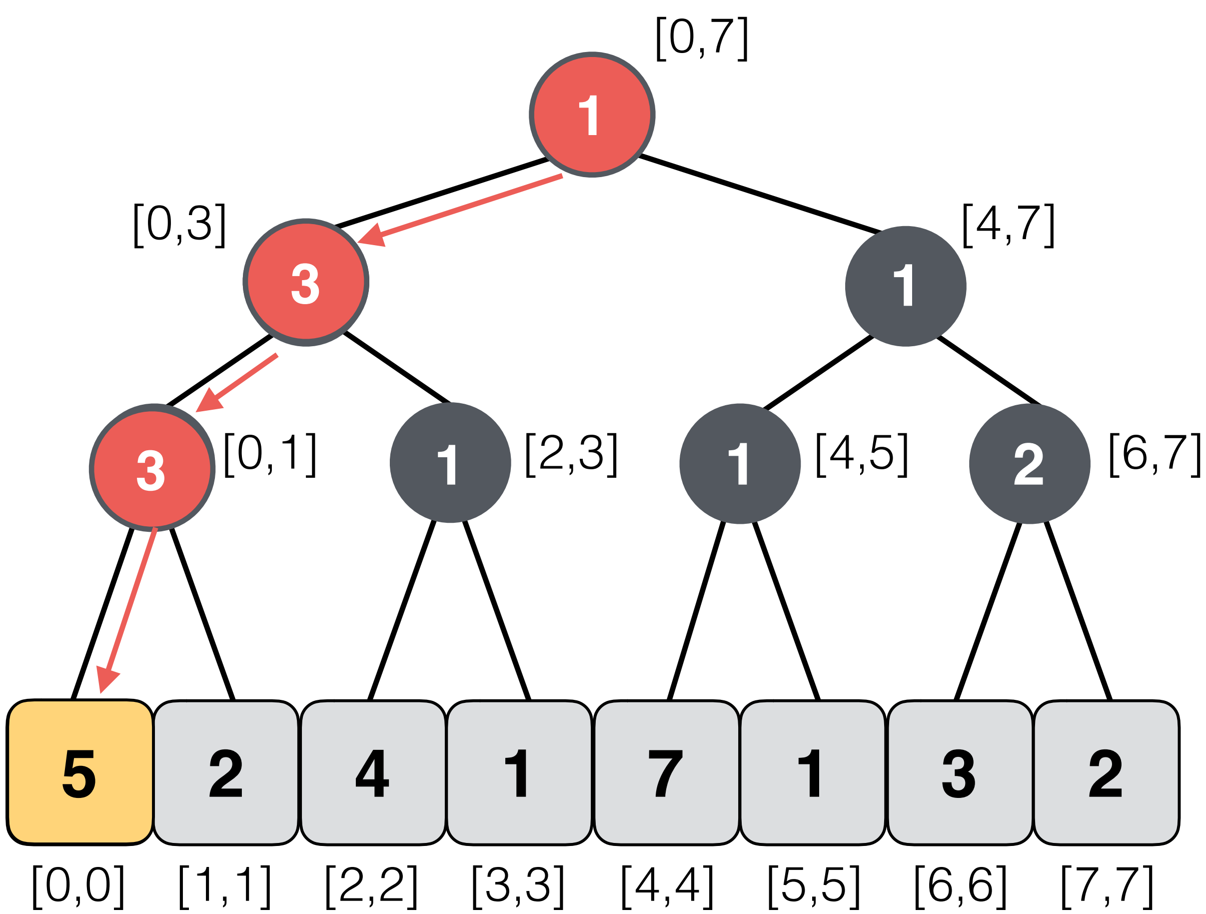


Lazy Tree

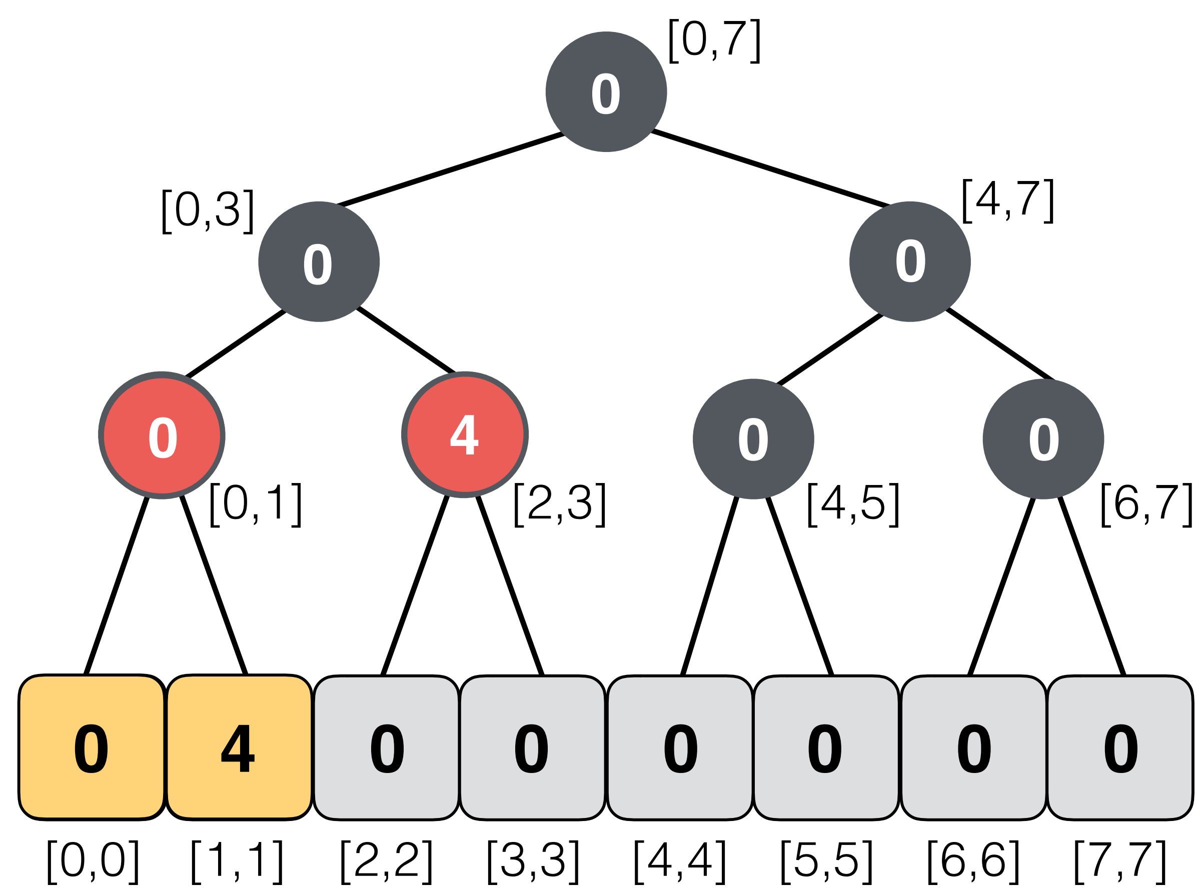
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

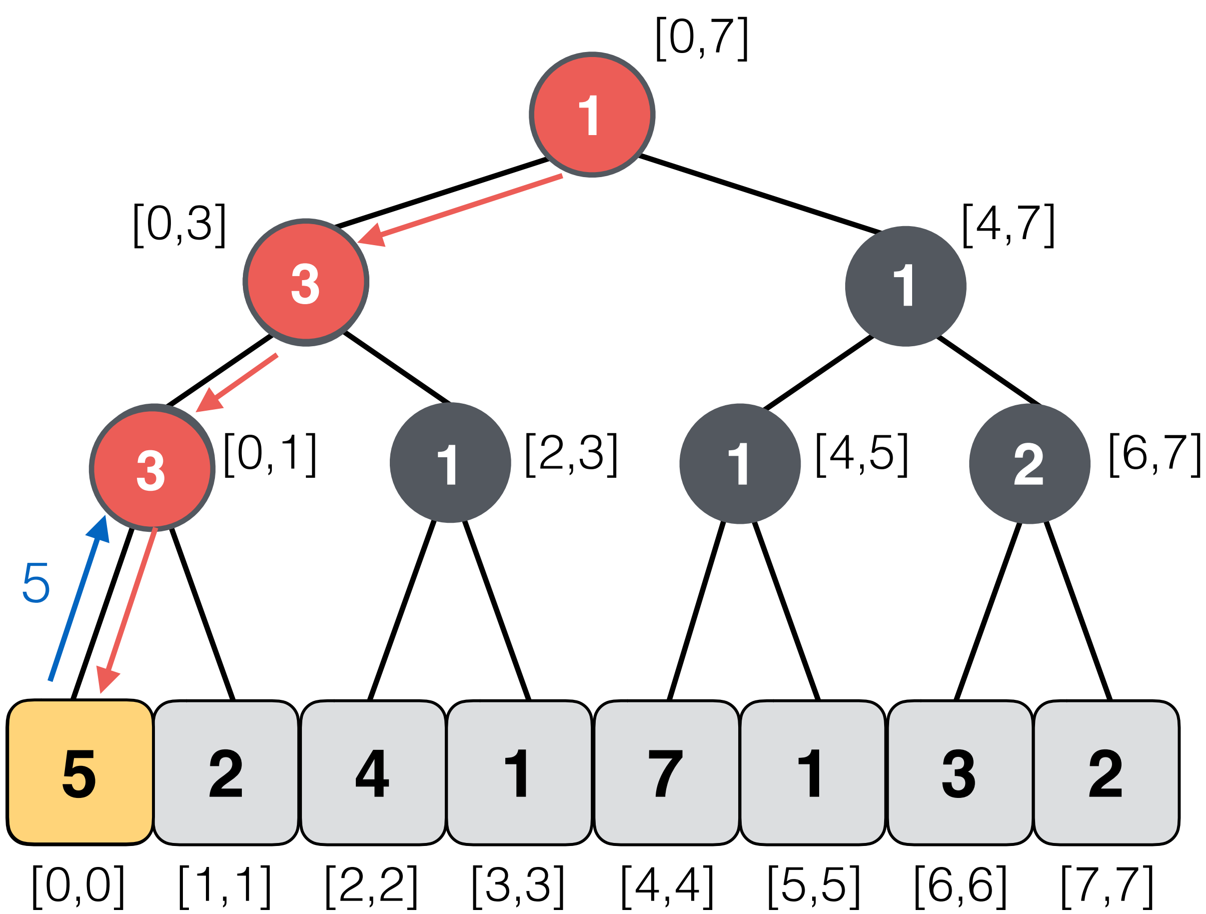


Lazy Tree

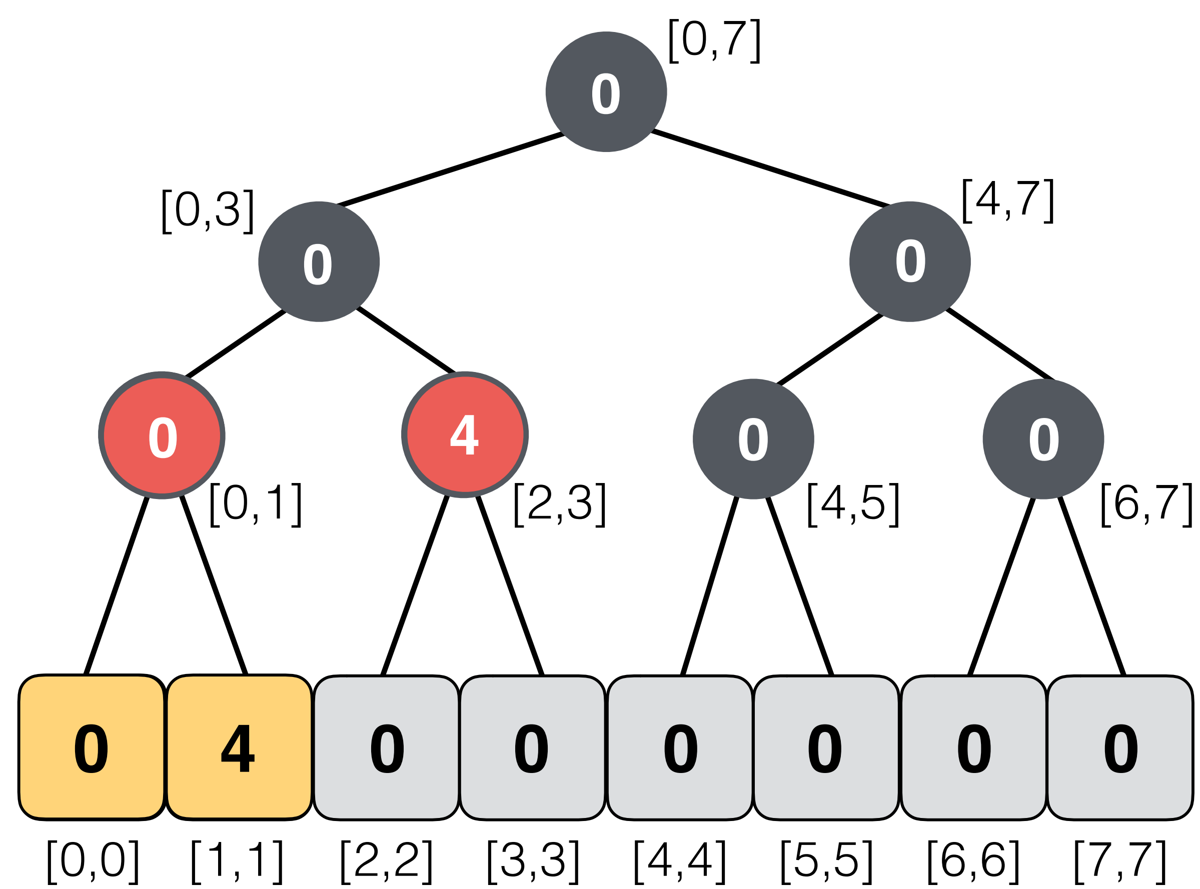
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

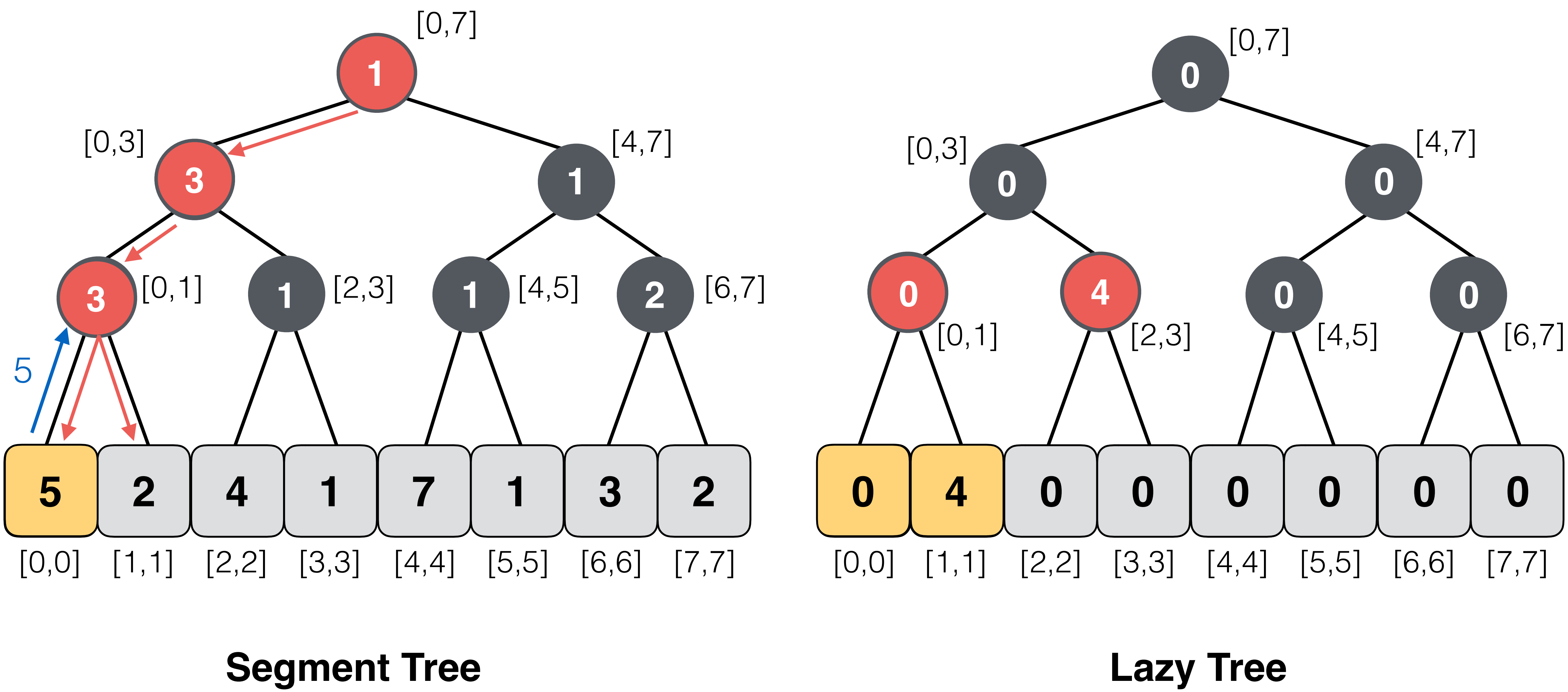


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

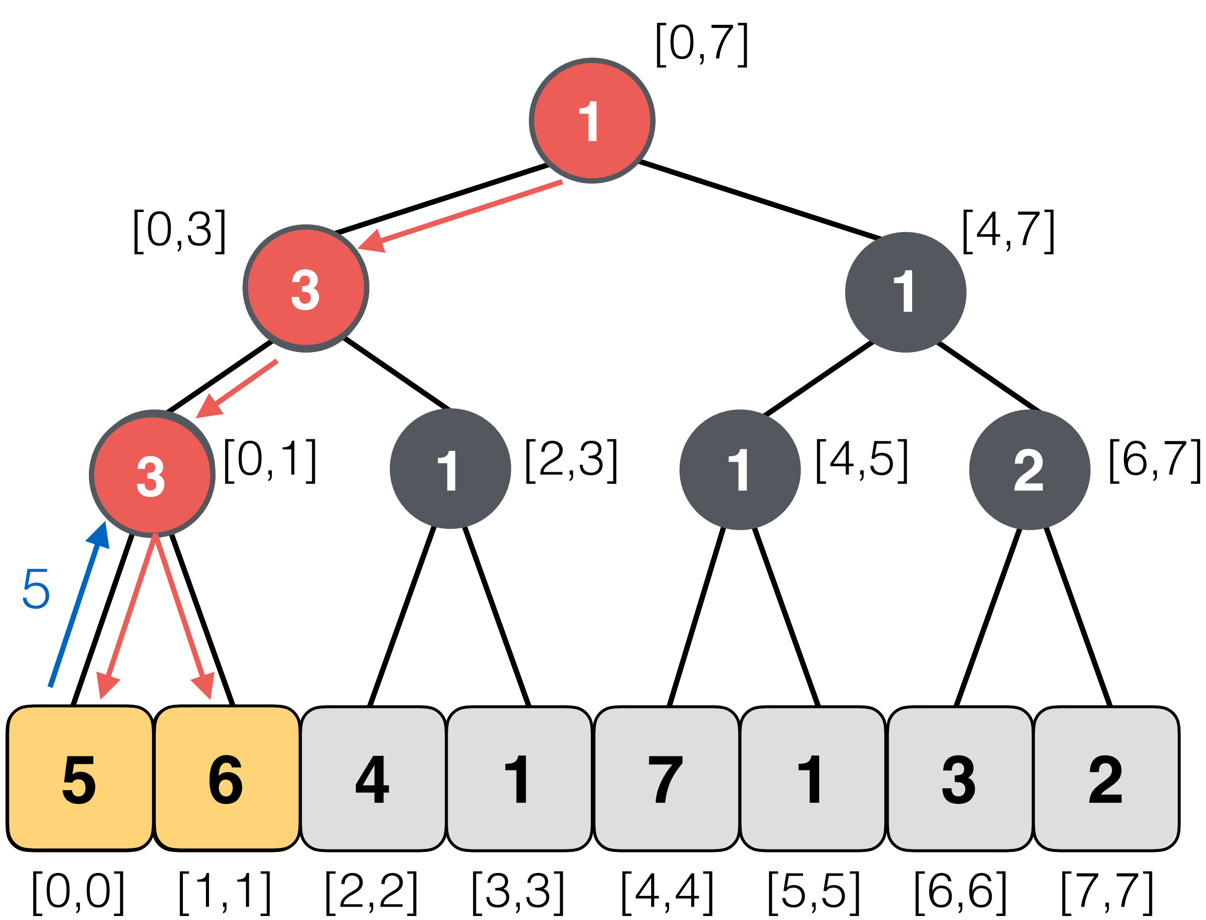
`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



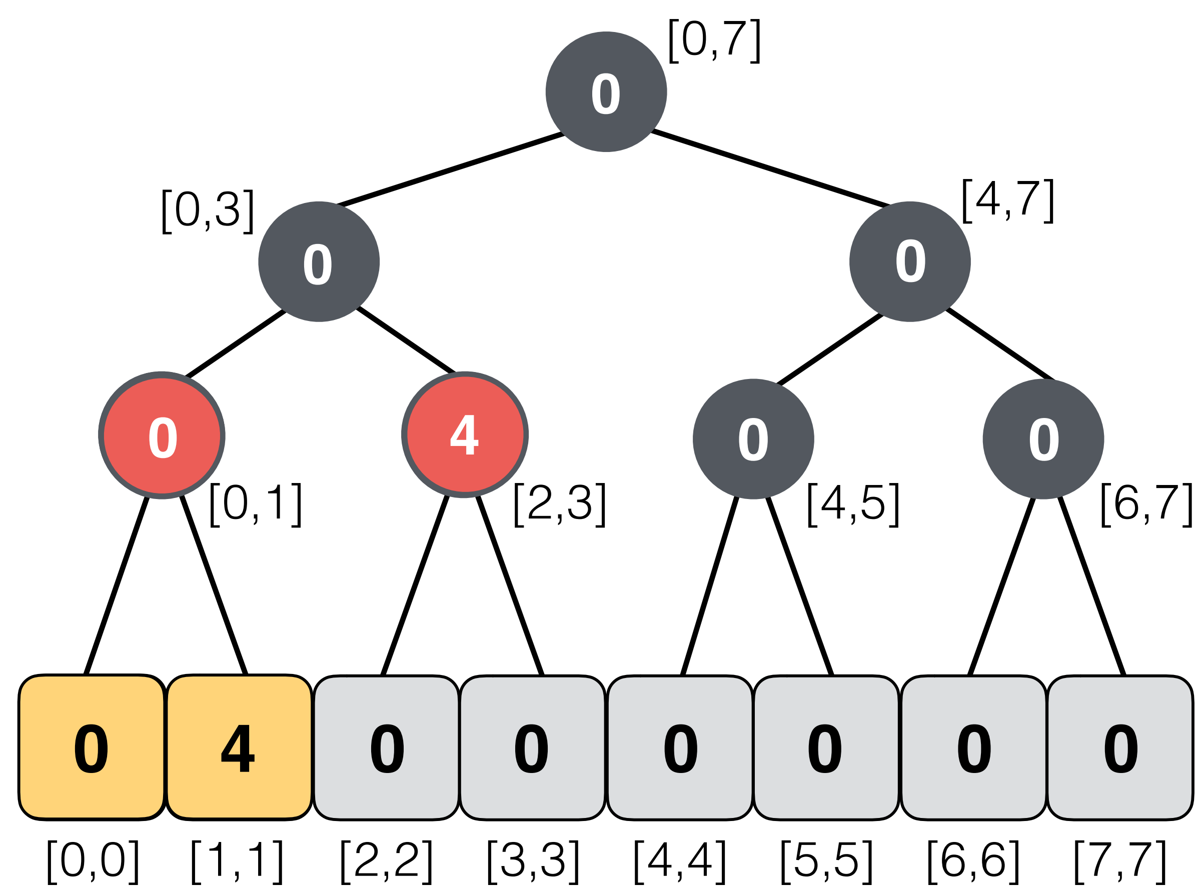
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

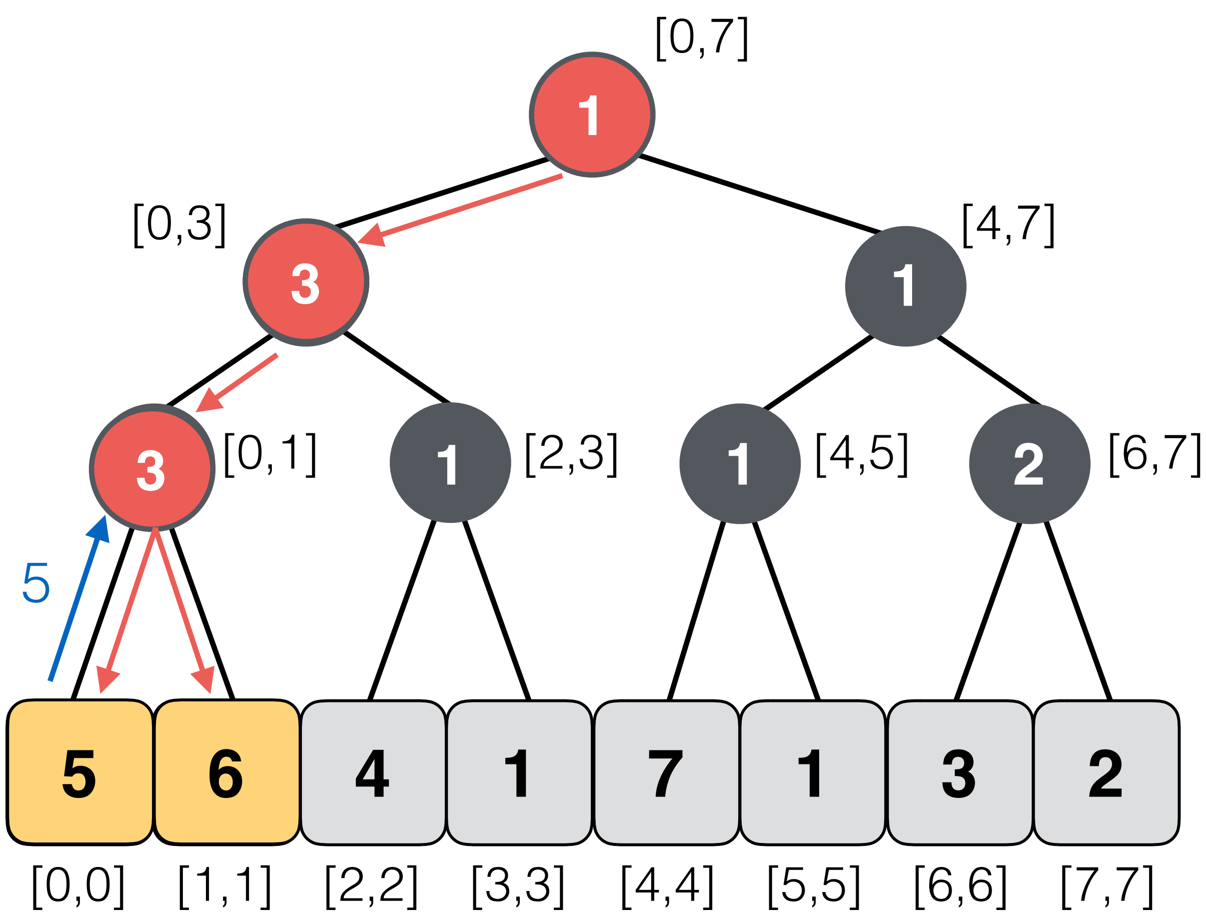


Lazy Tree

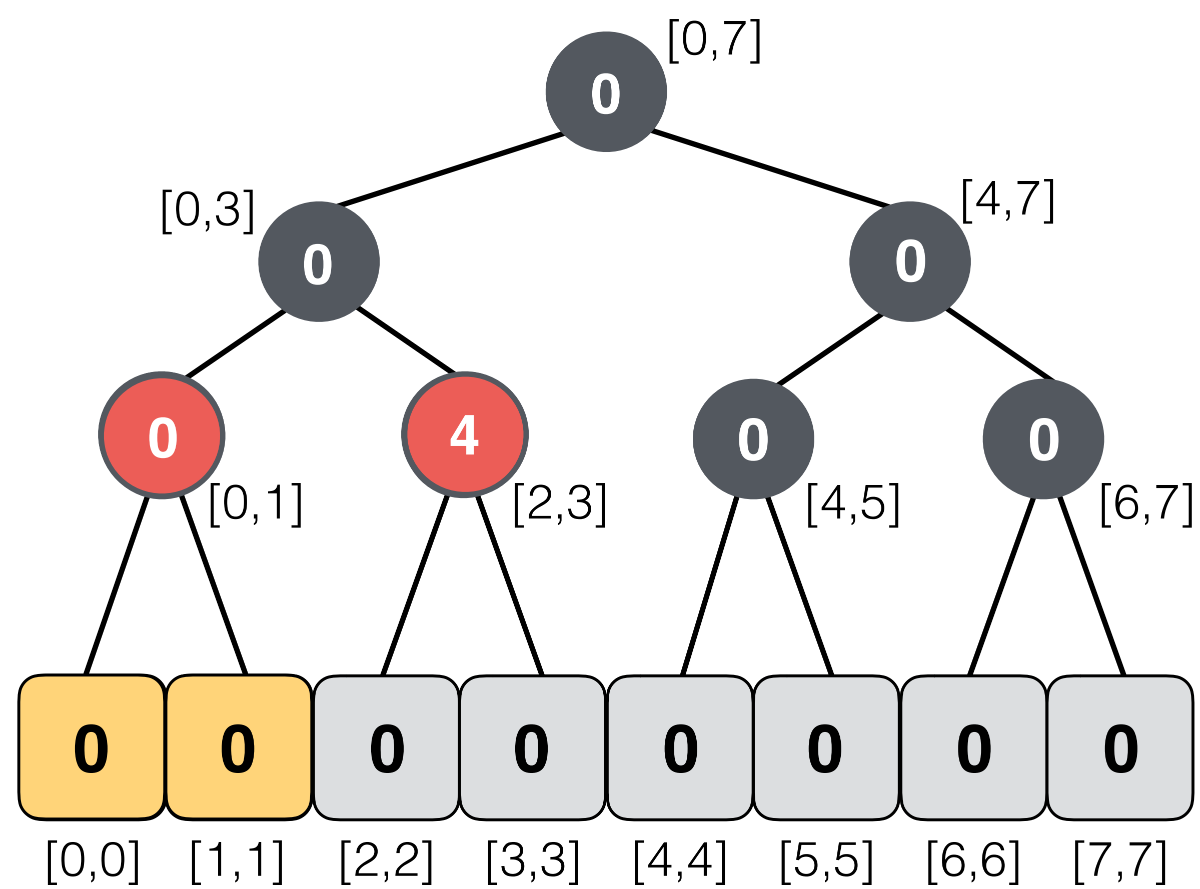
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

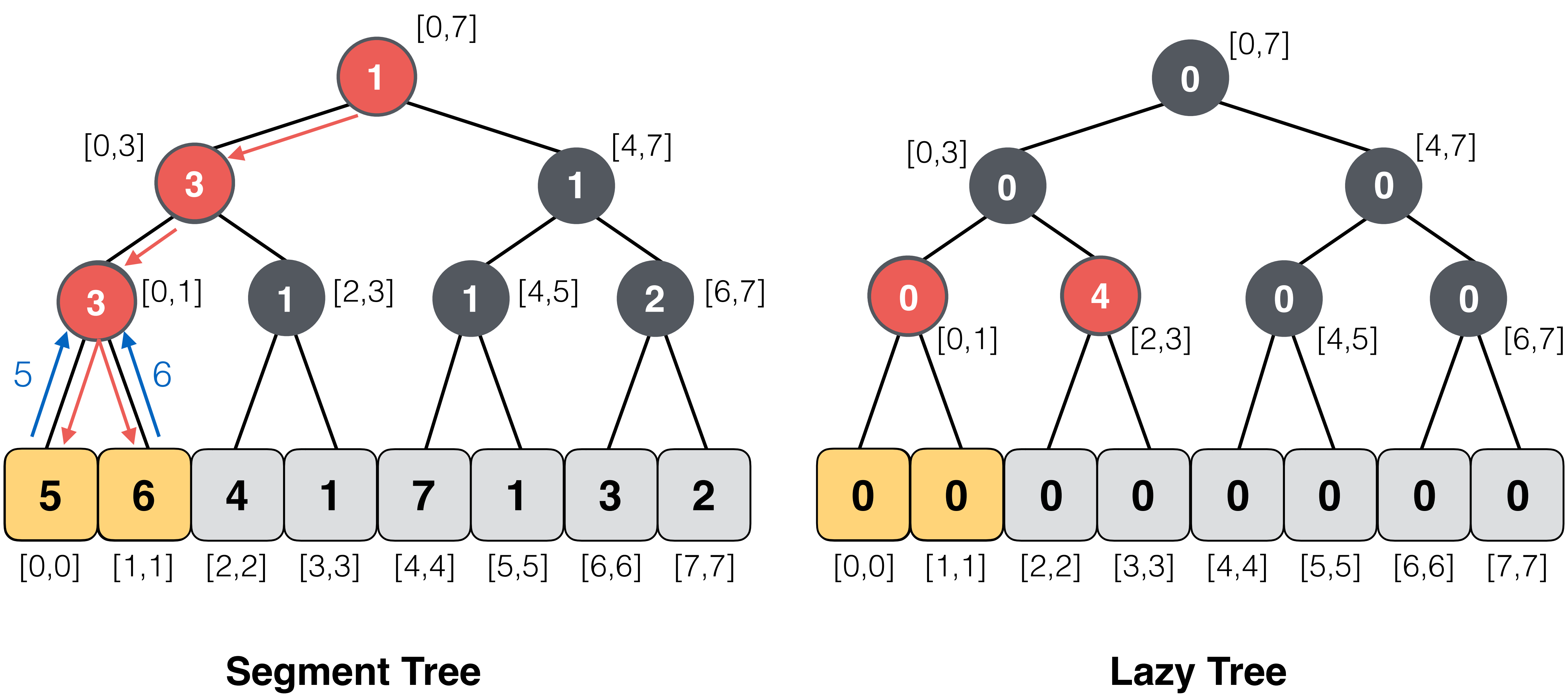


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

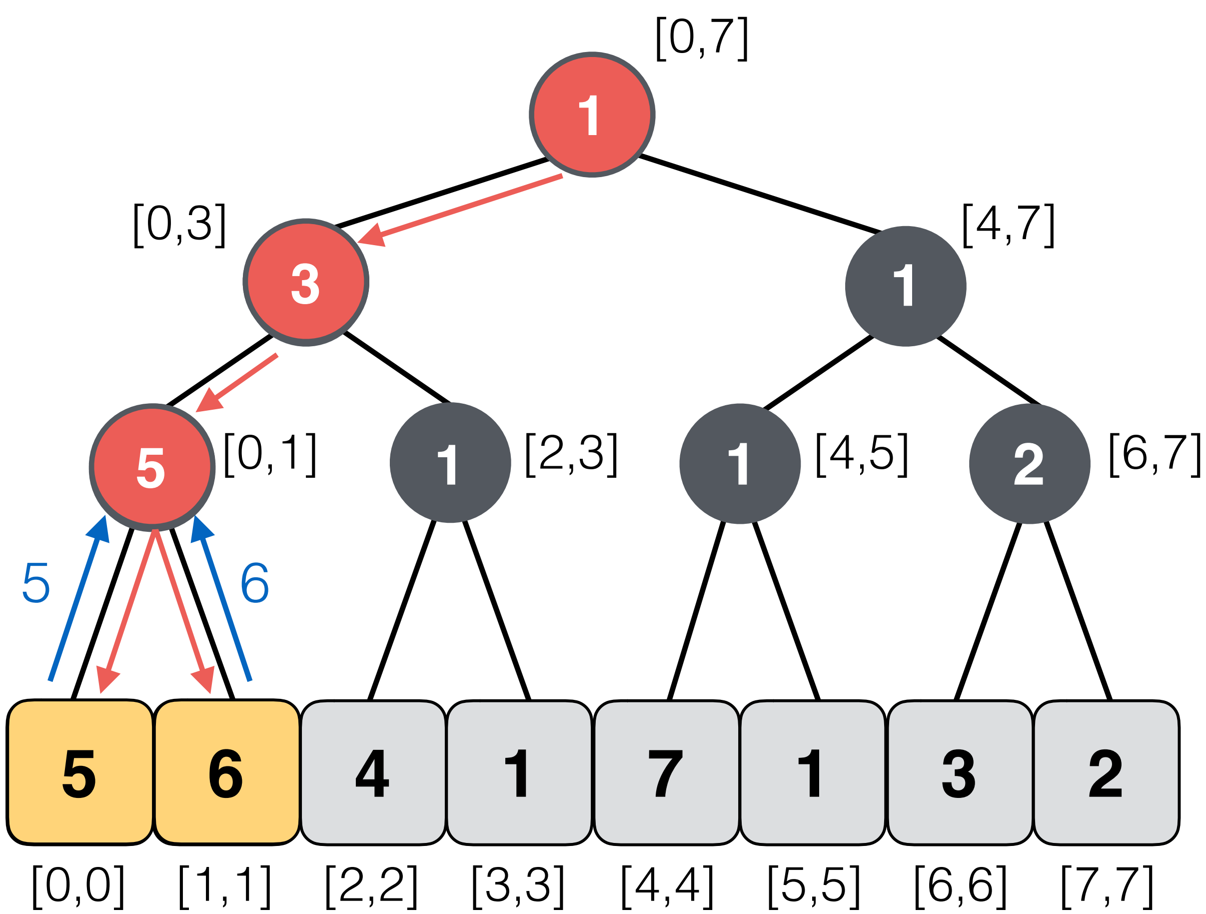
`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



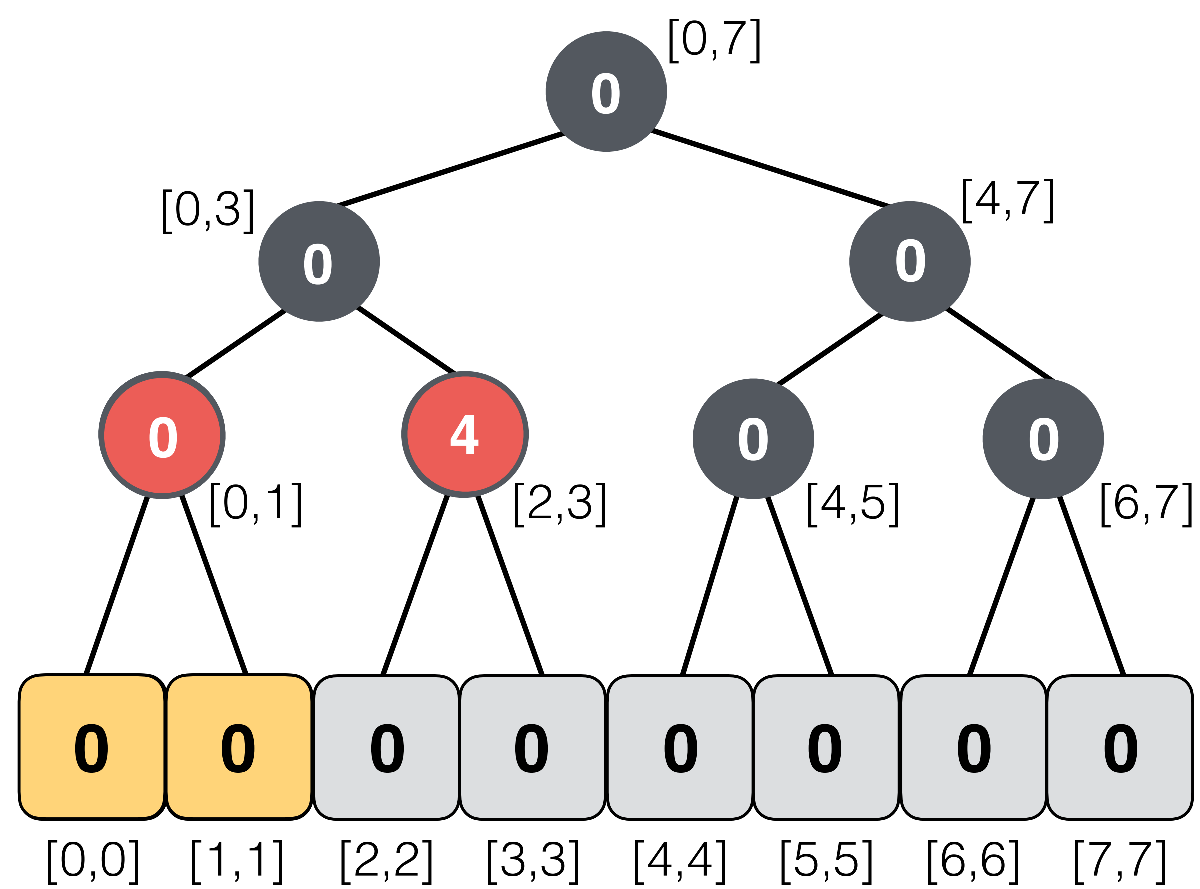
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

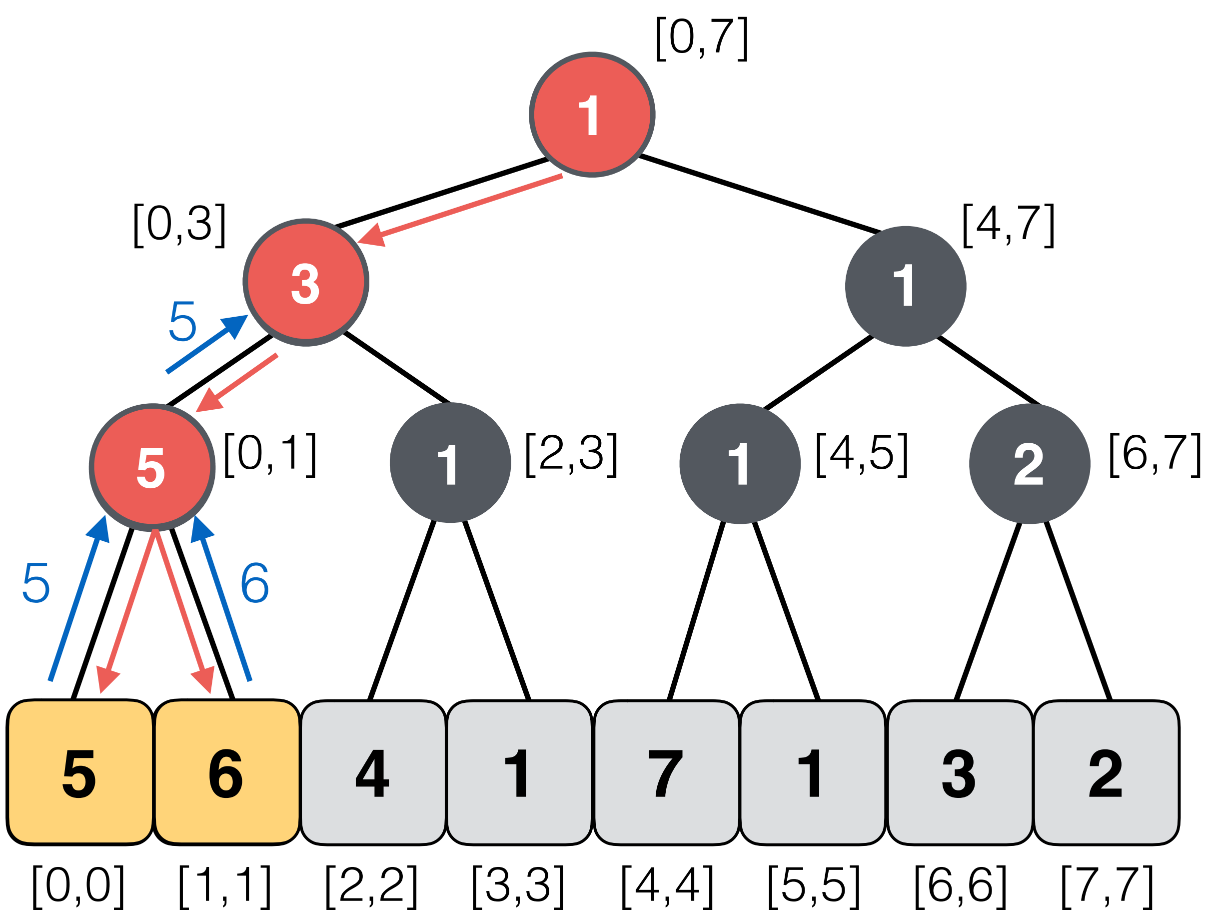


Lazy Tree

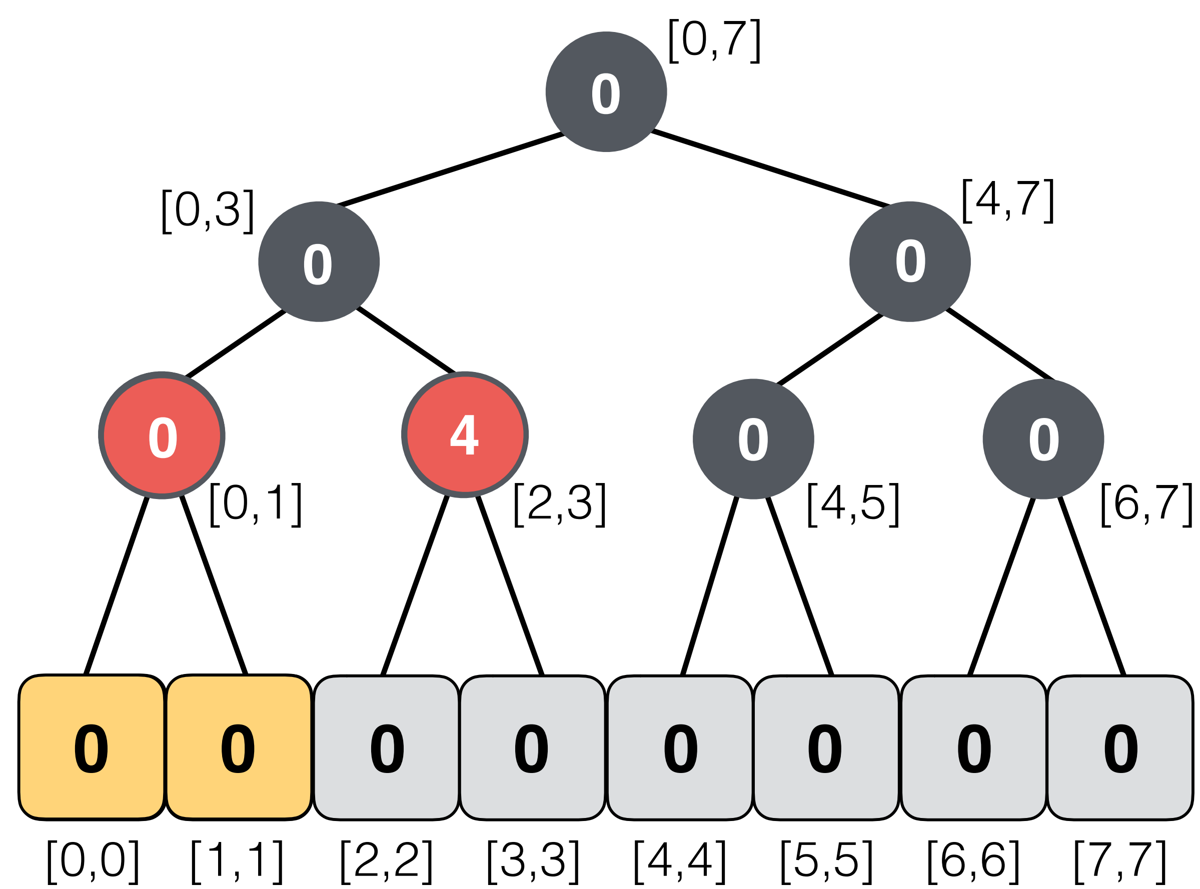
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

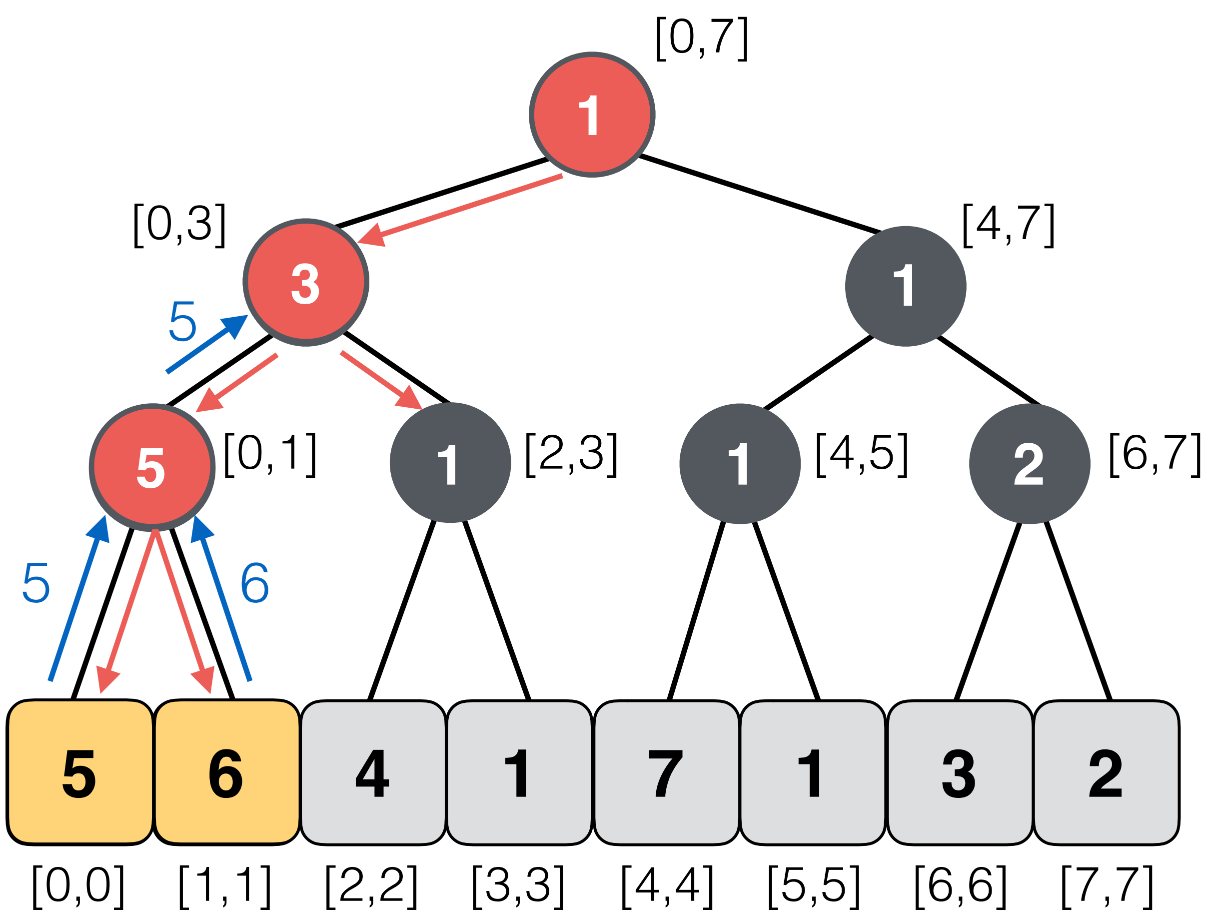


Lazy Tree

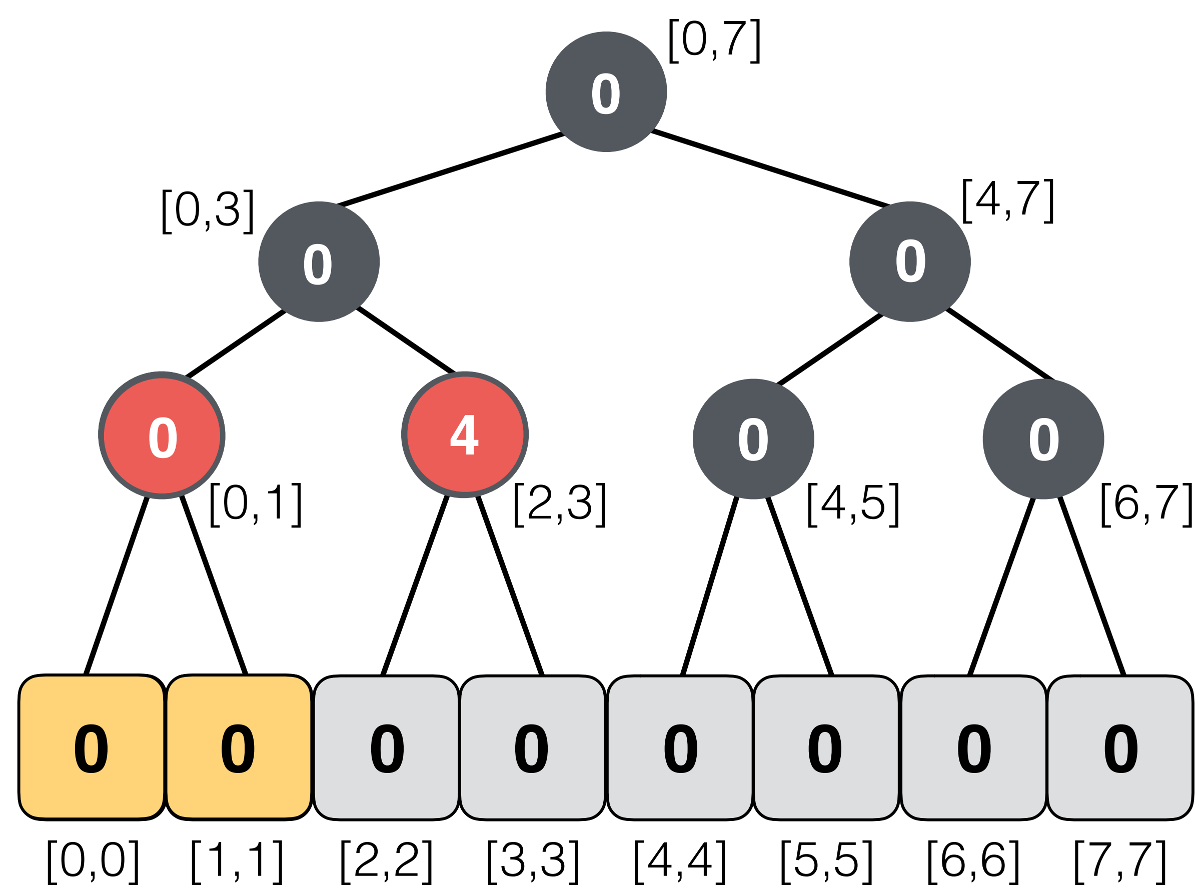
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

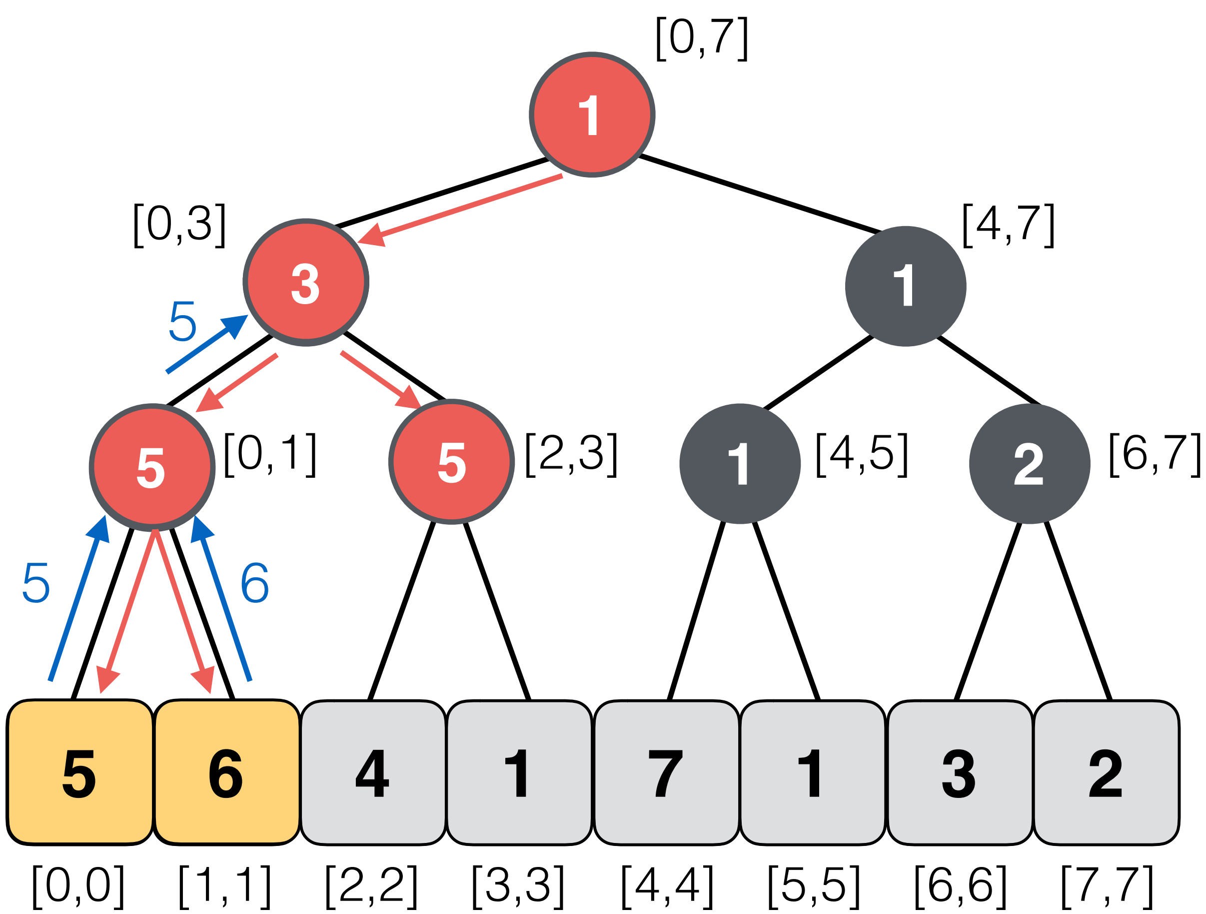


Lazy Tree

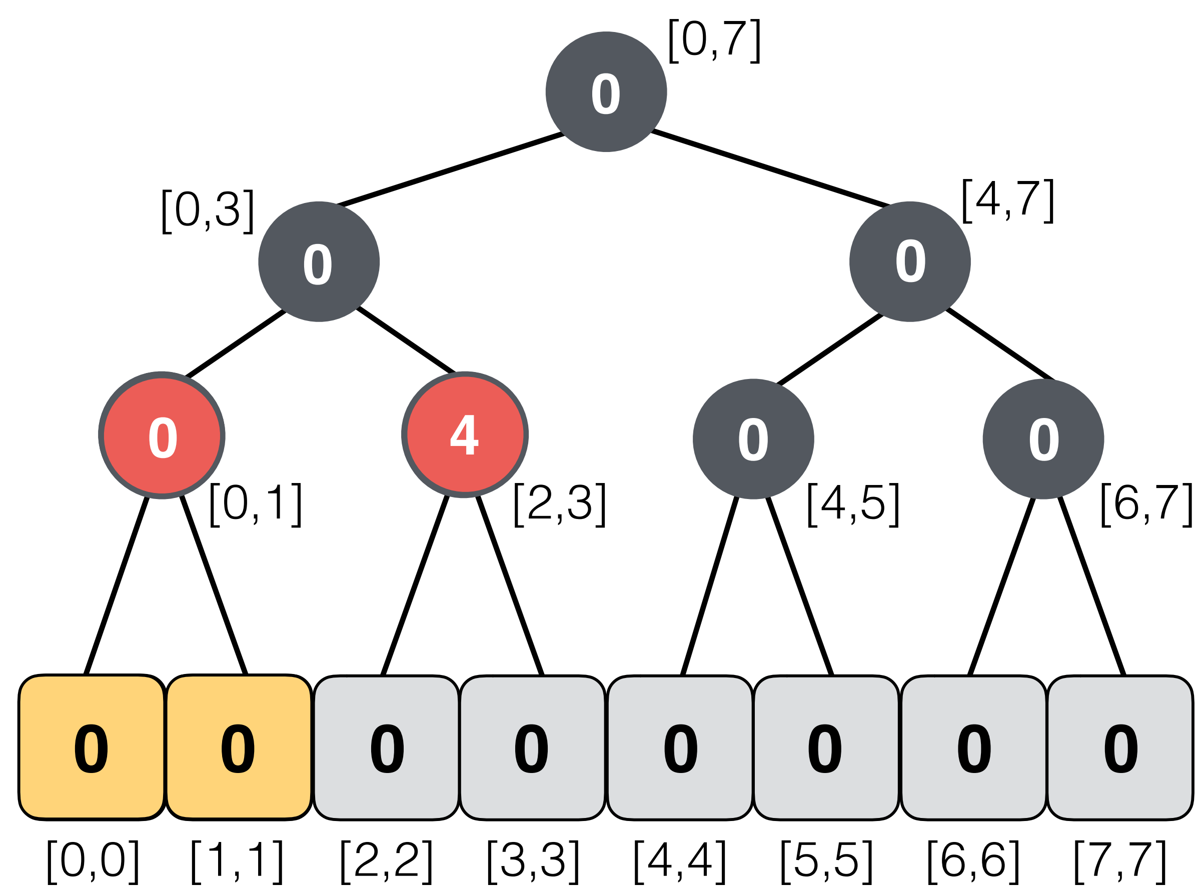
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

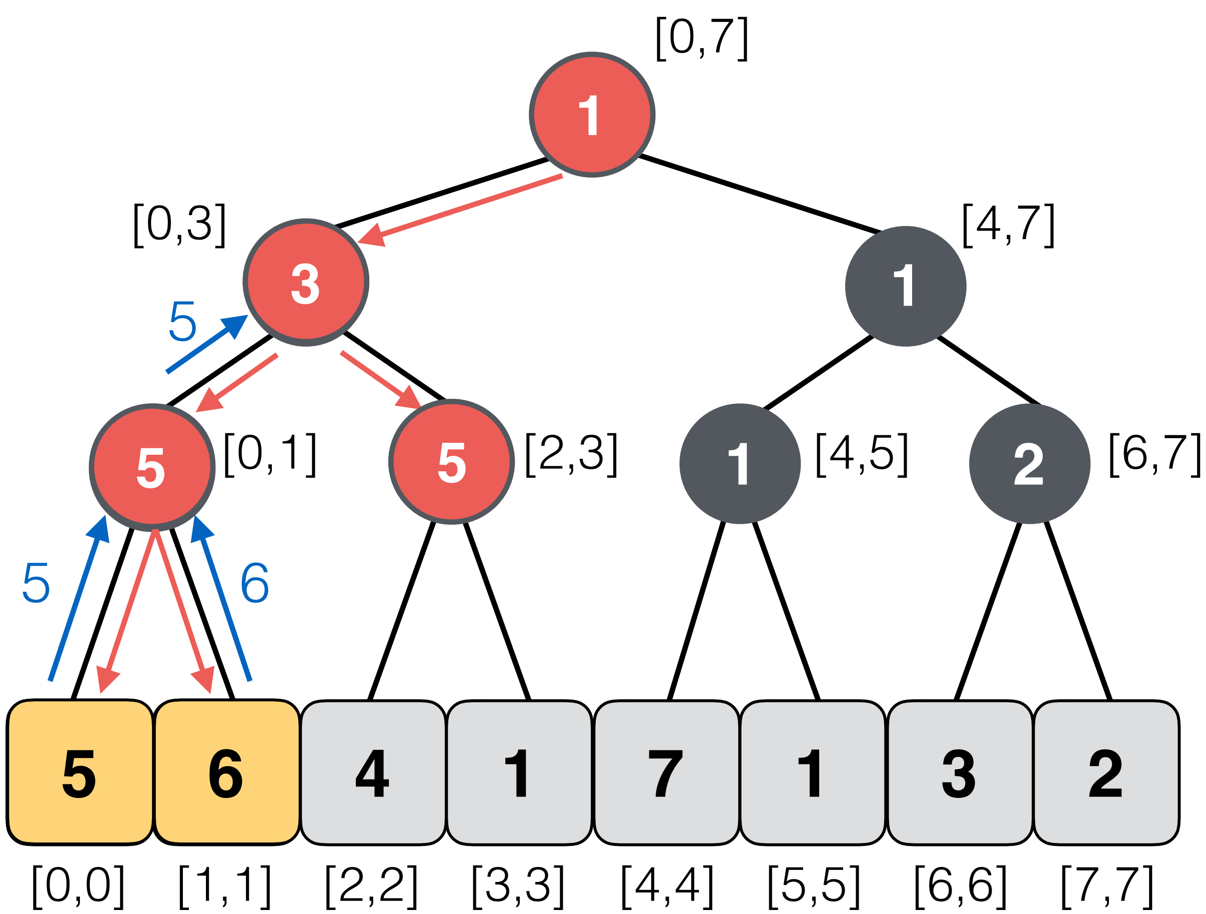


Lazy Tree

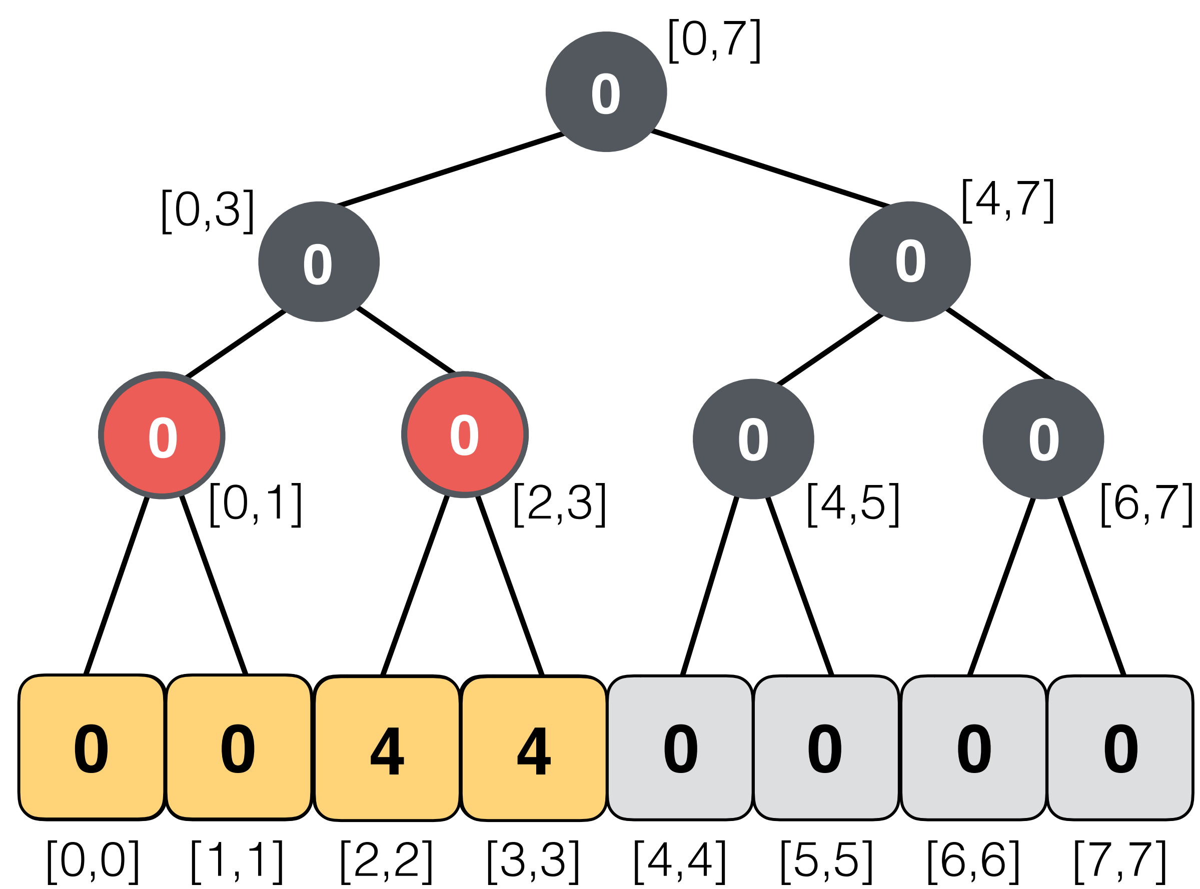
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

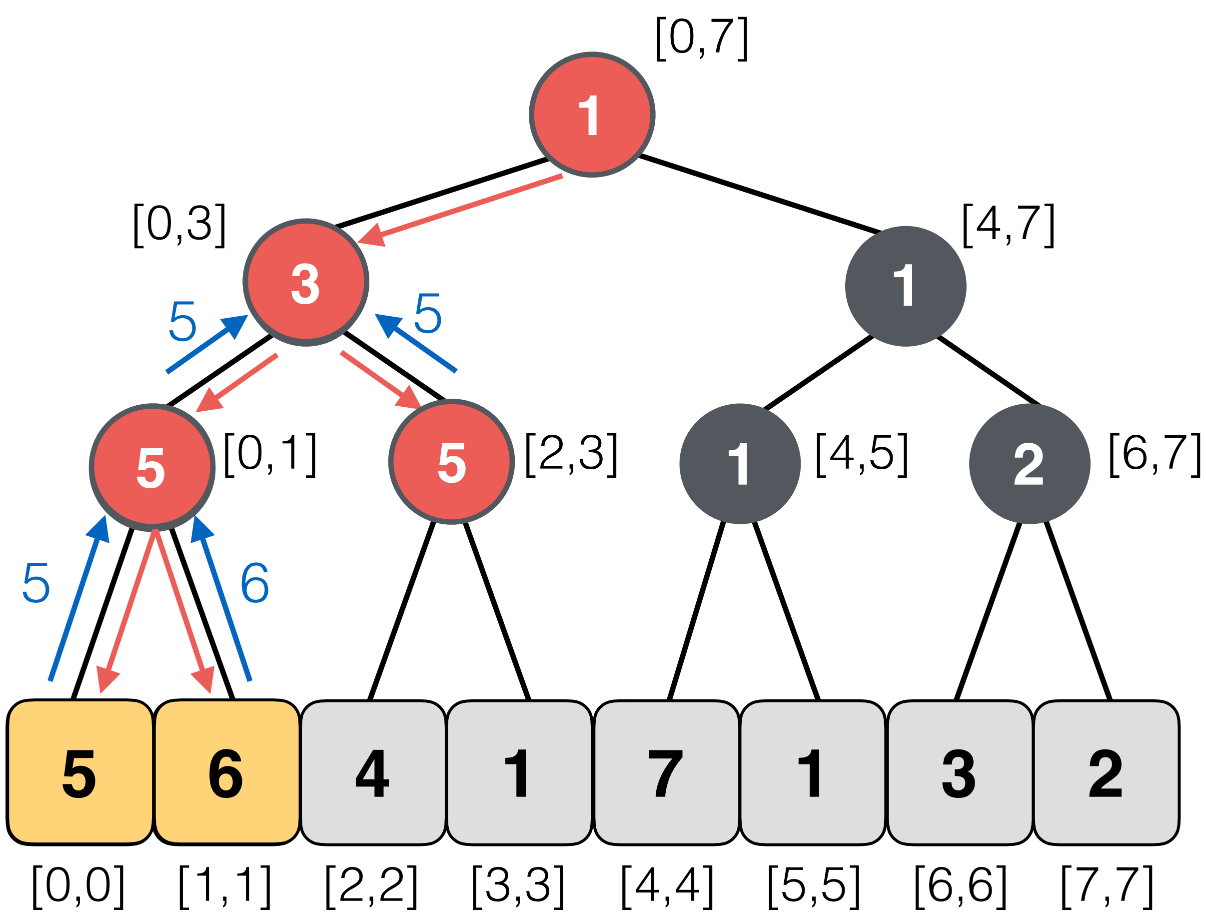


Lazy Tree

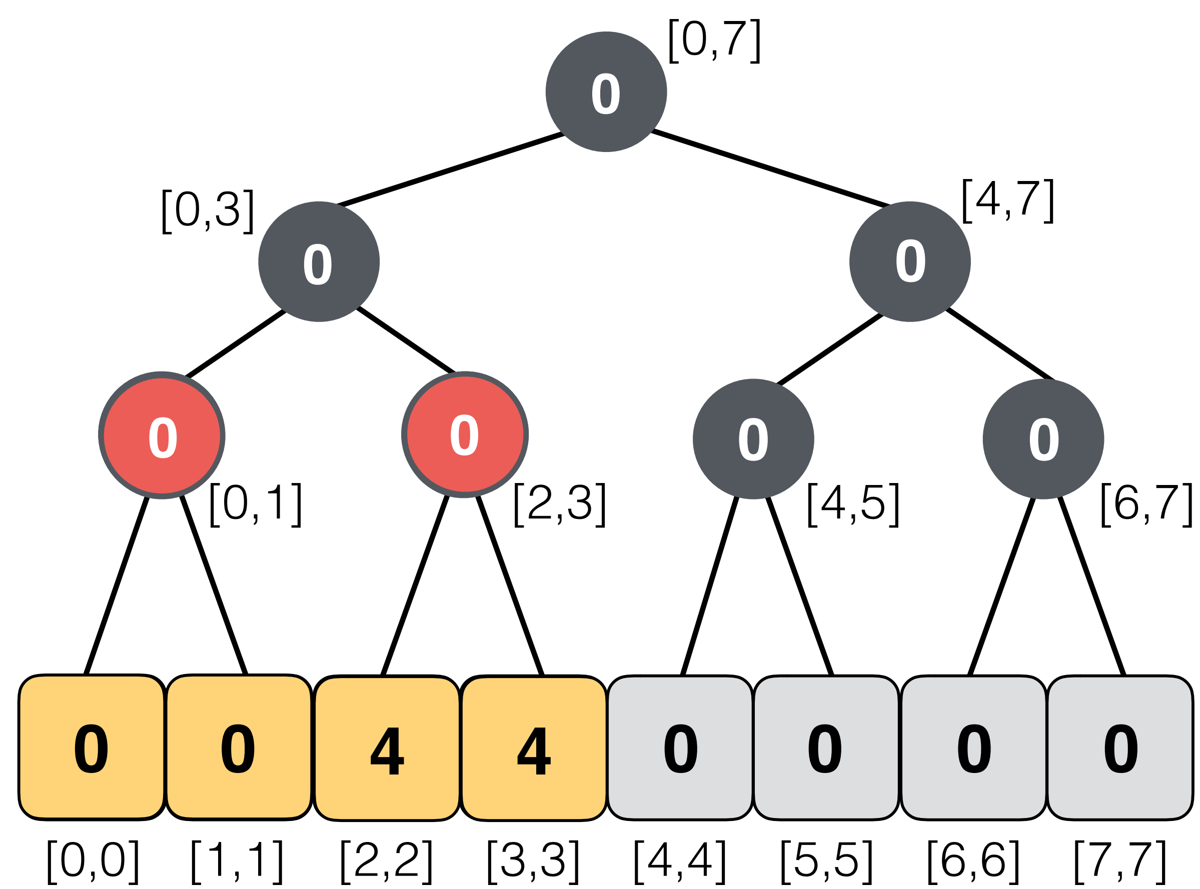
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

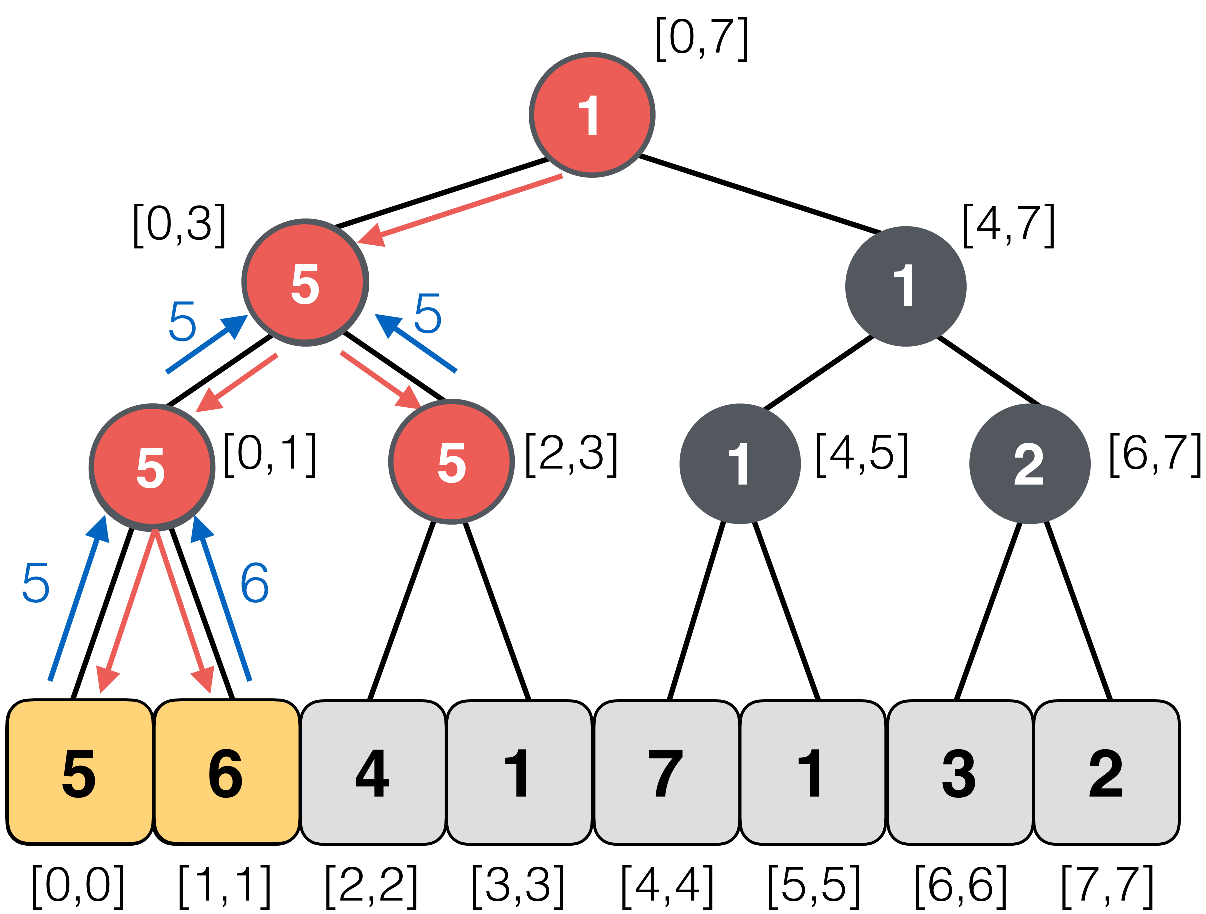


Lazy Tree

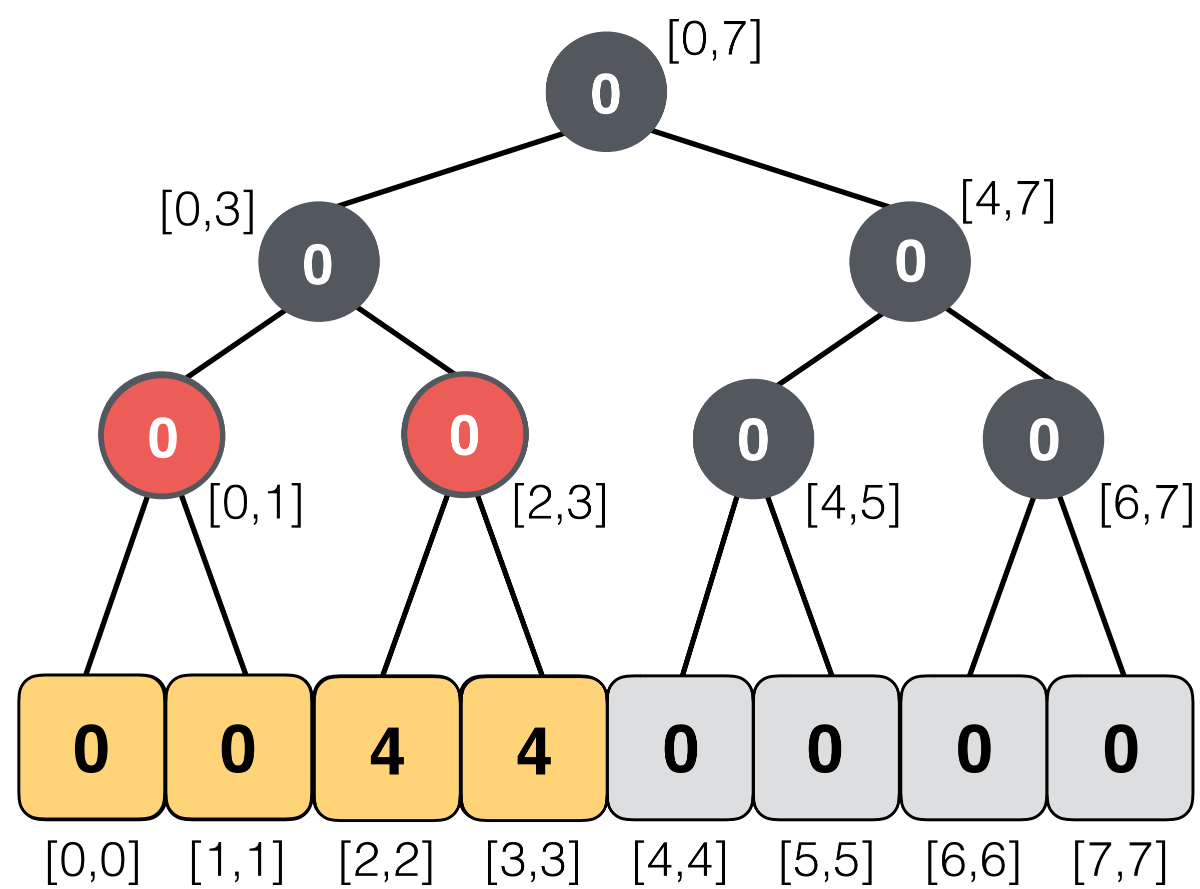
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

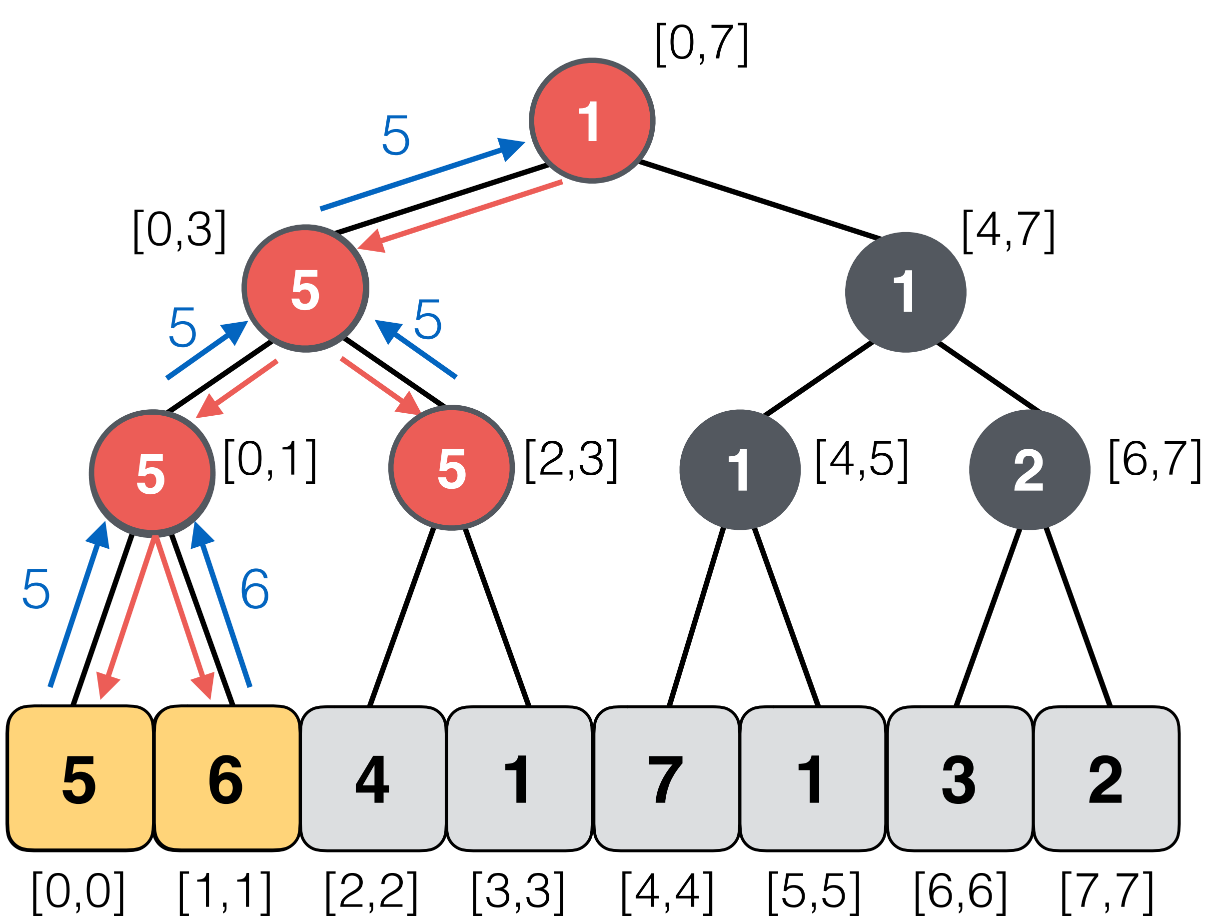


Lazy Tree

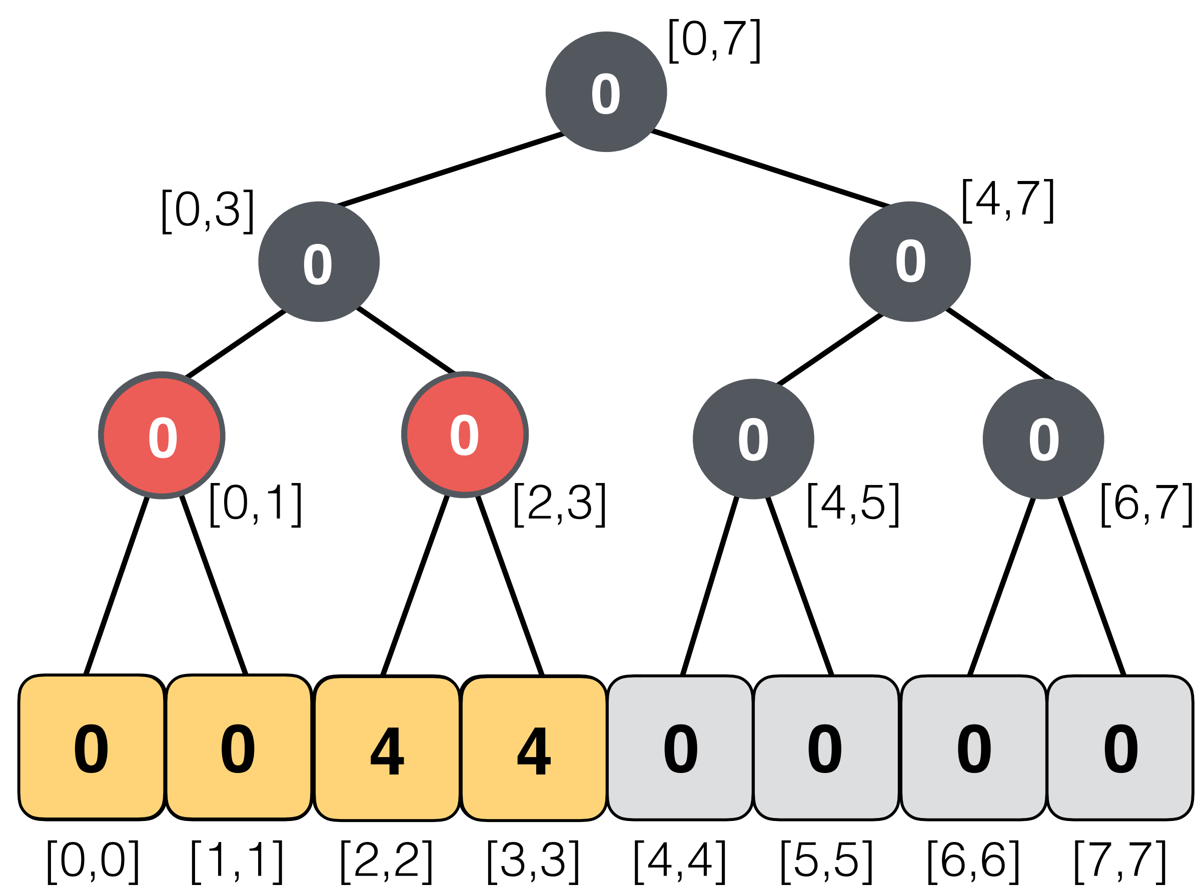
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

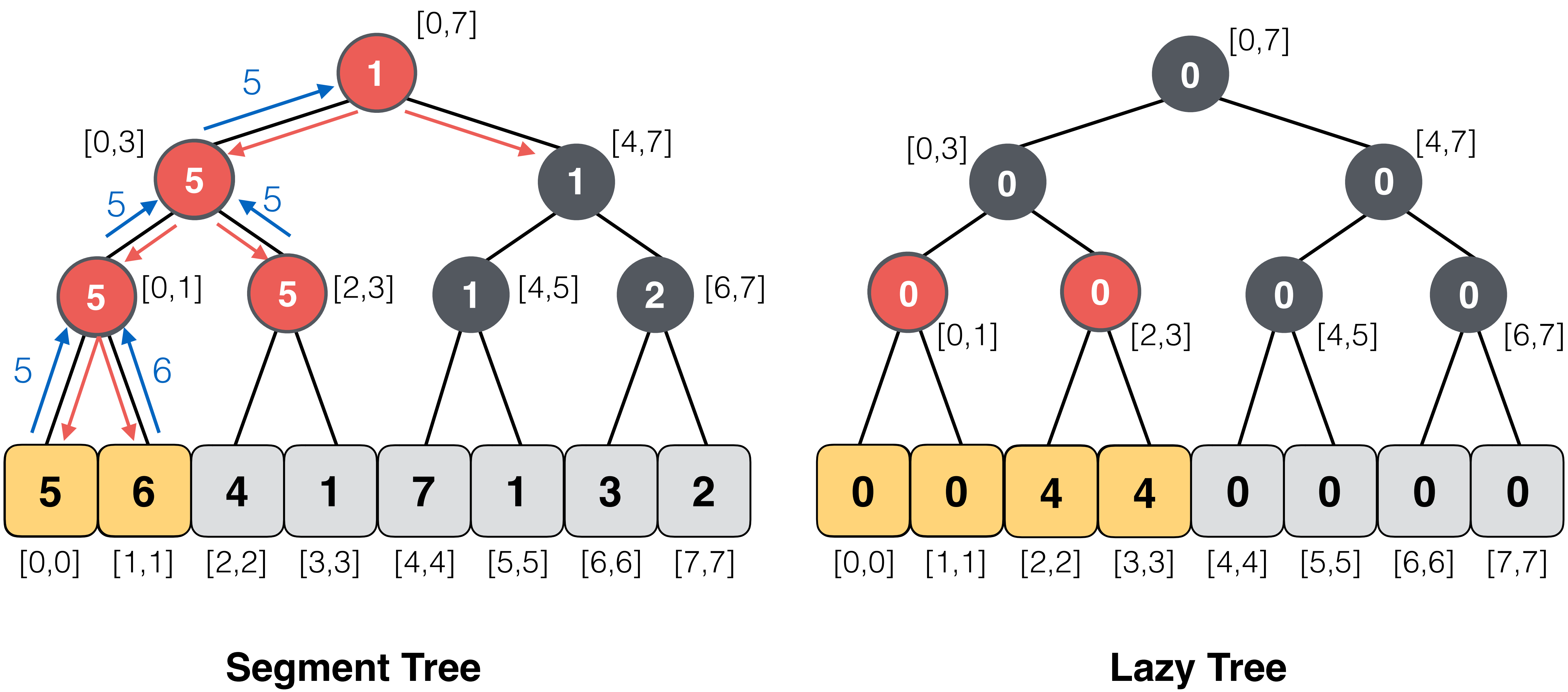


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

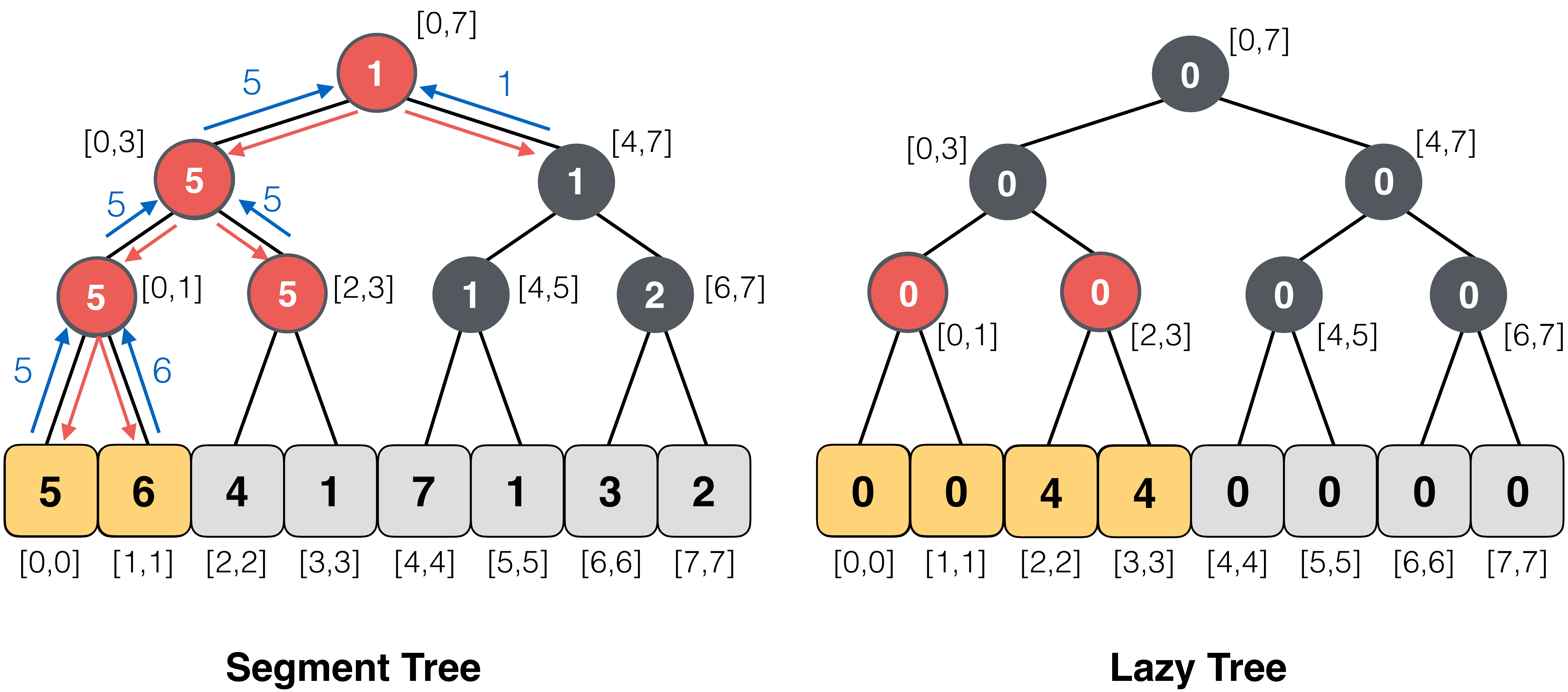
`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

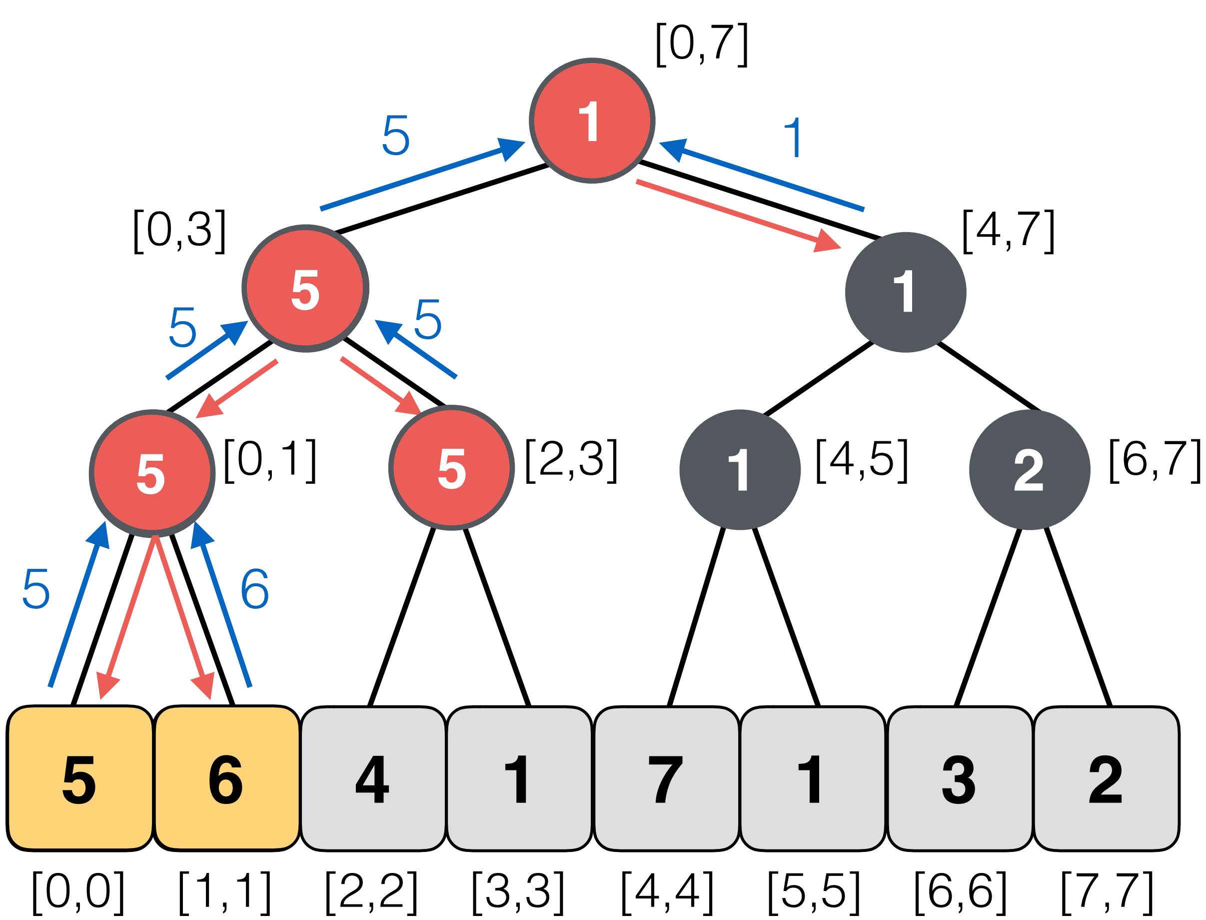
`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range(0,0,2)}}$
`rmq`(3,5) = ?



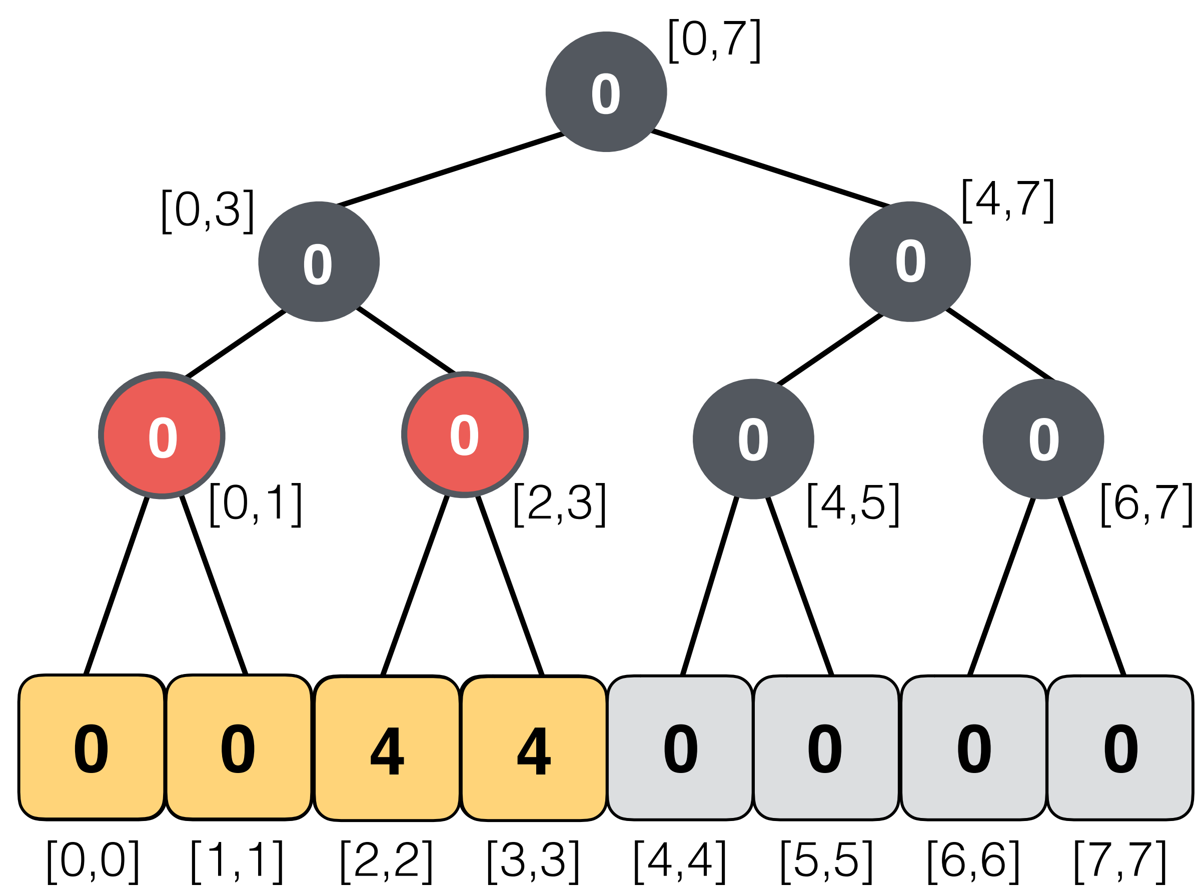
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

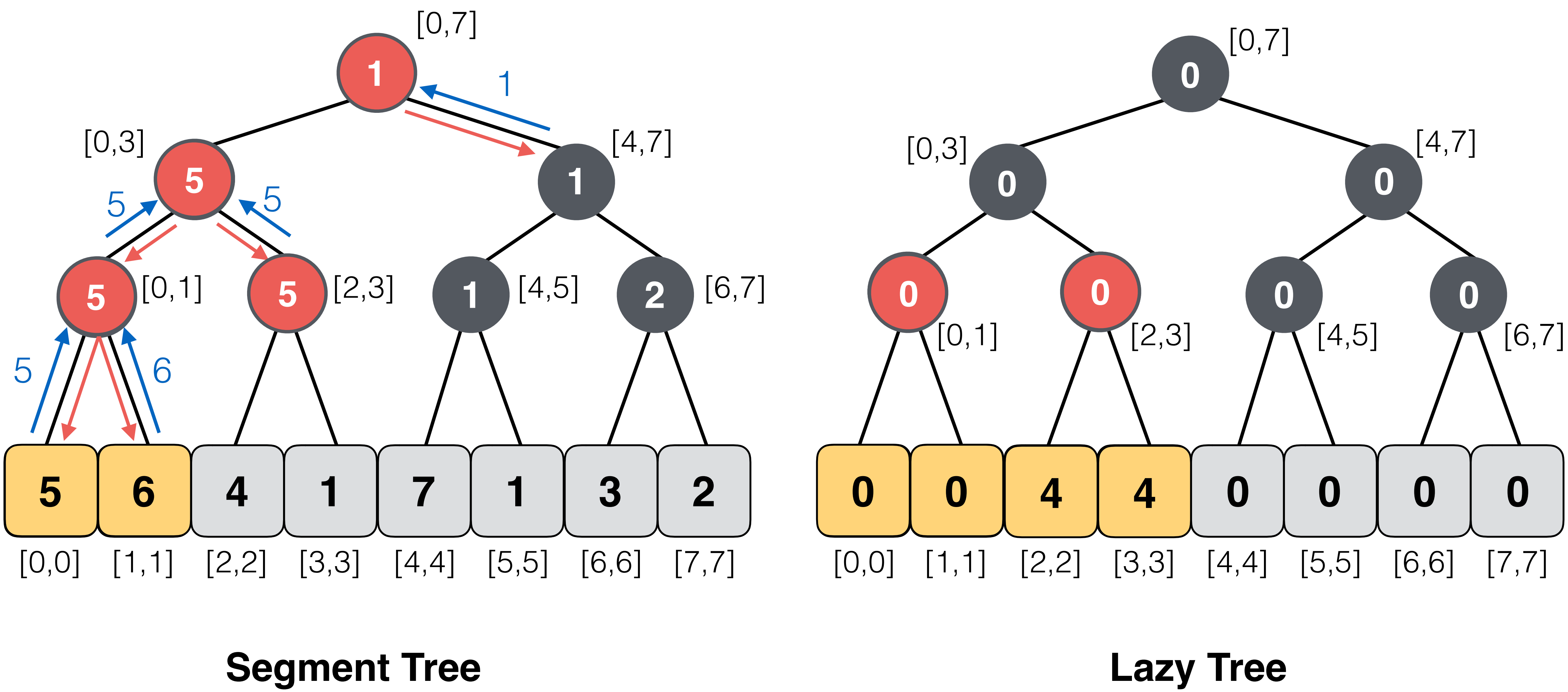


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

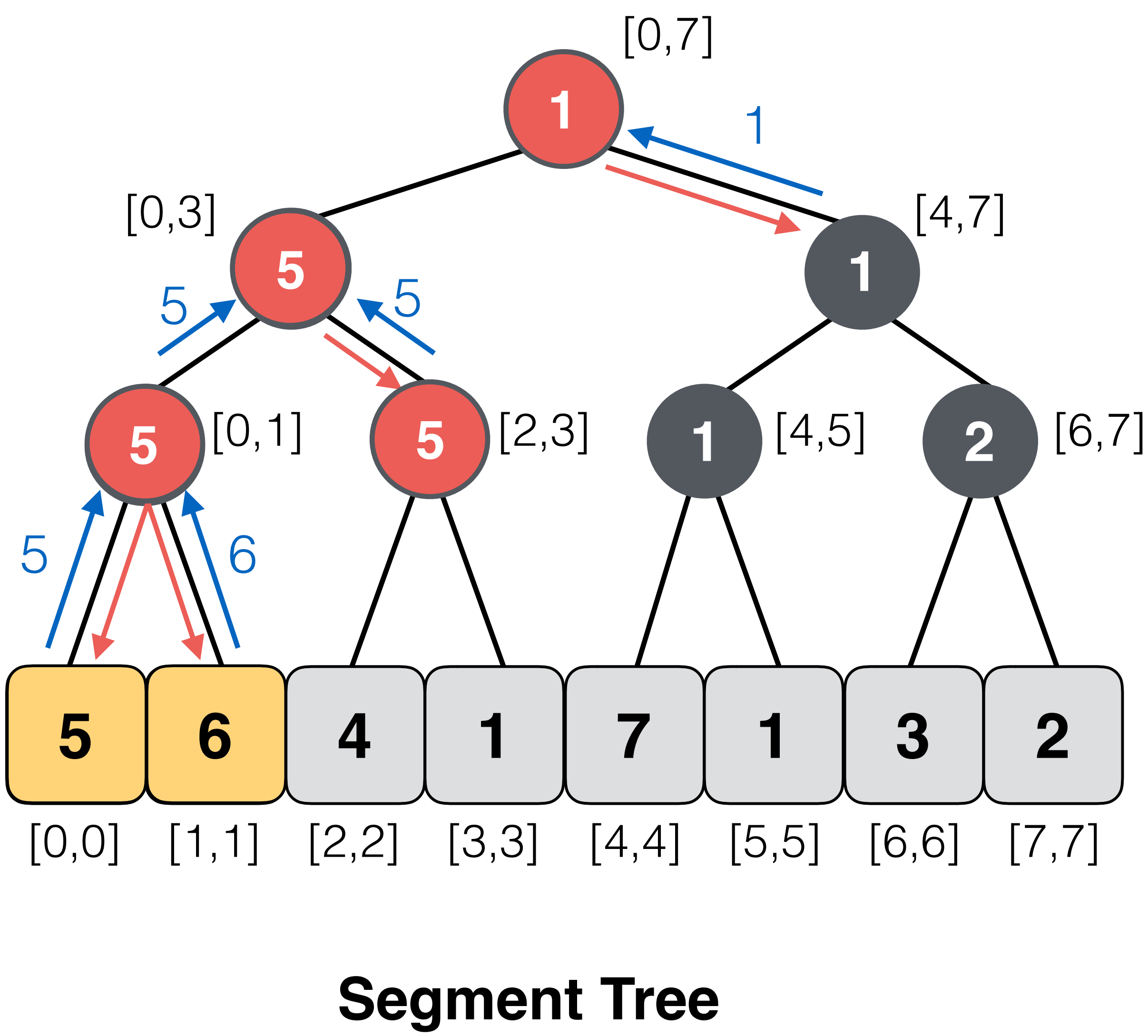
`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?

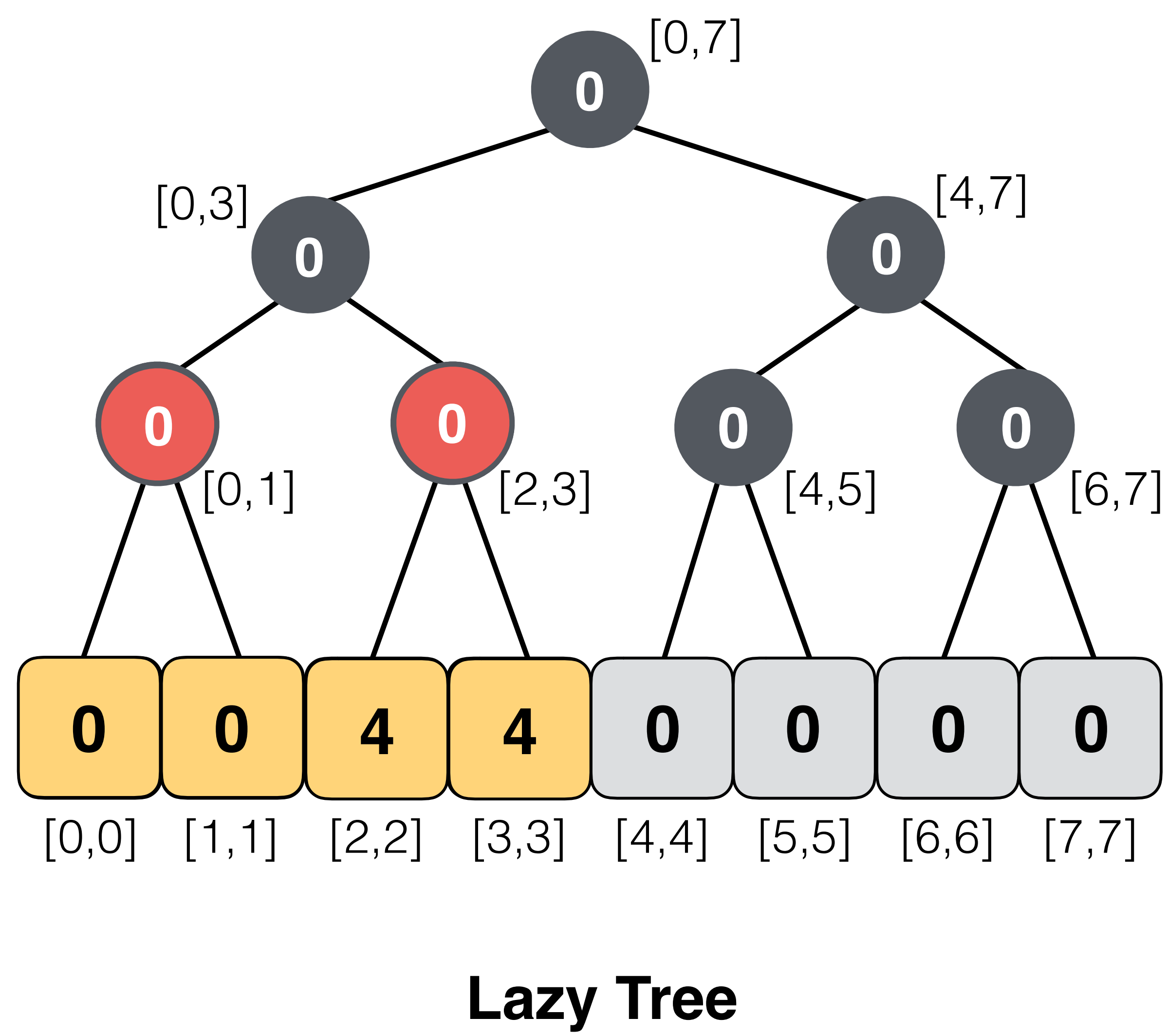


Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?

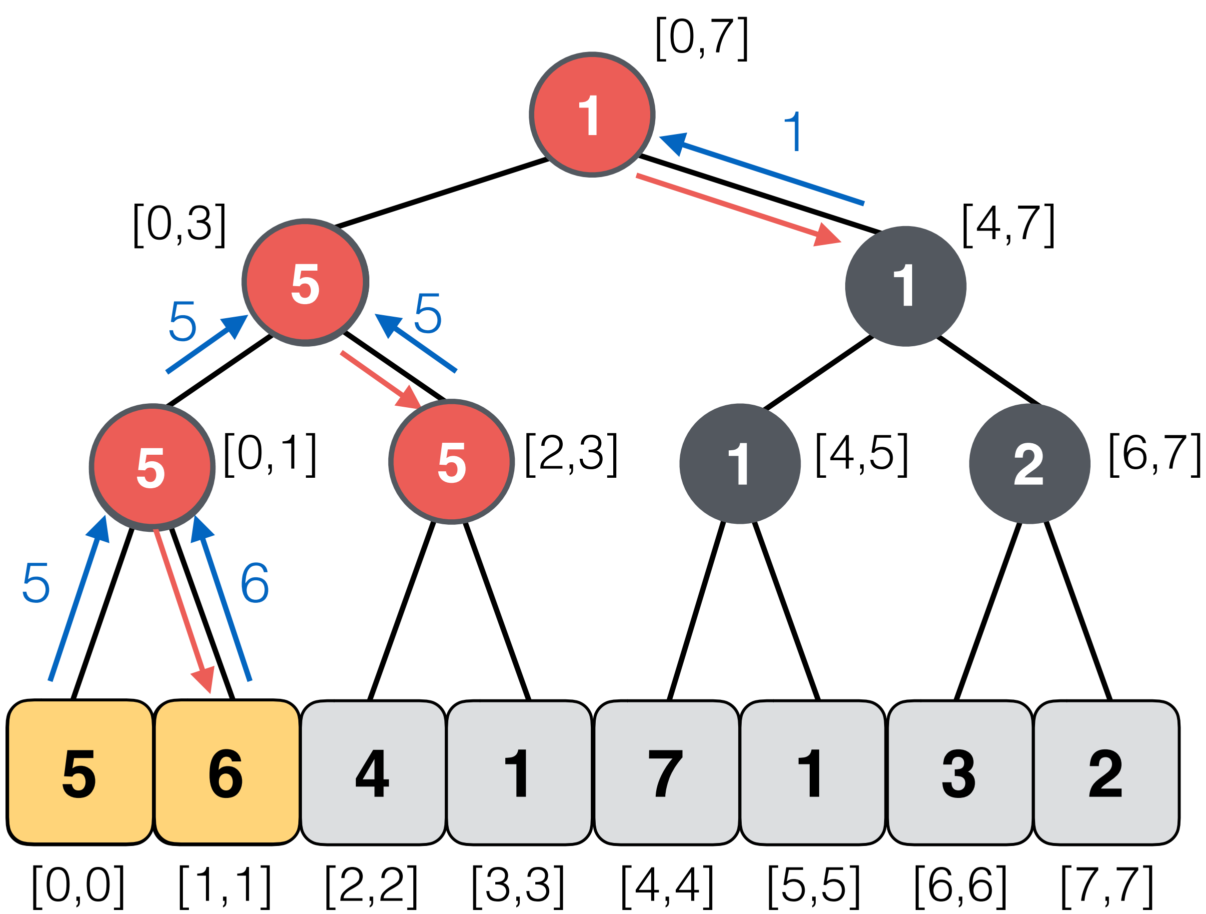




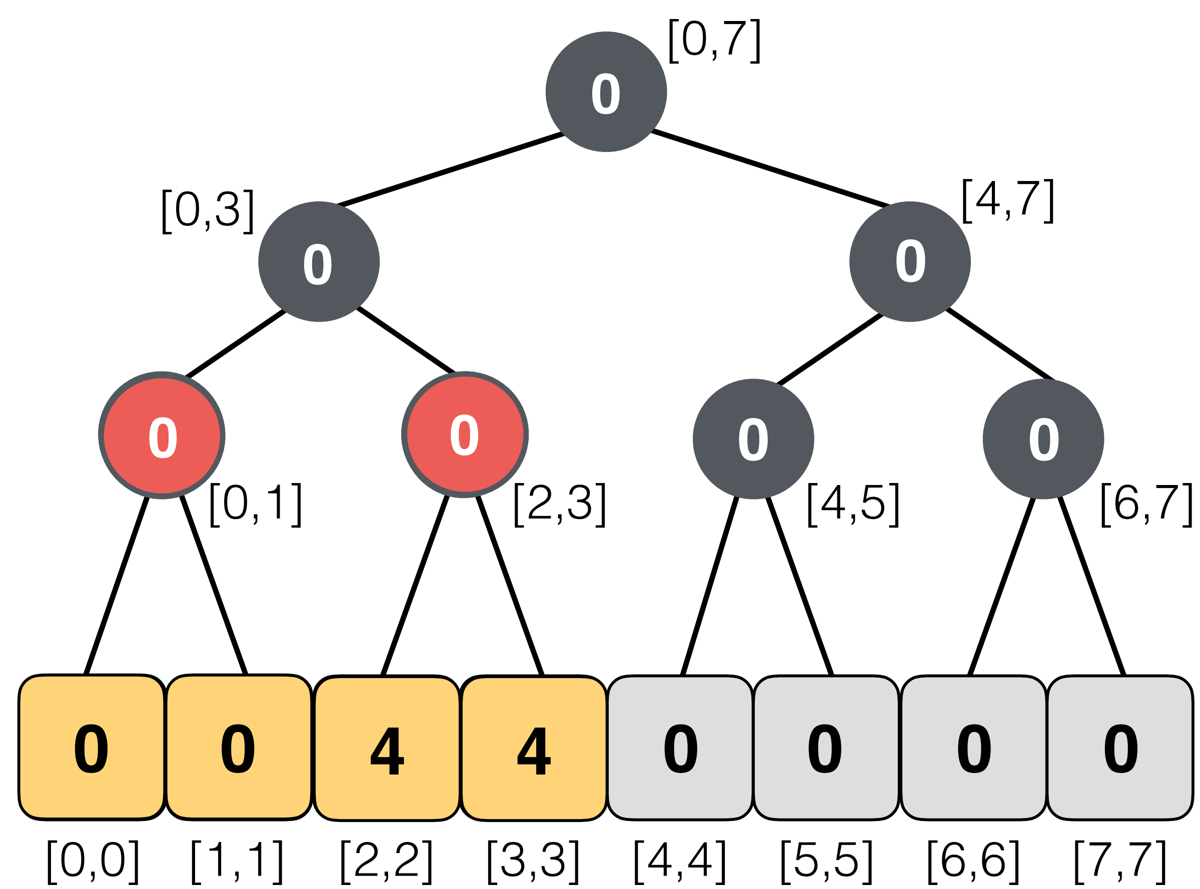
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

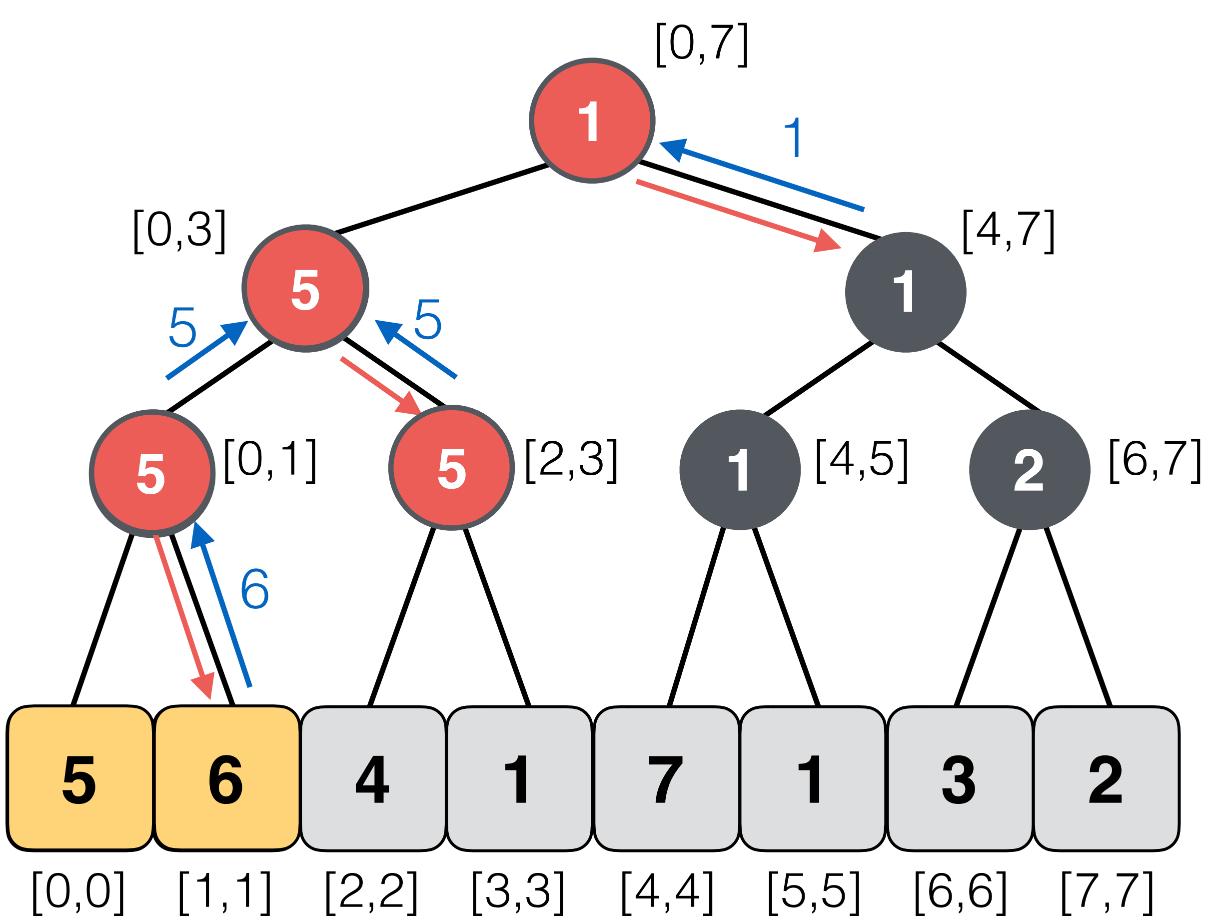


Lazy Tree

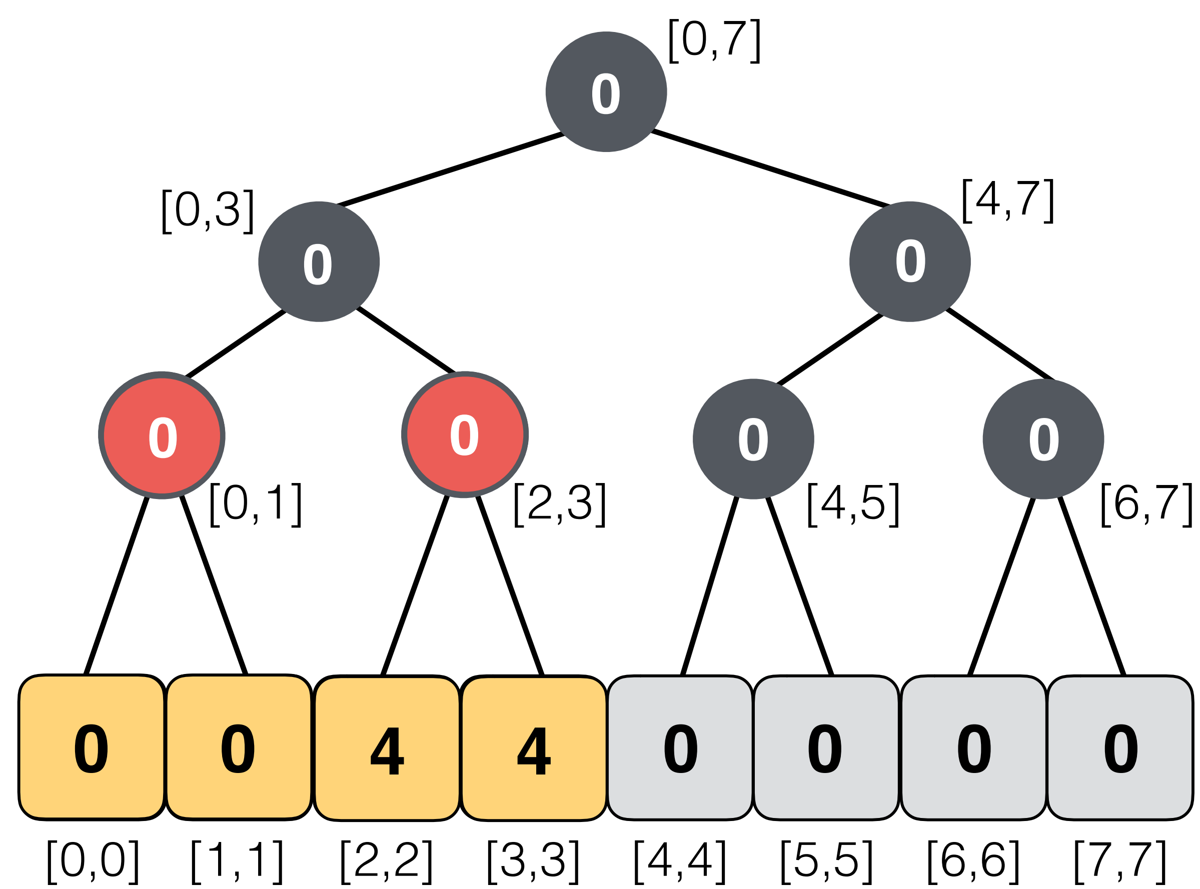
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

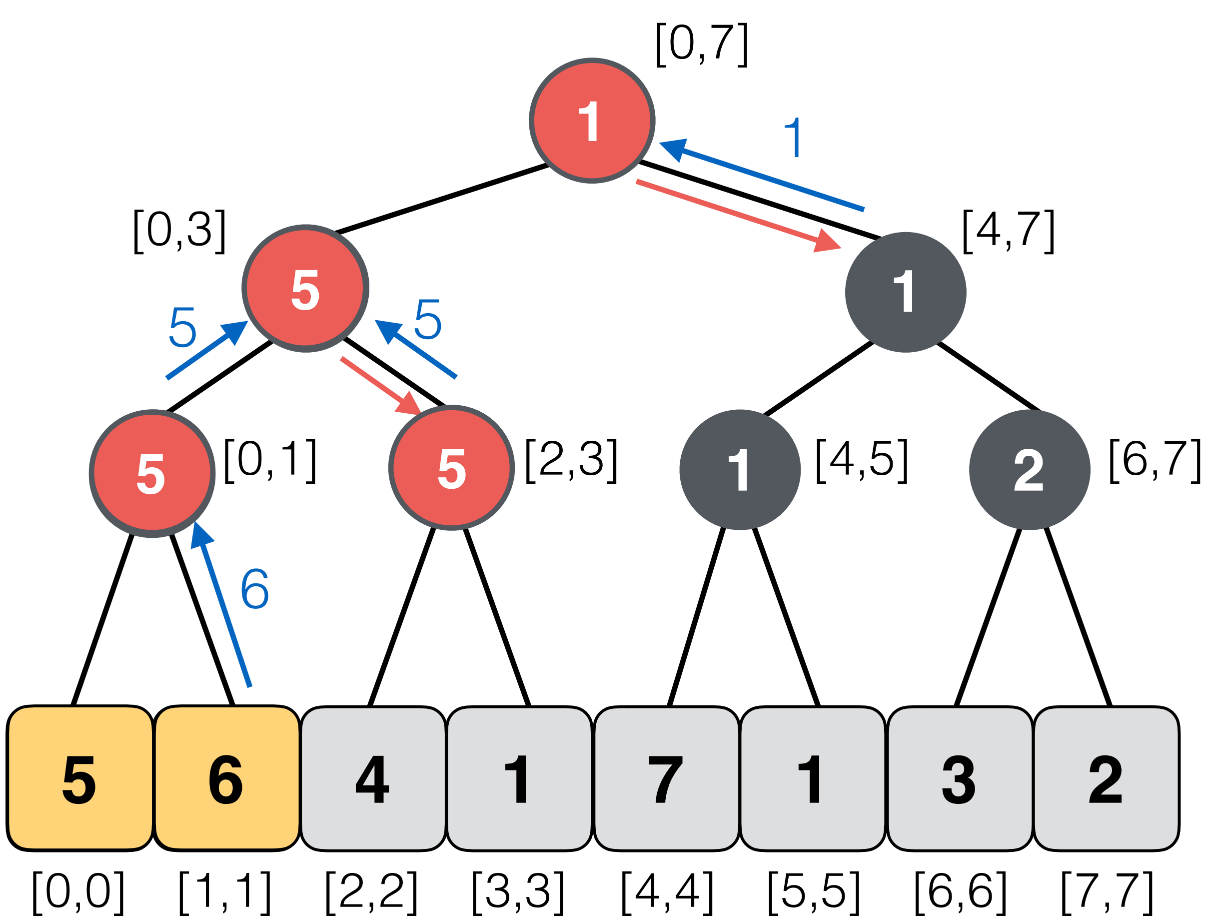


Lazy Tree

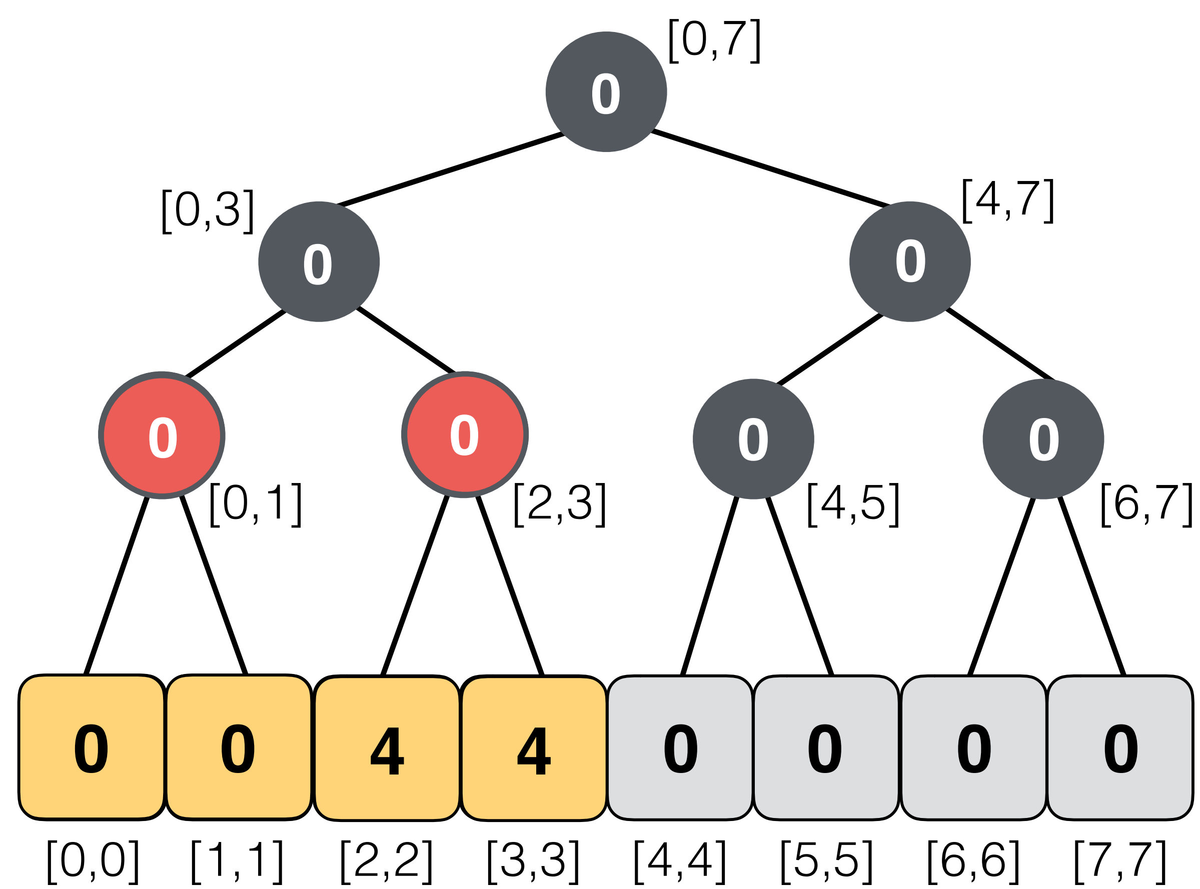
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

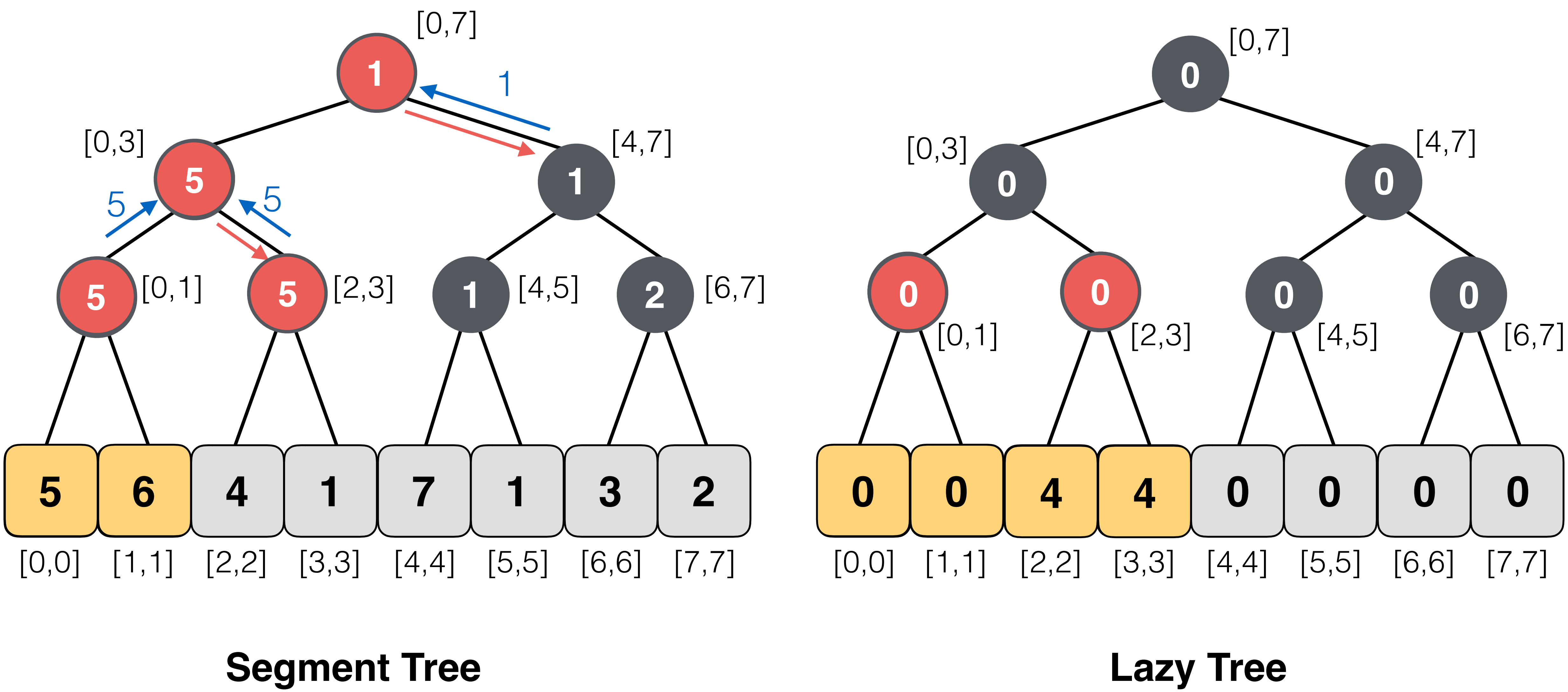


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

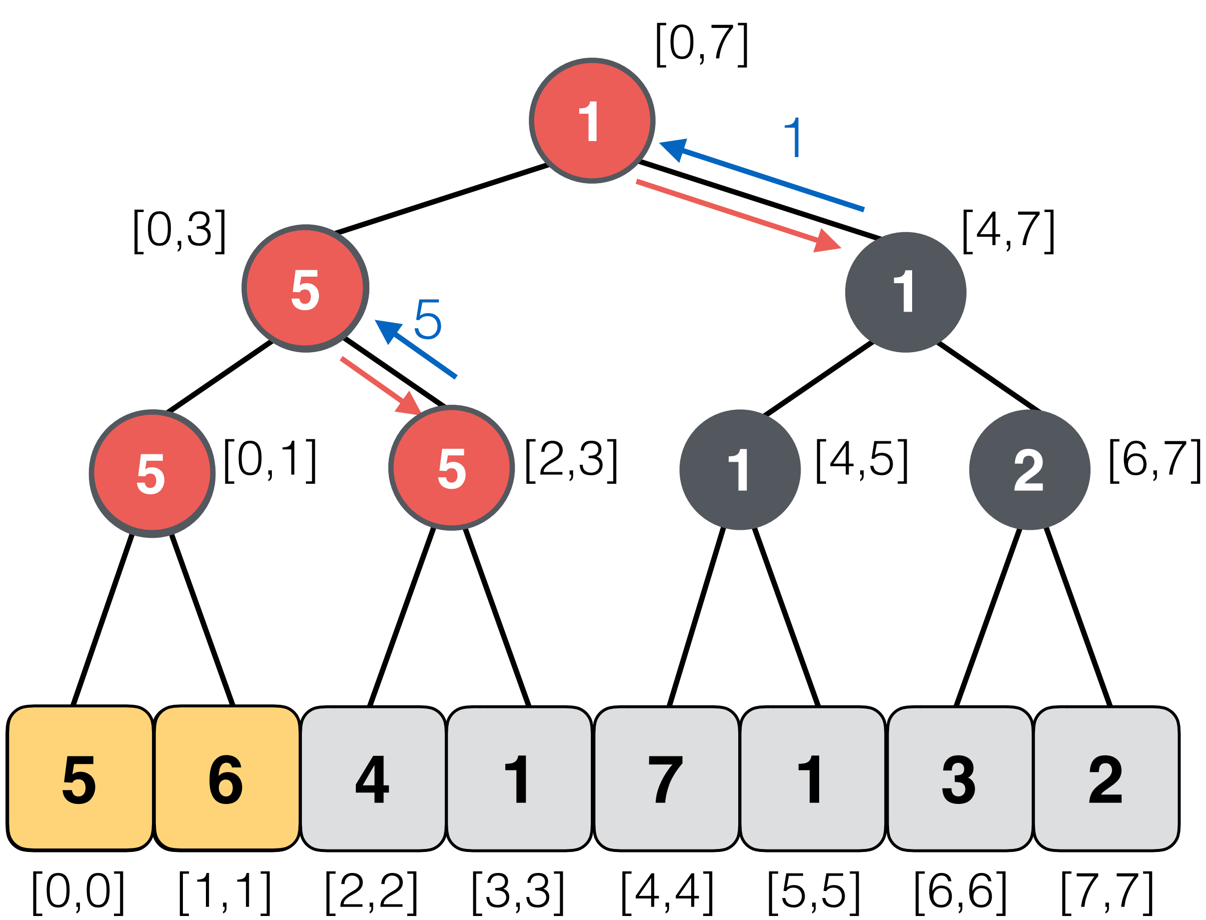
`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



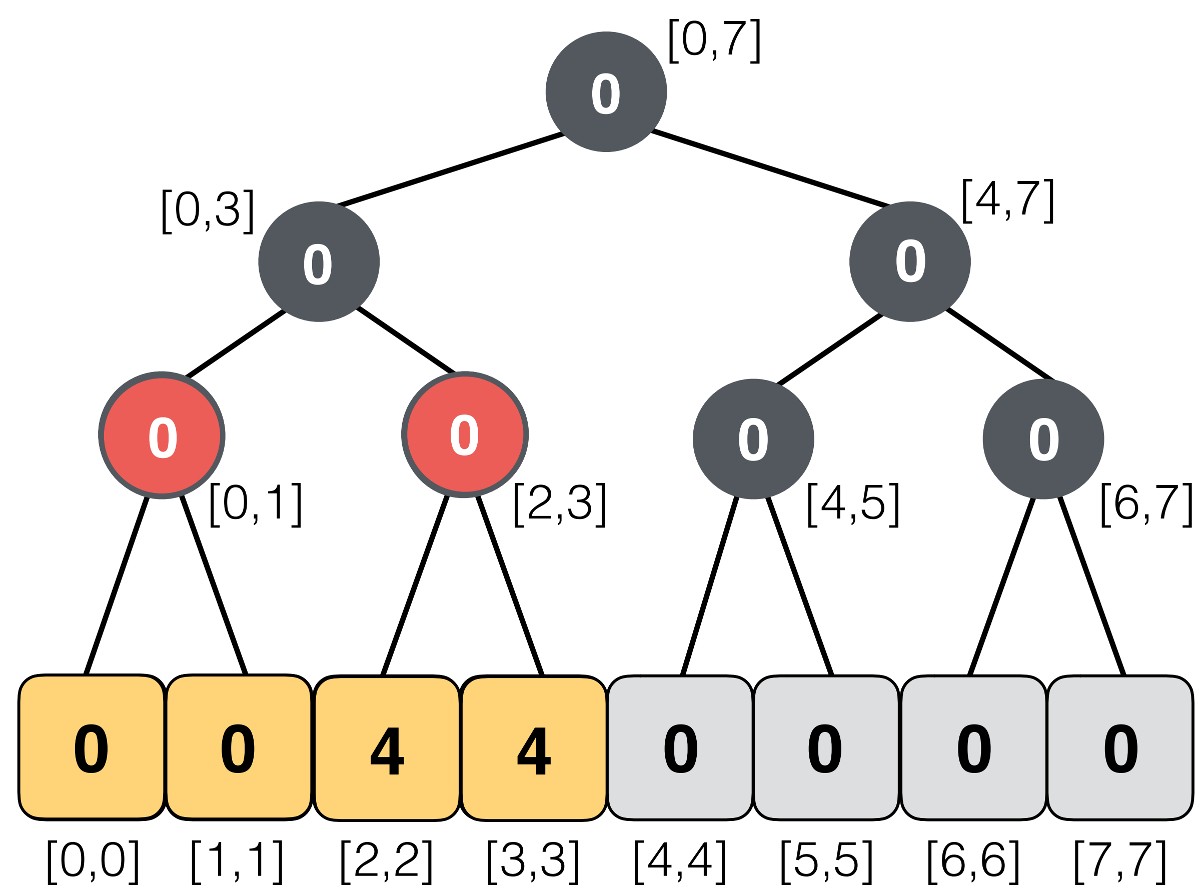
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

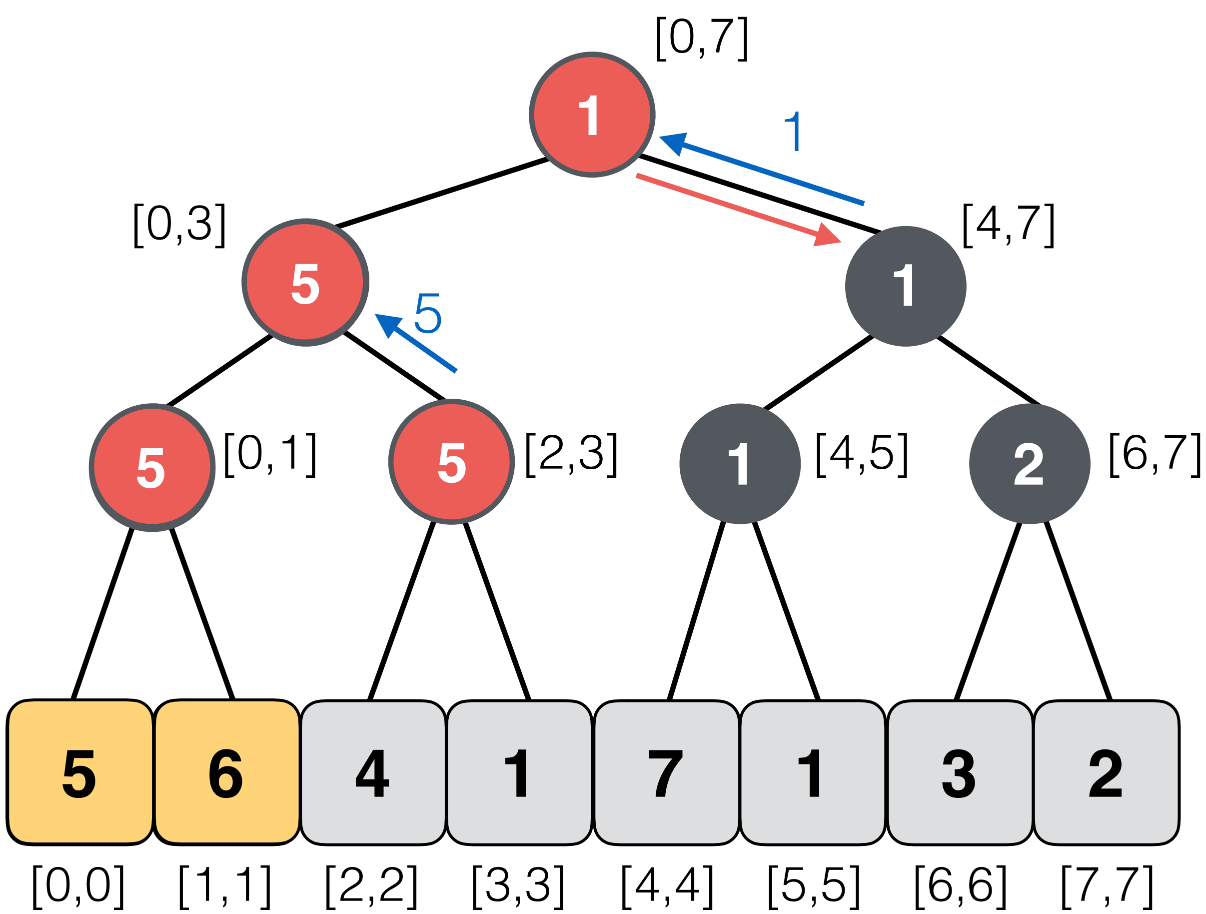


Lazy Tree

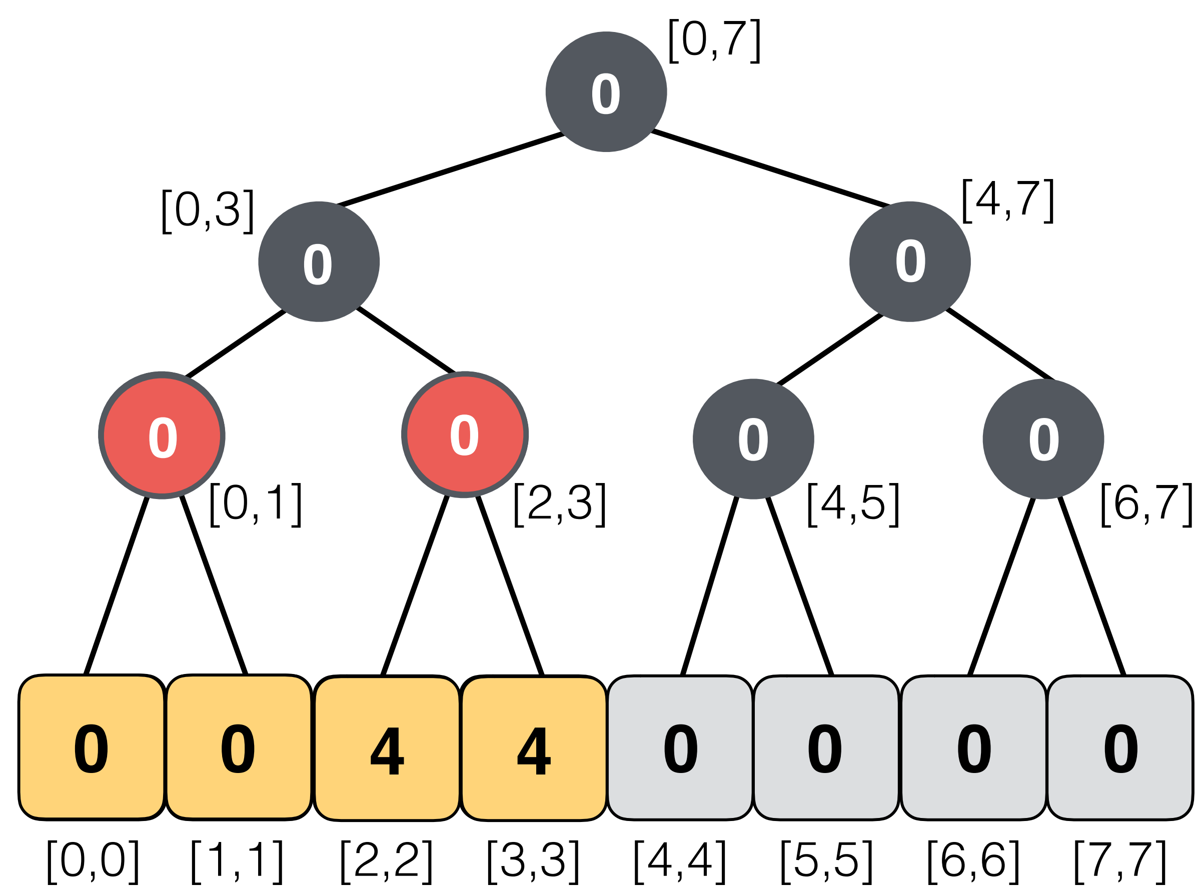
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

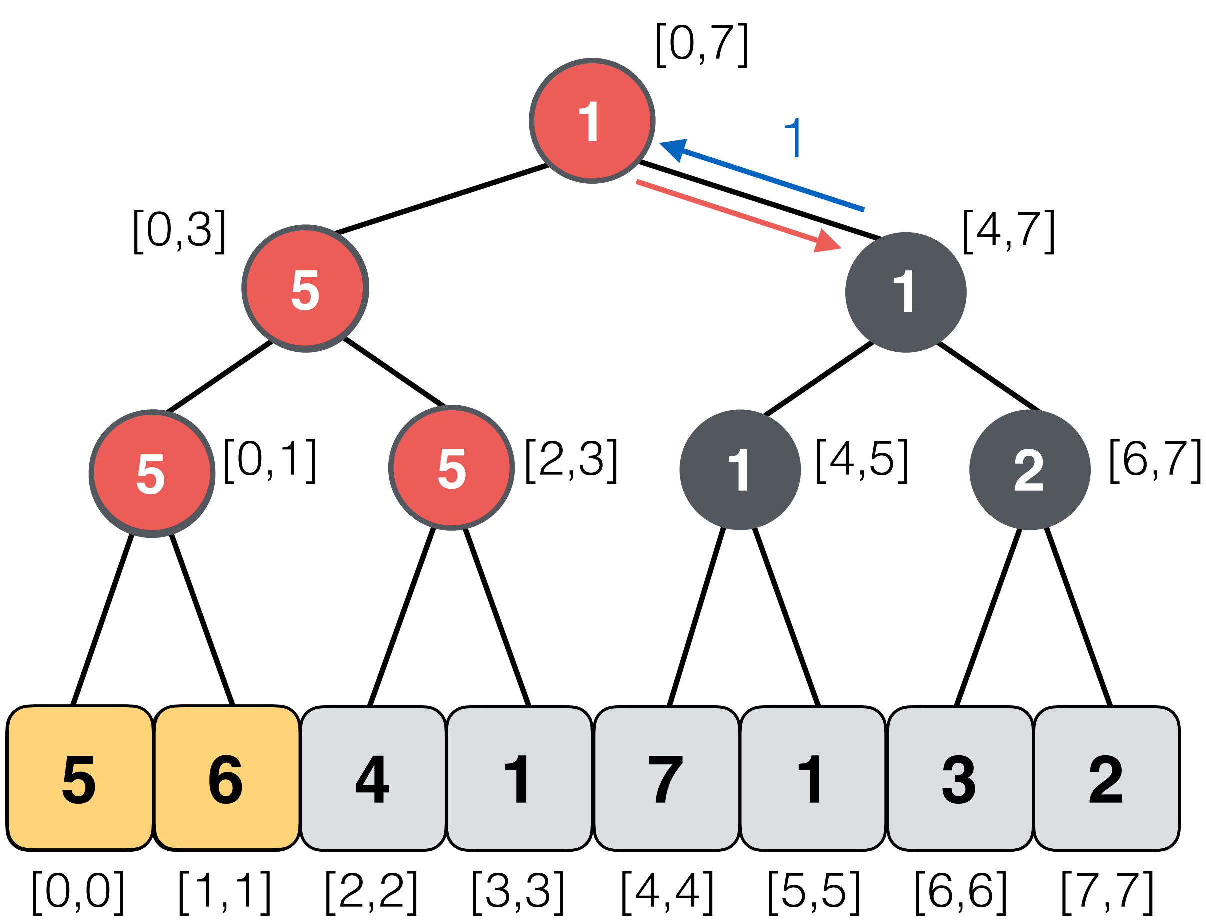


Lazy Tree

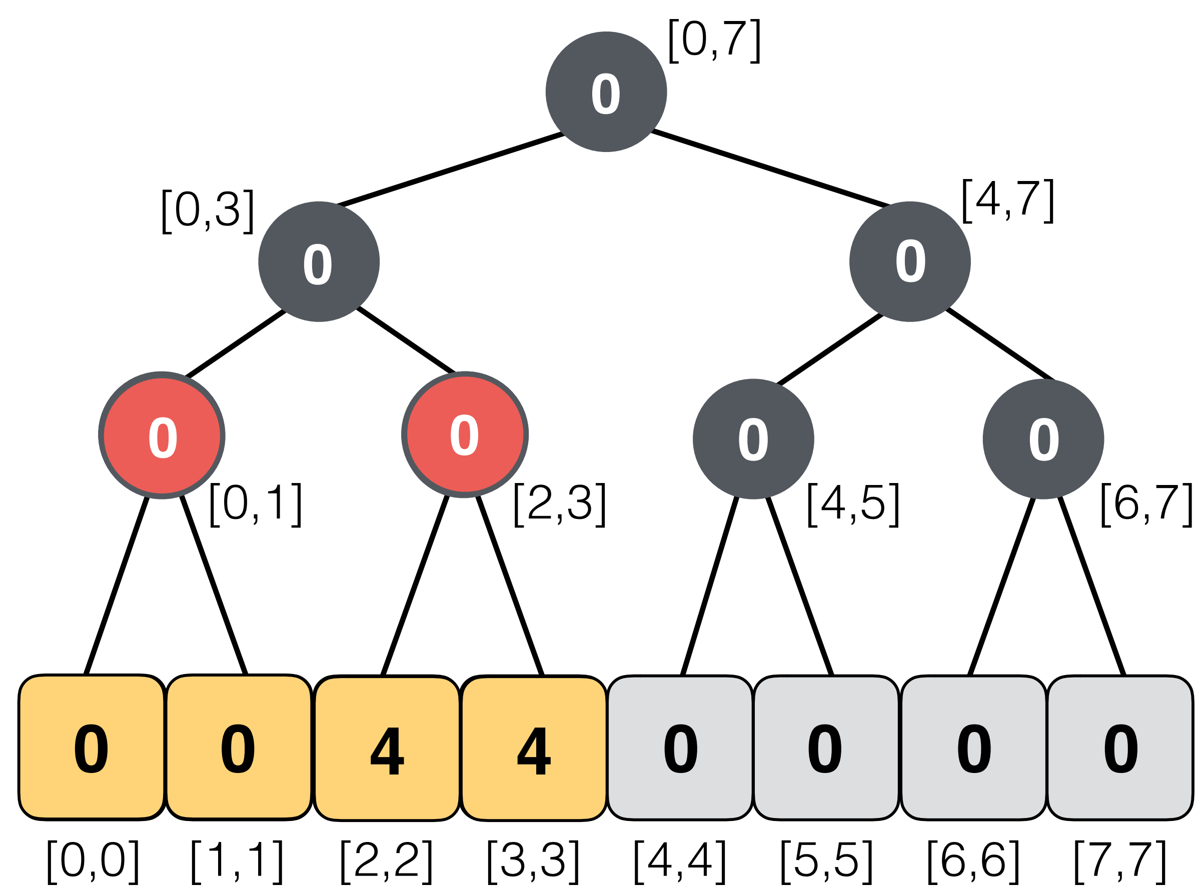
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

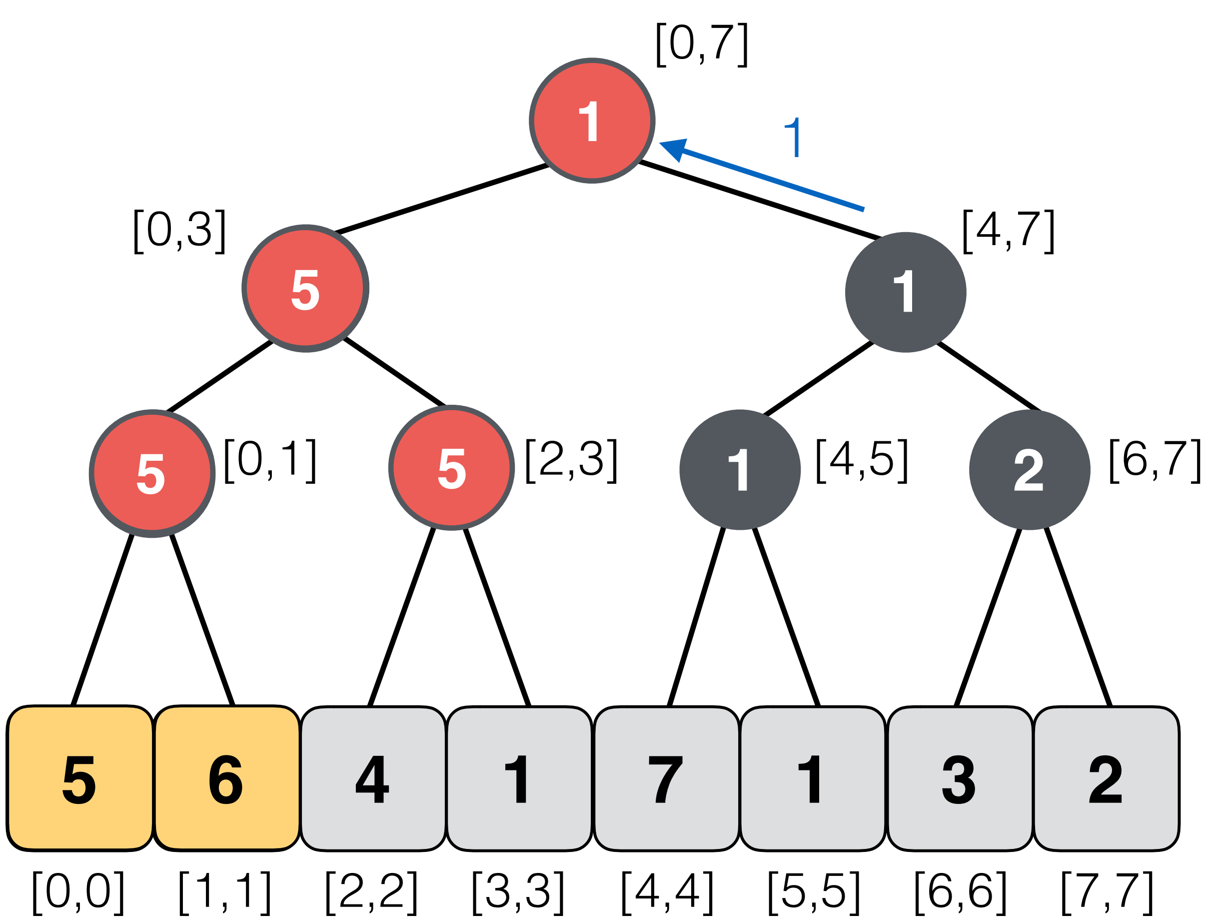


Lazy Tree

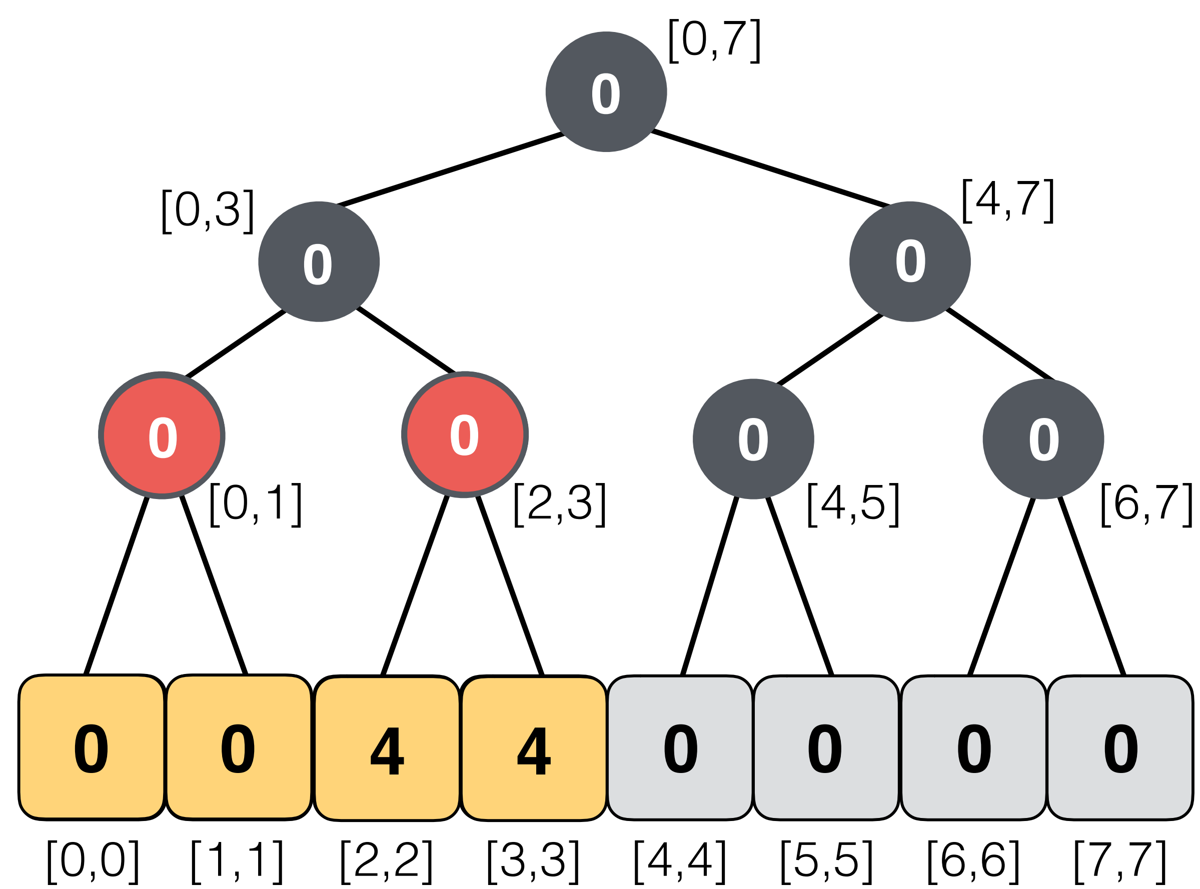
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
$\boxed{\texttt{update\_range}(0,0,2)}$
`rmq`(3,5) = ?



Segment Tree

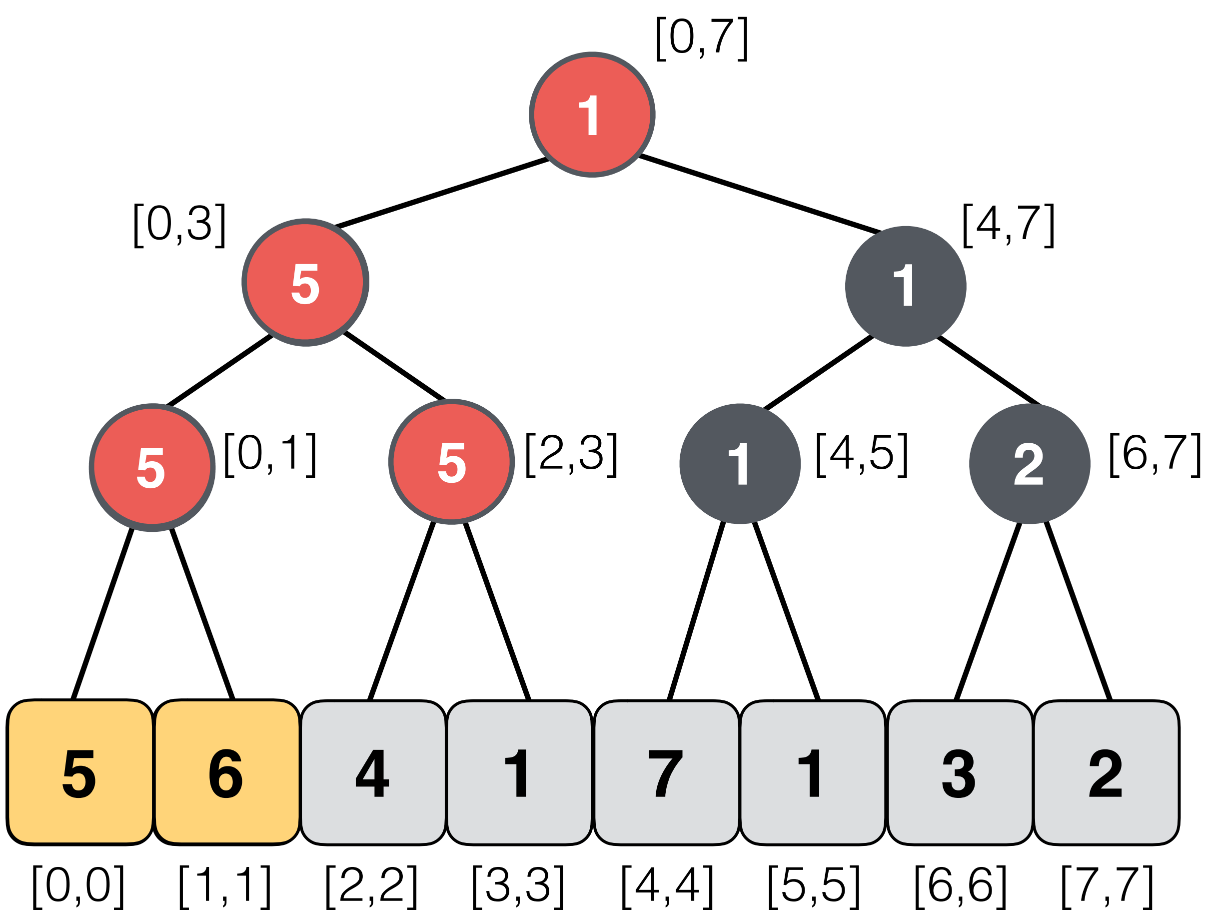


Lazy Tree

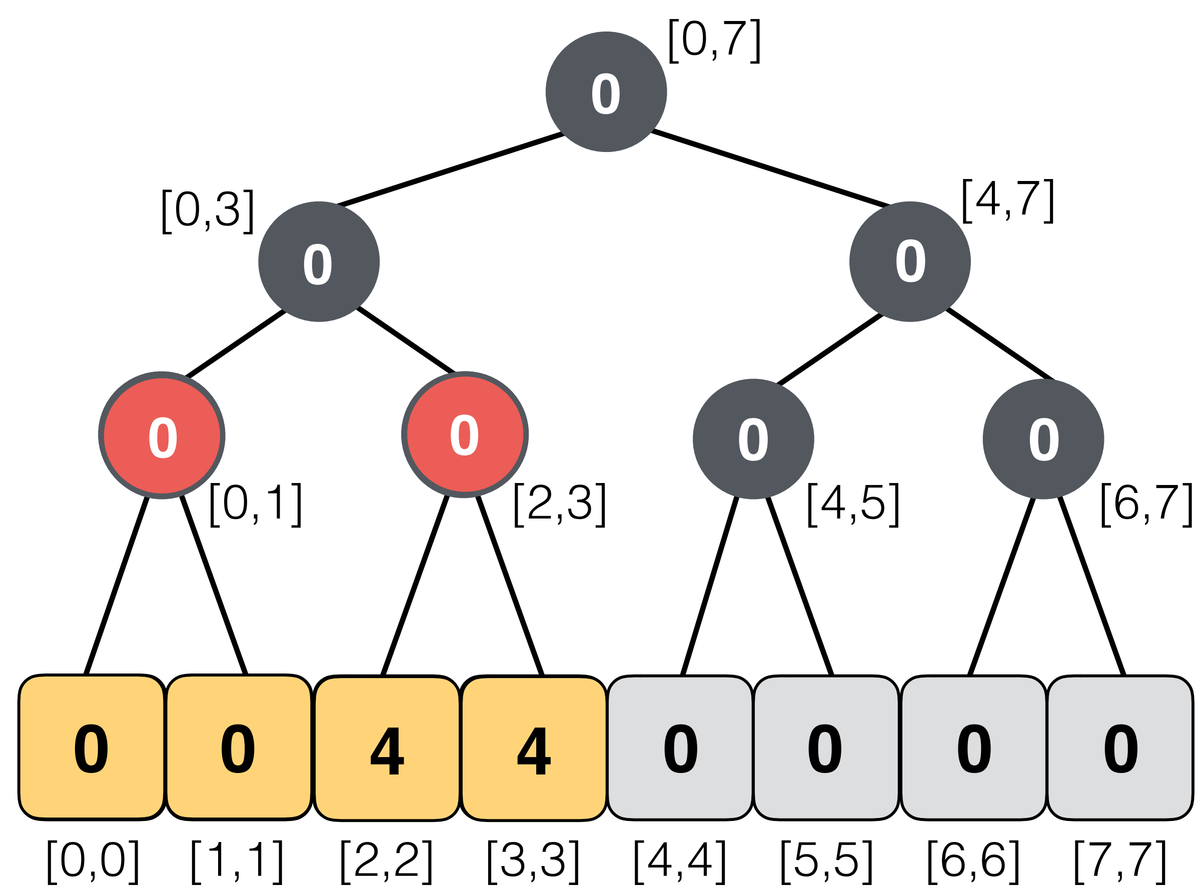
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

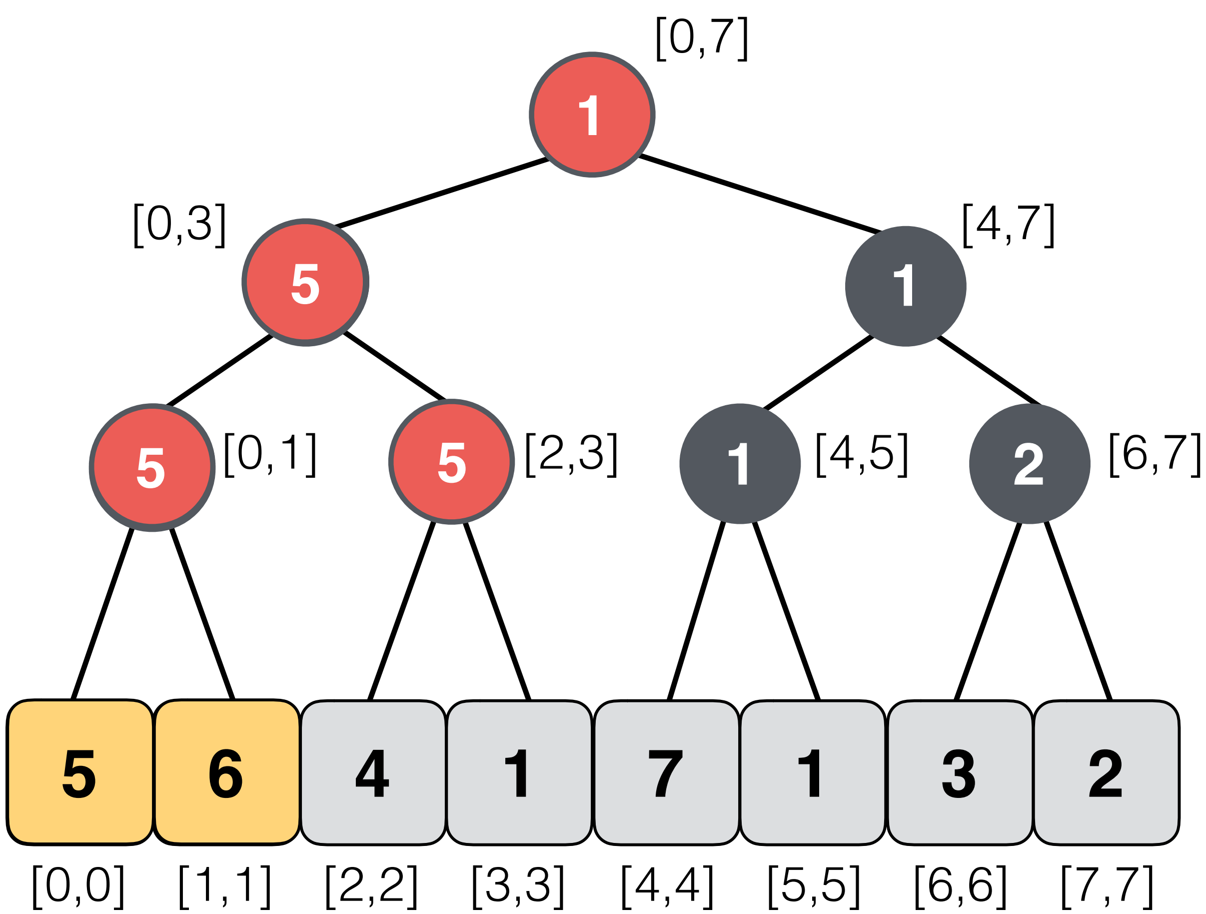


Lazy Tree

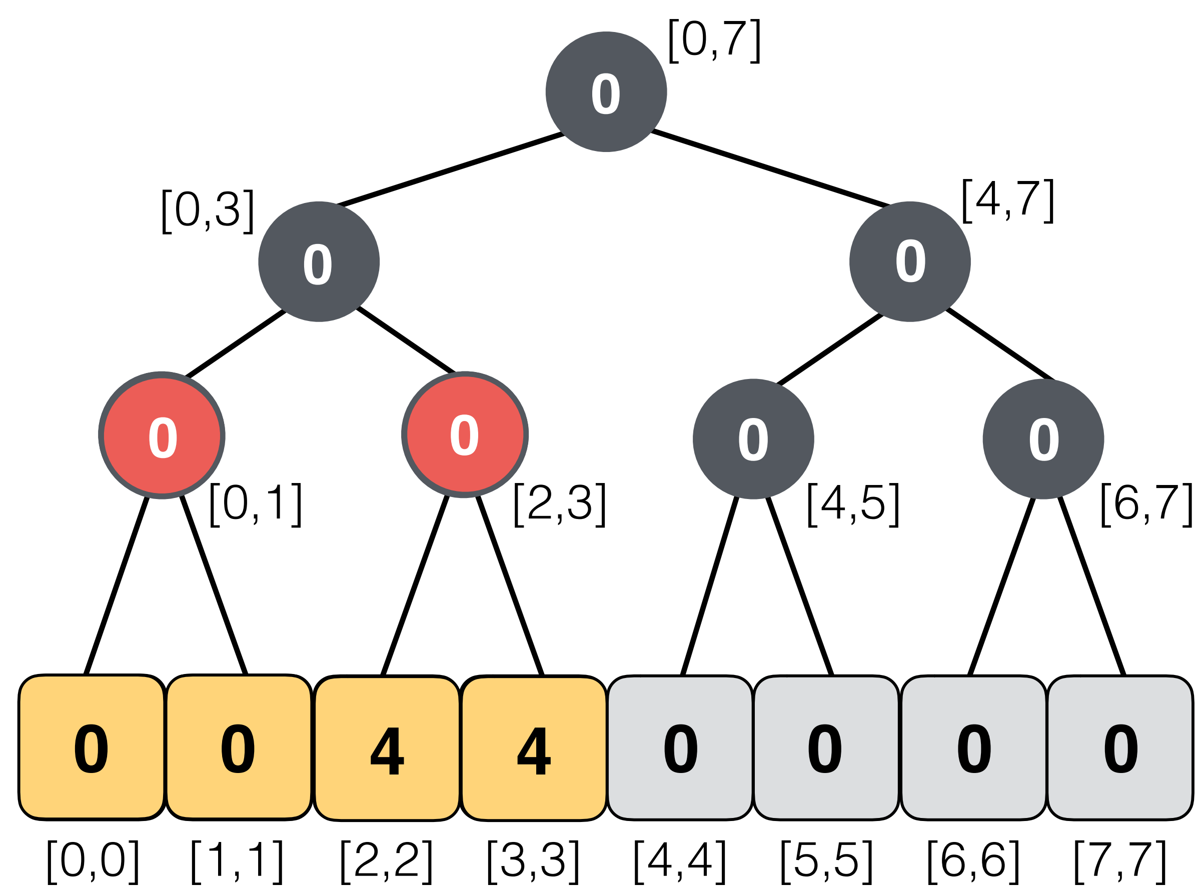
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
$\boxed{\texttt{rmq}(3,5) = ?}$



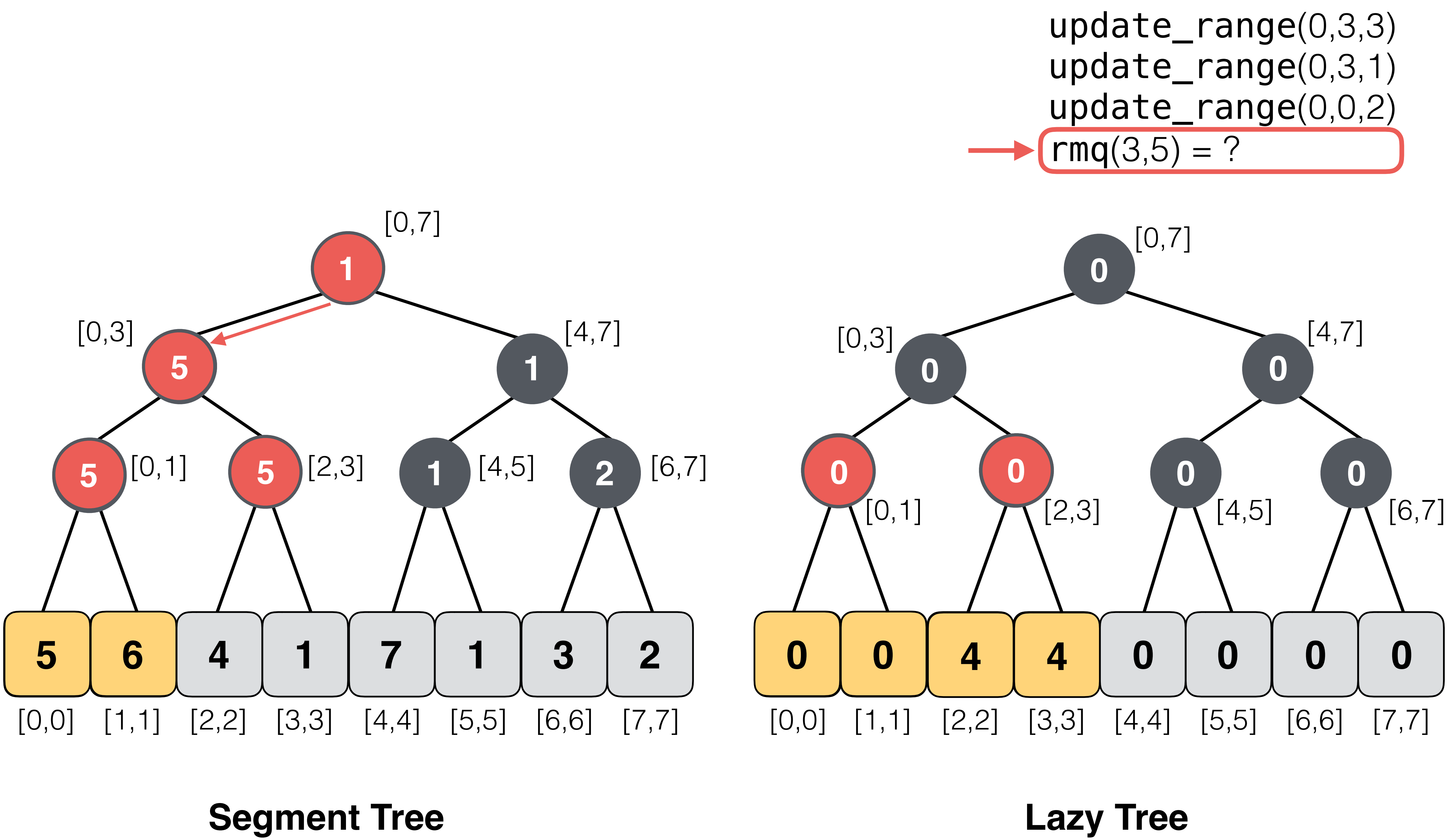
Segment Tree



Lazy Tree

Lazy Propagation in Segment Trees

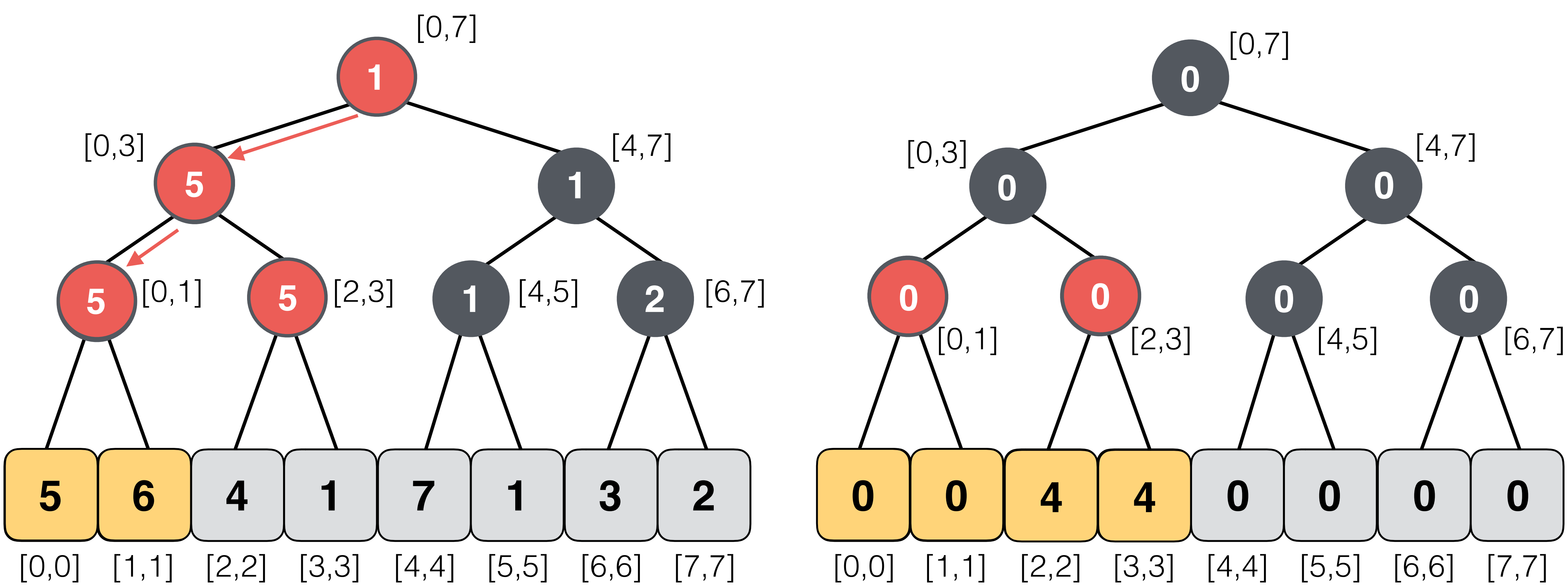
Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.



Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?

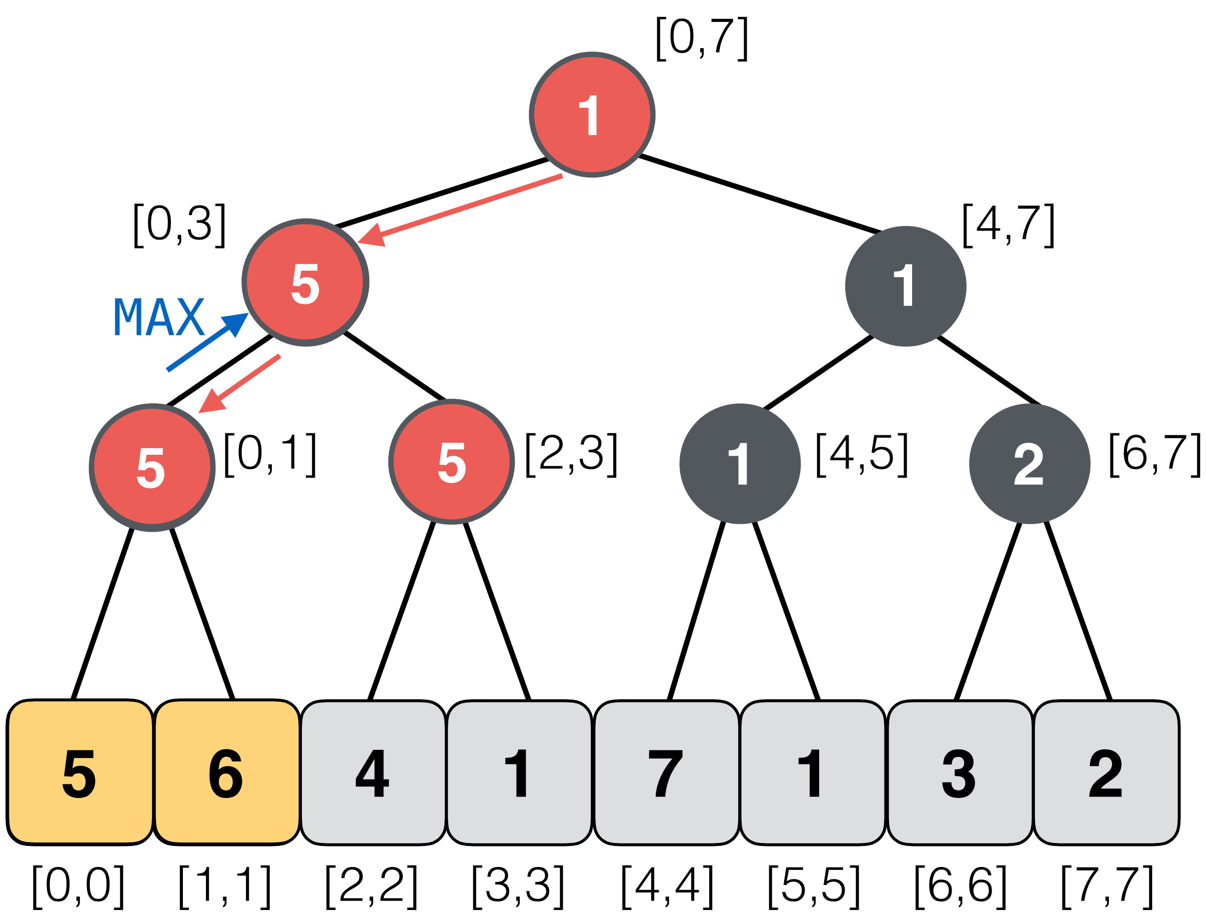


Segment Tree **Lazy Tree**

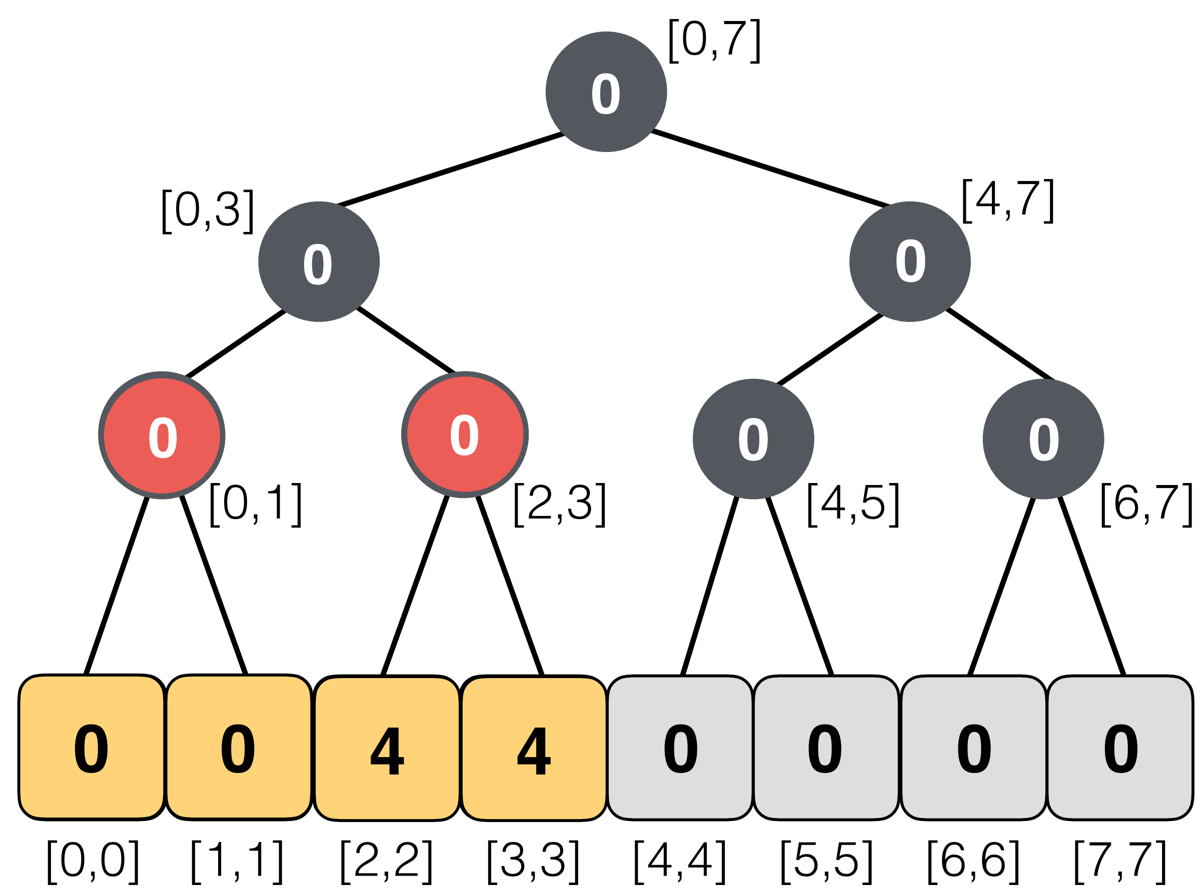
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

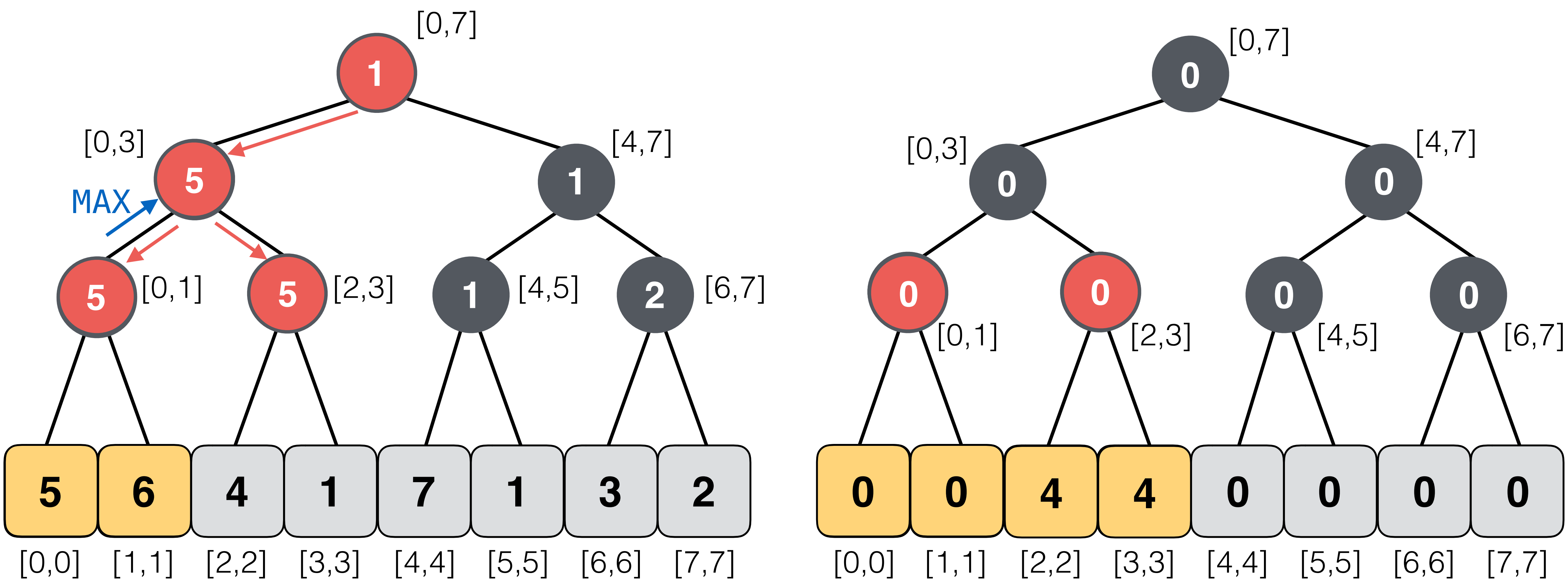


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?

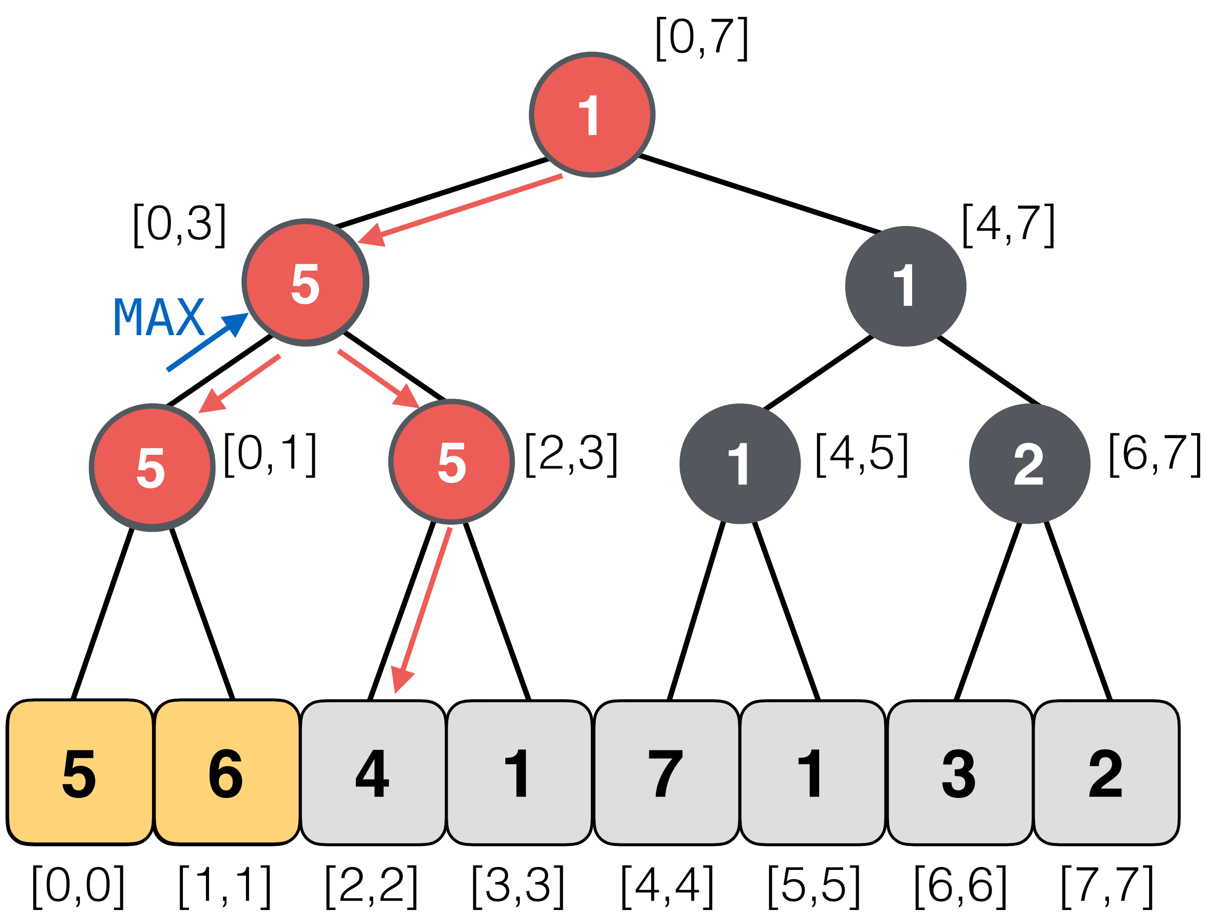


Segment Tree **Lazy Tree**

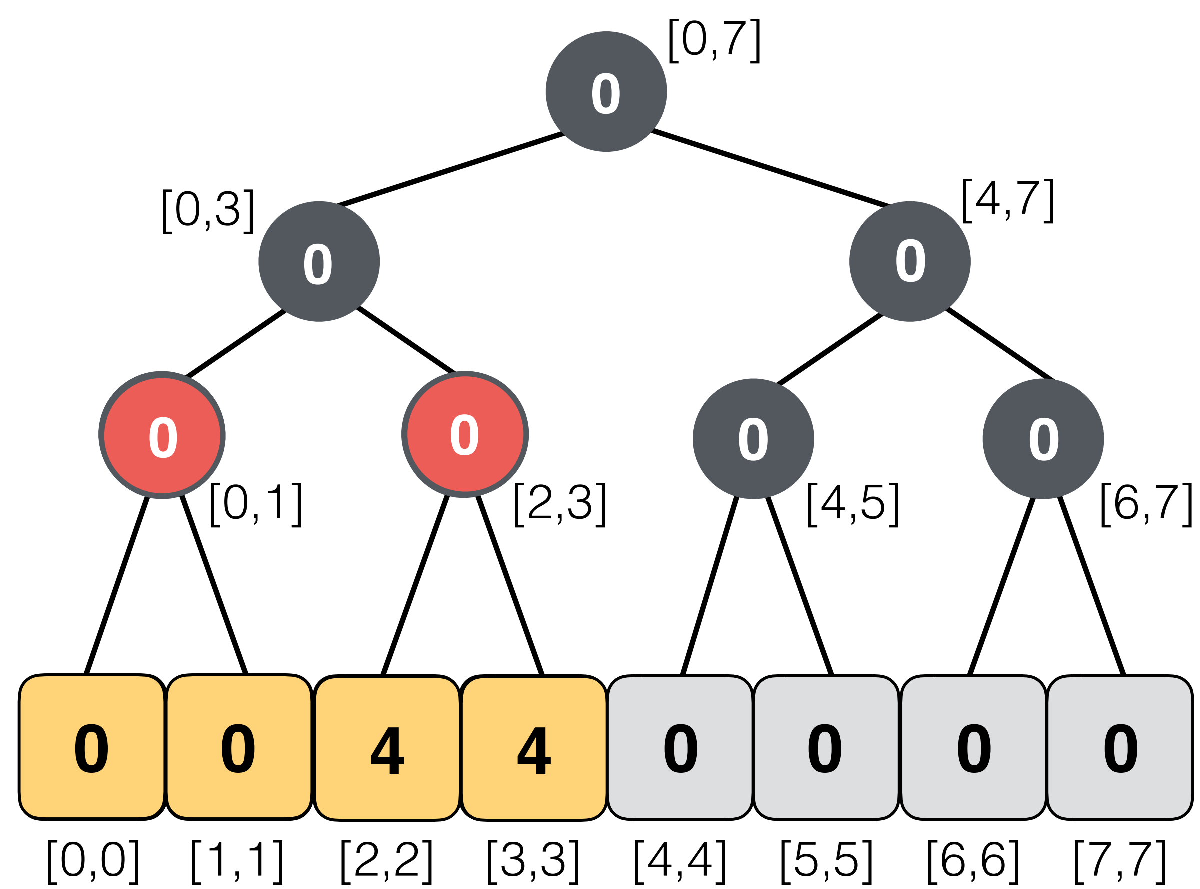
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

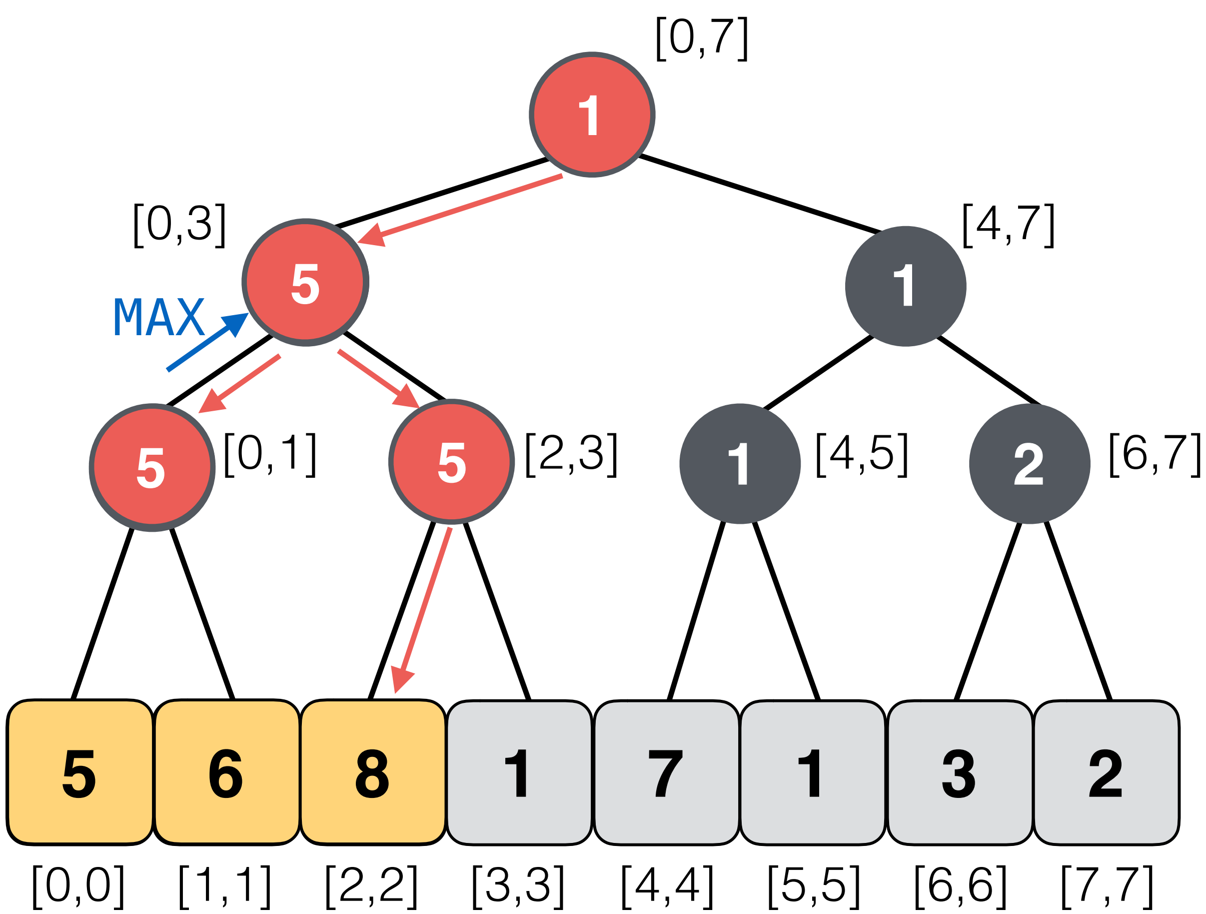


Lazy Tree

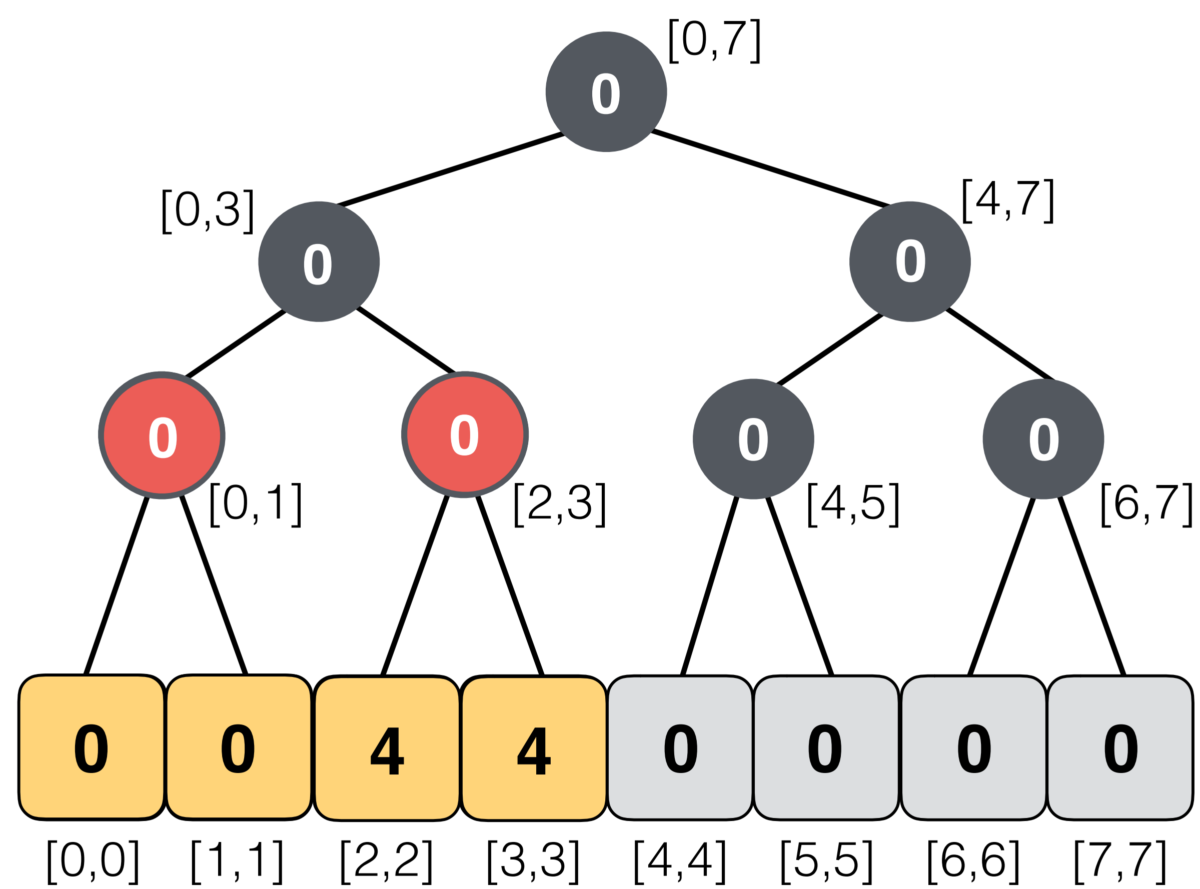
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

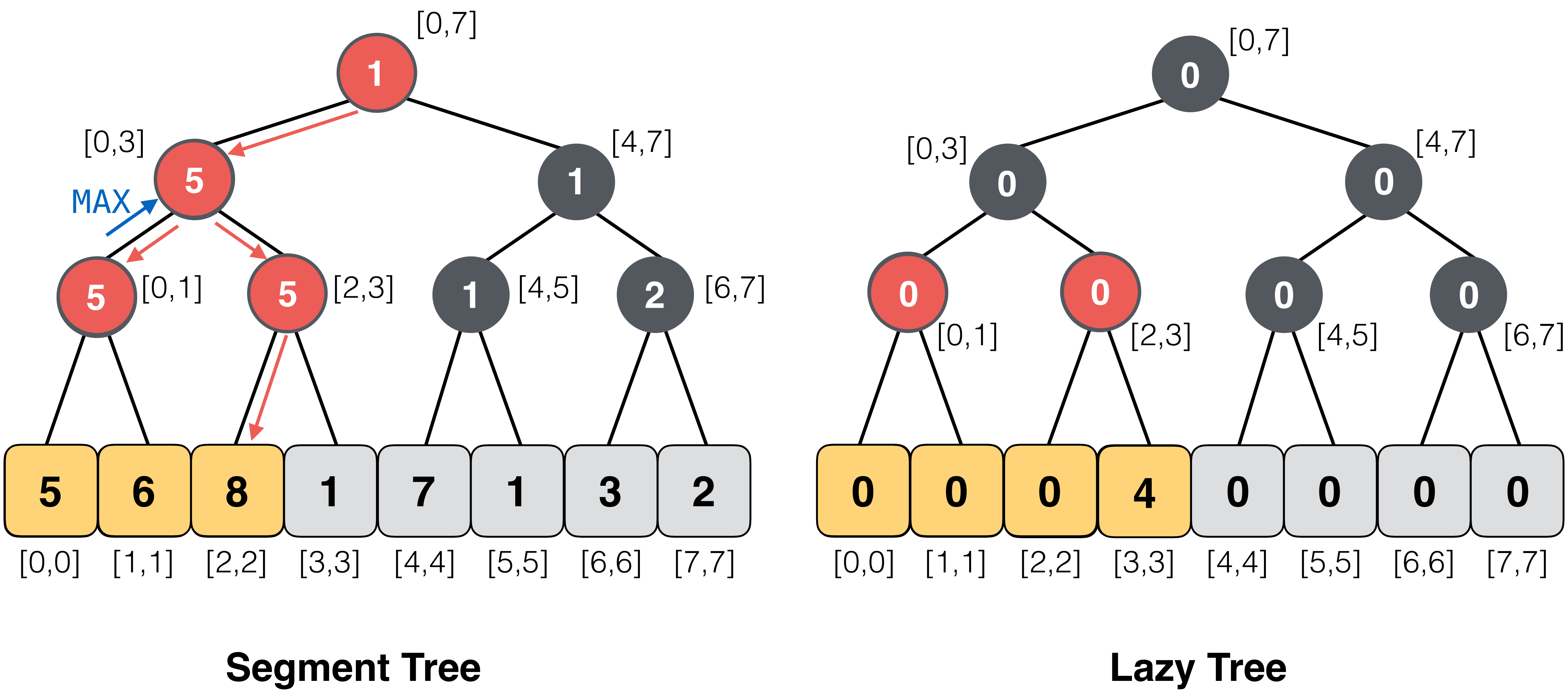


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

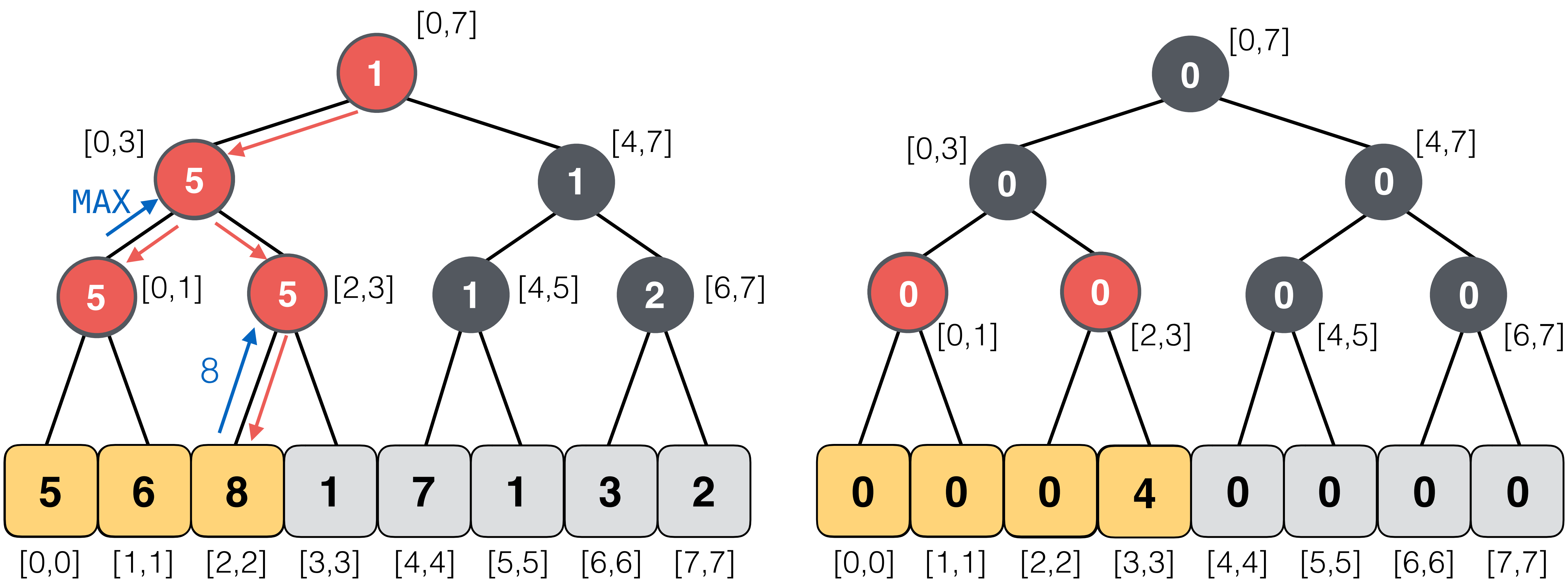
`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```

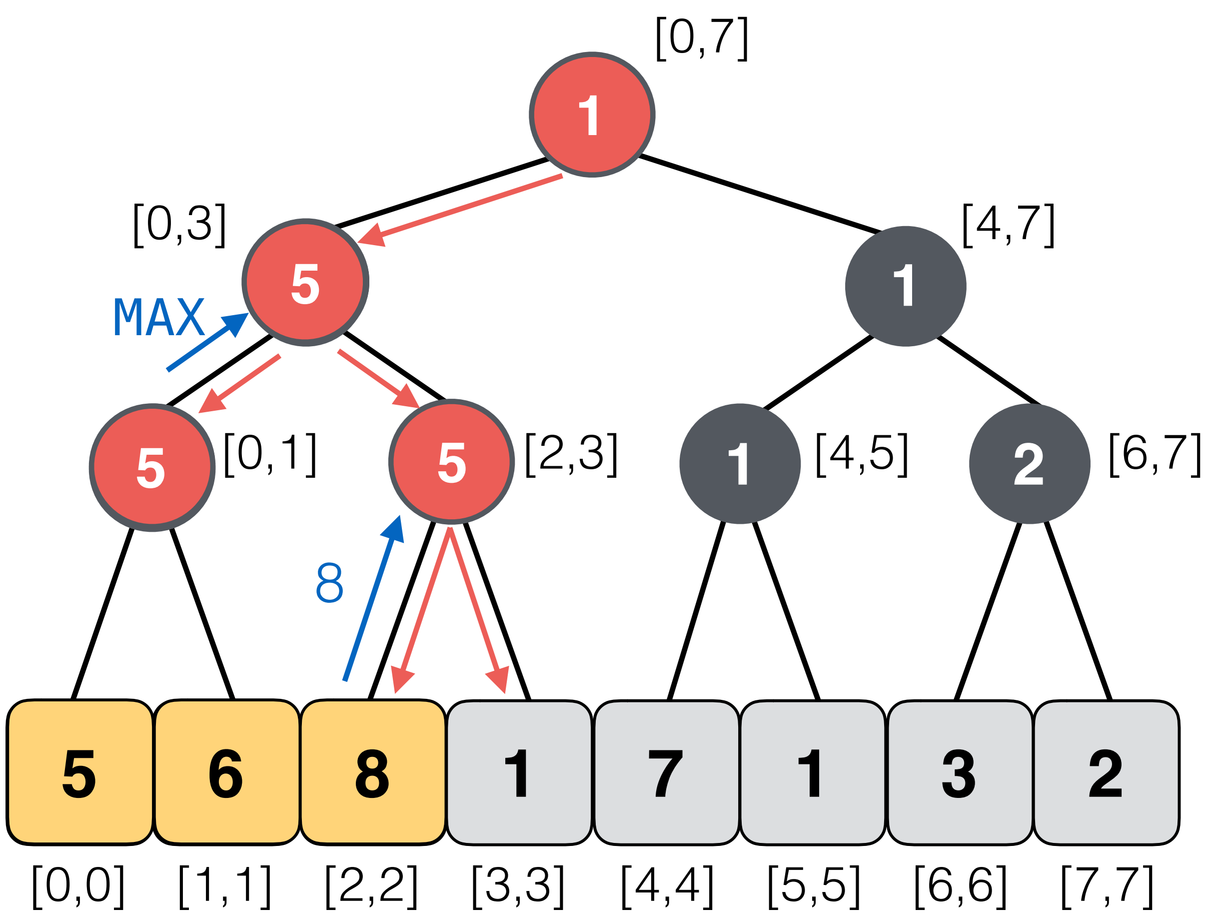


Segment Tree **Lazy Tree**

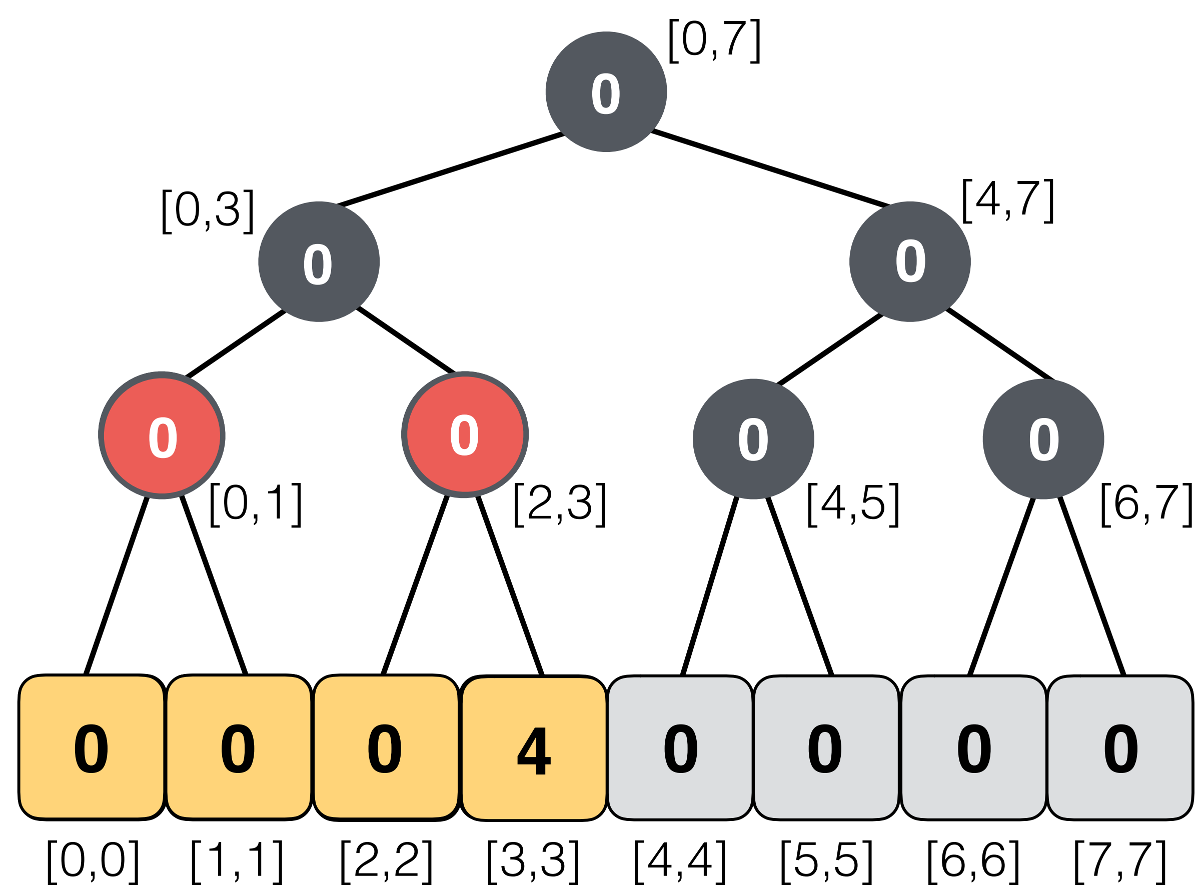
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

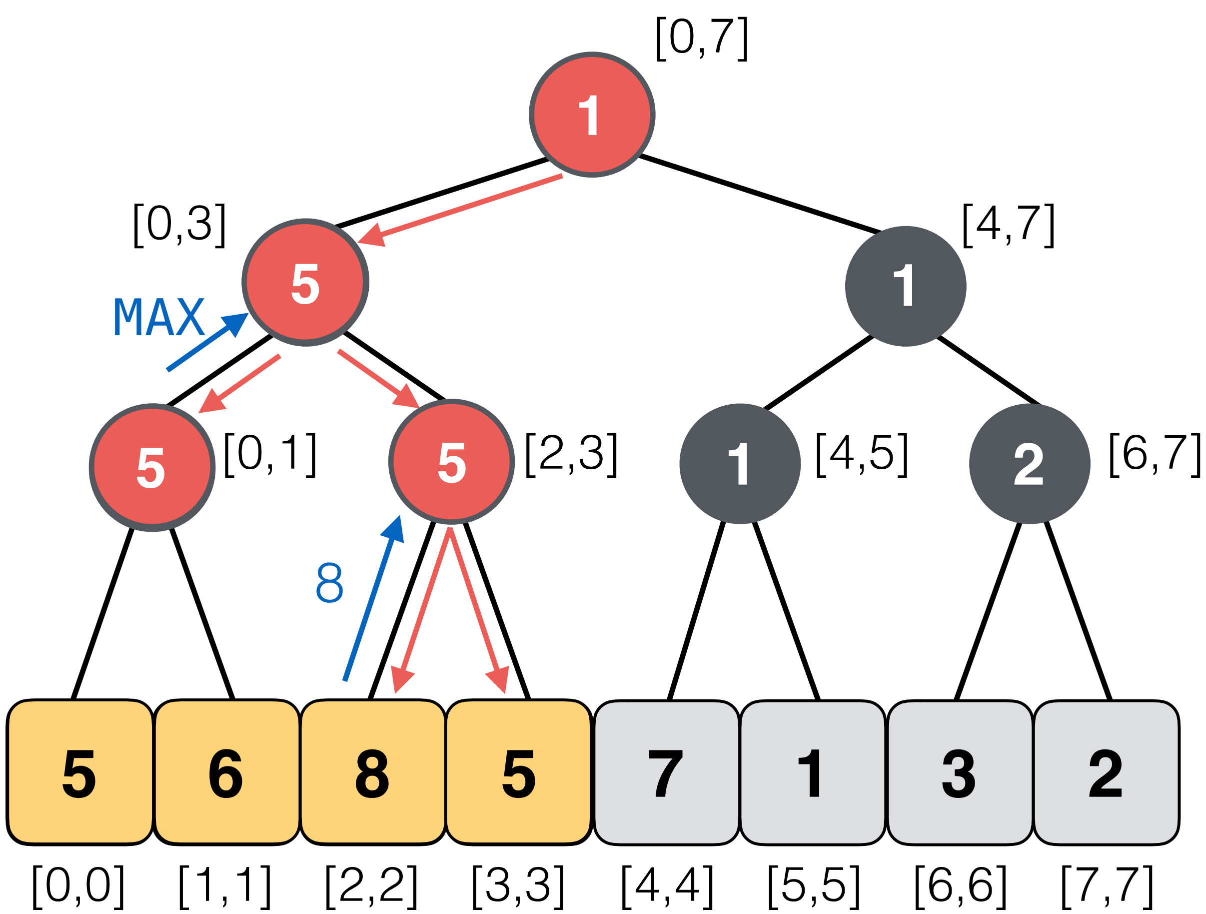


Lazy Tree

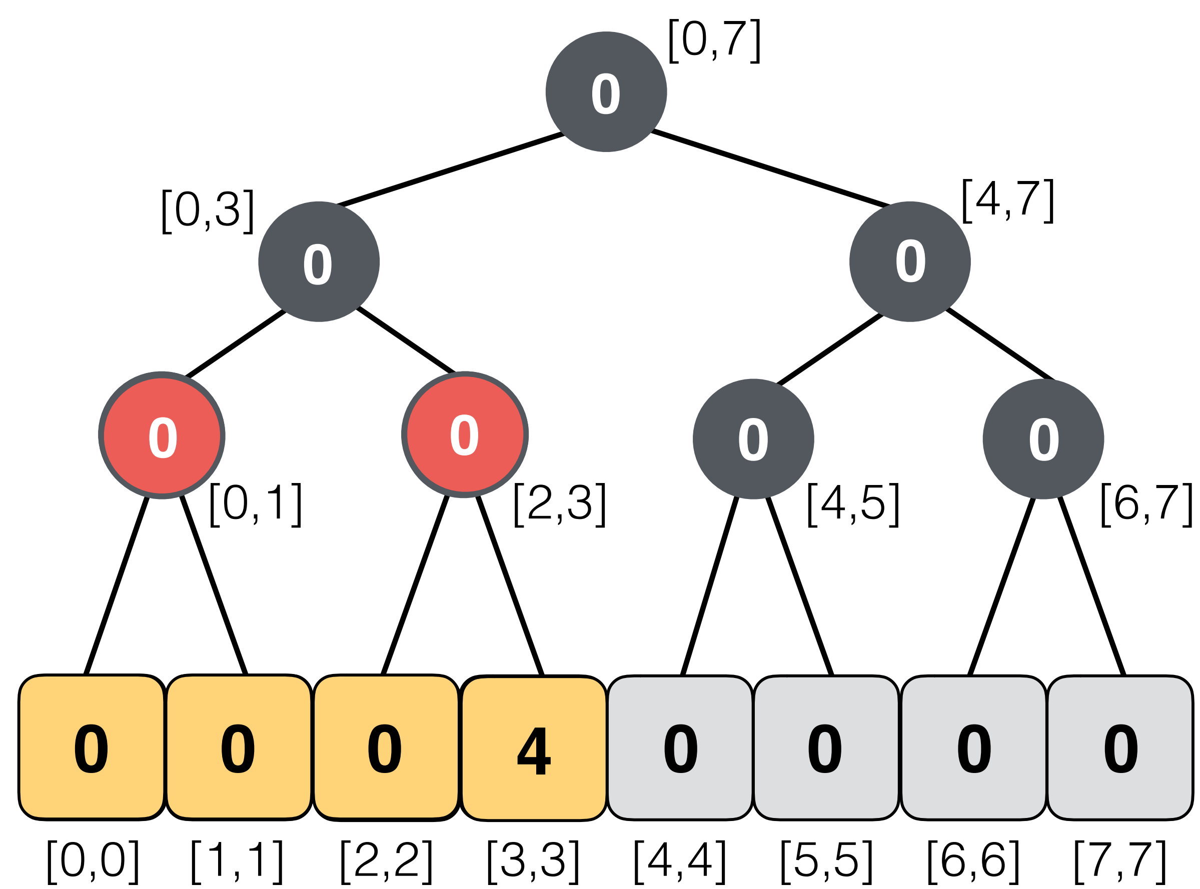
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

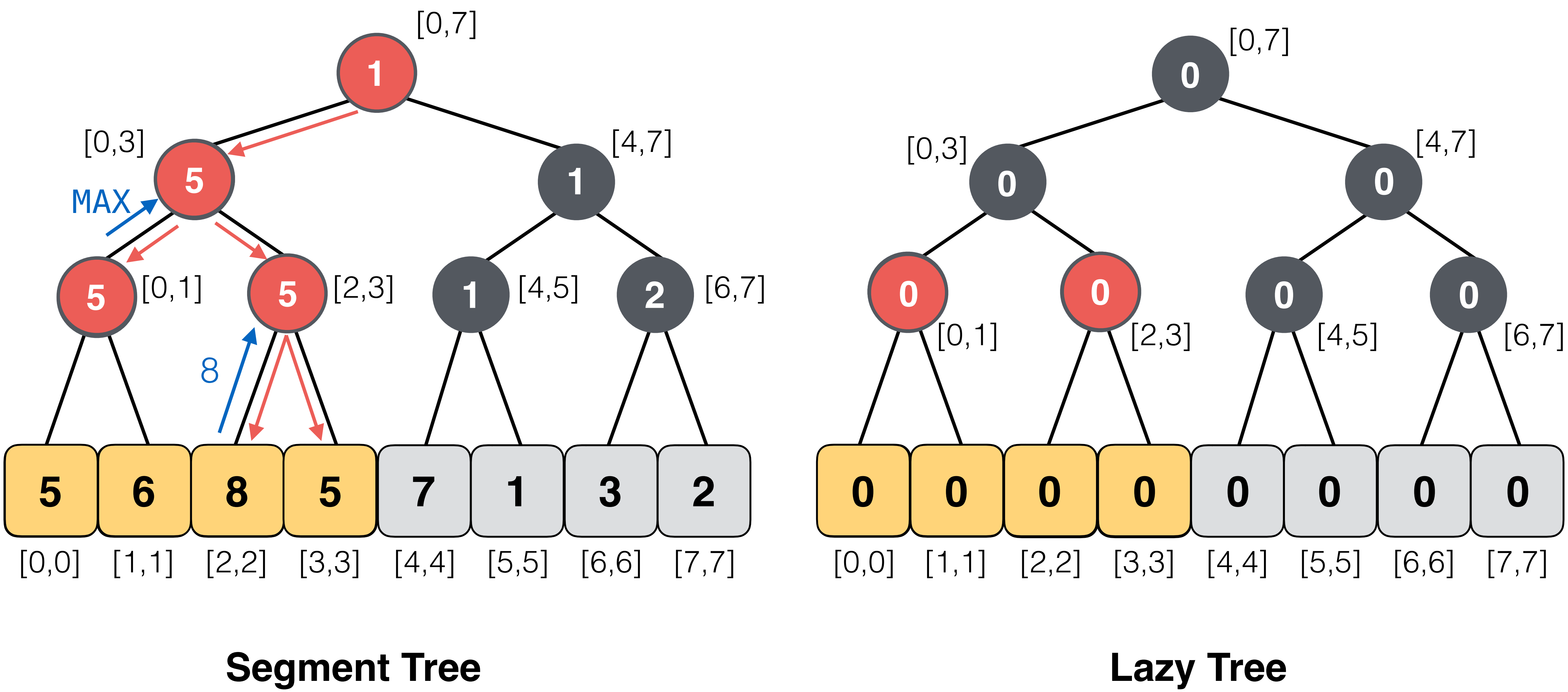


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

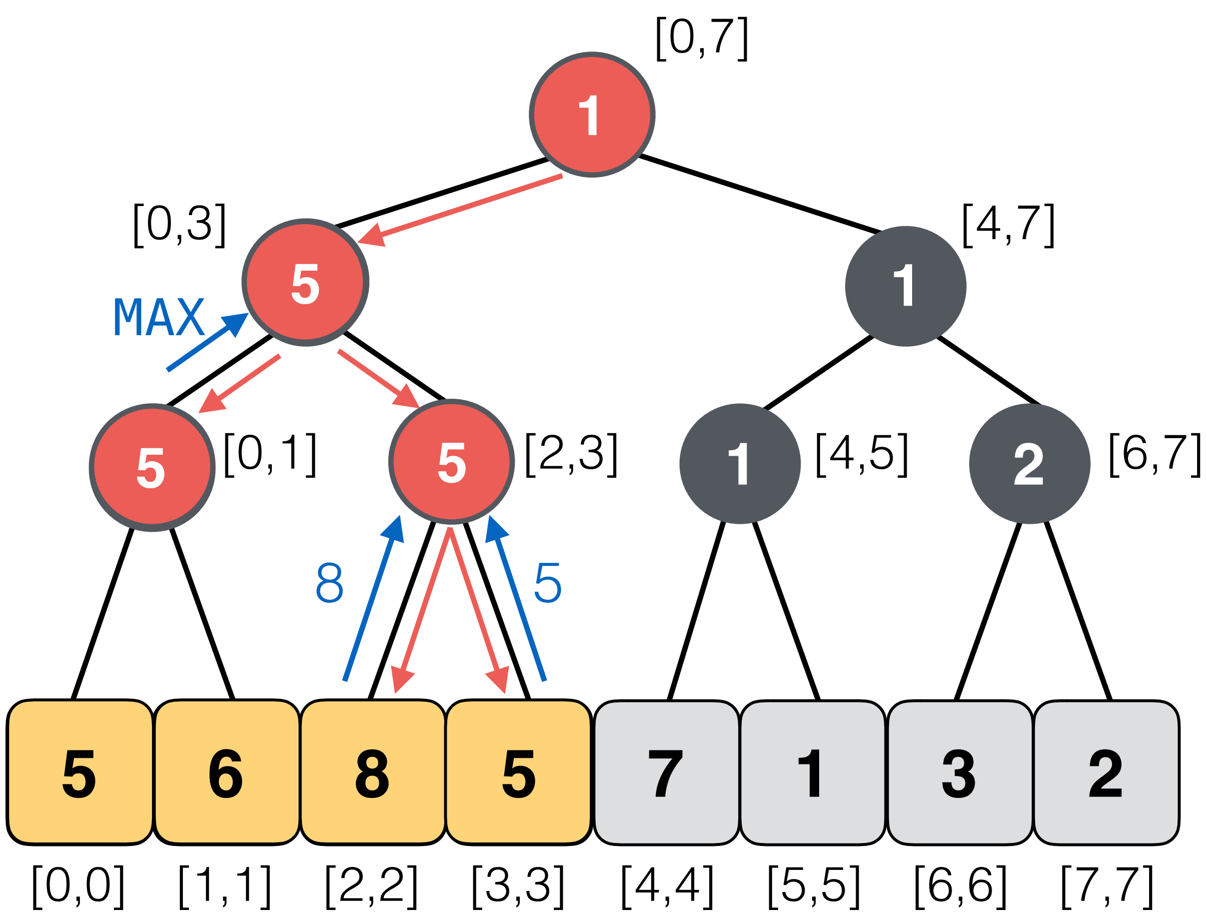
`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



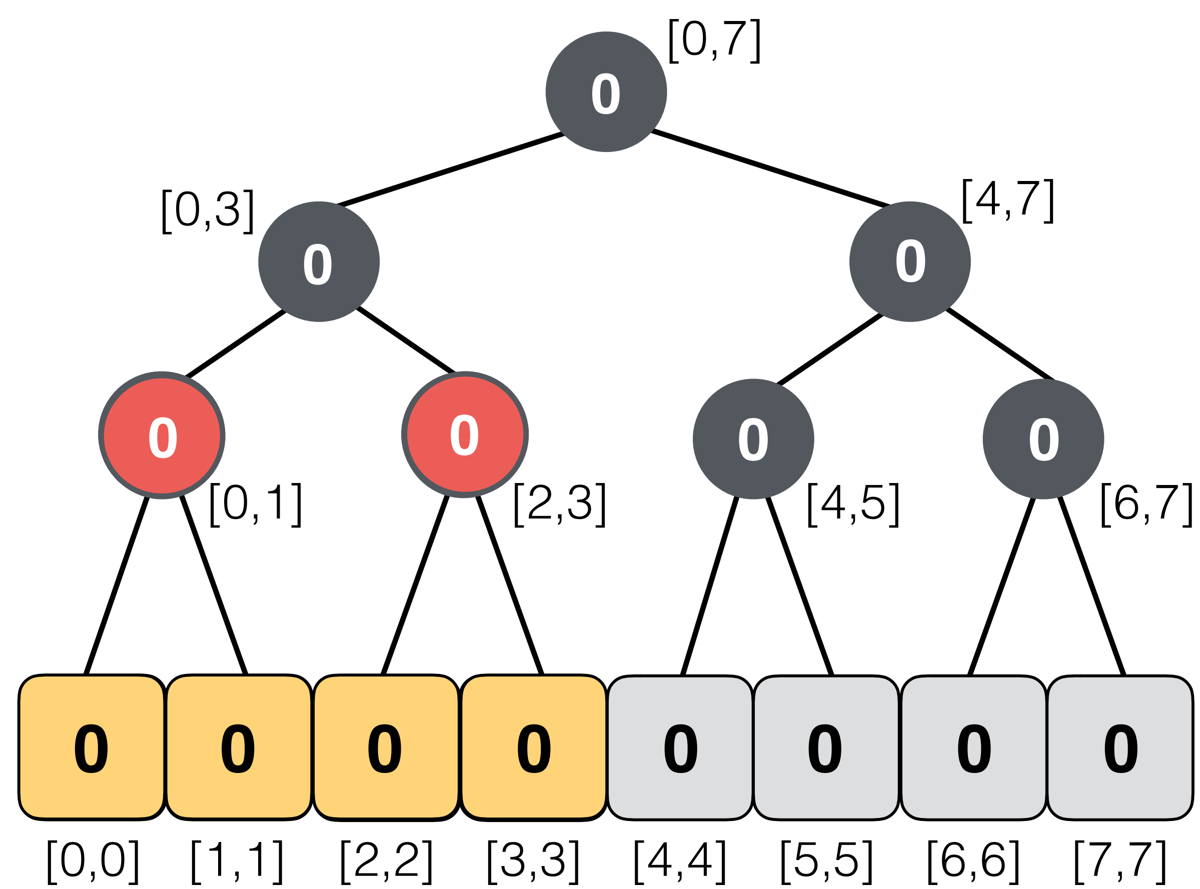
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

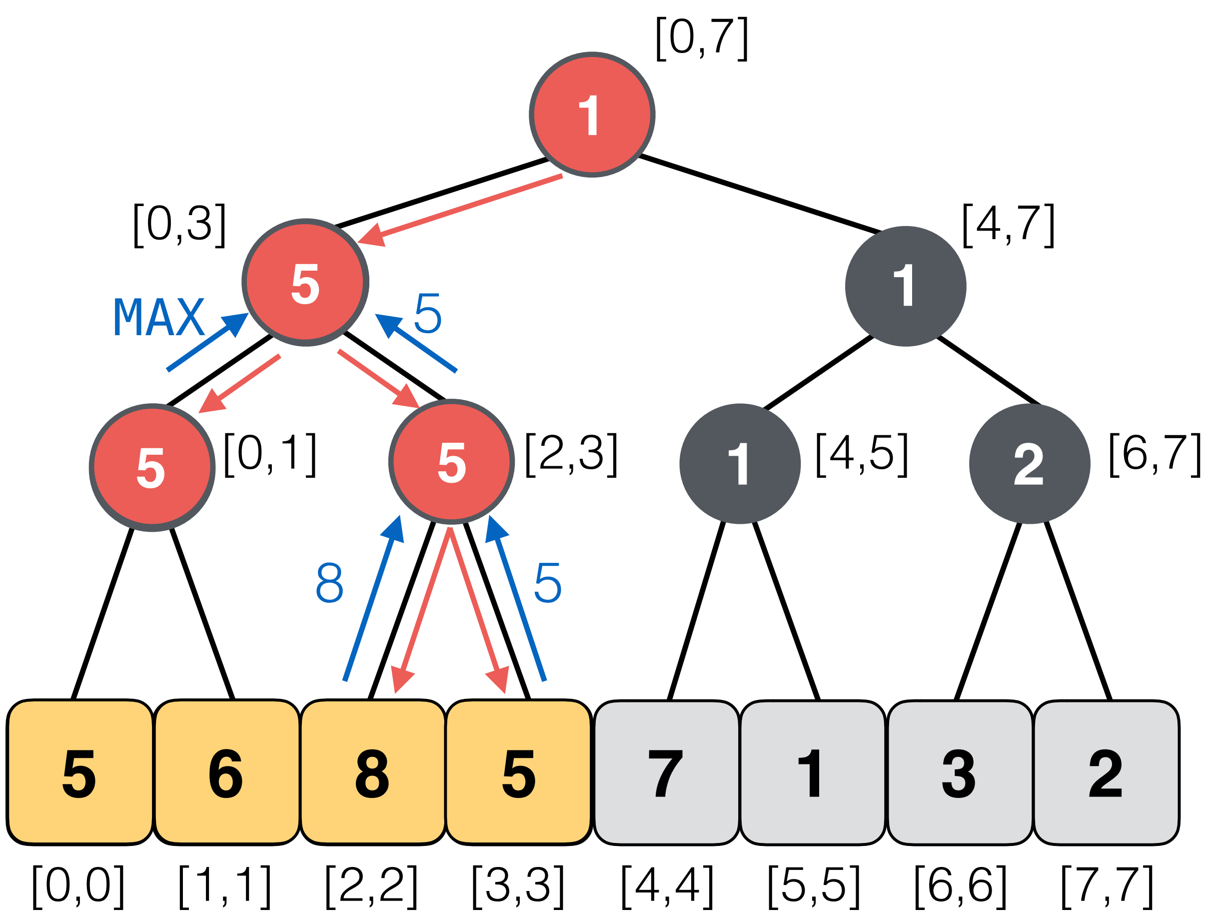


Lazy Tree

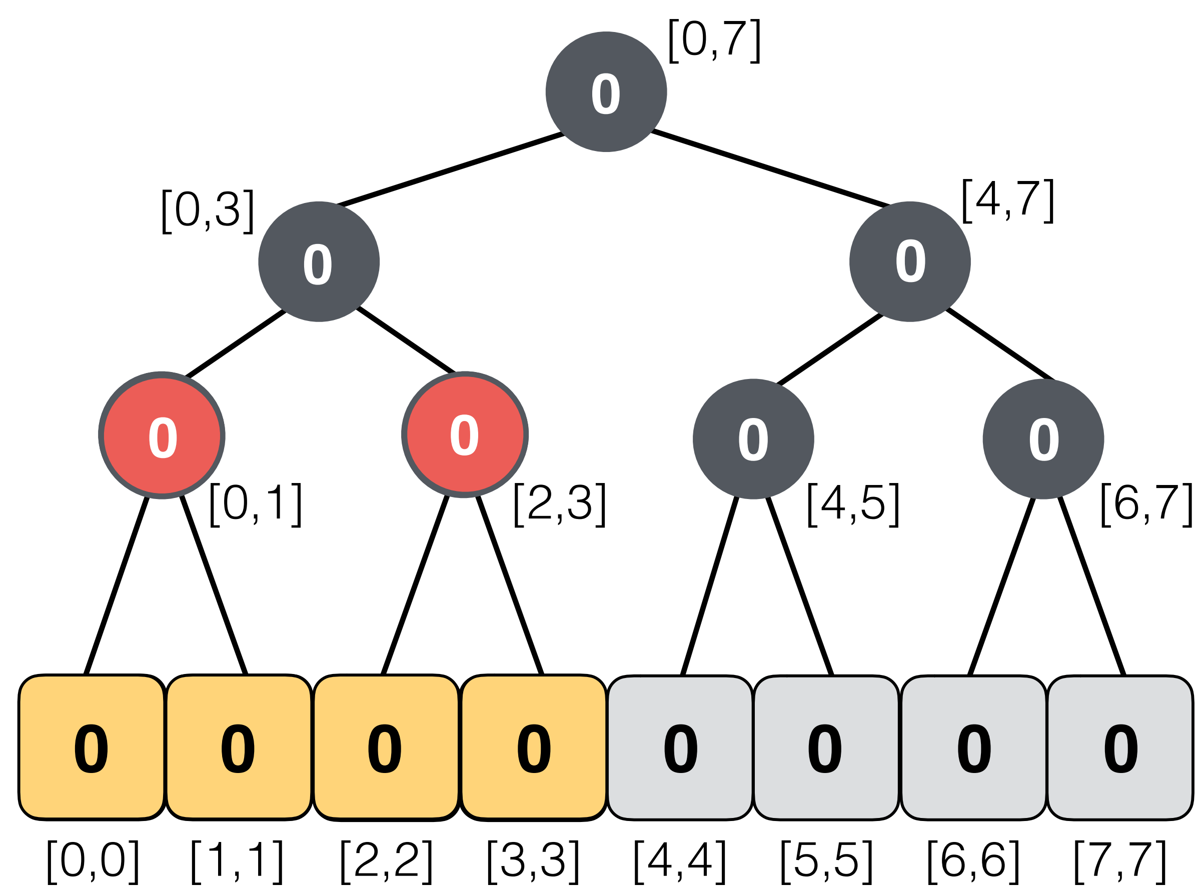
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

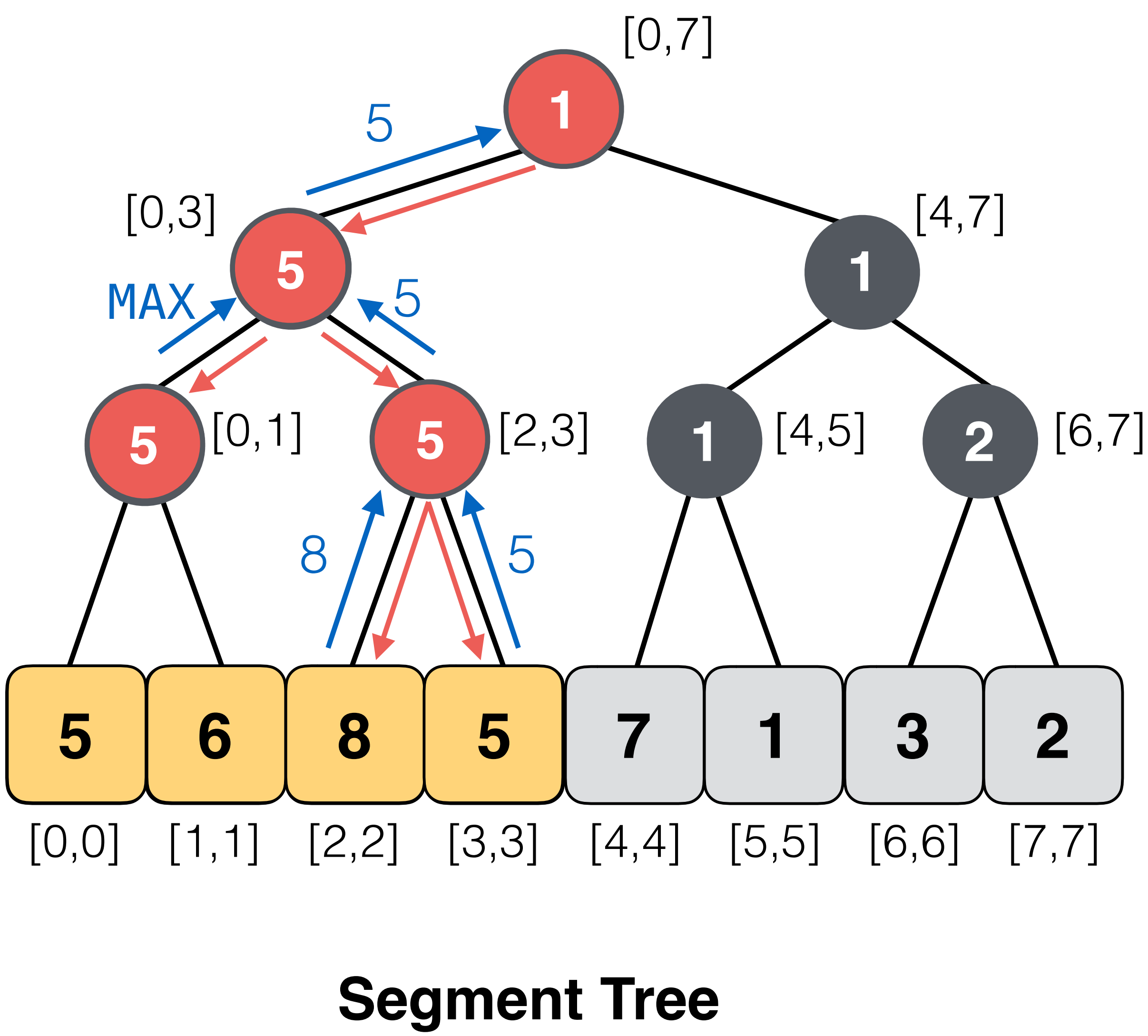


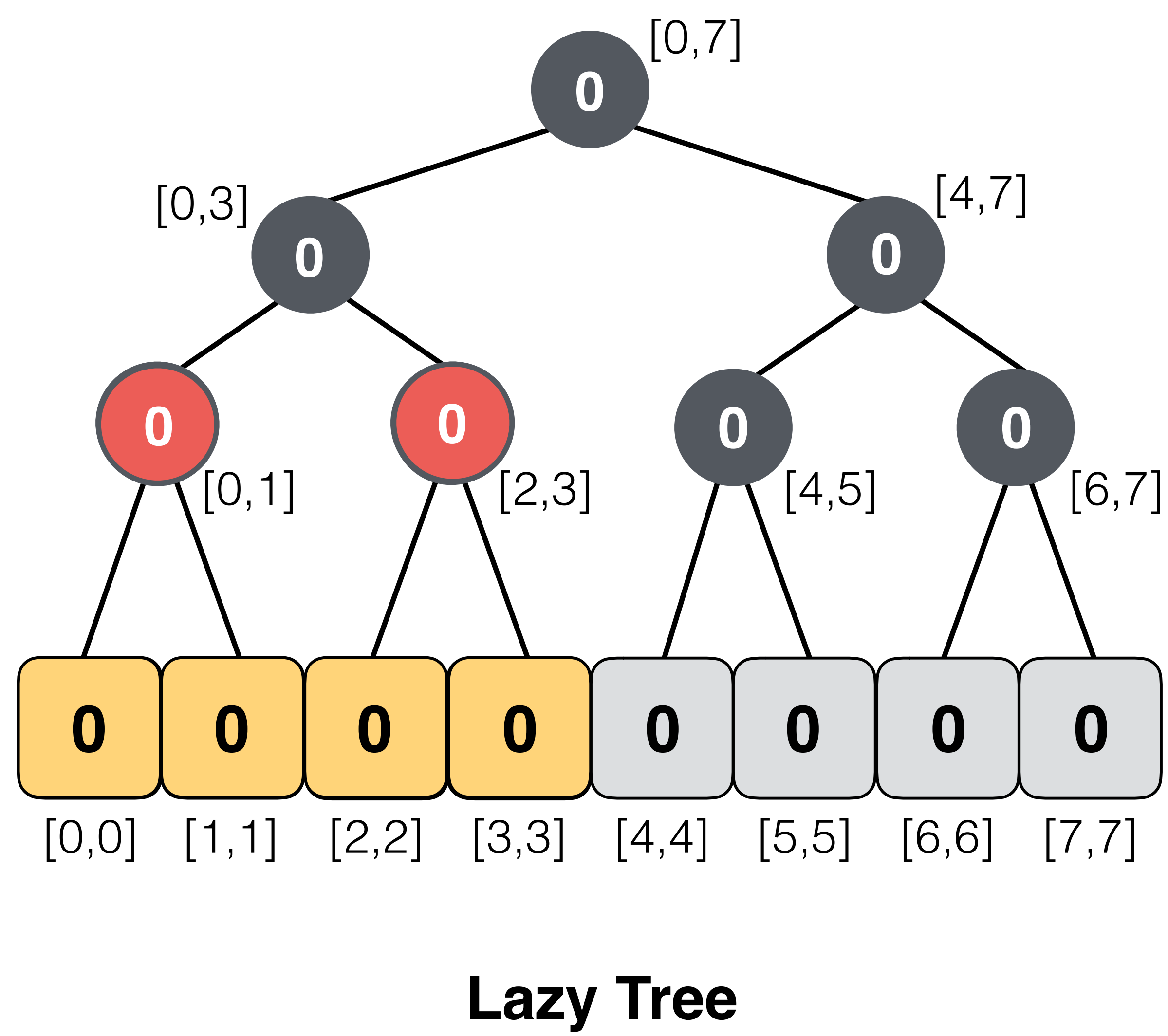
Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?

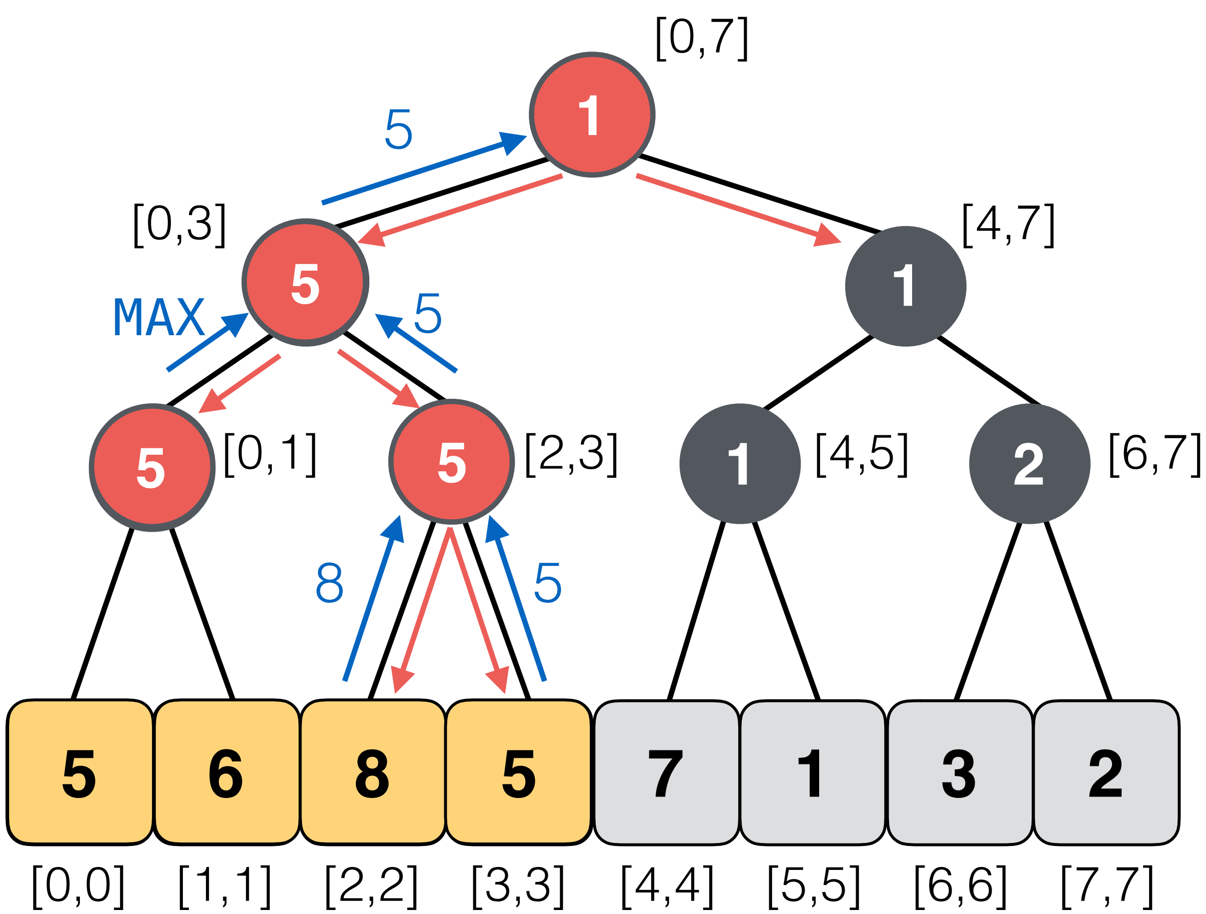




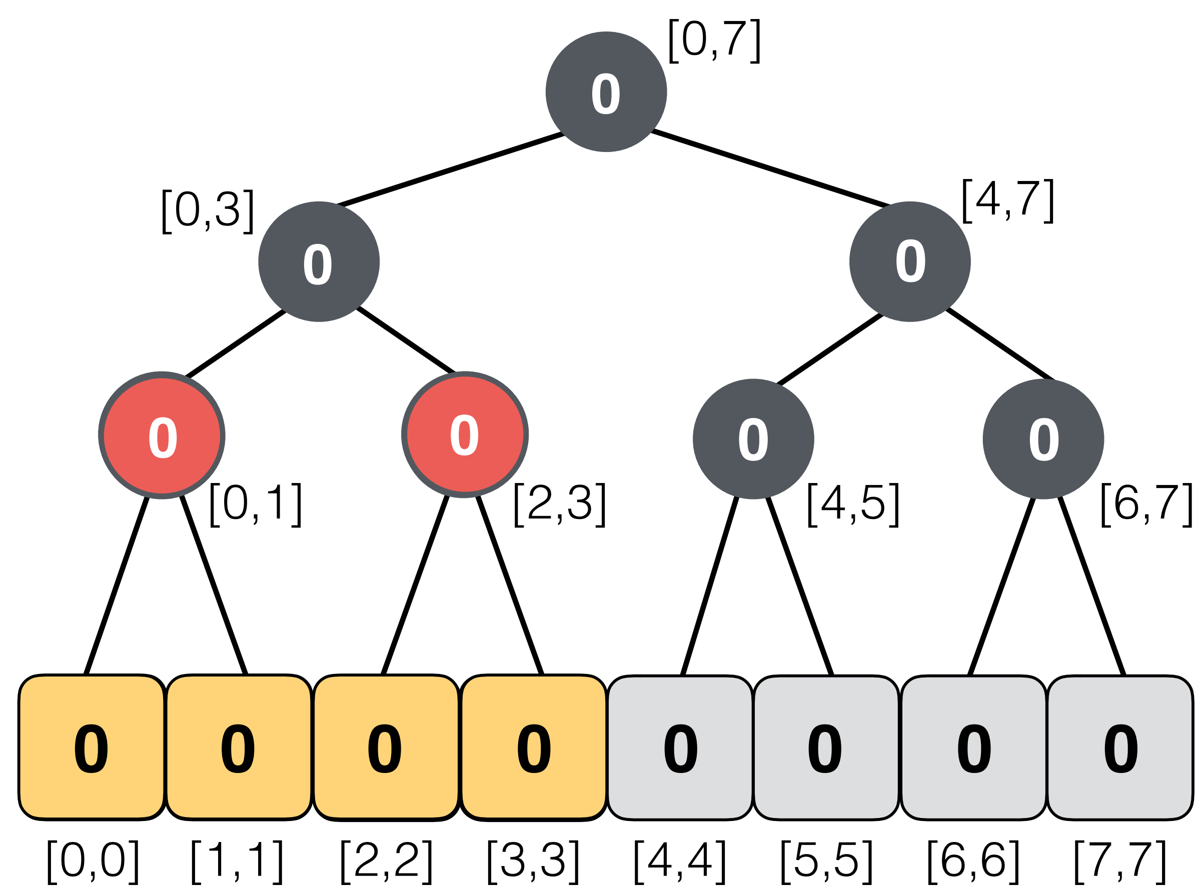
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Segment Tree

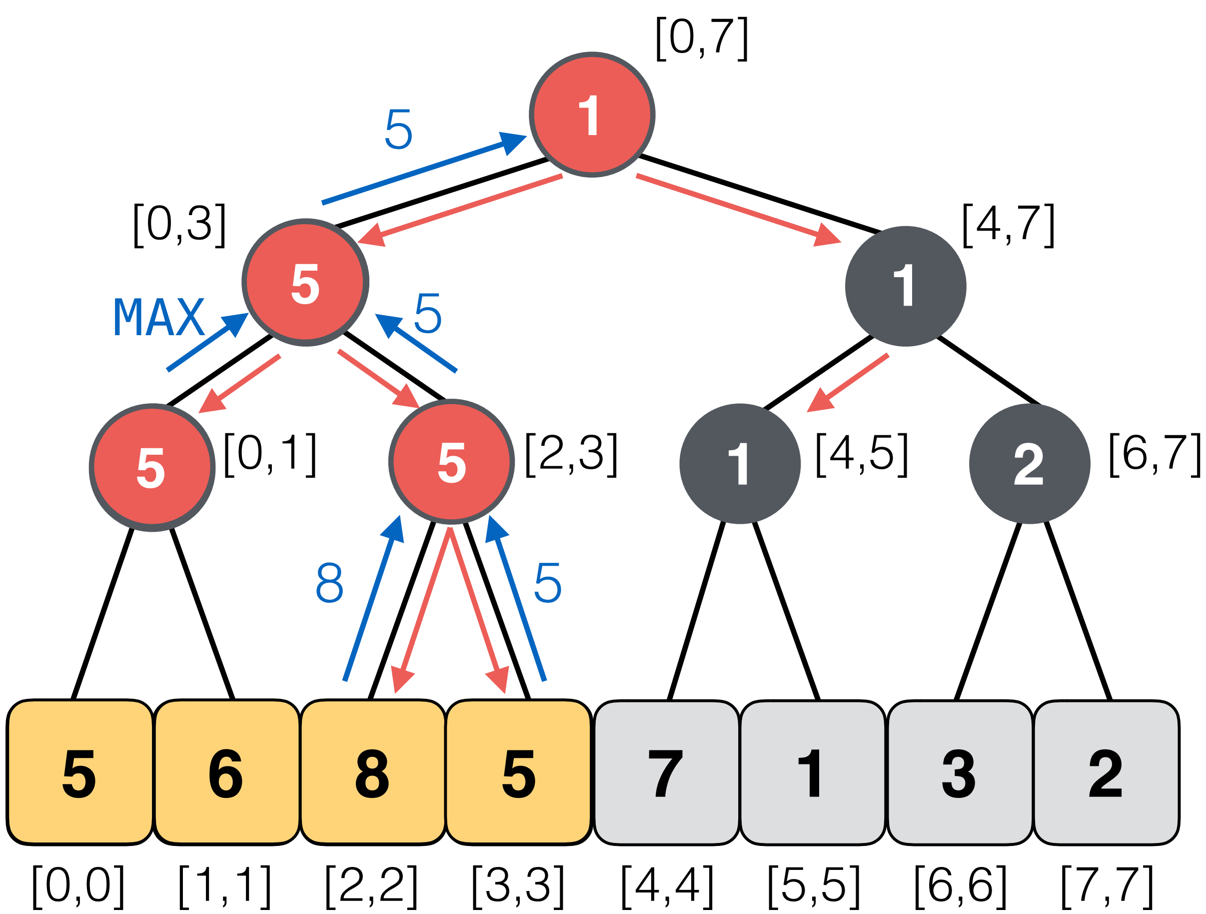


Lazy Tree

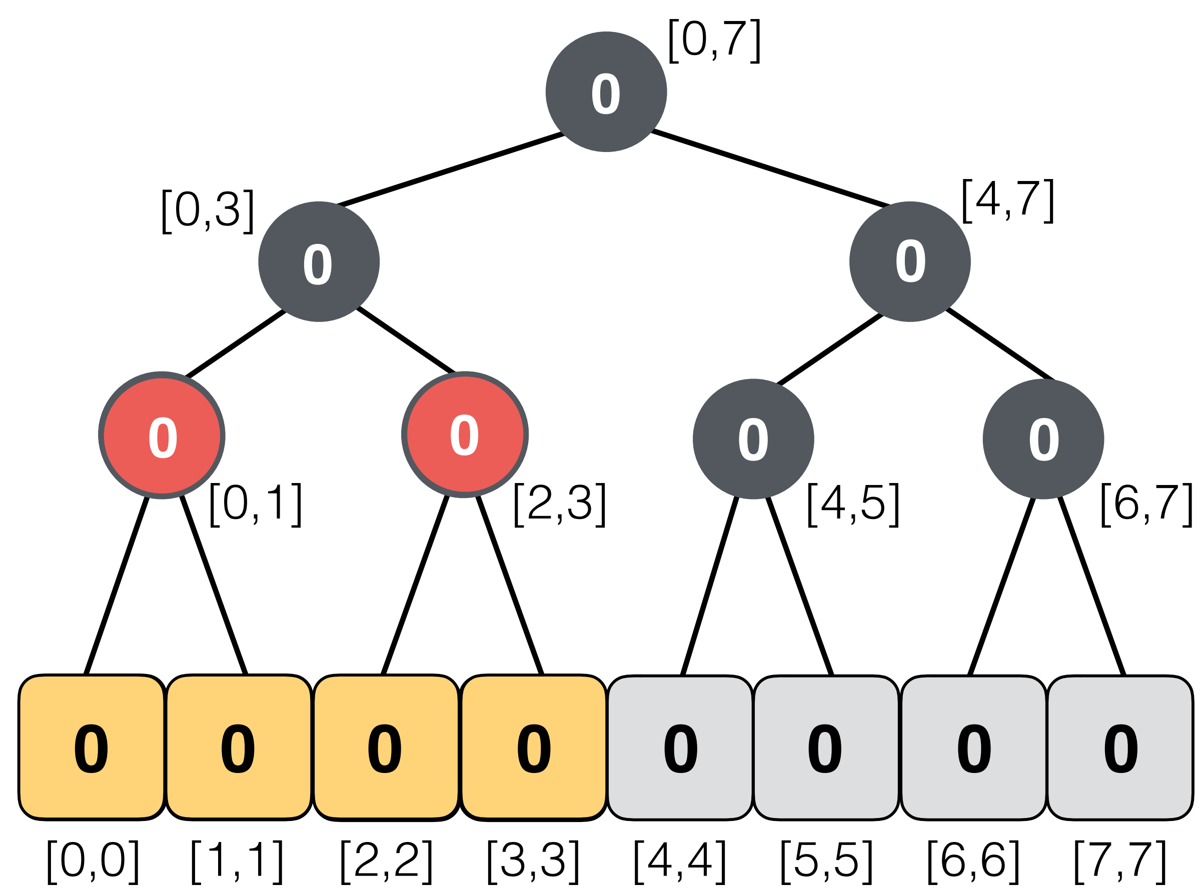
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Segment Tree

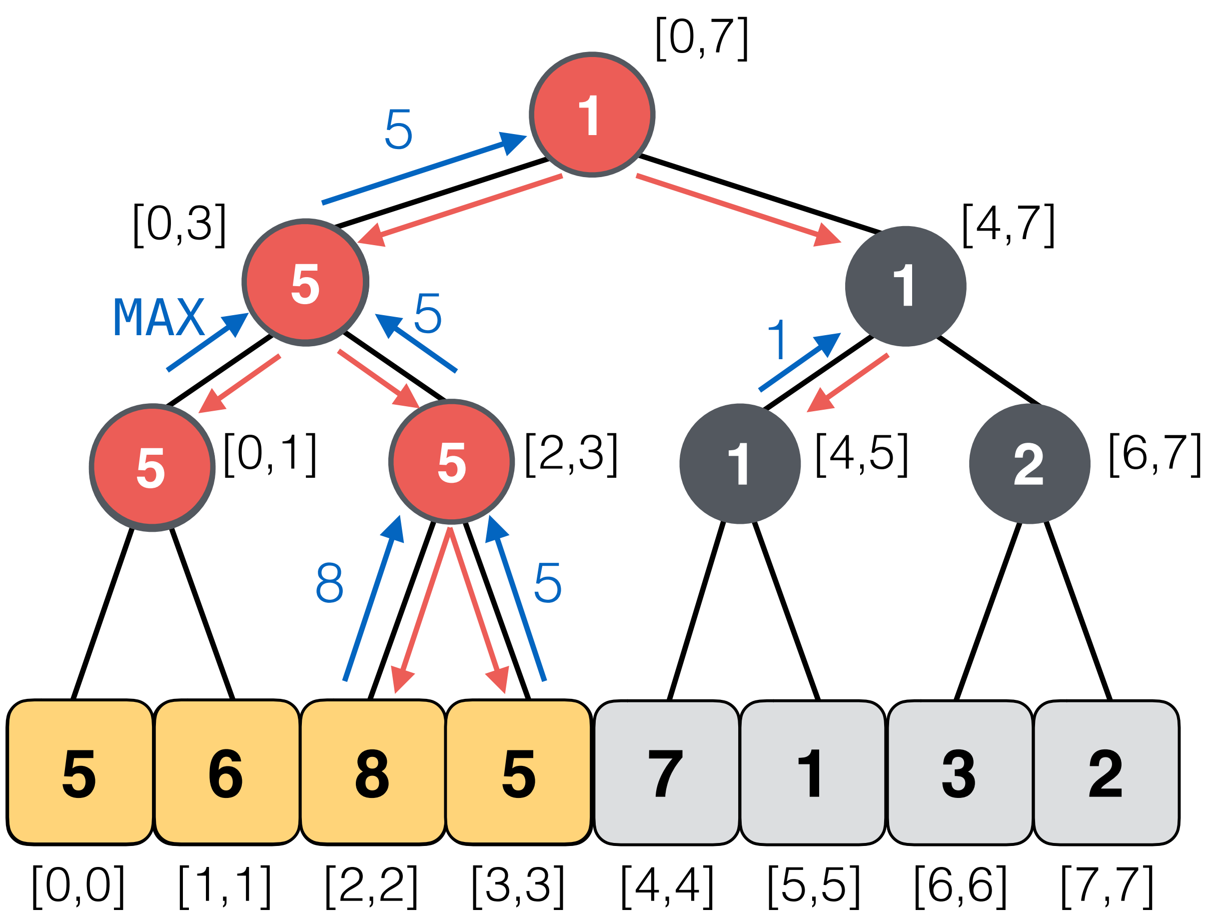


Lazy Tree

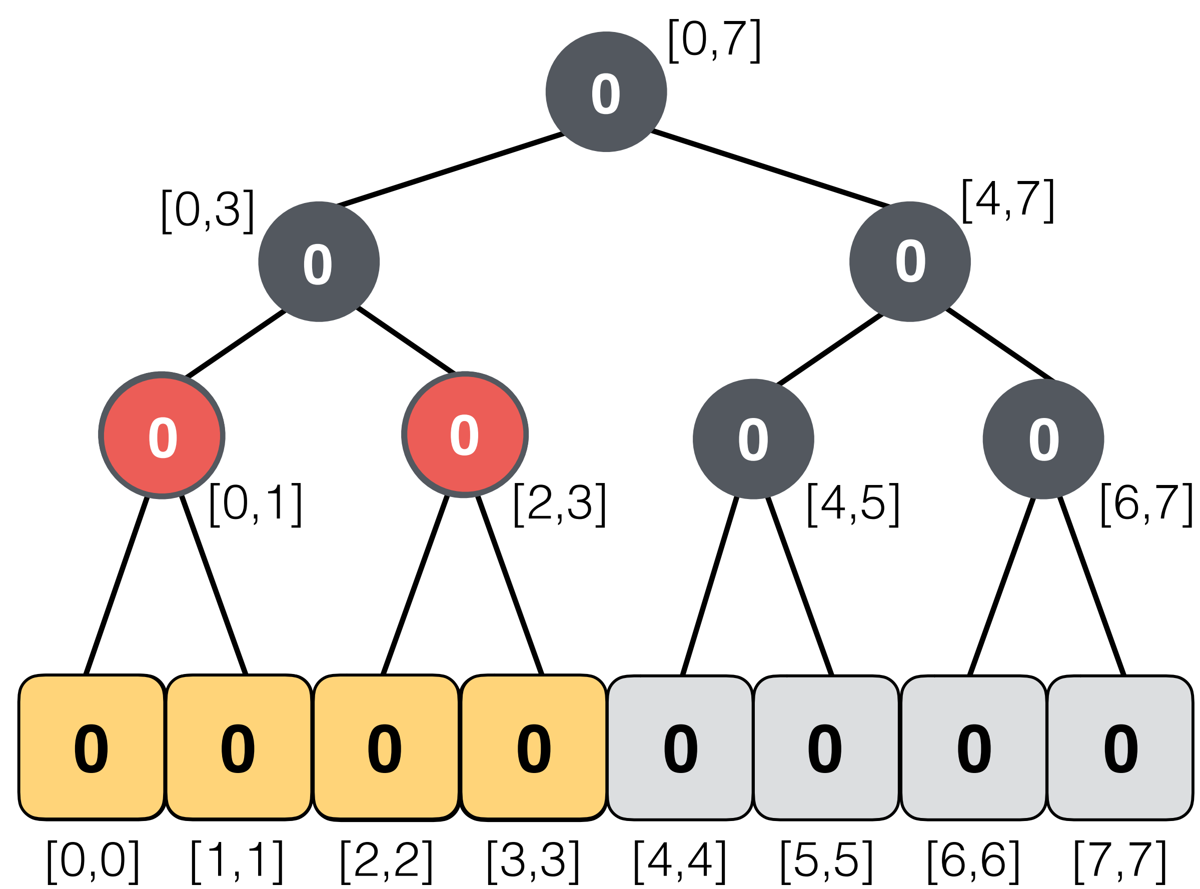
Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
$\boxed{\texttt{rmq}(3,5) = ?}$



Segment Tree

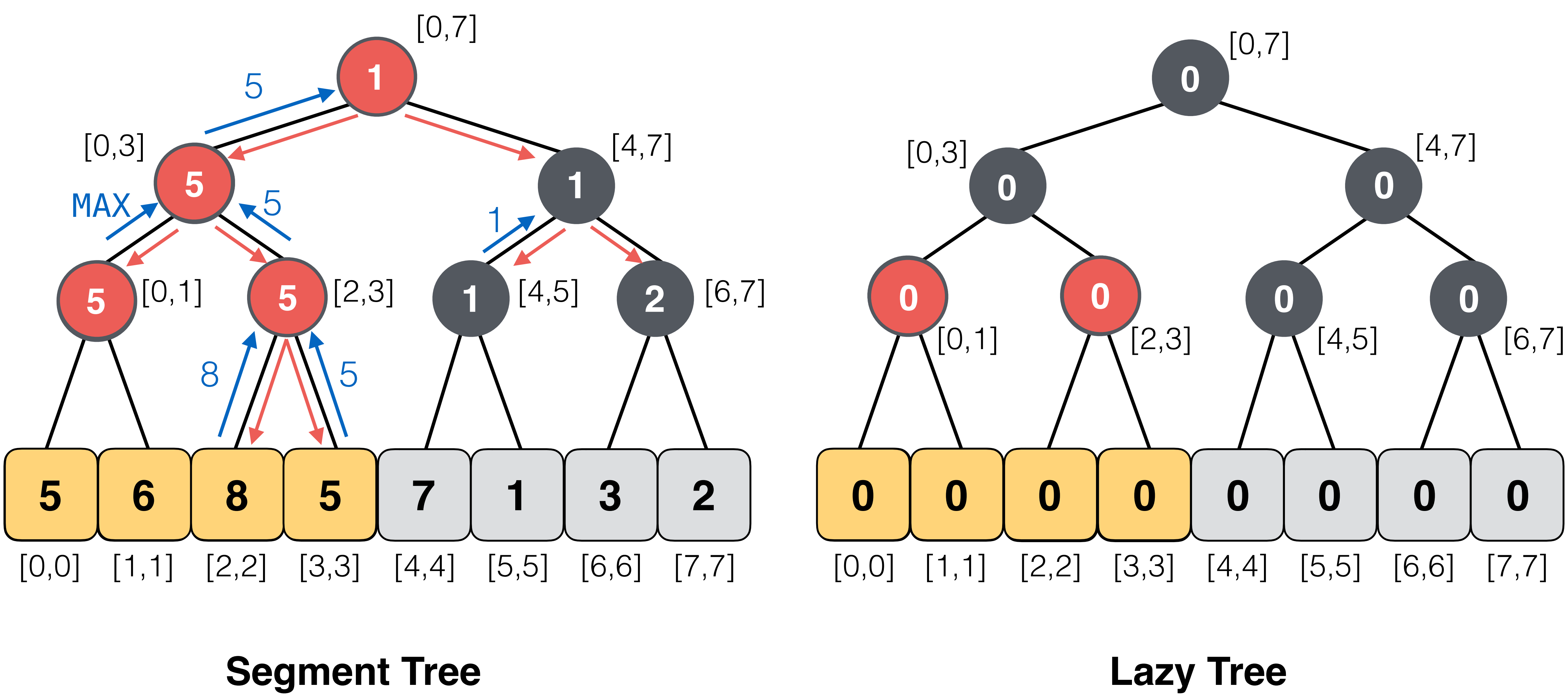


Lazy Tree

Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

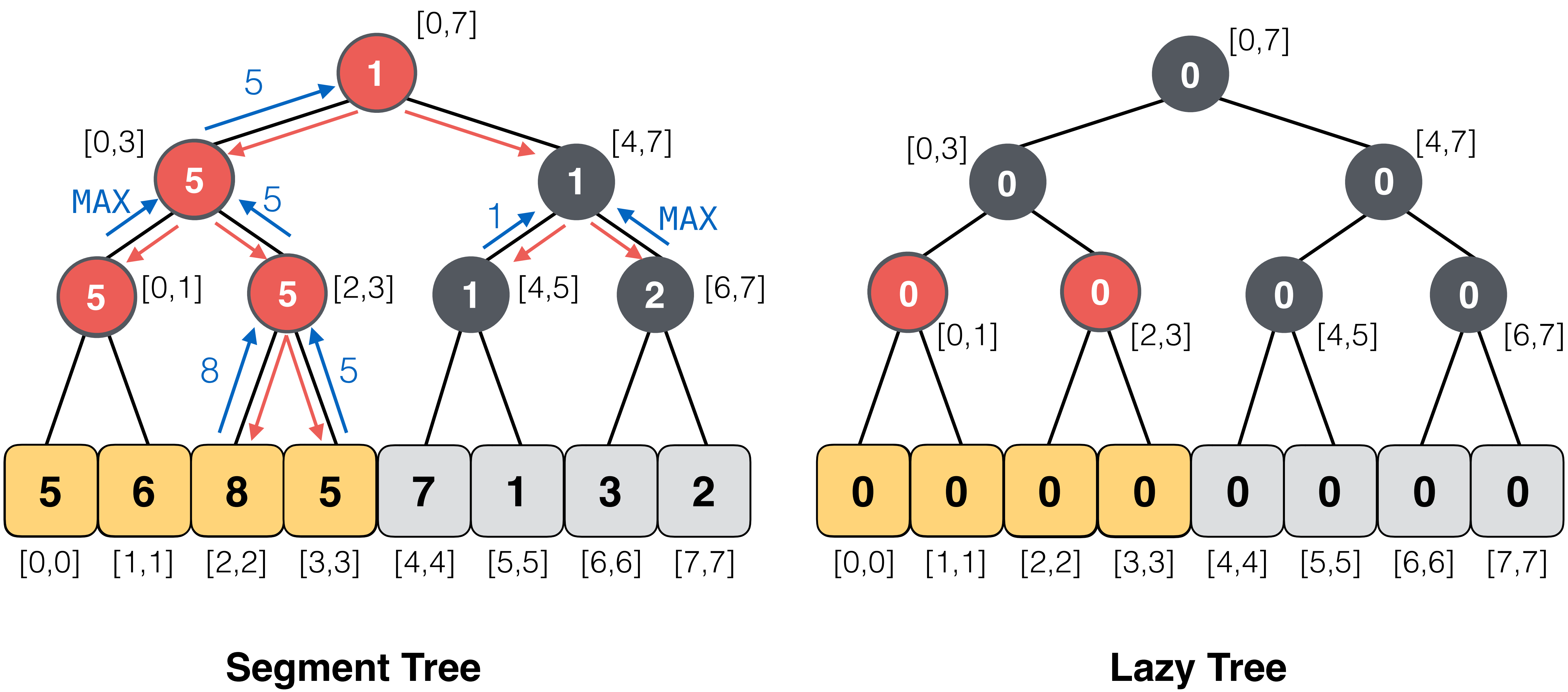
```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

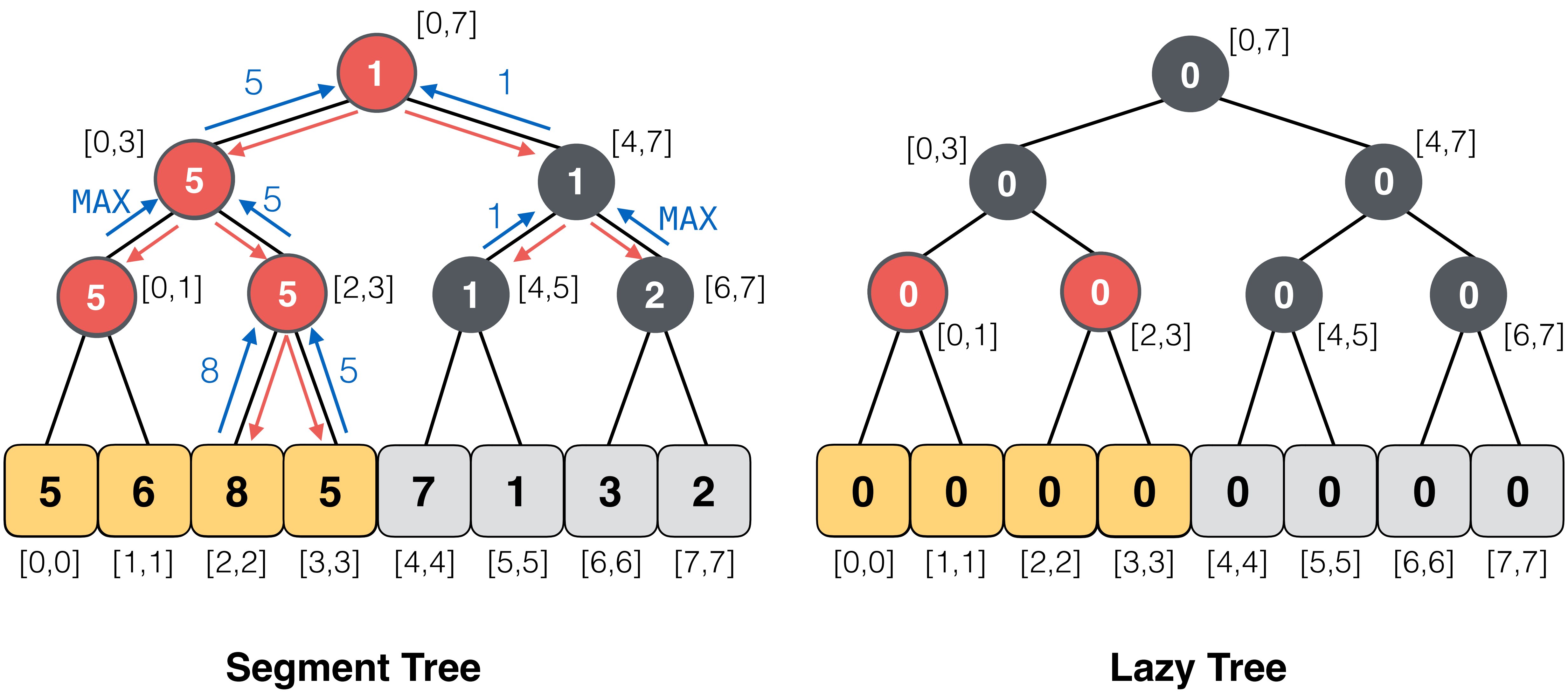
```
update_range(0,3,3)
update_range(0,3,1)
update_range(0,0,2)
rmq(3,5) = ?
```



Lazy Propagation in Segment Trees

Avoid going down to the leaves and then up updating the internal nodes.
Only update when needed.

`update_range`(0,3,3)
`update_range`(0,3,1)
`update_range`(0,0,2)
`rmq`(3,5) = ?



Exercises

Implement lazy propagation and test the difference in running time for a mix of updates/queries.

http://www.geeksforgeeks.org/lazy-propagation-in-segment-tree/

http://www.cdn.geeksforgeeks.org/segment-tree-set-1-sum-of-given-range/

References

Full segment tree code and benchmark at:
https://github.com/rossanoventurini/CompetitiveProgramming/tree/master/code/segment_trees

Video lectures:
https://www.youtube.com/watch?v=ZBHKZF5w4YU&list=PLrmLmBdmIlpv_jNDXtJGYTPNQ2L1gdHxu&index=22
https://www.youtube.com/watch?v=xuoQdt5pHj0&index=23&list=PLrmLmBdmIlpv_jNDXtJGYTPNQ2L1gdHxu